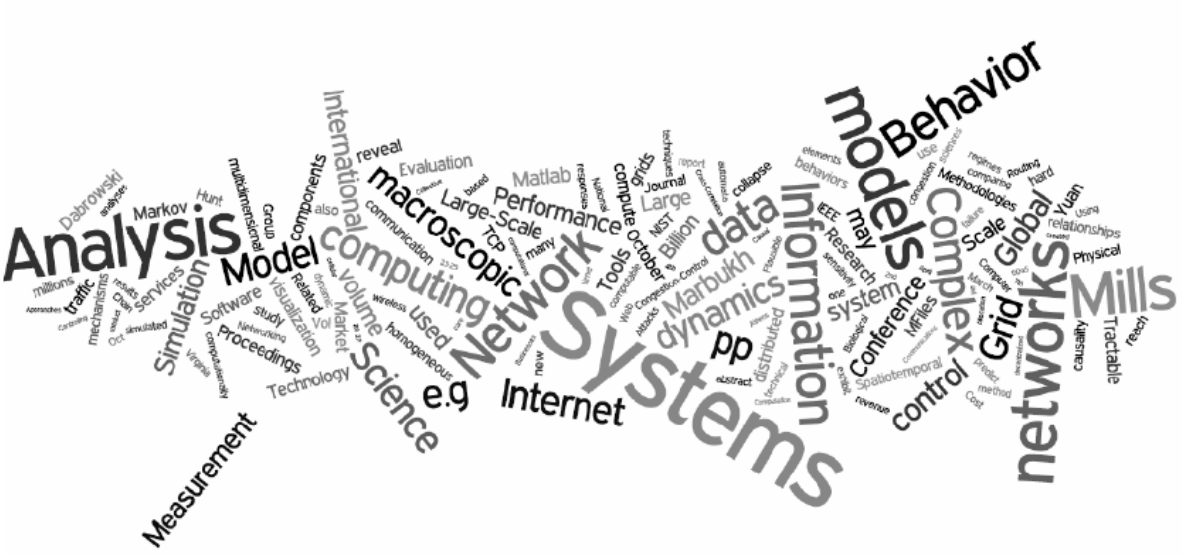
NIST Special Publication 500-282

Study of Proposed Internet Congestion Control Mechanisms

Kevin L. Mills
James J. Filliben
Dong Yeon Cho
Edward Schwartz
Daniel Genin



National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 500-282

Study of Proposed Internet Congestion Control Mechanisms

Kevin L. Mills
James J. Filliben
Dong Yeon Cho
Edward Schwartz
Daniel Genin
Information Technology Laboratory

May 2010

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick Gallagher, Director

National Institute of Standards and Technology Special Publication 500-282
Natl. Inst. Stand. Technol. Spec. Publ. 500-282, 534 pages (May 2010)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 2010

For sale by the Superintendent of Documents, U.S. Government Printing Office
Internet: bookstore.gpo.gov — Phone: (202) 512-1800 — Fax: (202) 512-2250
Mail: Stop SSOP, Washington, DC 20402-0001

Acknowledgments

The foundation for this work began in 2000, when Jian Yuan, now an associate professor in the department of electronic engineering at Tsinghua University, arrived at the National Institute for Standards and Technology (NIST) to begin four years of collaboration with Kevin Mills. Dr. Yuan introduced to NIST the idea of viewing large communication networks as complex systems, and worked together with Dr. Mills to explore the implications of investigating the Internet as a complex system. During that time, Dr. Yuan developed a cellular automata model of an Internet-like network, which served as the basis from which MesoNet was created. MesoNet is the reduced parameter discrete-event simulator used in the experiments reported in this study. The collaboration between Drs. Yuan and Mills led to creation of two proposals: (1) to develop a project on measurement science for complex information systems and (2) to formulate a complex systems research program within the Information Technology Laboratory (ITL) at NIST.

In 2006, Dr. Mills submitted a project proposal to an internal NIST competition, known as innovations in measurement science (IMS), aiming to gain approval from the NIST director to apply measurement science to complex information systems. William Jeffrey, the NIST director at that time, supported the project proposal, which also found favor with Cita Furlani, director of ITL. In parallel, a group of ITL researchers, including Chris Dabrowski, Fern Hunt, Vladimir Marbukh and Kevin Mills, proposed the creation of an ITL program to study complex systems in general. This second proposal was well received by the management team within ITL, which founded a complex systems (CxS) program and recruited Sandy Ressler as the program manager. Sandy has provided steadfast support to the work documented in this report. In fact, he added resources to the project so that we could explore the use of interactive visualization as a tool to reveal global behavior in large distributed systems. Further, he regularly attended the biweekly project meetings and provided many useful suggestions for improving the content and presentation of the work.

In 2007, while we were exploring potential challenge problems for a measurement science of complex systems, Vladimir Marbukh suggested that we consider the collection of congestion control algorithms being proposed to augment or replace congestion control procedures within the standard transmission control protocol (TCP), which operates within every computer connected to the Internet. Ultimately, Vladimir's suggestion led to the study documented in this report. Through the Internet Congestion Control Research Group (ICCRG) of the Internet Research Task Force (IRTF), we contacted the community of researchers involved in developing future congestion control mechanisms for the Internet. Several members of the ICCRG had performed empirical studies of selected congestion control algorithms operating within a small topology. These empirical results proved invaluable in verifying our models of individual congestion control algorithms. Wesley Eddy and Michael Welzl, co-chairs of the ICCRG, encouraged us to share our work with the group and provided ample agenda time at several meetings for us to first present our methods and later our findings. We also appreciate the general support for our work from researchers within the ICCRG community, as well as several researchers within the Transport Modeling Research Group (TMRG) of the IRTF.

In addition to support from Internet researchers, we also benefited from the work of several colleagues who made temporary visits to NIST. For example, Brittany Devine worked on the project for three months under the Summer University Research Fellowship (SURF) program administered by the National Science Foundation (NSF). Brittany contributed prototype code to make measurements on flow groups, as explained in Chapter 8. Edward Schwartz, one of the study authors, also worked on the project for three months under the SURF program. Ed, now a doctoral student at Carnegie Mellon University, developed the initial models of CTCP, FAST, HSTCP, H-TCP and Scalable TCP that were used within this study. Ed also conducted some preliminary experiments comparing CTCP, FAST and standard TCP under select scenarios. Cedric Houard contributed two years of work to the project as a guest scientist visiting from France. Specifically, Cedric developed DiVisa, an interactive system for visualizing multidimensional data.

Supported by these many valuable contributions, the authors produced the current report to document a set of methods, models, experiments and findings, which were produced largely over the period of 2006-2009. The study encompasses a wide scope and substantial breadth, which could not be documented completely in less than 500 pages. We are indebted to Christopher Dabrowski and Stefan Leigh, two NIST colleagues who made careful, independent readings of the entire report, and who also provided many useful suggestions for improving the manuscript. The review provided by Dabrowski and Leigh upheld the highest standards of NIST requirements for internal review prior to publication. Of course, in a report of such length, scope and depth, errors undoubtedly remain. Responsibility for such residual errors lies squarely at the feet of the authors.

Note: the document cover and chapter dividers contain images generated from the study text with http://www.worldle.net/

Executive Summary

When first introduced in the early 1980s the Internet appeared to be an interesting engineering curiosity, providing resource sharing and data communication services to support scientific researchers. Beginning with the introduction of the World Wide Web (circa 1995), the fundamental data communication services provided by the Internet transformed into a global infrastructure for commerce, education and entertainment. Later developments (circa 2005) built upon so-called Web Services to provide innovative social networking technologies that citizens the world over can use to organize and collaborate around collective interests. Along the way, innovative cell phones and other handheld devices were introduced and integrated with the Internet and the Web to extend available information and interaction services to people anytime, anywhere. The future promises a globe interconnected by large, distributed information networks, where people routinely interact in new and unexpected ways. Realizing this future relies in large part on our ability to understand and engineer globally distributed systems of interconnected hardware and software components, including their use by people. At present, society is technologically capable of building such systems but lacks the fundamental knowledge required to understand and predict macroscopic behaviors that may arise from complex interactions as such systems evolve with the addition of new technologies and new patterns of use. A similar lack of knowledge may impede progress with respect to other large systems engineered by society. For these reasons, researchers in the Information Technology Laboratory (ITL) of the National Institute of Standards and Technology (NIST) embarked upon a measurement science based program of research for complex systems. The NIST Complex Systems Program aims to investigate and evaluate methods and tools that system designers might adopt to improve scientific understanding of large distributed systems, such as information networks, electric grids and transportation webs.

As part of the NIST Complex Systems Program, this special publication investigates and evaluates modeling and analysis methods that can be applied to predict and understand macroscopic behavior and variations in user experience that may arise as engineers introduce changes in software components into a large information network, such as the Internet. The Internet consists of millions (someday billions) of interconnected components that may be changed independently. For example, every time vendors of major operating systems introduce software updates, millions of users download new software modules into computers connected to the Internet. As another example, users may download software to support new functions, such as social networking or distributed gaming. At the current state of the art, system designers lack techniques to predict global behaviors that may arise in the Internet as a result of interactions among existing and altered software components. Similarly, hardware faults and unexpected usage patterns may occur within the Internet. Engineers have insufficient methods and tools available to forecast global behaviors and resulting effects on individual users. The study described in this special publication aims to improve existing knowledge about a range of methods and tools that could be applied to understand and predict behavior in such complex information systems.

To give our study a concrete context, we selected a challenging problem of current interest and relevance for the Internet at large. Specifically, we study the likely consequences for macroscopic behavior and for individual users should any of several

proposed mechanisms be introduced to augment or replace congestion control procedures in the standard transmission control protocol (TCP), which is currently deployed to regulate the rate of information transfer among computers connected to the Internet. Congestion control procedures allow individual computers to measure available capacity on network paths and to attempt to transfer information as quickly as possible. Because conditions vary with time, congestion control procedures also enable detection of congestion that may arise as too many computers attempt to use a network path. Upon detecting congestion, TCP first substantially slows a computer's rate of transfer and then attempts to slowly increase the rate. Researchers have predicted that the standard TCP congestion control procedures will inhibit users from realizing increased transfer speeds as capacity expands in the Internet backbone from the current rate of 10 Gigabits per second (Gbps) to 100 Gbps and beyond. For this reason, various researchers have proposed changes to the congestion control procedures implemented in standard TCP. At the current state of the art, such proposed changes have been studied on individual long-lived flows using analytical methods and also studied using simulation and empirical measurements in small topologies with limited types of data traffic. Though researchers and engineers would like to predict the effects of such changes on macroscopic behavior and on individual users, no techniques are currently available to make such extrapolations to large, fast topologies transporting hundreds of thousands of simultaneous data transfers of various sizes under a wide range of network conditions. The study documented in this special publication describes and evaluates modeling and analysis techniques applied to make such extrapolations for seven proposed alternatives to standard TCP congestion control procedures.

We apply techniques often used by scientists at NIST when studying physical systems. First, we propose an abstract simulation model, representing a data communications network (including TCP) with only 20 parameters, as compared with the hundreds of parameters typically used in detailed Internet simulators. Second, we adopt 2-level-per-factor experimental designs, which consider each parameter at only two values, as compared with the billion or so values that each parameter could possibly take on. Third, we leverage orthogonal fractional factorial (OFF) experiment designs that enable us to model a sparse but balanced set of parameter combinations spread widely throughout the space of possible combinations. Reducing the number of parameters, parameter levels and combinations enables feasible simulation of large networks under a wide range of conditions. Third, we use a variety of statistical analysis and visualization techniques designed to explore multidimensional data sets. Fourth, we use detailed analyses of time series as required to supplement findings from statistical analyses. We demonstrate that our proposed combination of modeling and analysis techniques allows us to predict the influence of seven proposed congestion control mechanisms on macroscopic network behavior and individual user experience.

In summary, this special publication contributes to current knowledge about modeling and analysis techniques for complex information systems and also contributes to the body of knowledge surrounding proposals for improving congestion control mechanisms considered for deployment in the Internet. Six specific contributions improve current knowledge regarding techniques to understand and predict behavior in complex information systems. First, we summarize the current state of the art in modeling and analysis of communication networks and we identify several hard problems

that inhibit the study of large, fast networks. Second, we propose an approach to construct simulation models with a reduced parameter space. As a corollary contribution, we identify and explore some alternative, promising modeling approaches, including fluid flow models and hybrid models, which combine quantized time calculations with discrete events. Third, we describe and demonstrate how two-level OFF experiment designs can be applied to reduce the number of parameter combinations that must be considered, while yielding maximum information from available simulation resources. Fourth, we describe and apply a variety of analysis and visualization techniques for interpreting multidimensional data. We first use these techniques to conduct sensitivity analyses of our simulation model and then apply the techniques to compare congestion control mechanisms. Fifth, we evaluate our proposed modeling and analysis techniques, discussing the strengths and weaknesses of various methods and identifying those methods that proved most effective for our study. Sixth, we outline future research needed to improve upon the methods we evaluated. Our six contributions enhance understanding of methods and tools available to designers of complex systems.

Four specific contributions add to the body of knowledge surrounding proposals for improving Internet congestion control. First, we characterize likely macroscopic behavior and user performance for seven proposed alternatives to TCP congestion control procedures. In doing so, we reveal that proposed improvements to TCP congestion control would benefit individual users under a specific combination of circumstances unlikely to arise very often in the general Internet. We also identify some cautionary findings with respect to various congestion control mechanisms we study. Second, we identify key behavioral characteristics to be considered when comparing congestion control mechanisms. We found these characteristics through analyses of experiment data, rather than through a priori analyses. Part of our method was to collect as much measurement data as possible and then to use statistical techniques (e.g., correlation and principal components analyses) to identify those measures representing different facets of system behavior. Then, given selected measures, we could determine the key factors influencing macroscopic behavior and user experience. Previous studies of congestion control mechanisms did not reflect these key factors. Third, we identify and compare the main differences among the congestion control mechanisms we studied. We show that, for the key behavioral factors we identify, one of the seven mechanisms we studied fares better than the others. Fourth, we suggest some future research directions related to Internet congestion control. Our four contributions should help researchers to better understand the problem space surrounding congestion control in the Internet.

While the current study is quite comprehensive with respect to the study of large distributed systems, we have certainly not covered every method and technique that might prove useful. For example, a related project in the NIST Complex Systems Program is investigating how Markov models, coupled with perturbation analysis, Eigenanalysis and graph theory, can be used to identify specific aspects of system designs that might significantly degrade performance when subjected to failures. Further, while some of the methods we applied appear quite effective in the context of Internet congestion control, we also need to demonstrate effectiveness in other applications. In summary, this study makes substantial contributions to methods for modeling and analyzing complex information systems and also provides significant information to the community of researchers studying Internet congestion control.

Table of Contents

List of Figures

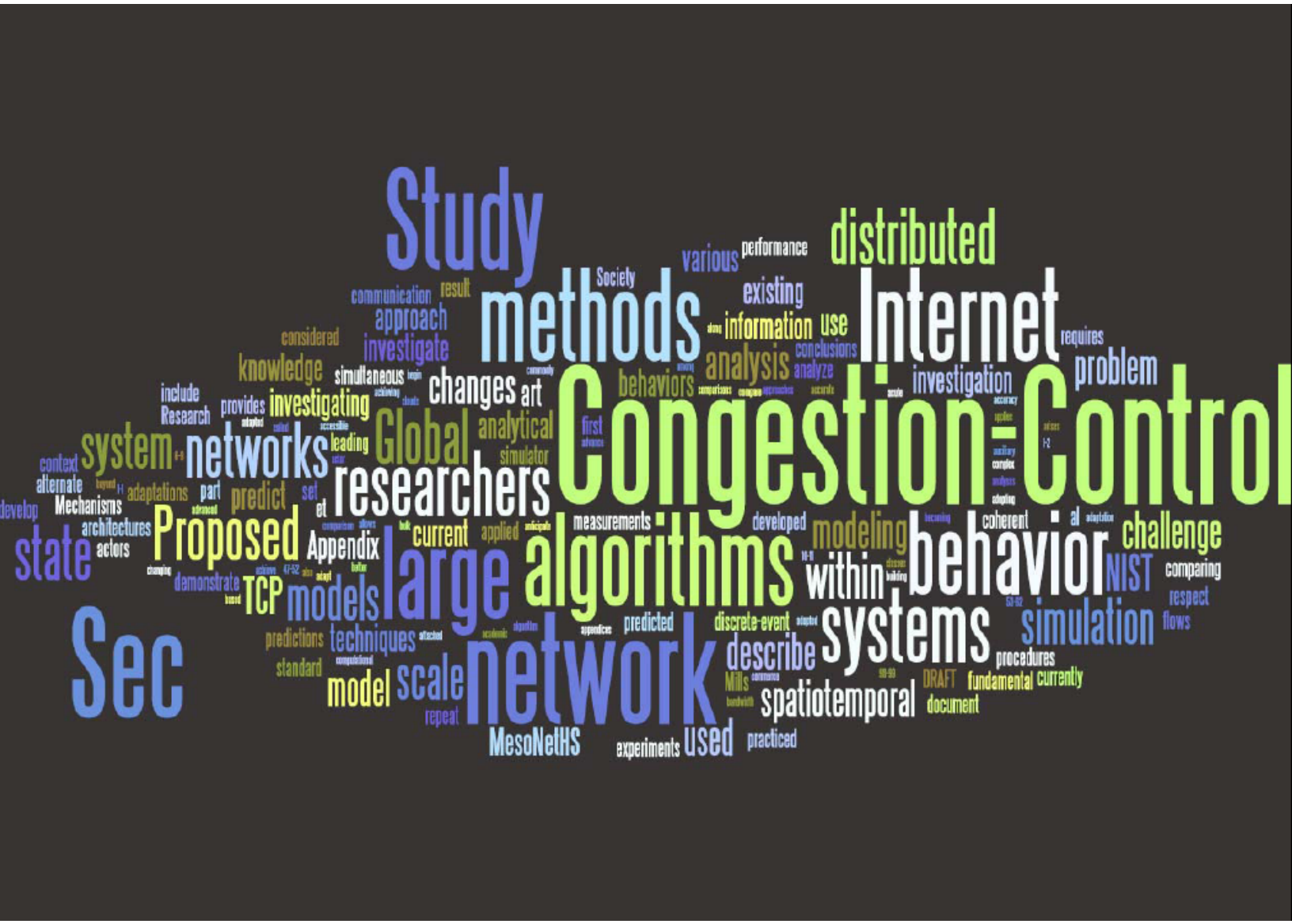
List of Tables

Table of Acronyms

Acronym	Definition
ACK	Acknowledgment
AvFSWO	Average File Size for Web Objects
AvThT	Average Think Time
AWND	Advertised Window
BIC	Binary Increase Congestion control
BRS	Backbone Router Speed
C	Capacity
CA	Cellular Automata
CAC	Correlation Analysis and Clustering
CPU	Central Processing Unit
CRTO	Current Retransmission Time Out
CTCP	Compound TCP
CWND	Congestion Window
DES	Discrete-Event Simulation
DT	Data segment (or packet)
DWND	Delay Window (used by CTCP)
FAST	Fast Active Queue Management Scalable TCP
FAST-AT	Fast Active Queue Management Scalable TCP with Alpha Tuning
GB	Gigabytes (Giga denotes billion)
Gbps	Gigabits per second (Giga denotes billion)
GENI	Global Environment for Network Innovation
GHz	Giga Hertz (Giga denotes billion)
HSTCP	High Speed TCP
HTCP	Hamilton TCP
ICCRG	Internet Congestion Control Research Group
IP	Internet Protocol
IRTF	Internet Research Task Force
ISP	Internet Service Provider
ITL	Information Technology Laboratory
Kbytes	Kilobtyes
Kbytes	Kilobytes
MB	Megabytes
Mbps	Megabits per second
Mbytes	Megabytes
MCMP	Multicore Multiprocessor
NAK	Negative Acknowledgment
NIST	National Institute of Standards and Technology
NSF	National Science Foundation
OFF	Orthogonal Fractional Factorial
P2P	Peer-to-Peer

PC	Personal Computer
PC	Principal Component
PCA	Principal Components Analysis
PDM	Propagation Delay in Milliseconds
POP	Point Of Presence
ppts	packets per time step
pps	packets per second
p/ms	packets per millisecond
PrFH	Probability of a Fast Host
PrLF	Probability of a Larger File
QSA	Queue Sizing Algorithm
RDist	Receiver Distribution
RTO	Retransmission Time Out
RTT	Round Trip Time
RWND	Receiver Window
SD	Standard Deviation
SDist	Source Distribution
SFSR	Scaling For Sources and Receivers
SLX	Simulation Language with eXtensibility
SQR	Square Root
SQRT	Square Root
SRTT	Smoothed Round Trip Time
SST	slow start threshold
STCP	Scalable TCP
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WEB	World-wide Web

Chapter 1 – Introduction



1 Introduction

Society is becoming increasingly reliant on large networked information systems for commerce, communication, education, entertainment and government. "[Despite] society's profound dependence on networks, fundamental knowledge about them is primitive. [Global] communication…networks have quite advanced technological implementations but their behavior under stress still cannot be predicted reliably…. *There is no science today that offers the fundamental knowledge necessary to design large complex networks [so] that their behaviors can be predicted prior to building them*." [104] This lack of knowledge grows more acute as society moves toward service-oriented architectures [102-103] that deploy software, platforms and infrastructure as distributed services accessible through networks.

Why are large distributed systems so difficult to predict? Such systems exhibit global behavior that arises from independent decisions made by many simultaneous actors, which adapt their behavior based on local measurements of system state. As a result of actor adaptations, global system behavior may change, influencing subsequent measurements, and leading to further adaptations. This continuous cycle of measurement and adaptation produces a time-varying global behavior that drives the performance experienced by individual actors within spatiotemporal regions of a large distributed system. Thus, to truly understand and predict behaviors in such systems requires techniques to model and analyze designs at large scale. Such techniques are currently beyond the state of the art, as practiced by network researchers.

As part of a team of researchers [105] at NIST, we are investigating methods to model and analyze distributed systems, such as the Internet, computational grids, service-oriented architectures and computing clouds. As part of this investigation, the study reported here develops, applies and evaluates a coherent set of modeling and analysis methods for distributed information systems of large spatiotemporal scale. The methods are adapted from techniques often applied by NIST scientists to study physical systems.

In this study, we develop methods to investigate global system behavior within the context of a challenge problem: comparing some proposed changes to the standard congestion control algorithm [9-10] for the Internet. Congestion control procedures are implemented as part of the transmission control protocol (TCP) that operates within every computer attached to the global Internet. Numerous researchers [46-51] have forecast changes in relationships among bandwidth and propagation delay as the speed of network links increases. These researchers predict that the current version of TCP will prove inadequate, leading to substantial underutilization in network resources and preventing end users from achieving high transfer rates. Such predictions have stimulated researchers to propose alternate congestion control algorithms [52-61] intended to achieve higher network utilization and better user performance. Evaluating the implications of adopting proposed changes to TCP congestion control procedures requires investigating global behaviors that result when such changes are deployed on a large scale throughout an Internet-like network. The current study provides such an investigation.

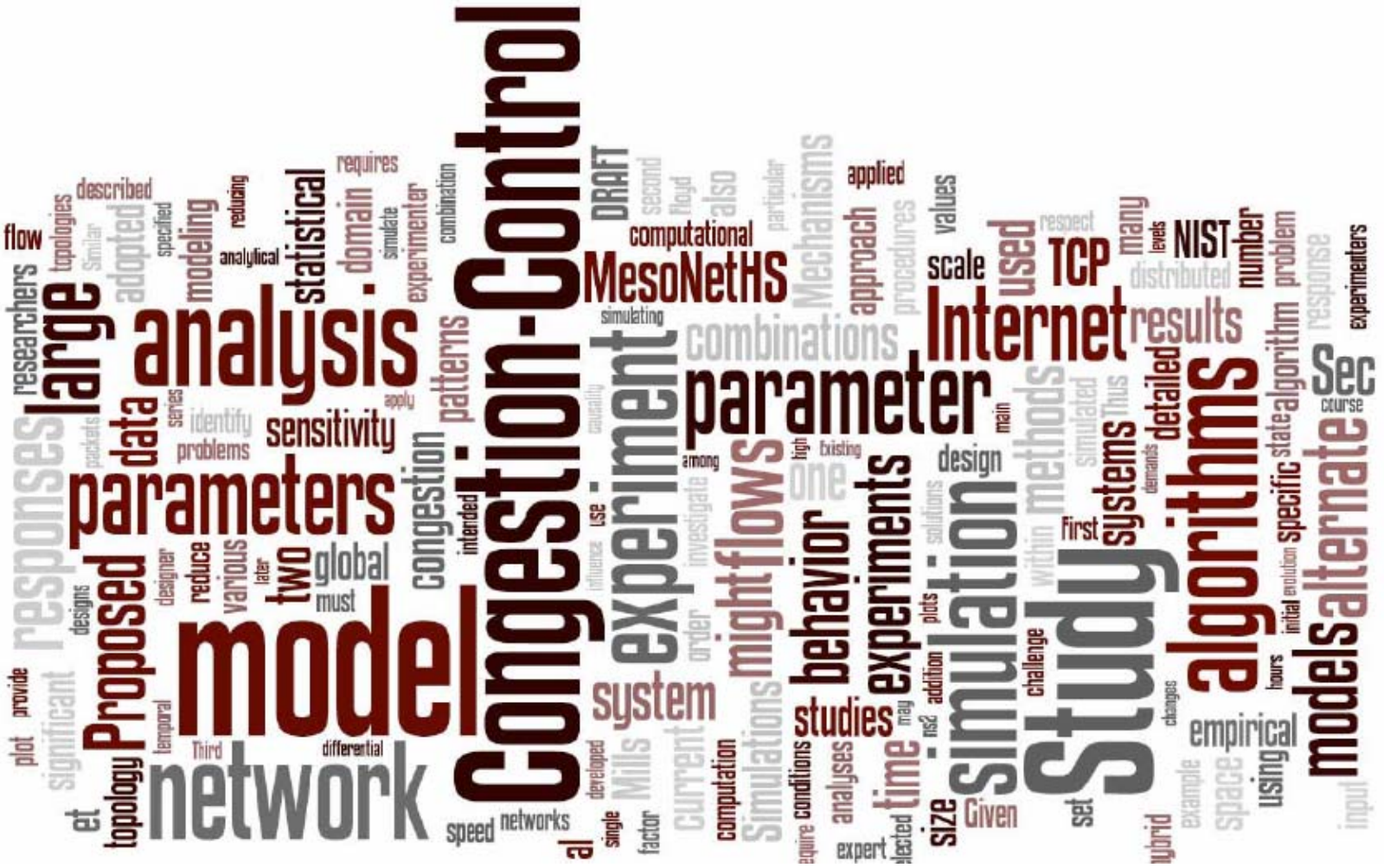
We begin (in Chapter 2) with a discussion of the challenge problem and the current state of the art with respect to investigating proposed Internet congestion control algorithms. We outline various approaches that we considered for modeling and analysis

and we describe the approach we selected. We introduce five hard problems we needed to solve in order to implement our approach and we discuss the solutions we adopted. In Chapter 3, we describe MesoNet, a medium scale, discrete-event simulation model that we created for use in this study. MesoNet allowed us to expose candidate congestion control algorithms to a wide variety of network conditions. We subjected MesoNet to sensitivity analyses, as documented in Chapter 4 and in Appendix C. These sensitivity analyses helped us to gain confidence that MesoNet provides a suitable model for TCP networks, and also enabled us to identify the most important parameters influencing MesoNet behavior. As part of our sensitivity analyses, we employ a NIST-developed, 10-step graphical analysis process, which is described in Appendix D. In Chapter 5, we explain our models for various congestion control algorithms and we document key empirical comparisons used to verify model correctness. The bulk of the study consists of six experiments, which we describe in Chapters 6 through 9. As we discuss in Chapter 2, these experiments were not constructed as an integral campaign, but rather arose through a process of iterative refinement, where findings from previous experiments suggested useful directions for subsequent experiments. We first compare (Chapter 6) congestion control regimes in a large, fast network simulation and then repeat the comparison (Chapter 7) in a simulated network with smaller size and slower speeds. In Chapter 8, we enlarge the traffic classes considered, while comparing the congestion control algorithms in a network where some flows use standard TCP and some use alternate algorithms. In Chapter 9, we repeat an experiment from Chapter 8 but in a larger, faster simulated network, where theorists suggest alternate congestion control algorithms could provide best advantage. Taken together, these experiments compare the behavior of seven congestion control algorithms under a wide range of simulated conditions. We generate sufficient information to draw some conclusions in Chapter 10 about the congestion control algorithms. Chapter 10 also provides an evaluation of the methods that we developed and applied. We include some appendices to document auxiliary investigation of analytical (Appendix A) and hybrid (Appendix B) models of TCP networks.

This study may interest two different audiences: (1) those seeking to understand and evaluate methods to model and analyze behavior in large, distributed information systems and (2) those aiming to compare proposed changes in algorithms for the Internet. Readers in the first audience can expect to learn about various modeling, experiment design and statistical analysis techniques applied to study dynamics in complex systems. In addition, such readers may benefit from our findings with regard to the strengths and weaknesses of the techniques we applied. Readers in the second audience can expect to learn how to model a data communications network with a manageable set of parameters. In addition, such readers may benefit from learning how we let measurement data (rather than preconceived metrics) drive our comparison of alternative congestion control algorithms. Mindful of these two different audiences, we attempt to provide a sufficient level of explanation to engage every reader. We explain our modeling and analysis methods in detail so that networking experts can follow our methods. And we provide sufficient tutorial information to allow those readers who are not networking experts to follow our challenge problem and related findings. Where appropriate, we also provide references to additional sources where readers in each audience can pursue more information.

We can summarize the contributions of this study along several lines. First, we define and demonstrate a coherent set of modeling and analysis methods that can be used to investigate behavior in distributed systems of large spatiotemporal scale. The methods we develop represent an advance in the state of the art, as currently practiced by network researchers. Second, we evaluate our modeling and analysis methods in the context of a challenge problem that investigates behavior of various proposed Internet congestion control algorithms. The challenge problem is of current interest to industrial and academic researchers within the Internet Congestion Control Research Group (ICCRG) of the Internet Research Task Force (IRTF). Third, we provide conclusions and recommendations with respect to the congestion control algorithms that we study. We demonstrate that our methods lead to insights that have not been obtained using existing methods. Fourth, we describe a medium-scale, discrete-event network simulator that we developed for our study. The simulator, called MesoNet, can be efficiently parameterized and allows feasible simulation of high-speed networks transporting hundreds of thousands of simultaneous flows. The most commonly used network simulators are incapable of supporting such large-scale models. Fifth, we suggest an approach that might improve the accuracy of existing analytical models for Internet congestion control algorithms. We anticipate future work to include improved analytical models within existing fluid-flow simulation frameworks in an effort to obtain accurate predictions regarding spatiotemporal behavior in large networks.

Chapter 2 – Method and Related Work



2 Method and Related Work

The work described in this report supports an overarching goal to develop and evaluate a coherent set of methods that can be applied to understand behavior in large distributed systems, such as the Internet, computational grids, service-oriented architectures and computing clouds. Large distributed systems may exhibit emergent behaviors, which are global behaviors arising from independent decisions made by many simultaneous actors, which adapt their behavior based on local measurements of system state. As a result of actor adaptations, system state shifts, influencing subsequent measurements made by the actors, which leads to further adaptations. This continuous cycle of measurement, adaptation and changing system state produces a time-varying (emergent) global behavior that influences performance experienced by individual actors within specific spatiotemporal regions of a large distributed system. For this reason, any proposed changes in decision algorithms taken by actors must be examined within the context of a large spatiotemporal scale in order to predict the effects of such algorithms on overall system behavior, as well as the resulting implication for individual actors.

In this study, we develop methods to investigate global system behavior within the context of a challenge problem: comparing selected proposed changes to the standard congestion control algorithm [9-10] for the Internet. As we show later, in Chapter 10, using our methods we were able to draw conclusions (1) about likely network-wide behaviors and user experiences that may arise if the Internet adopts any one of the algorithms we studied and (2) about the efficacy of the methods we used. In this chapter, we introduce the challenge problem, describe the current state-of-the-art techniques used to address the problem and outline a proposed advance in the state of the art. We consider some approaches that might be adopted to achieve our intended improvement in practice and then we explain the approach we adopted for the current study. We identify five hard problems we had to solve to develop our approach and we discuss some possible solutions to the problems and identify the solutions we adopted for the current study. We conclude with an argument that the methods we develop and apply in the current study should be generally applicable to a wide array of large distributed systems.

2.1 Challenge Problem

The fundamental design of the Internet protocol suite [3] assumes that network elements, such as routers, are relatively simple – receiving, buffering and forwarding packets among connected links and dropping packets when buffers are insufficient to accommodate arriving packets. Under this assumption, computers connected to the Internet must implement decision algorithms to pace the rate at which packets are injected into the network. Such decision algorithms, known typically as congestion control mechanisms, operate independently for each network flow between a source and receiver. The overall network, with a goal of achieving satisfactory service and a fair distribution of resources among all simultaneously active flows, relies upon each network source to measure congestion and then adapt the rate at which the source injects packets into the network – injecting faster when congestion is low and slower when congestion is high. Thus, congestion in the Internet is an emergent property of the simultaneous operation of many independent sources.

In current practice, congestion control mechanisms are implemented as part of the transmission control protocol (TCP) that operates within every computer attached to the global Internet. While TCP congestion control procedures have proven quite successful [2] at achieving desired global properties, numerous researchers [46-51] have postulated potential changes in relationships among bandwidth and propagation delay as the speed of network links increases toward 10s and 100s of gigabits per second (Gbps). Under the envisioned circumstances, researchers predict that TCP congestion control procedures will prove insufficient, leading to substantial underutilization in network resources and preventing end users from achieving high transfer rates, potentially reaching or surpassing one Gbps. These predictions have stimulated researchers to propose alternate congestion control procedures [52-61] that might achieve higher network utilization and better user performance as network speeds increase. Given the increasing number of proposals, interest is growing [62-68] in developing procedures to fairly and effectively evaluate properties of the various proposals. Evaluating the implications of adopting proposed changes to TCP congestion control procedures requires investigating global behaviors that result when such changes are deployed on a large scale throughout an Internet-like network. This is the challenge problem we tackle within the current study.

2.1.1 Current State of the Art

As part of proposing changes to standard congestion control procedures, researchers typically model, simulate and implement prototypes and then explore how candidate congestion control mechanisms might affect the Internet and its users. In general, researchers investigate candidate algorithms using three primary approaches: (1) empirical studies, (2) simulation studies and (3) analytical studies. In this section we outline and critique the current state of the art with respect to these approaches.

2.1.1.1 Empirical Studies. A fundamental approach to studying proposed changes to TCP congestion control procedures involves deploying a few computers, acting as sources and receivers, connected to Gigabit Ethernet switches in a dumbbell topology, where the network links are represented by a single computer that can be parameterized with specific bottleneck speed, buffer size and propagation delay. Several, typically two to ten, long-lived flows share the bottleneck path in the dumbbell topology and various measurements are made regarding traits such as fairness of resource allocation, responsiveness to changing conditions and link and buffer utilizations. Additional sources are often added to investigate the response of the proposed congestion control algorithms in the presence of background traffic, sometimes TCP flows and sometimes user-datagram protocol (UDP) streams. Usually the background traffic crosses the bottleneck link in an orthogonal direction to the long-lived flows. Several examples of such studies appear in the literature [65-68] and the basic approach is being considered by a group of researchers [62] intending to standardize procedures to characterize proposed changes to TCP.

Simple empirical studies have several merits. First, a topology involving few computing elements can be constructed conveniently in a laboratory setting. Further, the fundamental characteristics (speed, buffer size and propagation delay) of a bottleneck path can be reliably established to provide a suitable basis for head-to-head comparisons of alternate congestion control algorithms in identical, controlled situations. Third,

empirical studies investigate actual implementations of proposed algorithms as would be distributed in code used by computers on the Internet; thus, there is no room for modeling error. Of course, code bugs can exist; however, those code bugs, if undiscovered, would be actually deployed on the Internet. Fourth, as demonstrated in a recent study [67], empirical measurements in a simply topology can reveal behavioral properties that might well have significant implications. For purposes of our study, the major shortcoming of simple empirical studies is an inability to investigate the influence on global network behavior should proposed congestion control algorithms be adopted on a large scale. As we discuss below in Sec. 2.2.1, some researchers are investigating techniques to support configuration of larger empirical networks, which might be used to study global network behavior.

2.1.1.2 Simulation Studies. Another approach is to implement proposed congestion control algorithms within a simulation modeling framework and then to construct (or generate) simulated topologies representing large networks and conduct experiments to evaluate global behavior and user experience. In fact, many of the proposed congestion controls we investigate in the current study have been implemented within a widely accepted simulation framework, *ns2* [79].

Simulation provides a convenient vehicle for defining controlled experiments that can be used to compare alternate congestion control algorithms, head-to-head, with respect to potential effects on global network behavior. Further, using simulation, an experimenter can measure many network-wide properties that might be difficult or impossible to measure in a large, empirical network. Several simulation studies [46, 48, 54-56, 61, 64] have been conducted using the *ns2* framework, but all of these studies have simulated small topologies with a limited number of flows, perhaps up to hundreds. Simulations at such small scales are unlikely to reveal the influence of alternate congestion control algorithms on global network behavior. Large topologies must be simulated while simultaneously transporting up to hundreds of thousands of active flows. The computational and memory demands of *ns2* are significant, which discourages experimenters from attempting large simulations. We certainly decided that we could not achieve our goals using *ns2* simulations.

There exist many other network simulation frameworks [76-78, 80-83], both commercial and academic, that might be adopted. Most of these frameworks would need to be expanded to include simulations of the alternate congestion control algorithms. Still, at least one of these simulation frameworks has been used in an experiment transporting up to 10^5 TCP flows [69]. Unfortunately, configuring such a simulation experiment requires setting values for thousands of parameters, which must be managed by a database. To define a comprehensive set of experiments, such as we envisioned, would require significant, perhaps insurmountable effort. Further, we would need to specify settings for many parameters that require values but that would not necessarily be germane to our experiments. As we discuss below in Sec. 2.2.2, some researchers are investigating techniques to support faster simulations of large networks, which might be used to study global network behavior.

2.1.1.3 Analytical Studies. A third approach is to construct a model as a system of differential equations that represent network flows as fluids rather than as a sequence of

discrete packets. A fluid approximation is justified using an argument that when the number of packets flowing through routers is very large and when the packets move at very high speed then packet flows can be well approximated by a smoothly varying continuous data steam with an average flow rate. Continuous systems can be modeled with differential equations. Several researchers [107, 112, 114, 119, 122-127] have proposed differential equation models for standard TCP congestion control procedures.

Existing differential equation models for TCP exhibit some significant shortcomings for our purposes. First, existing models yield inaccurate results [112-113], which derive largely from difficulties in modeling the loss-estimation function in the differential equations. Existing models have tried estimating loss probability using an M/M/1/B queuing system; however, when considering many flows transiting a single router, such models give inaccurate results for many parameter combinations. Second, the difficulty of estimating loss probability increases with increasing complexity in network topology. For this reason, employing differential equations to model network flows in large topologies has received little attention. Third, current differential equation models treat only standard TCP congestion control procedures. For our purposes, such models must be augmented to include congestion control procedures for the set of alternate congestion control algorithms we studied. As we discuss below in Sec. 2.2.3, we are investigating techniques to model network flows with differential equations that have improved accuracy and that incorporate alternate congestion control procedures. When combined with fluid-flow simulators [73], our extended models might prove applicable to study global network behavior under various congestion control regimes.

2.1.2 Proposed Advance in the State of the Art

As described in the preceding overview, the current state of the art in modeling and analysis of distributed systems is limited to relatively small scales. We propose to define and apply modeling and analysis techniques that enable measurement and investigation of global behavior and actor experience in relatively large models of distributed systems. In this study, we explain and investigate our proposed modeling and analysis techniques in the context of comparing alternate congestion control algorithms suggested for use on the Internet. As a result of our investigation, we provide new insights into likely global behavior and user experience should the Internet deploy any of the alternate algorithms we study. Further, we characterize strengths and weaknesses of the modeling and analysis methods we use. Finally, we suggest additional directions for future investigations in modeling and analyzing global behavior in large distributed systems. We expect the methods illustrated in our study to advance the state of the art in modeling and analyzing large distributed systems because we believe our methods can be applied beyond the current study to consider other large-scale feedback processes, such as might arise in computational grids, computing clouds and service-oriented architectures.

2.2 Potential Approaches

In this section, we consider some potential approaches to achieve our goal to define and apply modeling and analysis techniques that enable measurement and investigation of the global behavior and actor experience in relatively large models of distributed systems. We discuss the possibility of expanding either empirical, simulation or analytical studies in order to consider larger systems than possible within the current state of the art.

2.2.1 Expanded Empirical Studies

In increasing numbers, researchers are investigating management frameworks that allow experimenters to configure collections of hardware and software components into specified, reproducible topologies that enable empirical experiments with distributed systems. Emulab [99], perhaps the earliest instance of such a framework, provides access to hundreds of configurable nodes made available to experimenters. Emulab has been replicated at several sites around the world. Perhaps inspired by Emulab and its clones, PlanetLab [101] has connected a network of sites across the globe that allocates hardware and software components for configuration into distributed-system topologies for controlled experiments. Similar management frameworks, though limited to controlling configurations within a single laboratory, have been developed by various university researhers [65, 97]. While most of these efforts include innovations with respect to system configuration and experiment control, the scale of topologies that can be constructed is limited to a few hundred nodes and often such a topology exhausts the resources of a given facility. In an effort to overcome such scaling concerns, some researchers have reported progress in emulating virtual topologies an order of magnitude larger than the available physical hardware [98].

Recognizing the limitations of current facilities for configuring topologies in support of experiments with large distributed systems, a community of researchers is pursuing GENI [100] (Global Environment for Network Innovations), a project sponsored by the National Science Foundation (NSF). GENI aims to create a virtual laboratory for exploring future distributed systems at scale. The GENI facility promises to provide a platform on which we could conduct empirical investigations into global implications of deploying various congestion control algorithms. Unfortunately, GENI is in early stages of development, so the timing is inopportune for our study. We hope that we can use GENI at some later time to validate findings from the experiments we conducted.

2.2.2 Expanded Simulation Studies

While explaining that simulation models hold a key position when attempting to understand behavior in large distributed systems, Paxson and Floyd [72] identify several impediments to simulating the Internet. First, the Internet is big; simulating a model at scale can require significant, perhaps impractical, computational resources. Second, the Internet is diverse with respect to administrative policies and technologies deployed. Third, the Internet is evolving in size, technologies, traffic patterns and applications. While these impediments might prove difficult to overcome, Paxson and Floyd discuss some possible coping strategies. First, they suggest that researchers search for invariants that can reduce the parameter space of the models. They identify two candidate invariants: (1) model session arrivals with a Poisson distribution and (2) model session sizes with heavy-tailed distributions such as log-normal or Pareto ($\alpha < 2$). Second, Paxson and Floyd suggest judiciously exploring the parameter space of a simulation model in order to identify parameters to which the model is sensitive. In a subsequent paper, six years later, Floyd and Kohler [70], critique the continued poor state of Internet modeling and admit that the research community does not know whether their models are valid. Floyd and Kohler prescribe four steps that could improve the situation. First, models should be limited in scope to the specific research questions under investigation in

particular studies. That is, the research community should abandon hopes of developing a single model of the global Internet. Second, any model used should be subjected to a sensitivity analysis in order to understand how parameter settings affect results. Third, with respect to important parameter settings, models should be compared against measurements in order to further increase confidence in real-world relevance. Fourth, a continuous program of measurements should be conducted with the aim of distinguishing between invariant and rapidly changing parameters in the real Internet. This enables models to be kept current and allows previous modeling studies to be placed in a temporal context.

The perceptive critique from Floyd and colleagues provides a context for considering current research into network simulation. Much of the current work revolves around developing improved simulation frameworks intended to provide a model of the global Internet. For example, a group of open-source developers is working on *ns3* [80], a replacement for *ns2* that is motivated by overcoming some perceived software limitations that make *ns2* difficult to use. Another group of researchers [82] is creating a parallel version of *ns2* in an effort to allow simulation of larger systems by dividing work among multiple processors. Constructing parallel network simulators has been a research topic for some time [69, 76, 83]; unfortunately, successful use of such simulators has proven elusive for network models. Some researchers [71] have begun to investigate hybrid models as a technique to reduce the resources required by network models, which might allow larger topologies to be simulated. Various other simulators [77, 78, 81] have been developed in order to teach university students about both networks and about software development.

While parallel simulators and hybrid models hold promise to increase the size of systems that can be simulated, few existing simulation research projects are motivated specifically to address the critique of Floyd and colleagues. Under some preliminary work leading up to the current study, Yuan and Mills [74] conducted a judicious search of a model parameter space to investigate the influence of various transport protocols on correlation structure in network traffic. The study, which adopted a cellular automaton model of a network of sources and receivers interacting within a grid topology, also adopted some of the invariants identified by Paxson and Floyd. Later, Yuan and Mills [75] converted the model to include a four-tier topology, which was used to study detection methods for distributed denial of service attacks within the Internet. The cellular automaton model, including the four-tier topology, was used to conduct preliminary experiments for the current study. Specifically, the model was subjected to a sensitivity analysis (using an approach explained in Chapter 4). Unfortunately, as explained in Sec. 3.1, scaling the cellular automaton model to represent a network with Internet-like speed and size proved computationally infeasible.

A hybrid network simulation model [71], combining discrete events with continuous approximation of discrete variables, may provide an attractive alternative because published computational and memory requirements appear quite promising. Specifically, the model appears to have required about 2 hours of processor time to simulate 30 long-lived flows operating for 11 simulated hours in a partial topological model of the Abilene backbone with 10 Gbps links. The published results for the hybrid model indicated the scenario was infeasible using *ns2*. Further, the hybrid model included many of the alternate congestion control models we studied. On the other hand, the

hybrid model included only a two-tier topology with sources and receivers connected directly to the backbone. In addition, the hybrid model did not include the existing measurement and parameterization capabilities included within our cellular automaton model. We did not explore the computational implications of expanding the hybrid model to add (access and point-of-presence) tiers to the topology, to expand the number of sources to hundreds of thousands, to introduce the TCP connection phase, to add various scenarios and to incorporate addition measurement code. Based on comparing the published computation requirements of the hybrid model and with the measured computational requirements of a discrete-event simulator (see Appendix B), we conclude that the hybrid model warrants further investigation (as discussed below in 2.5.1).

2.2.3 Expanded Analytical Studies

To employ analytical methods for our study, we needed to overcome three limitations of existing fluid-flow models: (1) improve accuracy in modeling queue evolution in order to obtain realistic estimates of loss probability, (2) construct differential equation models for alternate congestion control algorithms under study and (3) determine how to evaluate the resulting models in topologies of sufficient size and complexity. We believe we can leverage existing fluid-flow simulators [73], solved using numerical methods, to scale fluid-flow models to large networks. Solving the other two problems required more research, but we believed we could make some progress. Unfortunately, to complete our study in a timely fashion we needed solutions sooner rather than later. We decided to investigate analytical solutions to more accurately model queue evolution and then to construct differential equation models for some alternate congestion control algorithms. These investigations, documented in Appendix A, ran in parallel with our main study. At a later date we could use our analytical models to repeat our current studies and compare the predictions obtained and the resources required.

2.3 Selected Approach

Lacking a sufficiently large emulation facility and without an accurate fluid-flow model for the congestion control algorithms under study, we had little choice but to adopt simulation for our study. Following recommendations from Floyd and colleagues [70, 72], we aimed to reduce model scale and improve model quality by focusing the model on the study at hand, by adopting some recommended invariants, by conducting a sensitivity analysis of model parameters and by comparing key model behaviors with empirical measurements. We did not adopt *ns2*, or similar existing simulation frameworks, because the parameter spaces of such models are too large for our needs and because the computational and memory requirements are too costly for tractable simulations of systems of the size we envisioned. We also did not adopt our existing cellular automaton model because the computational costs made it infeasible to simulate networks of very high speeds and large sizes. On the other hand, the parameter space of our cellular automaton model was quite concise and appropriate for the studies we envisioned.

We decided to convert our cellular automaton model to a discrete-event simulation (DES), which we call MesoNet (see Chapter 3 for a description of the simulator). The DES simulator proved significantly faster than the cellular automaton, especially as simulated network size and speed increased. As described in Chapter 5, we

extended MesoNet, adding six alternate congestion control algorithms to the standard TCP congestion control procedures. We also found it necessary to add TCP connection-establishment procedures to the simulator. To allow exploration of flow dynamics under a range of network conditions, we adopted a single, heterogeneous topology for our study. As explained in Sec. 3.1.2, we modeled the topology after characteristics of existing network topologies and traits revealed by studies of topologies used by commercial Internet service providers. Given a concise set of model parameters, we applied statistically-based, experiment design techniques, as used typically in rigorous scientific and engineering studies. We then leveraged statistical properties of the experiment designs to conduct statistical analyses of response data. We say more regarding our approach below in Sec. 2.5, where we outline solutions (and possible alternatives) to five hard problems we had to solve in order to develop the details of our approach. First, though, we introduce the hard problems we faced.

2.4 Hard Problems

The approach we adopted for our study could not be developed and applied without devising solutions to five hard problems. We describe these problems below.

2.4.1 Model Scale

Simulating networks of the size and speed we envisioned presents two significant impediments: substantial computation and large parameter space, which combine to create a significant scaling problem. Models with thousands of parameters usually include many details. Such detailed simulators, while perhaps easy to relate to real systems, can require substantial computation to process each packet. The more packets processed the greater the computational time required for a simulation run. For networks of large size and high speed that are simulated over minutes or hours of operation, the number of packets per simulated second can prove quite high, so reducing per-packet processing time can substantially reduce computation demands for a given simulation run. A large parameter space requires simulating many runs to cover various parameter combinations. The computational requirements for simulating a specific combination multiplied by a large number of combinations can lead to an infeasible demand for computation. Reducing the number of runs required can substantially reduce computation demands for a particular simulation study. Beyond influencing computation demands, large parameter spaces require significant intellectual and practical effort from experimenters who must design and configure experiment runs. Experimenters must select the parameters of interest to vary for a given study. For other parameters, experimenters must determine fixed values for use throughout the study. Further, under a large parameter set, configuring parameters requires significant automation aid, such as databases or scripting. Even with automation, erroneous configurations can lead to incorrect experiment executions, which not only waste precious processing resources but also require significant intellectual effort and time to detect. A similar problem (see 2.4.3 below) can arise when simulations produce a large response space. Analyzing multidimensional data can prove intellectually challenging and can consume substantial processing resources. These problems of scale should be quite familiar to experimenters who use network simulations.

2.4.2 Model Validation

Whatever solution is adopted to reduce model scale, the resulting representation is likely more abstract than that provided by a typical detailed simulation model, such as *ns2*, and certainly less realistic than a deployed network. With a reduced model in hand, an experimenter faces the problem of establishing that the model is valid for purposes of an intended study. Often, experimenters compare results from two models, one more detailed, in order to establish confidence in a reduced model. Sometimes, experimenters compare predictions from simulations against predictions from widely accepted analytical models. Such comparisons between two models leave an uneasy feeling that perhaps both models might be wrong. Of course, the widely accepted model could be wrong and the newly created model correct, so changing the newly created model until it yields results aligned with the accepted model might be the wrong course. Further, even when two models are compared, experimenters typically consider only a small subset of parameter configurations. Fundamentally, when an experimenter produces a reduced-scale model, some means must be found to determine that the model is error free and that the model represents valid behaviors for purposes of the intended study. As software developers understand, producing error-free software is quite difficult; demonstrating that software is without error even more so. System modeling adds on top of that hard problem, the challenge of demonstrating a model is valid for its intended purposes.

2.4.3 Tractable Analysis

One of the significant advantages of modeling a large system using simulation is that any conceivable response can be measured at any time. Measuring responses from an empirical system can be much more difficult; impossible for some desired responses. In an analytical model, the level of detail may be insufficient even to represent various behaviors one might wish to measure. The measurement advantage of simulation can quickly introduce two challenges: identifying which responses are significant (or redundant) and analyzing large volumes of multidimensional response data. The first challenge might be rephrased as: What responses should one analyze? The second challenge might be rephrased as: Over what spatiotemporal extent should one analyze responses?

2.4.4 Causal Analysis

Analyzing measurement data from large systems allows various spatiotemporal patterns to be discerned. In general, the patterns appear from statistical analyses applied to various dimensions of the model parameter space. Existence of significant patterns can be detected and revealed to an experimenter using statistical analyses. Such patterns suggest that a model behaves in particular fashion under specified conditions. An experimenter usually desires to understand causality underlying significant patterns. Bridging the gap between statistical patterns and underlying causes represents a significant challenge with respect to any large system, including models of such systems.

2.4.5 Experiment Selection

Assuming existence of a valid, scalable simulation model where data can be analyzed tractably and causality can be established, a remaining challenge relates to selecting experiments that will probe a system in a manner needed to reveal key aspects of global

behavior that are germane to a particular study. Should an experimenter fail to include the necessary parameter values and scenarios then a simulation study may miss significant aspects of system behavior that would arise in an analogous real system.

2.5 Selected Solutions and Possible Alternatives

The modeling and analysis methods we developed and applied in the current study were intended to provide some level of solutions for the hard problems we described above. Below, we describe the solutions we adopted to address each problem. The combined solutions to the hard problems comprise the modeling and analysis methods we used, so we introduce our solutions in some detail. Where applicable, we also identify potential alternatives that might be used to address each problem. Before discussing our solutions, we provide a general mathematical introduction to the modeling state-space problem.

$$y_1, \ldots, y_m = \mathbf{f}(x_{1|[1,\ldots,k]}, \ldots, x_{n|[1,\ldots,k]})$$

Response State-Space **Stimulus State-Space**

n	Number of inputs (i.e., stimulus factors)
k	Factor range (i.e., number of values each factor can assume)
m	Number of outputs (i.e., responses)

Figure 2-1. The Modeling State-Space Problem

The mathematical transformation shown in Fig. 2-1 represents the modeling state-space problem as a mathematical equation transforming a set of input parameters into a vector of responses. This equation underlies the hard problems associated with model scale (2.4.1) and tractable analysis (2.4.3). The equation represents a simulation model as a function (**f**) returning a set of responses (y_1 to y_m) given a specified combination of input parameters (x_1 to x_n). Each input factor can take on a range (***k***) of values, often referred to in the experiment design literature as levels. The state-space transformation presents few problems when values of ***n***, ***k*** and ***m*** are small; however, for detailed network simulators the values can be large. For example, in a network model with 10^3 parameters, and assuming each parameter may be represented as a 32-bit integer[1], the stimulus state space would comprise (k^n =) $(2^{32})^{1000}$ combinations, which is on the order of 10^{9633}. Even if each simulation run can be made quite efficient, this is an infeasible parameter space to compute. The response state-space can also present a challenge when each of the ***m*** responses can be assigned to spatial partitions. For example, in a model with ***a*** actors each characterized by ***m*** responses, the response state-space becomes m^a. For a model with 10^5 actors and 20 responses, the response state-space becomes 20^{100000}. The response state-space can grow in other dimensions. For example, the ***m*** responses might be assigned to specific time intervals or to particular logical partitions. Spatial, temporal and logical partitions can be combined to further increase the response state-space.

[1] Model parameters, when represented using floating point notation, may take on even more values.

2.5.1 Scale Reduction

To constrain computational demands for our study, we adopted two main strategies: reduce the parameter space and use a two-level (or two-value) per factor orthogonal fractional factorial (OFF) experiment design method. Reducing the parameter space requires lowering the value of ***n***, while using a two-level OFF requires reducing the values of both ***n*** and ***k***. In two-level per factor experiments each parameter is assigned only 2 of its possible ***k*** values. Fig. 2-2 illustrates the theory underlying these strategies.

As a first step in reducing the scale of computational demands, a domain expert creates a model that can be specified with a reduced number (***n*** – ***r***1) of input parameters. One practical means to achieve this step is to construct a model that includes only parameters germane to an intended study. For us, this amounted to designing MesoNet (see Chapter 3), which can be parameterized with around 56 parameters, depending on how one chooses to count. Thus, assuming that a model such as *ns2* requires 10^3 parameters to configure the experiments we might design, our initial reduction factor was quite impressive (***r***1 = 944). Unfortunately, even with such a large reduction, the parameter state space remains infeasible to compute, as shown in Fig. 2-3.

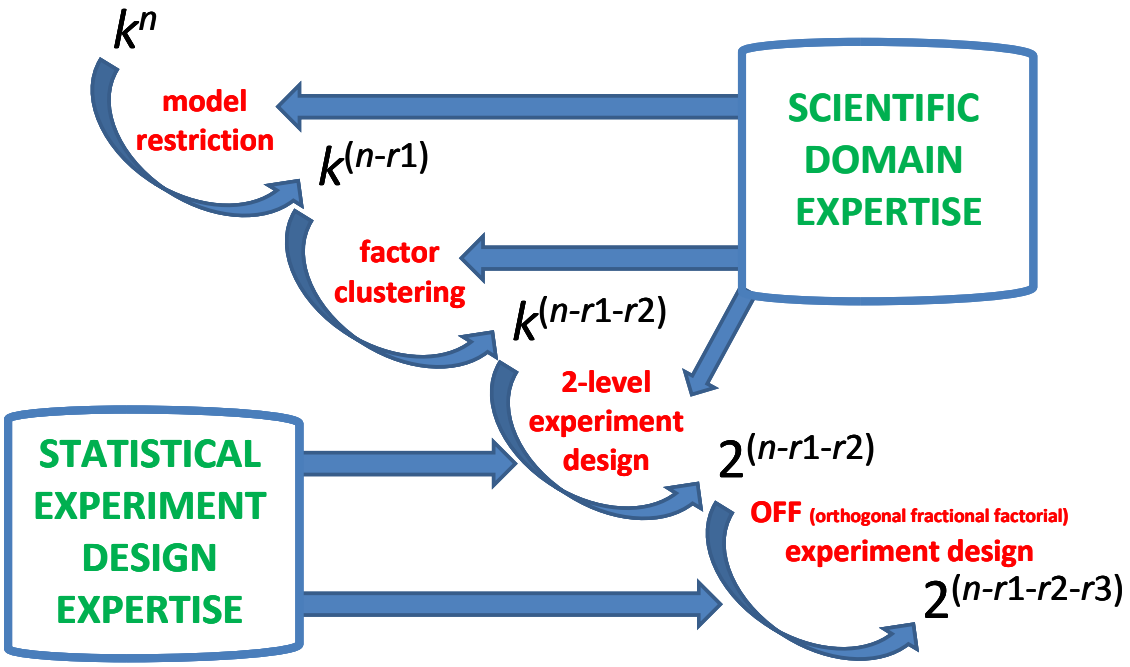


Figure 2-2. A Method to Reduce the Scale of Computational Demands

As a second step in reducing computational demands, a domain expert reexamines model parameters to identify those that can be grouped together to represent aspects of a single factor. This step requires both domain knowledge and creativity. In particular cases, parameter grouping might be guided by knowledge of the type of experiments envisioned for a study. We illustrate examples of parameter grouping in Chapter 4, where we conduct a sensitivity analysis of selected MesoNet parameters. When planning a complete sensitivity analysis of MesoNet we used parameter grouping to lower the

parameter count to 20 (***r***2 = 36). As shown in Fig. 2-3, the reduced parameter space still requires an infeasible amount of computation.

Having reduced ***n*** significantly, the next step involves reducing ***k***, the range of values each parameter may by assigned. Here, we are guided by experiment design theory [89], as developed by statistical researchers and as applied in rigorous scientific and engineering studies [95]. A major scale reduction is achieved by lowering ***k*** to a small number of levels (or values), typically 2 or 3. This reduction has two positive effects. First, the number of parameter combinations to simulate falls substantially – to a number that may be computationally feasible. Second, experiment design theory includes procedures for specifying 2-level and 3-level designs that can be subjected to statistical analyses that yield fundamental insights into system behavior. For our study, we set ***k*** to 2; as shown in Fig. 2-3, this reduces the parameter space to a point where we need to simulate only 10^6 parameter combinations.

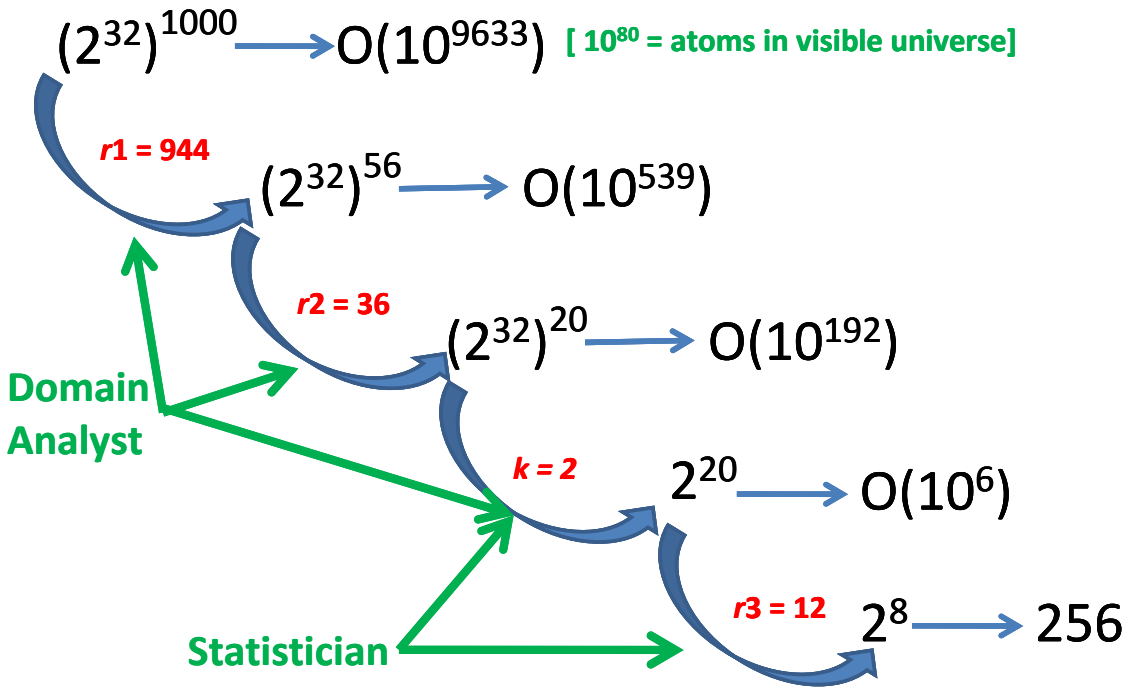


Figure 2-3. An Example Applying Scale Reduction to MesoNet Simulations – we use O( ) notation to denote "on the order of"

Suppose that using MesoNet to simulate one combination of parameters for a specified scenario requires 8 processor hours. Further suppose that we have 48 processors available for our simulations. Under those assumptions, we could simulate 2^{20} parameter combinations in about 20 years[2] (2^{20} runs x 8 processor-hours per run / 48 processors = about 1.748 x 10^5 hours). Obviously, we need to further reduce the computational demands because no one would be willing to wait two decades to learn the results of our

[2] If 10^3 processors were available, then the 2^{20} simulations could be completed in about one year, but we would still need to pay for the use of the processors.

study. We applied orthogonal fractional factorial (OFF) experiment design methods to further reduce ***n***.

To develop an OFF design, the experimenter first determines how many experiment repetitions are feasible. Suppose the experimenter decides to devote at most 2.048×10^3 processor hours to our sample experiment. Since 8 processor hours are needed to run each simulation, the experimenter can afford to investigate only ($2.048 \times 10^3 / 8 =$) 256 different combinations of parameters. Thus, given that ***k*** = 2, the experimenter requires that ***n*** = 8. This means that 2^{20} parameter combinations must be divided by 2^{12}, leaving only 2^8 combinations. In this example, as shown in Fig. 2-3, the reduction factor ***r***3 is 12.

Experiment design theorists have developed rules [89] to select which subset of parameter combinations to examine. The rules compose parameter combinations in a balanced and orthogonal form. A balanced design ensures that every factor is assigned each of its levels an equal number of times. An orthogonal design ensures that the subset of parameter combinations selected is spread evenly throughout the full set of parameter combinations. Further, the resulting design can be assessed precisely with regard to any confounding that may occur. Confounding means that given results cannot be attributed clearly to a main effect (i.e., single parameter) because the results might be due to interactions between two or more parameters. An experimenter should strive to select a Resolution IV design [89], where main effects are not confounded with two-parameter interactions; confounding between main effects and three-parameter interactions is usually acceptable because most systems are not driven by three- parameter interactions. When data analysis points to two-parameter interactions as drivers of system behavior, then a domain expert can usually determine which of the two is most likely.

A Resolution IV experiment design requires a sufficient number of simulations (n) to estimate a leading constant, each parameter (p) and each pair of parameters (p choose 2). This means the example given in Fig. 2-3 requires a least 211 ($n = 1 + 20 + 190$) simulations for a Resolution IV design. For a two-level design, choose the next higher power of 2 above n, i.e., 256 runs, which identifies the need for a 2^{20-12} design. Reducing the number of simulations below this would lead to confounding between main effects and two-parameter interactions.

The ultimate result of statistically based experiment design is that all experiment parameters (often called factors) are varied simultaneously. This powerful technique for reducing the search space stands in stark contrast to the approach typically adopted in networking simulation studies. For example, Paxson and Floyd [72] recommend holding all factors fixed except for one element, which becomes a single factor that is varied over a range during a particular simulation experiment. Varying only one factor at a time yields little information about overall system dynamics. Instead, such experiments indicate only the influence of the single varied factor given the fixed combination of other parameters. By varying all factors simultaneously, the system being investigated is examined over a much wider range of conditions. Of course, to derive useful information the resulting system responses must be analyzed with statistical methods matched to two-level OFF designs. We say more about this aspect of our approach in Sec. 2.5.3.

As shown above, reducing the parameter space of a model and applying two-level OFF designs can significantly reduce computation demands when simulating large systems. Of course, interpreting results from such experiments entails a key assumption

that system behavior is monotonic in regions between selected parameter values. OFF experiment designs will not reveal any nonmonotonicity that occurs within such regions. For this reason, an experimenter might wish to cover more than two levels in a design. In addition, the processing cost of running each simulation might require an experimenter to select an ***r***3 reduction value that provides an insufficient number of runs for a Resolution IV design, leading to an undesirable confounding structure. Thus, there exists a tradeoff between the cost of running each simulation and the number of simulations that might be desired.

In our study, we found that experiments with MesoNet require substantial computation when simulated network speeds mirror modern Internet speeds and when network size reaches hundreds of thousands of sources. Fortunately, though costly (averaging about 420 processor hours per simulation), such large simulations were not infeasible[3] with MesoNet, as is typically the case with more detailed simulators. Further, we found that we could obtain similar results when simulating a network with an order of magnitude lower router speed and only tens of thousands of sources. The smaller scale simulations were much less costly (averaging about 24 processor hours per simulation). Even with the smaller scale simulations, simulating 256 combinations of parameters can require more than 6×10^3 processor hours. Given 48 processors, the necessary simulations can be carried out within a week. Increasing the number of processors used to 256 would allow such simulations to be completed in about a day. Thus, smaller scale simulations are quite affordable and computationally feasible. Running a larger scale simulation experiment with 256 parameter combinations would take about 3 months when 48 processors are available and about 3 weeks when 256 processors are available. Thus, running larger simulations and more parameter combinations could be made more cost effective if the computation requirements for each simulation could be reduced. An alternative modeling method, known as hybrid systems, promises to reduce computation demand for network simulations.

Lee and colleagues [71] apply a combination of continuous-time dynamics and event-based logic to model long-lived flows transiting a portion of the Abilene backbone. In addition to TCP congestion control procedures, the hybrid model includes three of the congestion control algorithms examined in our study. As described in Appendix B, we used MesoNet to replicate an experiment reported by Lee and colleagues where 30 long-lived flows transmitted packets for 11 simulated hours. While MesoNet and the hybrid model obtained similar results, the hybrid model appears to require about two orders of magnitude less computation. Thus, for a study such as ours, a hybrid model combining continuous-time dynamics with event-based logic might offer a promising alternative to reduced-scale discrete-event simulation.

2.5.2 Sensitivity Analysis and Key Empirical Comparisons

To establish the validity of MesoNet for our study, we adopted two main strategies: (1) sensitivity analysis and (2) key empirical comparisons. Paxson and Floyd [72] recommend conducting a judicious exploration of a model's parameter space. Floyd and Kohler [70] repeat this advice. Floyd and colleagues also indicate that such explorations are seldom, if ever, practiced in the network simulation community. A main motivation

[3] Of course, the model code, the simulation framework, the underlying operating system and all required hardware must be highly reliable in order to run such large simulations without failures.

for exploring a model's parameter space is to understand how responses change with combinations of input parameters. Generating such knowledge can help identify parameter combinations for which experiments could yield insights and can build confidence in the operation of a model. We implemented the aims of Floyd and colleagues by conducting a sensitivity analysis of MesoNet. We designed the sensitivity analysis (described in Chapter 4) as a 2^{11-5} orthogonal fractional factorial (OFF) experiment[4], which requires 64 individual simulations. We interpreted the results of the experiment using statistical analysis methods, which we outline in Sec. 2.5.3 and explain in detail in Chapter 4. Given results from a 2^{11-5} OFF experiment, a domain expert determined if the results were as expected for a valid network model. In the case of unexpected results, the domain expert established whether the new insights were legitimate or whether the model exhibited errors. The sensitivity analysis of MesoNet uncovered both legitimate and illegitimate unexpected results. Where illegitimate results were identified, MesoNet was corrected and the sensitivity analysis was conducted again. In the end, the sensitivity analysis helped us to reduce the parameter space in later experiments so that we could focus on the top handful of factors influencing model behavior. In addition, related analyses (see Sec. 2.5.3) allowed us to reduce the response space we needed to examine in subsequent experiments. And, of course, the sensitivity analysis increased our confidence in MesoNet as a simulation platform on which to base further experiments.

One shortcoming of our sensitivity analysis arose from limiting parameter settings to only two levels. As mentioned earlier, behaviors might not be monotonic between the chosen levels. In addition, outside the range of the chosen levels a model might not exhibit the same behaviors. To address these shortcomings to some extent, we conducted a second sensitivity analysis where we chose different values to represent the two levels for each parameter. We document this second sensitivity analysis in Appendix C. We also took further steps to explore MesoNet. Some researchers [71] had conducted a hybrid simulation aimed at replicating findings expected from accepted analytical insights into TCP flows. As described in Appendix B, we repeated this published experiment using MesoNet and we compared our results to the published results. Demonstrating that MesoNet could reproduce findings predicted from accepted analytical models also raised our confidence.

Establishing MesoNet as an effective simulator of TCP flows transiting a network topology was a necessary, but insufficient, validation for the experiments we intended to conduct. Since we added six proposed alternate congestion control algorithms into MesoNet, we needed some means to establish that our models correctly implemented the algorithms. Fortunately, in a recently published paper, researchers [67] reported some empirical results from studying five of the six algorithms in a dumbbell topology. Another paper [66] provided empirical results for the sixth algorithm in a similar setting,

[4] The actual process involved multiple repetitions of sensitivity analyses, which enabled us to eliminate some model parameters from our experiments. In addition, we chose to fix selected parameters because they were not germane to the issues we intended to study. This is the reason we focused our sensitivity analysis on 11 parameters instead of 20. In later work we conducted a full sensitivity analysis using all 20 parameters in MesoNet. This later sensitivity analysis used a 2^{20-12} OFF design, which required simulating 256 parameter combinations. This later sensitivity analysis revealed that MesoNet is driven primarily by 6 or 7 of the 20 model parameters. This result confirmed our choice to use about 32 conditions for each of our experiments, described in Chapters 6-9.

though with different parameter combinations. Since MesoNet could be configured with various topologies, we were able to simulate parameter combinations from the empirical study within a dumbbell topology. As described in Chapter 5, we compared behaviors generated from our MesoNet simulations against the published empirical results. In fact, we continued to improve our models of the alternate congestion control algorithms until the simulated behavior generated by MesoNet matched the empirical behavior reported in the literature. By making these key empirical comparisons, we gained confidence that we had correctly simulated the congestion control algorithms under study.

2.5.3 Statistical Analysis Methods

By adopting two-level OFF designs as the basis for all of our experiments, we enabled the application of numerous statistical analyses that could provide insights into relationships between patterns of input parameters and observed responses. We explain the analysis methods we used in detail at each point in our study where we apply them (see Chapter 4 and Chapters 6 through 9). Here, we introduce the key analysis methods in outline form, focusing on the major contributions of each method. As a general contribution to tractable analysis, all methods we adopt either reduce the dimension of multidimensional data, concisely summarize multidimensional data in succinct form or both.

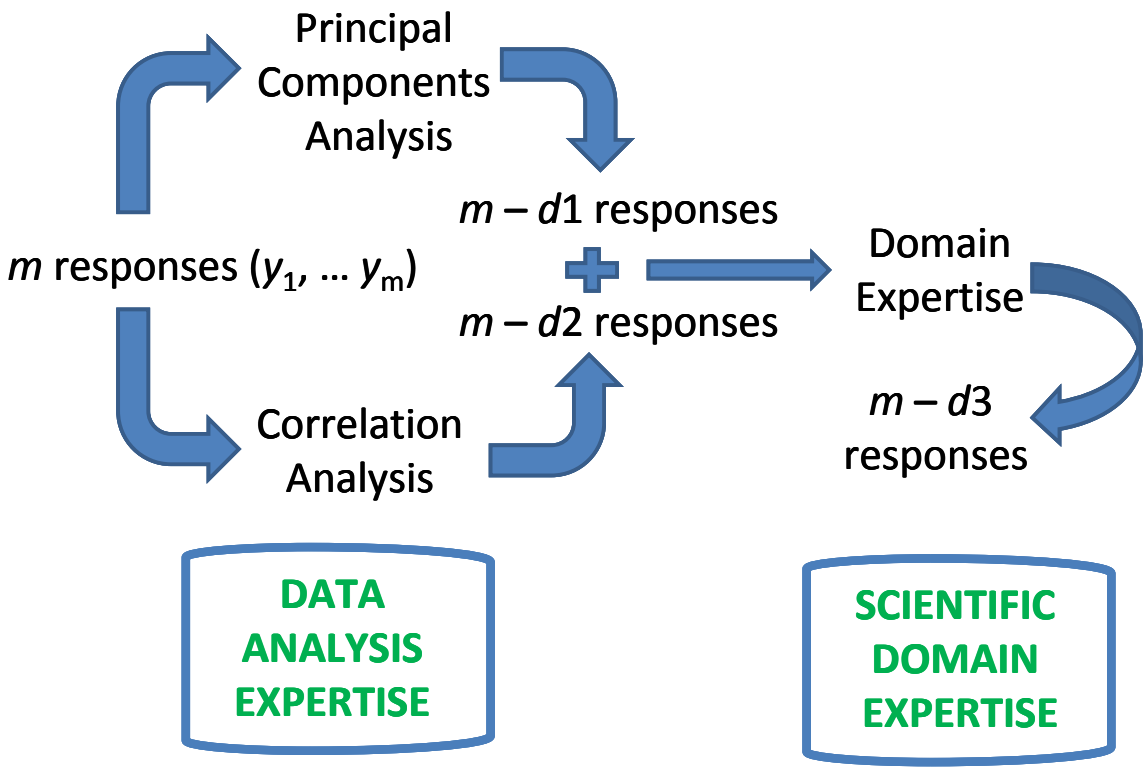


Figure 2-4. Reducing the Space of Model Responses using Correlation and/or Principal Components Analysis

One statistical method we adopted was correlation analysis, which we used to reduce the dimension of multidimensional response data. Simulation models provide

experimenters with the possibility to measure many different system responses. Correlation analysis can identify where only a subset of responses need be analyzed in order to portray system behavior. As an alternate or complementary approach one could also apply principal components analysis. Fig. 2-4 illustrates the general idea behind these two analyses. Given a set of m responses, principal components analysis can find that a lower dimension $m - d1$ can capture the variation in response data. Alternatively, correlation analysis can suggest that fewer responses $m - d2$ can suitably represent variation in system behavior. Given two different proposals for reducing the response space of a model, a domain expert may then choose which responses to analyze for subsequent experiments. Chapter 4 provides a specific example where we applied these methods to MesoNet.

Another reason to apply correlation and principal components analyses is to help validate a model. A domain expert likely has expectations that selected responses mirror similar aspects of system behavior. Correlation analysis can provide clusters of similar responses that an expert can verify. Surprising correlations may indicate errors in a model or else new findings. Similarly, a domain expert may have expectations about general network characteristics that drive behavior. By examining results from a principal components analysis, an expert can verify these overall aspects of system behavior, as represented by a model.

We also adopted a 10-step technique developed at NIST [92] to support analysis of data generated by two-level experiments designed using OFF. Each of the 10 steps produces a graphic (or plot) aimed at concisely summarizing some aspect of response data. The plots include: (1) ordered data plot, (2) scatter plot, (3) main effects plot, (4) interaction effects matrix, (5) block plot, (6) Youden plot, (7) |effects| plot, (8) half-normal probability plot of |effects|, (9) cumulative residual standard deviation plot and (10) contour plot of two dominant factors. We explain each of the plots in detail in Appendix D, and we apply selected plots in Chapter 4 to support of our sensitivity analysis of MesoNet. We also demonstrate in Chapter 4 how two-level OFF designs can aid other exploratory data plots to reveal insights about system behavior.

As demonstrated in Chapters 6 through 9, we used cluster analysis in many of our experiments to provide a concise summary of multidimensional response data. Cluster analysis computes multidimensional distances between sets of responses associated with specified parameters, or combinations of parameters, and then groups the combinations based upon similarities in distances. For example, we might cluster responses associated with each congestion control algorithm for each combination of experiment parameters (i.e., experiment condition) to produce a set of dendrograms[5], one per condition (e.g., Fig. 6-4). Clustering can help us identify patterns among responses. For example, we might find that two algorithms always cluster near each other (i.e., have similar responses). Or we might find that three different algorithms cluster together only under selected conditions. We sometimes combine cluster analysis with other methods to reveal additional patterns. For example, we use an ordered data plot to classify the relative levels of congestion associated with each condition and then overlay the classification onto a set of clustering dendrograms (e.g. Fig. 6-8) to identify how congestion level influences clustering.

[5] A dendrogram is a tree diagram frequently used to depict the arrangement of clusters, as produced by some hierarchical clustering algorithm.

In addition to clustering, we also used time series plots (e.g. Fig. 6-6) to investigate particular response dimensions highlighted by other analyses. Given a time series plot, we could determine if specific responses can be adequately summarized with an average value over particular time periods of interest. In addition, comparing time series of aggregate flow states allowed us to see how changing parameter combinations influence the pattern of flow states. We also applied time series of selected responses in selected situations (i.e., time periods and conditions) to investigate particular detailed behaviors.

Aside from the standard available statistical analyses, we combined various statistical methods into custom analysis visualizations aimed at revealing patterns for our particular experiments. We constructed such custom visualizations to provide concise summaries of multidimensional response data. We used a specific visualization (e.g., Fig. 6-9), which we designated a detailed analysis plot, to compare residuals about the mean (y axis) for each congestion control algorithm under each experimental condition, sorting the conditions on the x axis in increasing magnitude of difference. We generated one such plot for each response of interest. We labeled each condition (x axis) with a multidimensional set that included: experiment factor settings, algorithm identifier, order of magnitude in dispersion among the residuals, percentage difference in absolute response and a discriminating statistic derived from a test for outliners. We introduce and explain the details of our custom analysis plots in Chapter 6. We also used custom plots in Chapters 7 through 9.

Though each custom detailed analysis plot provided a significant amount of information about a single response, the plots, when taken together, obscured any overall pattern that might arise across responses. To overcome this limitation, we designed a second custom visualization, which we called condition-response summaries – introduced in Chapter 6 and applied also in Chapter 7. Condition-response summaries (for example see Fig. 6-10) are constructed as matrices of responses (columns) by conditions (rows), where each cell is either blank or contains the identifier assigned to a specific congestion control algorithm under test. To generate a condition-response matrix, we analyzed (automatically) each detailed response to identify cases where some algorithm was identified as a significant outlier under the corresponding combination of parameters. In such cases, we placed the identifier assigned to the outlying algorithm into the corresponding condition-response cell of the summary matrix. If the algorithm was a high outlier then the identifier was colored green. If the algorithm was a low outlier then the identifier was colored red. Further, we could apply filtering so an outlier would be included in the summary only where additional criteria were satisfied. For example, as shown in Fig. 6-11, we might specify that the outlier must show at least a 10% absolute difference from the mean of all responses for the same condition. Condition-response summaries enabled us to identify patterns where algorithms became outliers under specific conditions.

We generated several other custom visualizations, introduced in Chapter 8 and also applied in Chapter 9, to support our analyses comparing relative goodput[6] of TCP flows and competing flows running alternate congestion control algorithms. One

[6]Goodput is application level throughput, i.e. the number of useful packets per unit of time forwarded by the network from a certain source to a certain destination, excluding protocol overhead, and excluding retransmitted data packets.

visualization (e.g. Fig. 8-34) comprised a set of scatter plots (one per alternate congestion control algorithm) of goodput on alternate flows (x axis) vs. goodput on TCP flows (y axis), where each data point represented the goodputs associated with a specific combination of input parameters for a particular flow group[7]. We used these plots to identify algorithms that outperformed competing TCP flows. We augmented the scatter plots with a set of bar graphs (e.g., Fig. 8-35), one per combination of input parameters, ordered by increasing congestion. Each bar graph contained seven bars, one per alternate congestion control algorithm. One end of the bar represented the higher of the average goodputs (either on TCP flows or alternate flows) and the other end represented the lower. The bar was colored green when the alternate algorithm gave flows higher goodput and colored red when TCP flows gave higher goodput. These bar graphs enabled us to discern patterns associated with conditions under which specific alternate congestion control algorithms gave better goodputs than TCP. Finally, we designed a custom visualization (e.g., Fig. 8-12), which we called rank matrices[8], to explore patterns associated for relative goodput among alternate congestion control algorithms and among TCP flows competing with alternate algorithms under the same conditions. We generated a pair of rank matrices for each alternate congestion control algorithm. One matrix in the pair analyzed flows running the alternate algorithm and the second matrix in the pair analyzed TCP flows competing with the alternate algorithm. Each matrix intersected flow groups (columns), sorted by decreasing file size, with conditions (rows), sorted by increasing congestion. Each cell in a matrix contained a number representing the relative rank in goodput when considering all alternate congestion control algorithms under the same flow group and combination of input parameters. We used rank matrices to compare goodput among the alternate congestion control algorithms and to judge their relative influence on competing TCP flows.

2.5.4 Data-Supported Domain Expertise

The statistical analysis methods we adopted proved quite effective at identifying overall patterns from summarizations of experiment data. Further, the analysis methods that incorporated level settings for experiment input parameters could sometimes yield information that suggested causality. Unfortunately, not all of the statistical analysis methods we adopted considered input parameters. Further, in many cases specific input parameters did not directly identify causes. For example, while we inferred that varying levels of congestion in our simulations caused behavioral differences with respect to some measured responses, various parameter combinations tended to influence observed congestion. Thus, we sometimes established causality by classifying combinations of parameters with respect to responses indicating congestion (e.g., packet loss or retransmission rates). In that way, we could match patterns in other response data to changes in congestion. A similar approach could be used to classify combinations of parameters with respect to other macroscopic patterns, such as delay and demand from packets or flows. Domain expertise was required to decide which larger patterns to classify and which responses represented such patterns. Similarly, results from statistical

[7] A flow group is defined by three attributes: the relative potential for congestion on a path through the topology, the file size and the maximum achievable goodput on the path.

[8] Rank matrices report the relative ordering from lowest (1) to highest (7) goodput achieved by a particular congestion control algorithm when compared against the competing congestion control algorithms.

methods, such as correlation and principal components analyses, provided a basis for patterns on which domain experts needed to superimpose interpretations.

Establishing causality in our study always required a domain expert to interpret data. In many cases, the summarized data we used for statistical analyses was insufficient to establish causality. For this reason, we instrumented our simulation model to capture data at more detailed levels. For example, most responses reported by MesoNet were captured as time series[9], so that we could observe temporal evolution. In addition we were able to capture spatiotemporal evolution by producing time series of many characteristics (e.g., queue size, utilization, losses and count of transiting flows) for every router in our simulated topology. We did not capture spatiotemporal evolution for every source or flow. The reason we did not is twofold: (1) the model could potentially simulate hundreds of thousands of sources and (2) billions of flows could come and go over the course of a simulation. In order to gather insight into detailed behavior of flows and sources, we took two steps. First, we enabled experimenters to define long-lived flows between specified pairs of routers. The experimenter also specified when each long-lived flow would start and stop. We coded MesoNet to capture detailed temporal information (such as congestion window size, goodput, window increases, losses and timeouts) for each long-lived flow. Second, we enabled experimenters to capture message and state changes for a random sample of flows. We also instrumented MesoNet to capture temporal evolution with respect to logical classifications such as flow types. Finally, we included the option for an experimenter to capture time series of packet counts on selected links in a simulated topology. We explain these detailed measurements in Chapter 3.

Given patterns revealed by statistical analyses and armed with detailed temporal and spatiotemporal data captured from the same simulations, we investigated causality using the scientific method. When statistical analyses revealed a significant pattern, we developed a hypothesis regarding the cause. We then used detailed data to test the hypothesis; usually positing evidence that should exist if the hypothesis proved correct. If detailed data provided supporting evidence, then we considered the hypothesis confirmed. Where detailed data did not provide supporting evidence, we developed a different hypothesis and sought supporting evidence among the detailed data. For a given pattern of interest, we iterated the approach until we found a hypothesis supported by the data. For the patterns investigated in the current study we were able to find detailed data providing evidence of causality.

In addition to supporting the findings in the study, our approach to establishing causality also proved useful in identifying occasional errors within our simulation. For example, during one experiment, statistical analysis of summary data revealed that under lightly loaded conditions one of the alternate congestion control algorithms exhibited both a higher retransmission rate and a larger average congestion window[10] size. These two patterns seemed unlikely to occur simultaneously, so we needed to determine a cause. Here, we adopted an exploratory approach. We turned to detailed data mapping temporal evolution of average congestion window size for a particular flow type under a

[9] Data subjected to statistical analyses were derived from summarizations of time series captured by the simulation model.

[10] Congestion window defines the number of packets that may be sent prior to receiving an acknowledgment. A larger congestion window generally means a higher potential goodput.

specific combination of conditions. We compared related time series for eight congestion control algorithms. In seven of the algorithms, the temporal evolution of the average congestion window was flat. For the eighth algorithm, congestion window increased linearly with time. The eighth algorithm was the same algorithm identified by the statistical analysis. The comparison of time series indicated that something was wrong with respect to congestion window adjustment in the eighth algorithm. Armed with this information we examined the specific temporal evolution of the congestion window for a long-lived flow managed by each of the algorithms under the same conditions. Comparing these time series revealed that, early in the flow's life, the offending algorithm increased the congestion window much more quickly than the other algorithms. Detailed examination of the related time series identified the exact time when the incorrect algorithm began its unexpectedly sharp rate of increase in the congestion window. The time was coincident with the point where the flow had reached the initial slow-start threshold, which initiated congestion avoidance procedures even in the absence of a loss. Examination of the related code revealed that the model failed to record the time of the transition into a variable intended to hold the time of the last congestion event. After correcting the error, the experiment was rerun and the results (both summary and detailed) were compared with the previous erroneous results, which established that the cause had been identified and corrected.

2.5.5 Domain Expertise and Incremental Design

We adopted an incremental approach to experiment selection. We relied on domain expertise to design the initial experiment (described in Chapter 6). Designing an experiment involves three main activities: (1) deciding which input parameters to vary and which input parameters to fix, (2) selecting values for both the variable and fixed input parameters and (3) specifying any spatiotemporal scenario embodied within the experiment. In deciding which input parameters to vary and fix, we were guided initially by findings from the sensitivity analysis of the simulation model. The factors that most influenced model behavior were selected as input variables for the first experiment; less influential factors were assigned fixed values. In selecting values for fixed and variable parameters, we were guided by our understanding of networks, by the intended aims of the study and by results from the sensitivity analysis. For the initial experiment, we identified an interest in investigating: (1) how congestion control algorithms behave under normal Web-browsing traffic, (2) how congestion control algorithms respond to the onset of heavy spatiotemporal congestion caused by a period of large file transfers and (3) how well congestion control algorithms recover as congestion eases when traffic transitions back toward normal Web-browsing. This naturally led to an experiment scenario encompassing three separate time periods. In addition, we wished to show that MesoNet could simulate networks of significant speed and size. This led to selecting parameters to simulate a large, fast network. To investigate how congestion control algorithms behave on various types of paths within a network, we constructed a simulated topology that permitted heterogeneity in network paths.

The design for the second experiment (described in Chapter 7) was influenced by two main factors: (1) determining whether similar findings (to the first experiment) could be obtained when simulating a smaller, slower network and (2) investigating the influence of the initial slow-start threshold. Given these incremental objectives, we

simply repeated the first experiment while specifying lower values for network speed, network size and initial slow-start threshold.

After reflecting on results from the first two experiments, we decided to extend the range of simulated user traffic in the third experiment (described in Chapter 8). This decision tests the intended purpose of the alternate congestion control algorithms: to improve user performance on large files. We specified four sizes for user flows: increasing from Web objects to documents to service packs to movies. Of course, users could transmit such files over paths with various congestion profiles, and constrained by the maximum interface speed of a source or receiver. For this reason, we introduced new code to measure flows in groups, based upon three dimensions: (1) file size, (2) network path type and (3) interface speed. This enables goodputs to be compared with respect to flows sharing similar characteristics. Given that the previous two experiments exercised the algorithms under relatively heavy congestion, we also decided to significantly reduce overall congestion in the parameter combinations specified for the third experiment. We eliminated variation in temporal congestion; the scenario considered the network operating under the same traffic mix for a single, one-hour time period. Spatial congestion could still arise due to many flows transiting particular areas of the network, but the overall level of congestion was much reduced from previous experiments. To investigate the influence of various alternate congestion control algorithms on competing TCP flows, we decided to include a mix of flows: some operating under TCP and others operating under one of the alternative algorithms. To represent a network that might be moving incrementally toward replacing TCP, we introduced a new model parameter and selected two settings: (1) more TCP flows and (2) more alternate flows. Finally, because the first two experiments suggested that the initial slow-start threshold exerted significant influence on network behavior, we chose to replicate the third experiment twice: once with a high initial slow-start threshold and once with a low initial slow-start threshold.

After reflecting on results from the third experiment, we decided on a fourth experiment (described in Chapter 9) to increase the size and speed of the network by an order of magnitude and to retain other parameters from the third experiment. This decision reflects two main purposes: (1) to investigate behavior of alternate congestion control algorithms under networks of speed and size comparable to modern Internet-based networks and (2) to demonstrate that simulating large, fast networks with large files may be computationally feasible under MesoNet. Of course, the computational requirements proved substantial, so we chose to repeat only one instance of the third experiment: the case with a high initial slow-start threshold. The choice of a high initial slow-start threshold was motivated by desire to focus on the influence of loss/recovery procedures in the alternative congestion control algorithms.

Only a domain expert can decide on the specific experiments to run and the parameters and values to fix and vary. No general method exists for making these decisions. By designing experiments incrementally, the motives and results of preceding experiments can be considered when selecting the aims and designs for subsequent experiments.

2.6 Conclusions

In this chapter, we described the motivation underlying our goal to develop and evaluate a coherent set of methods that can be applied to understand behavior in large distributed

systems. We introduced a challenge problem: comparing alternative congestion control algorithms proposed for the Internet. We described and critiqued current state-of-the-art approaches adopted by researchers to compare congestion control algorithms. We outlined how our research aims to advance the state of the art. We considered several approaches that might be used to achieve our intended advances. We described the approach we developed, which required solving five hard problems. We explained the solutions we adopted to address each problem. In the remainder of this study, we use the challenge problem to develop and evaluate our methods.

While the current study investigates a specific challenge problem, the methods we apply should be generally applicable to a wide array of large distributed systems. Discrete-event simulation (DES) is applied to study a diverse range of scientific and engineering problems. Though DES requires substantial computation to represent large systems, computing power is becoming more cost-effective as system designers adopt multi-core, multiprocessor (MCMP) designs. Orthogonal fractional factorial (OFF) designs have a long history of application in rigorous scientific and engineering studies. Further, OFF experiment designs construct simulation runs that each considers a specified combination of input parameters. Such designs provide a good match for MCMP computer systems because each simulation can be run in parallel. Thus, the more processors available, the faster the simulation campaign can be completed. The statistical analysis methods we adopted are largely independent of the details of specific system models. Even our custom visualizations are based on general analysis approaches. Of course, the methods we developed and applied in the current study have limitations that must be considered. We discuss these limitations in Sec. 10.2, where we evaluate the methods we used to compare alternate congestion control algorithms.

Chapter 3 – Description of MesoNet

3 Description of MesoNet

MesoNet is a medium-scale (i.e., mesoscopic) packet-level simulator, which represents transport flows that regulate the generation of packets, routers that queue (or discard), forward and transmit packets, and links that subject packets to propagation delays. MesoNet achieves scale reduction by eliminating packet sizes (i.e., expressing capacities in terms of integral packets) and by simplifying routers to share forwarding capacity across all attached outgoing links. Each backbone router contains a single, drop-tail queue of finite length. Each lower-level router contains two, finite-length, drop-tail queues – one for packets moving toward the network core and one for packets moving toward the network edge. These simplifications prevent MesoNet from providing precise quantitative estimates but enable qualitative understanding of relationships driving behavior in networks of reasonably large size: hundreds of routers transporting hundreds of thousands of flows. In what follows, we explain the structure of MesoNet and then describe how to configure the model for particular simulations. We also define measurements that the model can produce, including the ability to provide detailed traces of flow state and packet exchanges. We close with a brief discussion of how MesoNet is coded in SLX [84-85], a discussion intended to aid those who wish to review the model's source code, which is freely available from the authors.

3.1 Model Structure

MesoNet was motivated by finite-element, discrete-time (FEDT) models, also called cellular automata (CA) models [74], which are often used by physical scientists when attempting to understand general relationships that drive spatiotemporal evolution in complex physical systems, such as collections of particles and structural arrangements of chemicals in materials. In CA models, the state of each element is updated at each discrete time step. MesoNet was originally designed and implemented as a CA model. In this form, MesoNet worked fairly well for networks of limited size (on the order of up to 3×10^4 sources) and forwarding speed (on the order of 13 million packets per second). When MesoNet simulated networks of larger size (hundreds of thousands of sources) and faster speed (hundreds of millions of packets per second), the CA modeling foundation proved quite inefficient. Simulating 1.5 million time steps of network operation could take more than two weeks on contemporary (circa 2007) PC-based servers. A CA model can require substantial useless processing when elements within the model have little to do for relatively long periods of simulated time. While the CA version of MesoNet was constructed as efficiently as possible, expanding the size and simulated speed of model elements produced a model with too much overhead. Ultimately, for models of a size and speed of interest for the studies reported in this document, the CA version of MesoNet needed about 5 CPU seconds to simulate 6 time steps of network operation.

MesoNet was subsequently recast into a discrete-event simulation (DES) model [94]. This means that model elements (i.e., all elements except for simulated flows and packets, which are transient) persist for the duration of the simulation and that the state of each model element is updated at varying time intervals dictated by the global sequence of events, where each event is generated by some model element. A DES model skips over idle time and processes only when events occur. As a result, processing speeds increase and the increase (over the CA version of MesoNet) increases with model size

and simulated network speed. For models of a size and speed used in the current study, the DES version of MesoNet needed about 5 CPU seconds to simulate 26 time steps of network operation. Thus, 1.5 million time steps of simulated network operation could be computed in about 4 days (instead of 16 days). To achieve this speedup, the DES version of MesoNet requires about double the memory of the CA version.

In what follows, we introduce the six fundamental elements of MesoNet and then describe how these elements are structured into a simulated network topology. We also introduce our representation of simulated packets, which are the mechanism through which MesoNet elements interact. Packets are originated by sources and receivers, flow through a topology of routers and links and then cease to exist. We provide with a brief discussion of our DES model, including one means to relate abstract time to the typical notion of real time in networks.

3.1.1 Model Elements

The persistent elements of MesoNet interact through two types of transient elements: packets and flows. Packets are the fundamental messages that move among the model's persistent elements. Flows are collections of packets that move between specific sources and receivers over a defined time period. Flows then represent logically related spatiotemporal sequences of packets. The operation of each flow is controlled jointly by three elements: a source, a receiver and an access router (to which the source is attached). Each source generates data packets, subject to various constraints. Each receiver generates acknowledgment packets as required to indicate reception of expected data. Each receiver also generates negative acknowledgment packets to indicate that expected data was missed. Access routers update appropriate state variables for subordinate sources based upon information in arriving acknowledgments and negative acknowledgments. Access routers, point-of-presence (POP) routers and backbone routers manage the forwarding of packets. The simulated propagation of packets over distance is achieved through backbone links. Below, we describe the detailed operation of each of these element classes.

3.1.1.1 Sources. MesoNet sources represent user behaviors associated with data sources. Sources alternate between ON and OFF periods. During each ON period, a source generates data packets under regulation of congestion control rules implemented by a specific, simulated transport protocol. During each OFF period, a source simply waits for simulated time to advance (an exponentially distributed value) to the beginning of the next ON period. The alternation of a source between ON and OFF periods simulates a user surfing the Web and downloading selected files of interest. Initially, sources are placed into the ON state with a specified probability. Sources that start in the OFF state are scheduled to enter the ON state at some random time after the model starts execution.

Upon entering the ON state, a source selects a file size, which designates the number of data packets to transfer to a receiver. The source remains in the ON state until the receiver has acknowledged the required number of data packets. The file size is selected from a distribution (usually a Pareto distribution with a designated mean and shape) that represents the aggregate size of objects that might typically be downloaded to display a Web page. With some probability, a given file size is multiplied by a factor,

which represents the situation where a Web-surfing user decides to download a particular file (e.g., document, service pack or movie).

After selecting a file size, a source selects a specific receiver with which to exchange the file. The source selects a receiver associated with a different backbone router than the source. This ensures that each simulated flow will transit the network backbone. Each eligible backbone (first-tier) router has a uniform probability of being selected. Once a backbone router is selected, the source chooses among one of the second-tier routers beneath the backbone router. For this selection, the probability of choosing a particular second-tier router is proportional to the number of receivers encompassed by each eligible router. If a selected second-tier router is a point-of-present (POP) router, then there will be third-tier (access) routers beneath it. In this case, the source selects a specific access router. For this selection, the probability of choosing a particular third-tier router is proportional to the number of receivers encompassed by each eligible router. Once a specific access router is determined, the source chooses any idle receiver encompassed by the access router. If no receiver is idle, then the source conducts another selection process, beginning with selection of a backbone router. The selection process continues until the source finds an available receiver.

After selecting a receiver, a source initiates a connection process, intended to simulate the establishment of transmission control protocol (TCP) flows [5, 8-9]. Without including a connection process, the model would tend to generate congestion patterns inconsistent with real TCP traffic, which must establish a connection before attempting to exchange data. MesoNet simulates a two-way handshake, with behavior patterned after TCP implementations deployed in Microsoft Windows™ operating systems [11]. The source sends a SYN packet and then waits 3×10^3 time steps for a connection to be established. If a connection is not established, then the source sends a second SYN and waits 6×10^3 time steps for a connection to be established. If a connection is not established, then the source sends a third SYN packet and waits 12×10^3 time steps for a connection to be established. If no connection is established after 21×10^3 time steps, then the connection is declared to have failed and the source enters the OFF state after selecting the next time for the source to reenter the ON state.

After successfully establishing a connection, a source enters the CONNECTED state and begins to transfer data packets. Upon initially entering the CONNECTED state, a source checks the time step and flow class (see 3.1.2.3 below) to determine whether the file size should be increased. MesoNet permits qualified flows to be designated as jumbo flows during particular time periods. The size of a jumbo flow is multiplied by an additional factor, intended to simulate the exchange of large data sets between research organizations. Upon transferring the first data packet of a file, the flow state is initialized – consistent with requirements of the particular transport protocol used on the flow. For all flows, both the congestion window and slow-start threshold are set to initial values and the flow is placed into the SLOW-START phase. The flow also establishes an initial estimate for the smoothed round-trip time (SRTT) between the source and receiver. This initial estimate is derived from the network topology used in the simulation (see 3.1.2 below). The first packet is transferred to the access router that encompasses the source. If the incoming queue in the access router is full, then the data packet is dropped.

After transferring the initial data packet of a flow, a source enters its main processing mode that attempts to transfer data packets whenever permitted and until all

required data packets have been sent. The source maintains a retransmission timeout (CRTO) that allows it to recover from situations where required feedback does not arrive. The first step of processing is to determine if the retransmission timer has expired. If so, following rules associated with a specific transport protocol, the source reduces the slow-start threshold and resets the congestion window to its initial value. The source also doubles and resets the retransmission timeout and then reenters the SLOW-START phase.

Before sending additional data packets, a source determines how many data packets it may send. Roughly, this is the congestion window minus the number of data packets sent but not yet acknowledged by the receiver. Of course, the number of data packets to be sent must also be bounded by the file size for the flow. If any data packets may be sent, then the source transfers one data packet to the access router that encompasses the source. If the incoming queue in the access router is full, then the data packet is dropped. Sending multiple data packets requires multiple invocations of a source. Thus, the number of packets that a source may send per time step is limited by the source's capacity (in packets per time step, or ppts), which is established by a configuration parameter (see 3.2.4 below).

3.1.1.2 Receivers. MesoNet receivers are responsible for sending positive and negative acknowledgment packets as directed by entries in an incoming queue. In MesoNet, each data packet associated with a flow is assigned a monotonically increasing (integer) sequence number. Acknowledgement (ACK) and negative acknowledgment (NAK) packets also contain a sequence number that identifies the sequence number of the next data packet expected by a receiver. In MesoNet, packets flow along a fixed route; thus, packets may be lost but not reordered. For this reason, the sequence numbers of packets on a flow can continually increase and still indicate packet losses and receptions. When a data packet arrives with an expected sequence number then an ACK packet should be sent with the next sequence number. When a data packet arrives with an unexpected sequence number then a NAK packet should be sent with the next sequence number. When a source receives either an ACK or a NAK, this means that a receiver got a data packet, though perhaps not the expected data packet. Thus, by counting the number of ACK and NAK packets from a receiver, a source can determine how many data packets were received. Data packets lost before reaching a receiver or ACK and NAK packets lost before reaching a source will cause a source to send additional data packets, which are retransmissions.

The processing of a receiver is quite simple. If there are no entries in its queue, then the receiver does nothing. If there are entries in its queue, then the receiver removes the first entry and generates an outgoing SYN+ACK, ACK or NAK packet as indicated. The outgoing packet is given to the access router associated with the receiver. If the incoming queue of the access router is full, then the packet is discarded. Sending multiple ACK or NAK packets requires multiple invocations of a receiver. Thus, the number of packets that a receiver may send per time step is capped by the receiver's capacity (in ppts), which is established by a configuration parameter (see 3.2.4 below).

3.1.1.3 Access Routers. Access routers are the most complex elements of a MesoNet model. Each access router has two queues: one for packets bound up toward the network

core and one for packets bound down toward sources and receivers. At each invocation, an access router processes at most one packet, alternating between the up and down queues. If no packet is available, then the access router simply does nothing. Access routers are also responsible for three roles: (a) forwarding outgoing packets from sources and receivers toward the network core, (b) processing incoming SYN and data packets on behalf of subordinate receivers, and (c) processing incoming SYN+ACK, ACK and NAK packets on behalf of subordinate sources. Each of these roles is addressed in turn.

If a packet is outbound toward the network core, then the access router removes the packet from the up queue and finds the next-hop router (either a POP router or a backbone router) and forwards the packet to that router. If the incoming queue of the destination router is full, then the packet is discarded.

If a packet is inbound toward the network edge, then the router removes the packet from the down queue and applies processing that depends upon the packet type. If the inbound packet is a SYN, then the access router creates an entry for the destination receiver requesting generation of a SYN+ACK and places the entry into the receiver's work queue. If the inbound packet is a SYN+ACK, then the access router changes the phase of the destination source to CONNECTED. If the inbound packet is a data packet, then the access router compares the packet sequence number to the sequence number expected by the receiver. If the sequence numbers match, then the access router creates an ACK entry and places it into the work queue of the destination receiver. If the sequence numbers do not match, then the access router creates a NAK entry and places it into the work queue of the destination receiver. The access router also updates the sequence number of the destination receiver to be one greater than the sequence number in the received data packet.

If the inbound packet is an ACK or NAK packet, then the access router updates the destination source state to reflect the feedback. First, the access router decrements the number of data packets that remain to be sent on the flow. If no more data packets remain to be sent, then the source is placed into the OFF state and a time is selected for the source to return to the ON state. If the inbound ACK or NAK is an interim packet in the flow, then the access router updates the SRTT and retransmission timeout for the source. The access router also sets the highest sequence number acknowledged for this flow to be the sequence number in the ACK or NAK. If the sequence number in the ACK or NAK is lower than a sequence number recorded when the last NAK was received for the source, then the remainder of the feedback processing is skipped because the packet provides outdated information. The feedback processing that remains is to update the state of the source's congestion control variables (notably the congestion window and slow-start threshold) based on (a) packet type (ACK or NAK), (b) current state of the congestion control variables, and (c) rules of the specific congestion control algorithm being simulated for the flow associated with the packet.

MesoNet increases the congestion window identically for all flows for which the congestion window is below the slow-start threshold. The procedures adopted correspond to limited slow-start, as suggested by Floyd [7]. Upon receiving an ACK, if the congestion window is below some MAX_SS_THRESHOLD, then the congestion window is increased by one, which amounts to an exponential increase. If the congestion window exceeds MAX_SS_THRESHOLD, then the congestion window (*cwnd*) is increased by 1/(*cwnd*/(0.5 x MAX_SS_THRESHOLD)), which amounts to a logarithmic

increase. When the congestion window exceeds the slow-start threshold, congestion avoidance procedures are adopted, where the increase resulting from an ACK depends upon the particular congestion control algorithm being simulated for the flow. For TCP Reno, receiving an ACK in congestion avoidance increases the congestion window by 1/*cwnd*, which results in a linear increase.

Upon receiving a NAK, the access router records the next data sequence number for the source when the NAK was received. This marks the ACK or NAK sequence number that must be received before feedback processing can recommence on this flow. In addition, receiving the NAK causes the access router to reduce the source's congestion window. For TCP Reno, the congestion window is reduced by ½ and the slow-start threshold is set to the reduced congestion window, which ensures that the flow enters congestion avoidance, where *cwnd* increases linearly upon receiving new ACKs. The explanation of other congestion control algorithms is postponed until Chapter 5.

Whenever an access router is invoked it processes only one packet. Thus, to process multiple packets an access router must be invoked multiple times. The rate at which an access router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

3.1.1.4 Point-Of-Presence Routers. Point-of-Presence (POP) routers provide intermediate points to connect access routers to the network backbone. (Note that some access routers may be connected directly to backbone routers, as discussed below in 3.1.2.2). POP routers serve only to forward packets from access routers toward backbone routers and to forward packets from backbone routers toward access routers. Thus, POP routers act as statistical multiplexor and demultiplexor elements. MesoNet topologies do not permit POP routers to be connected to each other or to be connected to multiple backbone routers.

Each POP router has two queues: one for packets bound up toward the network core and one for packets bound down toward the network edge. At each invocation, a POP router processes at most one packet, alternating between the up and down queues. If no packet is available from the appropriate queue, then the POP router is idle; otherwise, the POP router removes the first packet from the queue. If the packet is inbound, then the POP router looks up the next-hop access router and forwards the packet. If the incoming queue of the access router is full, then the packet is discarded. If the packet is outbound, then the POP router looks up the next-hop backbone router and forwards the packet. If the queue of the backbone router is full, then the packet is discarded. To process multiple packets, a POP router must be invoked multiple times. The rate at which a POP router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

3.1.1.5 Backbone Routers. Backbone routers connect to each other through backbone links and also connect to POP or access routers. Thus, backbone routers forward outbound and transit packets to backbone links and forward inbound packets to POP or access routers. Each backbone router has a single queue from which one packet is processed on each invocation. If the queue is empty, then the backbone router does nothing. If the queue is not empty, then the backbone router removes and processes the first packet. If the packet is inbound, then the backbone router looks up the next-hop POP

or access router and forwards the packet onward. If the incoming queue of the next-hop router is full, then the packet is discarded. If the packet is an outbound or transit packet, then the backbone router forwards it on to the appropriate backbone link. To process multiple packets, a backbone router must be invoked multiple times. The rate at which a backbone router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

3.1.1.6 Backbone Links. Backbone links are simplex links connecting (source and destination) pairs of backbone routers and delaying packets for a specified propagation time prior to delivering them to the next backbone router. Generally, backbone links are paired so that if one goes from source backbone router A to destination backbone router B, then a second goes from source backbone router B to destination backbone router A, Also, generally each of the paired backbone links exhibits the same propagation delay, though the model permits configuring asymmetric propagation times. Backbone links are simulated as queues sized to hold as many packets as a backbone router can forward in one time step times the number of time steps required to propagate packets on the link. At each invocation, a backbone link processes at most one packet. If the queue is empty, then the backbone link does nothing. If the queue is not empty, then the backbone link examines the first packet. If it is not yet time for the packet to arrive, then the backbone waits. Once the time has come for the packet to arrive at the destination backbone router, then the backbone link removes the packet from its queue and forwards it on to the backbone router. If the backbone router queue is full, then the packet is discarded.

3.1.2 Network Topology

Network topology (i.e., the arrangement of routers and links and routes that transit them) plays a key role in shaping macroscopic behavior in a packet network. The study of network topologies, particularly of the larger Internet topology, a network of networks, remains an object of extensive research [12-15, 18-22, 29, 32]. Given access to selected measurement points and little other information, researchers attempt to generate maps of network topologies [17, 23-24, 27-28, 31]. This becomes quite a difficult problem for the Internet as a whole, where a topology of autonomous systems is the measurement aim. Even generating maps of individual autonomous systems, each perhaps representing the topology of an Internet service provider (ISP), can be quite challenging when making measurements from outside the topology [16, 30].

MesoNet models a single ISP (Internet service provider) and thus the topology of interest is a collection of routers and links that compose a single autonomous system. MesoNet allows a user to define any network topology as long as a few restrictions are followed. First, POP routers may only connect to one parent (backbone) router, but may connect to multiple children (access) routers. Each access router may only connect to one parent router, either a POP router or a backbone router. This means that POP and access routers may not connect to routers of the same type. This results in fixed routing for packets flowing toward and away from the network backbone. Second, the network backbone must define fixed routes for packets flowing in a given direction between pairs of backbone routers. This means that packets flow on fixed routes among backbone routers. These restrictions deviate from some real network topologies in a couple of ways. First, some real network topologies provide multiple links between pairs of routers.

These multiple links provide increased capacity when all links are operating and also provide increased reliability when selected links are down. For MesoNet, higher capacity links would be modeled as higher capacity routers. MesoNet does not support capacity reductions that would occur when selected links within a multiple-link set fail. Second, some real network topologies allow links between routers at the same level. These same-level links may provide multiple paths to adapt around temporary outages. MesoNet does not represent such alternate routes. Also, by assigning link costs, multiple paths may be used to engineer traffic flows along various routes. In such cases, link costs are not associated with link propagation delay but are chosen to create desired traffic patterns. MesoNet can be used to represent such traffic engineering because MesoNet does not explicitly require that routes be assigned based on link propagation delays.

MesoNet topology restrictions have several implications. First, packets in each direction between a source and destination flow on the same fixed path. Second, the link capacities in MesoNet topologies remain fixed for the duration of a simulation.

For the experiments discussed in this report, MesoNet uses a single topology, shown in Fig. 3-1. The backbone for this topology was adapted from the original Abilene backbone [25]. Other aspects of the topology were derived from investigations of ISP topologies, as reported in the published literature [26]. The simulated topology defines a fixed route for packets flowing between each source-receiver pair. While MesoNet may specify routes based on any criteria, the routes specified in the experiments reported in this document are shortest path routes that are based typically on link propagation delays. The assigned paths in both directions between a given pair of routers are usually, but not always, mirrors of each other. This implies that routes are not always determined by link propagation delays. The use of asymmetric routes is similar to the definition of traffic-engineered routes defined for many real ISP topologies. The routes used in this study are specified below in Table 3-2.

The resulting simulated topology was compared against an example topology used by an ISP. The comparison provided confidence in the main features of the topology shown in Fig. 3-1. Differences arose from topological restrictions imposed by MesoNet, as well as from the need to execute simulations within a tractable amount of computing time. There were four main differences observed between the topology in Fig. 3-1 and the example ISP topology. First, the real ISP topology used fixed path routing with weights assigned to achieve traffic engineering objectives rather than assigning weights based on propagation delays. Thus, while routes taken through both topologies are fixed, none of the routes in the real topology were shortest-path routes based on propagation delay. Second, the real ISP topology used multiple links to connect many pairs of routers. This provided an increase in both capacity and reliability. The simulated topology included only single links between router pairs. Thus, the simulated topology could not support as much traffic as the real topology. This restriction was adopted to decrease simulation time. Further, the simulated topology could not represent capacity decreases resulting from link failures. This restriction was adopted because the periods of time simulated (between 20 and 60 minutes) were of short duration and because investigations were not focused on network changes arising from link failures. Third, the real ISP topology exhibited multiple links from POP routers to backbone routers and also between POP routers. These multiple links were used for traffic engineering and for improving reliability. Fourth, the real topology included around 60 backbone routers and 81 bi-

directional links, 85 POP routers and 663 access routers. The simulated topology includes only 11 backbone routers and 14 bi-directional links, 22 POP routers and 139 access routers. This restricts the simulated topology to carry less traffic than the real topology. Of course, restricting the size of the simulated topology reduces the computation time required to simulate the entire network over a given period of time. This is especially important when attempting to simulate high-speed forwarding capacities, ranging up to 384 Gbps in simulations described in this report.

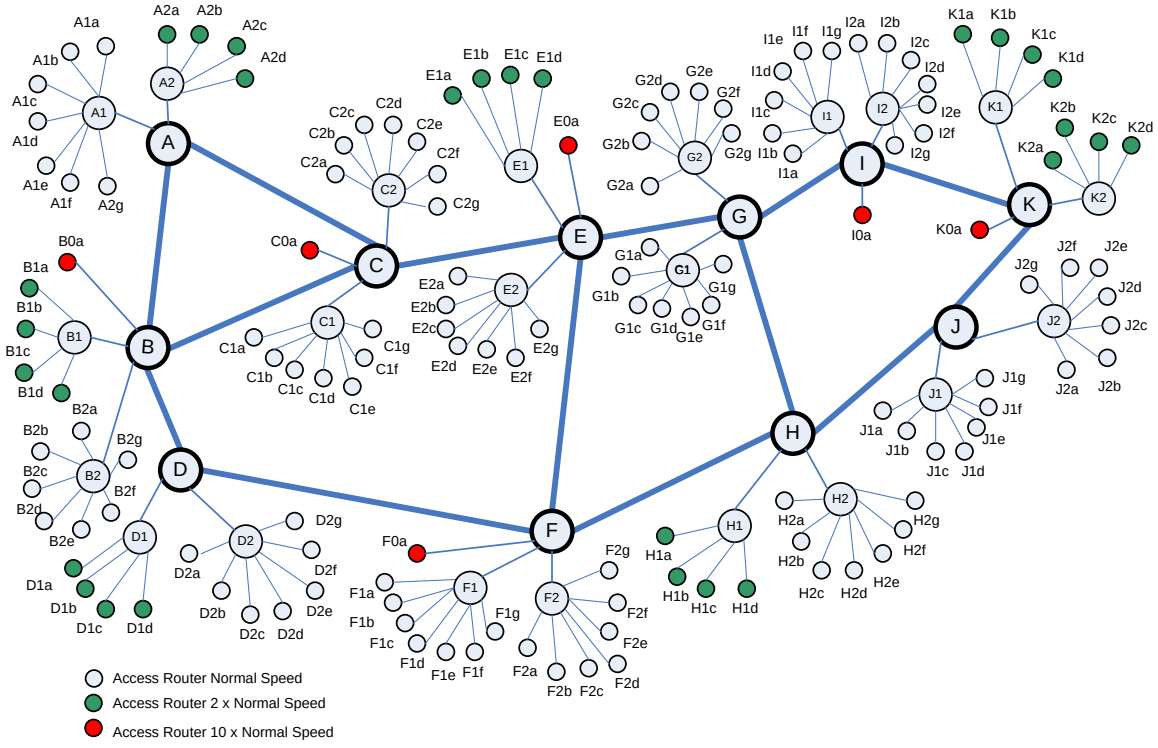


Figure 3-1. Topology used for simulation experiments discussed in this study

In summary, the topology simulated in this study represents one instance of a real backbone topology (the original Abilene network). The remainder of the topology was inspired by traits from real ISPs, as reported from investigations in the published literature. The main restrictions adopted in the simulated topology deal with reducing the number and capacity of routers and links. These restrictions were adopted to allow for feasible simulations of network operation within available computing resources. The sections that follow describe the key aspects of the topology used in experiments throughout this study.

3.1.2.1 Four-Tier Structure. The topology consists of a four-tier structure. The top tier is the network backbone, composed of 11 backbone routers (A-K in Fig. 3-1) and 14 bi-directional links. Routers forward packets at some specified rate. Propagation delays are imposed only when packets transit backbone links. The second tier comprises 22 POP routers (A1-K2 in Fig. 3-1). The third tier comprises 139 access routers (A0a-K2d in Fig.

3-1). The fourth tier (not shown in Fig. 3-1) consists of sources and receivers that represent computers exchanging data over the network.

Data packets are generated at a source, forwarded to an associated access router and then on to the access router's parent, either a POP router or a backbone router. Outgoing data packets that reach a backbone router are forwarded on a link connecting to the next-hop backbone router on the route and so on until reaching the backbone router under which the receiver may be found. The destination backbone router forwards the packet on to the destination child router (either POP or access router). An inbound data packet reaching a POP router is forwarded on to the destination access router and then to the destination receiver. Acknowledgment packets flow in a similar fashion but in the reverse direction from receiver to source. This four-tier structure is similar to the structure that exists in many ISP networks.

In general, backbone routers operate at higher speed than POP routers and POP routers operate at higher speed than access routers. Further, topologies are constructed so that backbone and POP routers can handle the expected load, leaving bottlenecks to arise at the level of access routers. For the topology in Fig. 3-1 to mimic this behavior requires that router speeds be configured properly.

3.1.2.2 Heterogeneous Composition. Recent research, which attempts to map the network topology for selected ISPs, suggests a heterogeneous composition for real networks [26]. For example, some access routers operate at higher speeds than others and some access routers connect directly to backbone routers rather than transit across POP routers. The topology in Fig. 3-1 includes both forms of heterogeneity. First, access routers operate at three different speeds: normal, twice normal speed (green routers in Fig. 3-1) and ten times normal speed (red routers in Fig. 3-1). Second, selected access routers connect directly to backbone routers. The main reason access routers connect directly to backbone routers is because they require higher bandwidth. For this reason, the topology in Fig. 3-1 has access routers, operating at ten times the normal speed, connect directly to backbone routers. Further, POP routers can handle fewer high-speed access routers than access routers of normal speed. The topology in Fig. 3-1 reflects this by allowing only four fast access routers to connect to a POP router, while a POP router can handle seven access routers of normal speed. Further heterogeneity is possible, for example, by allowing three normal access routers and two fast access routers to connect to POP routers.

3.1.2.3 Flow Classes. Access routers provide network access for sources and receivers. The division of access routers into three speed classes (normal, fast and directly connected) has the effect of creating six possible classes of flows, depending upon the location of a flow's source and receiver: NN[1] (normal-normal), FN (fast-normal), DN (directly connected-normal), FF (fast-fast), DF (directly connected-fast) and DD (directly connected-directly connected). Thus, heterogeneity among access routers leads to heterogeneity among flows, which allows measurements to be made regarding the performance of flows within each of the six classes.

[1] We color code flow-class designations to indicate their relationship to the types of access routers in the topology.

3.1.2.4 Fixed-Path Routing. Even when a choice exists among equal-cost routes, most deployed routers in the Internet attempt to ensure that all packets related to an individual flow transit the same fixed route in a given direction [41-43]. The aim is to have packets reaching receivers in sequence and thus to reduce problems associated with fragmentation of reassembly buffers. There exists debate on how successful routers are in achieving this aim [38-39, 45]. Since packet sequencing cannot be completely guaranteed on the Internet, receivers will sometimes indicate (via a duplicate acknowledgment) a packet has been missed, when in fact the packet may have been reordered rather than lost. For this reason, TCP sources typically wait for three duplicate acknowledgments before initiating retransmission of a lost packet. This allows time for reordered packets to arrive at a receiver and be acknowledged. The cost of this heuristic is that reaction to actual lost packets is somewhat delayed.

MesoNet topologies are totally successful in achieving packet sequencing along the route in each direction between a source and receiver. The upshot of this MesoNet feature is that packets received out of order can be assumed to signal a loss. For this reason, MesoNet sources take retransmission actions after receiving one, rather than three, duplicate acknowledgments. (Note that MesoNet replaces the use of three duplicate acknowledgments with one explicit negative acknowledgment, or NAK).

Table 3-1. Link Propagation Delays in the Base Simulated Topology

Router Pair	Link Pair	Link Propagation Delay (in time steps)
A-B	1-2	10
A-C	3-4	12
B-C	5-6	12
B-D	7-8	4
C-E	9-10	8
D-F	11-12	20
E-F	13-14	10
E-G	15-16	7
F-H	17-18	9
G-H	19-20	7
G-I	21-22	4
H-J	23-24	7
I-K	25-26	5
J-K	27-28	3

3.1.2.5 Simulated Abilene Backbone Characteristics. All experiments described in this study operate within the context of a simulated Abilene backbone network topology (recall Fig. 3-1) with the characteristics described in this section. First, the backbone links in the topology exhibit the base propagation delays shown in Table 3-1. The simulated topology encompasses (14 x 2 =) 28 unidirectional backbone links. The propagation delays, shown in Table 3-1, are the same for both backbone links (one in each direction) that connect the indicated router pairs. The odd numbered link of each pair flows in the forward direction (e.g., A to B) and the even numbered link flows in the reverse direction (e.g., B to A).

The simulated topology defines fixed routes, given in Table 3-2, for packets flowing between sources and receivers located under specific pairs of backbone routers.

The fixed routes indicate links that are followed and so the sum of the link propagation delays determines the route propagation delays (as indicated in Table 3-2). The round-trip propagation times between selected source and destination domains can be determined by summing, from Table 3-2, the route propagation delay in the forward direction and the route propagation delay in the reverse direction. The average round-trip propagation delay for the base topology is 41 time steps; the minimum roundtrip propagation delay is 6 time steps (i.e., route J-K-J) and the maximum roundtrip propagation delay is 100 time steps (i.e., route A-B-D-F-H-J-H-F-D-B-A). One-way paths that are part of the five asymmetric routes are highlighted in red in Table 3-2.

Table 3-2. Routes across the Backbone from Source (S) to Destination (D) Domain (One-Way Route Propagation Delays in Time Steps Given in Parentheses)

S\D	A	B	C	D	E	F	G	H	I	J	K
A	-	A-B (10)	A-C (12)	A-B-D (14)	A-C-E (20)	A-B-D-F (30)	A-C-E-G (19)	**A-C-E-G-H (34)**	A-C-E-G-I (31)	A-B-D-F-H-J (50)	A-C-E-G-I-K (36)
B	B-A (10)	-	B-C (12)	B-D (4)	B-C-E (20)	B-D-F (24)	B-C-E-G (27)	B-D-F-H (33)	B-C-E-G-I (31)	B-D-F-H-J (40)	**B-C-E-G-I-K (36)**
C	C-A (12)	C-B (12)	-	C-B-D (16)	C-E (8)	C-E-F (18)	C-E-G (15)	C-E-G-H (22)	C-E-G-I (19)	C-E-G-H-J (29)	C-E-G-I-K (24)
D	D-B-A (14)	D-B (4)	D-B-C (16)	-	**D-B-C-E (24)**	D-F (20)	**D-F-E-G (37)**	D-F-H (29)	D-F-H-G-I (40)	D-F-H-J (36)	D-F-H-J-K (39)
E	E-C-A (20)	E-C-B (20)	E-C (8)	**E-F-D (30)**	-	E-F (10)	E-G (7)	E-G-H (14)	E-G-I (11)	E-G-H-J (21)	E-G-I-K (16)
F	F-D-B-A (30)	F-D-B (24)	F-E-C (18)	F-D (20)	F-E (10)	-	**F-E-G (17)**	F-H (9)	F-H-G-I (20)	F-H-J (16)	F-H-J-K (19)
G	G-E-C-A (19)	G-E-C-B (27)	G-E-C (15)	**G-H-F-D (20)**	G-E (7)	**G-H-F (16)**	-	G-H (7)	G-I (4)	G-H-J (14)	G-I-K (9)
H	**H-F-D-B-A (43)**	H-F-D-B (33)	H-G-E-C (22)	H-F-D (29)	H-G-E (14)	H-F (9)	H-G (7)	-	H-G-I (11)	H-J (7)	H-J-K (10)
I	I-G-E-C-A (31)	I-G-E-C-B (31)	I-G-E-C (19)	I-G-H-F-D (40)	I-G-E (11)	I-G-H-F (20)	I-G (4)	I-G-H (11)	-	I-K-J (8)	I-K (5)
J	J-H-F-D-B-A (50)	J-H-F-D-B (40)	J-H-G-E-C (29)	J-H-F-D (36)	J-H-G-E (21)	J-F-H (16)	J-H-G (14)	J-H (7)	K-I (5)	-	J-K (3)
K	K-I-G-E-C-A (36)	**K-J-H-F-D-B (41)**	K-I-G-E-C (24)	K-J-H-F-D (39)	K-I-G-E (16)	K-J-H-F (19)	K-I-G (9)	K-J-H (10)	J-K-I (8)	K-J (3)	-

3.1.3 Simulated Packets

Simulated packets in MesoNet share many elements with real Internet packets, while also lacking some traits of real packets. Each simulated packet comprises six main fields: (a) packet type, (b) source address, (c) destination address, (d) flow identifier, (e) sequence number and (f) creation timestamp. Each simulated packet also contains a propagation time field used to control propagation across simulated backbone links. MesoNet packets can have one of the following types: SYN, SYN+ACK, DT, ACK or NAK. Each source

and destination address in MesoNet consists of three components: (1) domain identifier, (2) POP identifier, and (3) access-point identifier. Flow identifiers are represented as a pair of pointers: one to a source and one to a receiver. MesoNet packets are assigned monotonically increasing sequence numbers, but numbers are skipped whenever a packet in the flow is lost.

The main difference between MesoNet packets and real Internet packets is that MesoNet packets lack size. This simplification was adopted to allow MesoNet to represent packet-level behavior without having to simulate the lower level of octets or bits. This was intended to make MesoNet easier to code and understand and faster to simulate. One could then assume that all MesoNet packets have the same size – for example, one could assume that each packet is 1500 octets (i.e., 12×10^3 bits) in length. In this way, processing rates expressed as packets per time step can be converted into other bases, such as octets per time step or bits per time step.

Real Internet packet flows may consist of data going in both directions (i.e., flows may be full duplex), which also permits ACKs to be piggybacked on data packets. MesoNet flows simulate data moving in one direction only (i.e., flows are simplex) and thus ACKs are not piggybacked on data packets but instead are sent individually. The restriction to unidirectional data flows (with ACKs flowing in the reverse direction) combined with lack of packet size has the unrealistic effect that MesoNet processes simulated ACKs at the same rate as simulated data packets, which would not occur in the real Internet because ACK packets typically have a much smaller size than data packets. This is one reason why MesoNet cannot be used to derive precise quantitative measures of network performance.

Real Internet packets do not include a NAK type – but instead use duplicate acknowledgments. Real Internet packets also include a FIN type, used to close flows. MesoNet dispenses with the FIN exchange and instead closes flows implicitly when a source receives the last expected ACK or NAK packet from a receiver. Real Internet packets may also include a RESET type, which can abort a flow. MesoNet allows a source to abort a flow with resorting to a RESET packet.

Real TCP implementations may include several optional, optimization features, such as deferring ACKs, pacing data transmissions and selectively acknowledging packets. Further, real TCP implementations implement various procedures, such as keep-alive ACKs that ensure a flow remains active even in the absence of data packets. MesoNet does not simulate any of these features or procedures, unless required by particular simulated transport protocols.

3.1.4 Relating Abstract Time to Real Time

MesoNet simulation progresses with respect to abstract time steps that have no specific meaning within the domain of real time. Various parameters in MesoNet specify element capacities in terms of units per abstract time step. This abstract formulation of time is quite unusual for network designers and engineers, who tend to think in capacities dimensioned along real time (e.g., bytes per second, packets per second or bits per second) instead of abstract time. To achieve mapping between abstract time steps and real time, MesoNet includes an explicit parameter to specify the basic simulated time unit. By setting this parameter, an experimenter creates a link between abstract time steps and real time. Configured capacities, given in packets per time step, are converted into delays

(i.e., 1/capacity) per packet, and then scaled by the basic time unit. For example, suppose a router is defined as operating at 400 packets per time step and the basic simulated time unit is defined as one millisecond (0.001), then the time taken to process one packet is computed as $1/400 \times 0.001 = 2.5 \times 10^{-6}$ seconds and thus the router processes packets at $1/2.5 \times 10^{-6} = 4 \times 10^{5}$ packets per second.

3.2 Model Configuration

The operation of MesoNet elements is controlled by a number of parameters that must be specified by the experiment designer. This section describes these configuration parameters, divided into six categories: (1) simulation control, (2) definition of user behavior, (3) adaptation of the network topology, (4) description of sources and receivers, (5) specification of optional long-lived flows, and (6) transport protocols. The settings of MesoNet parameters associated with a specific simulation run are reported in an output file, as explained below in Sec. 3.2.8.

3.2.1 Simulation Control Parameters

Simulation control parameters allow an experiment designer to control various aspects of a simulation's operation, such as duration and measurement granularity. Parameter M defines the length of the measurement interval in abstract time steps. The model will make sample measurements (see Sec. 3.3 below) at the end of each measurement interval. Parameter MI defines the number of measurement intervals over which the simulation will execute. Thus, the total time simulated will be M * MI time steps. Parameter basicTimeUnit establishes a link between abstract time steps and simulated real time. This link is explained above in Sec. 3.1.4.

Parameter MB defines the number of measurement intervals that can be buffered by the model. When the measurement buffer fills, measurements are appended to appropriate files and the measurement buffer is cleared to accumulate subsequent measurements. This enables very long simulations to be conducted without using excessive memory.

Parameter exID can be used to assign a number to a simulation run. This number provides one means of distinguishing measurement file names and of associating a related set of measurement files. The system time is used as another means. Typically, the parameter exID would be set to specific run numbers associated with some experiment design. For example, a sensitivity analysis experiment requiring 64 runs would set exID sequentially from 1 to 64 to reflect each of the runs required by the experiment design. This would allow specific output files to be associated with specific experiment configurations.

Parameter RandOffset defines the offset used to parameterize seeds for the random number streams[2] used in MesoNet. Typically, for experiment designs that expose different system configurations to similar random conditions, the RandOffset should be set to the same value for all runs. Alternatively, when the same system configuration is to be exposed to varying random conditions, RandOffset should be set to different values for each run.

[2] MesoNet uses seven uniform pseudo-random number streams to control various aspects of the simulation, such as assigning congestion control algorithms to sources, generating think times, assigning network-interface speeds, starting sources, determining file sizes and file types, and selecting flow receivers.

3.2.2 Parameters Defining User Behavior

Parameters defining user behavior determine the demand that users will generate on the simulated network. The parameters defining user behavior represent two main aspects: think time and file size. Parameter lambdaOFF specifies the average number of time steps that a user delays between file transfers. This parameter is the mean of an exponential distribution. Parameter lambdaON specifies the average number of packets that a flow transfers. This parameter is the mean of a Pareto distribution. Parameter alpha is the shape of the Pareto distribution of file sizes. The Pareto distribution results in heavy-tailed file sizes, which some researchers have observed on the Internet [33-36].

The fundamental file size parameter (lambdaON) should be construed as representing files sizes of typical Web pages on the Internet. Sometimes, users may decide to download files, linked from web pages. For example, a user may download a report, a dataset, a song, a photograph or a video. The parameter Fp represents the probability with which a user decides to download a linked file. In general, linked files would be larger than typical web pages, so the parameter Fx represents the multiplier by which linked files will be increased beyond normal Web page sizes. Thus, a user will select a file size with an average of lambdaON. Then, with probability Fp, the file size will be multiplied by Fx to represent a linked file[3].

MesoNet can also simulate specific periods during which DD flows (between sources and receivers under pairs of directly connected access routers) exchange very large scientific datasets. Parameter Jon defines the proportion of total simulated time that must elapse before this mode of operation commences and parameter Joff defines the proportion of total simulated time that must elapse before this mode of operation ceases. Thus, this special mode commences when simulated time steps reach M x MI x Jon and ceases when simulated time steps reach M x MI x Joff. When this mode is operational, the selected file size of any DD flow will be multiplied by parameter Jx. Setting Jon $\geq$ 1.0 disables this mode of operation.

3.2.3 Parameters Adapting Network Topology

While requiring specification of a basic network topology (such as defined in Sec. 3.1.2), MesoNet permits an experiment designer to adapt that topology in several ways. Parameter settings allow adjustment of link propagation delays, router speeds and buffer sizes in routers.

Parameter deltaX is set to a factor that is used to multiply the link propagation delays that were specified in the base topology. Setting deltaX = 1.0 means that base propagation delays will remain unchanged. Setting deltaX > 1.0 increases propagation delays by the specified factor. Setting deltaX < 1.0 reduces propagation delays by the specified factor.

Parameter R1 establishes the basic forwarding speed (in packets per time step) of the backbone routers; however, since backbone routers handle transit traffic as well as inbound and outbound traffic, the actual capacity of backbone routers is set to R1 x BBspeedup, where BBspeedup could be set to reflect the number of transit links for a typical backbone router in a given topology. By default, BBspeedup = 1. MesoNet

[3] In Chapter 8 we introduce additional file-size probabilities and multipliers that allow simulation of larger files, such as software service packs and movie downloads.

enforces the usual relationship that backbone routers are faster than POP routers, which are faster than typical access routers. To ensure this relationship, parameters R2 and R3 are set to divisors[4] that are used to reduce R1. The speed of POP routers is set to R1/R2, while the speed of typical access routers is set to R1/R2/R3. Recall, though, that MesoNet topologies allow for some heterogeneity with respect to access routers: some access routers may be faster than typical access routers and some access routers may be connected directly to backbone routers. Parameters Bfast and Bdirect are used to increase the speed of such routers. Fast access routers are assigned a capacity of R1/R2/R3xBfast. The capacity of each directly connected access router is set to R1/R2/R3xBdirect.

In addition to assigning buffer sizes directly for routers, MesoNet also implements two alternative buffer-sizing algorithms. One algorithm, which embodies recommended practice [40, 44], sizes each router's buffer to accommodate the estimated average round-trip time (RTT) for routes transiting the router multiplied by the router's capacity (C). Thus, this algorithm sets the buffer size of each router to RTTxC, where RTT is the average round-trip time for the topology and C is the capacity of the specified router – so faster routers will have larger buffers and larger propagation delays will increase buffer sizes in all routers. The second algorithm, following the suggestion of McKeown and colleagues [37], divides the computed buffer size by the square root of the expected number of flows transiting a router. This algorithm then sets buffer sizes to RTTxC/SQRT(*N*), where *N* is the expected number of flows transiting a given router. MesoNet estimates *N* by aggregating the number of sources and receivers under a router (or under subordinate routers) and then dividing by two.

Parameter QszAlg specifies which buffer sizing algorithm to use: 1 means RTTxC; 2 means RTTxC/SQRT(*N*); 3 means (RTTxC + RTTxC/SQRT(*N*))/2; and a higher number means all buffer sizes are set directly to the given number. Setting QszAlg = 3 amounts to interpolating between algorithms 1 and 2. The buffer size, as set or as computed by the specified algorithm, is then multiplied by parameter Qfactor to establish the final buffer size of each router. Setting Qfactor = 1.0 uses the computed or assigned buffer size, while setting Qfactor > 1.0 increases buffer size and setting Qfactor < 1.0 decreases buffer size.

3.2.4 Parameters Describing Sources and Receivers

MesoNet uses several parameters to control the number, distribution and speed of sources and receivers in the fourth tier of the network topology. Every access router in the simulated topology contains some number of sources and receivers. Parameter baseSources defines the relative scale of the number of sources (and receivers) under a typical access router. For example, if baseSources = 100, then access routers will have on the order of 100 sources each, subject to variations arising through specification of related parameters, as discussed below. If one knows the number of access routers (*Na*) in a given topology, then one can compute *Na* x baseSources, which gives the approximate maximum number of active flows. For example, the topology in Fig. 3-1 has 139 access routers, so for baseSources = 100 the maximum number of active flows would be about 13.9×10^3. In order to reduce the blocking probability for a source seeking a receiver in any given domain, MesoNet ensures that the base number of receivers under each access

[4] We use the symbol "/" to designate division

routers is 4 x baseSources. In the example used here, then, the number of receivers would be on the order of 4 x 100 x 139 = 55.6 x 10^3.

Parameter deltaU permits linear scaling of baseSources in order to increase or reduce the approximate number of sources and receivers in a given topology. The approximate number of sources under each access router is then computed as deltaU x baseSources. Continuing the example, assume that deltaU = 2.0; then the approximate maximum number of active flows would become 2 x 100 x 139 = 27.8 x 10^3 and the number of receivers would be on the order of 2 x 4 x 100 x 139 = 111.2 x 10^3.

The sources and receivers in a topology must be distributed in some fashion. One possibility is to assign an equal number of sources and receivers to each access router. This would suggest a peer-to-peer world [20], where flow patterns are equally likely between any pair of access routers. Another possibility is to concentrate sources in a selected set of access routers and to place most receivers in a different set of access routers. This would suggest a client-server world [23], where flow patterns reflect the location of customers and the popularity of content providers. MesoNet provides six parameters that can alter the distribution of sources and receivers among access routers in each of three classes: normal (N-class), fast (F-class) and directly connected (D-class).

Given a set of sources and one access router of each class, parameters probNs, probNsf and probNsd specify, respectively, the probability that a source is located under the N-class router, the F-class router and the D-class router. Since each access router could contain deltaU x baseSources sources, three access routers would contain 3 x deltaU x baseSources sources. To distribute those sources under each class of access router, one multiplies the appropriate probability by the number of available sources. Continuing the example, given one access router in each of the three classes, one would find 3 x 2 x 100 = 600 sources. Suppose that probNs = 0.1, probNsf = 0.6 and probNsd = 0.3. Each N-class router would contain 600 x 0.1 = 60 sources, each F-class router would contain 600 x 0.6 = 360 sources and each D-class router would contain 600 x 0.3 = 180 sources. Given the topology shown in Fig. 3-1, this distribution changes the maximum number of active flows to 17.46 x 10^3, i.e., 105 (N-class routers) x 60 (sources) + 28 (F-class routers) x 360 (sources) + 6 (D-class routers) x 180 (sources). This means that more flows will originate from fast and directly connected access routers than from normal access routers. Similar adjustments can be made with respect to the distribution of receivers.

Given a set of receivers and one access router of each class, parameters probNr, probNrf and probNrd specify, respectively, the probability that a receiver is located under the N-class router, the F-class router and the D-class router. Since each access router could contain 4 x deltaU x baseSources receivers, three access routers would contain 3 x 4 x deltaU x baseSources receivers. To distribute those receivers under each class of access router, one multiplies the appropriate probability by the number of available receivers. Continuing the example, given one access router in each of the three classes, one would find 3 x 4 x 2 x 100 = 2.4 x 10^3 receivers. Suppose that probNr = 0.8, probNrf = 0.1 and probNrd = 0.1. Each N-class router would contain 2.4 x 10^3 x 0.8 = 1.92 x 10^3 receivers, each F-class router would contain 2.4 x 10^3 x 0.1 = 240 sources and each D-class router would contain 2.4 x 10^3 x 0.1 = 240 sources. Given the topology shown in Fig. 3-1, this distribution changes the maximum number of active receivers to 209.76 x 10^3, i.e., 105 (N-class routers) x 1.92 x 10^3 (receivers) + 28 (F-class routers) x 240 (receivers) + 6 (D-

class routers) x 240 (receivers). This means that more flows will have receivers in normal access routers than in fast and directly connected access routers.

The example used here might describe a client-server view of the distribution of sources and receivers. First, 58 % of data sources reside under the 28 F-class routers while the remaining 42 % of data sources reside under the other 111 access routers. Second, 96 % of all data sinks are under the 105 N-class routers, while the remaining 4 % of data sinks are under the 34 remaining access routers. This distribution of sources and receivers also influences the probability of various flow classes (recall the six flow classes described above in 3.1.2.3). For the example, 57 % of all flows will be FN flows and 35 % will be NN flows. The remaining 8% of flows are distributed as follows: 6.2 % DN flows, 1.9 % FF flows, 0.6 % DF flows, and the remaining 0.04 % DD flows.

Aside from differences in location, sources and receivers might also transmit packets at different speeds. For example, many users have computers that transmit at 100 million bits per second (Mbps), while some users have computers that transmit at 1 Gbps. MesoNet provides three parameters to specify speed differences among sources and receivers. Parameter Hbase defines the capacity (in packets per time step) of normal sources and receivers, while parameter Hfast defines the capacity of fast sources and receivers. Parameter FastHostProb specifies the probability that any given source or receiver is fast. Upon generation of a source or receiver, capacity is set to Hfast with probability FastHostProb and to Hbase with probability 1 – FastHostProb.

Another key characteristic of sources is the mechanism used to react to congestion on flows (see Chapter 5). MesoNet simulates seven alternate congestion control algorithms: standard TCP Reno [8-9], Binary Increase Congestion control (BIC) [61], Compound TCP (CTCP) [58], FAST [60], High Speed TCP (HSTCP) [52], Hamilton TCP (HTCP) [54], and Scalable TCP [53]. Sources are assigned one of these seven algorithms with probabilities specified by the following parameters: prTCP, prBICTCP, prCTCP, prFAST, prHSTCP, prHTCP and prSCALABLE. Each of these parameters can be set to any real value between 0 and 1, provided that the sum of the seven parameter values is 1. Each source retains its assigned congestion control algorithm for the duration of a simulation run.

Finally, MesoNet uses a probability distribution to determine the initial activation time for sources. Parameter prON specifies the probability that a source is activated immediately. Parameter prONsecond defines the probability that a source is activated as part of a second wave. Parameter prONthird defines the probability that a source is activated with the third wave. Remaining sources are activated in a fourth and final wave. Sources in the second wave are activated after a random delay, using an exponential distribution with a mean 1/3 x lambdaOFF. Sources in the third wave are activated after an average random delay of 2/3 x lambdaOFF. The remaining sources are activated after an average random delay of lambdaOFF.

3.2.5 Parameters Specifying Special Long-Lived Flows

MesoNet allows selected flows to be designated as long-lived flows, which start at a specified time and then transmit continuously, subject to congestion control constraints. Selected measurements are taken so that the behavior of long-lived flows can be monitored individually. Specification of long-lived flows requires the use of six configuration parameters.

Parameter maxLongLivedFlows (default value 3) determines the maximum number of long-lived flows that can be described, which dimensions the compile-time arrays used to hold associated measurements. One may change this parameter if the need arises to define more than three long-lived flows. Parameter LL_FLOWS defines the actual number of long-lived flows that are specified in a given configuration file. Note that the following constraint must hold: LL_FLOWS ≤ maxLongLivedFlows.

Four parameters describe the characteristics of a long-lived flow. Each of these parameters is an array with dimension LL_FLOWS, where a given array index specifies a particular long-lived flow and can be used to find the four, associated characteristics. Array LL_FLOW_SOURCES contains the access-router identifier under which the source of each long-lived flow can be found. Array LL_FLOW_RECEIVERS contains the access-router identifier under which the receiver of each long-lived flow can be found. Access-router identifiers are sequentially assigned integers that range between 1 and the number of access routers defined in a topology. For the topology defined in Fig. 3-1, access-router identifiers range over 1 (A1a) to 139 (K2d). For example, suppose LL_FLOW_SOURCES = {12, 24, 50} and LL_FLOW_RECEIVERS = {131, 102, 62}. These parameters specify three long-lived flows: (a) a flow where a source under access router 12 (B0a) transmits to a receiver under access router 131 (K0a), (b) a flow where a source under access router 24 (C0a) transmits to a receiver under access router 102 (I0a), and (c) a flow where a source under access router 50 (E0a) transmits to a receiver under access router 62 (F0a). Note that a specific source-receiver pair is created and pre-connected for each long-lived flow, so connection establishment procedures are not used.

Array LLon specifies the proportion of simulation time that must elapse before each designated long-lived flow will be activated. For example, given a configuration where M = 200 and MI = 7.5×10^3 (i.e., a simulation requiring 1.5 million time steps), then a long-lived flow with LLon = 0.4 will start at measurement interval number 3.0×10^3 (i.e., at time step 60.0×10^4). If the specified LLon value is 0, then the flow begins immediately. If the specified LLon value is 1.0 or greater, then the flow will not be activated.

Array SOURCE_TYPE specifies the type of congestion control mechanism to be used by the source of each long-lived flow. Acceptable values for the type of congestion control mechanism are shown in Table 3-3. Note that any other value for the type of congestion control mechanism causes MesoNet to assign a type probabilistically; probabilities are as specified in Sec. 3.2.4.

Table 3-3. Specification of Values for Parameter SOURCE_TYPE

SOURCE_TYPE Value	Congestion Control Mechanism
1	TCP Reno
2	HSTCP
3	CTCP
4	Scalable TCP
5	FAST
6	HTCP
7	BIC

3.2.6 Parameters for Transport Protocols

The remaining parameters required to configure MesoNet simulations relate to transport protocols. Most of these parameters concern congestion control algorithms, and also have recommended values. We defer discussion of these until Chapter 5. Three parameters relate to initial slow-start procedures, where MesoNet adopts the same behavior for all sources, regardless of congestion control algorithm.

Any TCP flow starts without knowledge of the surrounding environment, including network-interface speed, access-link speed and congestion on the network path to a receiver. For this reason, TCP adopts a probing procedure, known as slow start. During initial slow start, a TCP flow transmits relatively few packets. As acknowledgments are received successfully, a TCP flow then quickly increases its sending rate until a packet loss is signaled. After a packet loss, a TCP flow reduces its sending rate. Subsequently, receipt of acknowledgments causes a TCP flow to increase its sending rate less quickly than occurred prior to the loss.

The behavior of initial slow start is dimensioned primarily by two parameters. First, the initial congestion window (parameter INITIAL_TCP_CWND) determines the number of packets that will be transmitted prior to receiving an acknowledgment. As a default, MesoNet sets INITIAL_TCP_CWND = 2, which corresponds to the initial congestion window used in several Microsoft Windows™ operating systems [11]. During initial slow start, TCP sources increase the congestion window by one packet for each acknowledgment received successfully. This amounts to an exponential increase in the congestion window (and corresponding sending rate). After the congestion window reaches a specified value, known as initial slow-start threshold, further acknowledgments increase the congestion window (*cwnd*) by 1/*cwnd*, which amounts to a linear increase. Thus, a second parameter, INITIAL_TCP_SS_THRESHOLD, defines the value of the congestion window after which the increase becomes linear. An appropriate setting for this value is not widely agreed upon. Some experts [6] suggest that the threshold should be set arbitrarily large in order to quickly discover available bandwidth. Mark Carson (personal communication, November 12, 2008) reports that Linux sets the threshold based on feedback about the maximum number of packets that can be buffered by the receiver. Still others [10] suggest a low threshold might be prudent for its positive influence in reducing network-wide congestion.

Floyd observes [7] that using an arbitrarily large threshold in high-speed networks can lead to conditions where a long sequence of packets could be lost on a new flow, which results in wasted network capacity and poor user performance. Floyd also observes that using a low threshold in high-speed networks can lead to conditions where insufficient packets are sent early in a flow, which gives poor bandwidth utilization and poor user performance. Given the tradeoffs between an initial slow-start threshold that is too small and one that is too large, Floyd recommends using a limited slow start, which introduces a third parameter, MAX_SS_THRESHOLD. In limited slow start, a TCP source increases the congestion window exponentially with successive acknowledgments up to MAX_SS_THRESHOLD. Subsequently, receipt of successive acknowledgments causes the congestion window to be increased by 1/*cwnd*/(0.5xMAX_SS_THRESHOLD) until INITIAL_TCP_SS_THRESHOLD. In this second stage of initial slow start the congestion window increases logarithmically. After the congestion window reaches

INITIAL_TCP_SS_THRESHOLD, the congestion window increases linearly with new acknowledgments. Floyd recommends a value of 100 for MAX_SS_THRESHOLD.

MesoNet adopts limited slow start procedures, as recommended by Floyd. By default, the relevant MesoNet parameters are set as follows: MAX_SS_THRESHOLD = 100 and INITIAL_TCP_SS_THRESHOLD = an arbitrarily large integer. To deactivate limited slow start set MAX_SS_THRESHOLD ≥ INITIAL_TCP_SS_THRESHOLD.

Given the relatively small size of Web objects and high (and increasing) network speeds and relatively long propagation delays in networks, initial slow-start procedures can have a dominating effect on user throughputs. This is because sources must wait for feedback before increasing congestion window size and corresponding sending rate. The initial size of the congestion window coupled with the rate of increase can dictate observed throughput for small to medium sized files. Thus, observed throughputs could be biased by file size on individual flows that use variations in initial slow-start procedures. For this reason, MesoNet ensures that all TCP flows use the same initial slow-start procedures.

3.2.7 Parameters Identifying Monitored Links

The configuration file closes with parameters that may be used to identify links in the topology for which MesoNet should count packet transmissions over time. The specific meaning of these parameters is given below in Sec. 3.3.7.

3.2.8 Reporting Parameter Settings

Prior to commencing a simulation, MesoNet writes the settings of parameter values to a text file in the directory in which model measurements will be reported. The file and directory naming conventions mirror those used for measurement files (see Sec. 3.3.1 below). The first blank in the general file naming form is replaced by the token "ConfigurationSettingsFor". So, for example, parameter settings will be written to a file with a name similar to "ConfigurationSettingsForRun0TimeStamp41540.txt". This file also reports the value of various derived parameters, which are computed from combinations of parameter settings.

3.3 Model Measurements

MesoNet measures many facets of the spatiotemporal behavior of a simulated network. This section outlines the measurement approach and details some specific measures taken and reported during MesoNet simulations. The measures fall into six general categories. Summary measures report selected information that indicates general workload and performance during a simulation. Aggregate measures describe the temporal behavior of the entire network, as it unfolds during a simulation. Flow-class measures indicate the average temporal behavior of each of the six flow classes (recall 3.1.2.3) simulated by MesoNet. Long-lived flow measures follow the temporal behavior of any specific long-lived flows (recall 3.2.5) configured in a particular simulation. Per-router measures monitor the temporal behavior of each individual router within a simulated topology. Optional, link-level measures report the temporal progression of traffic on selected links within a simulated topology. To support the needs of specific experiments, MesoNet can also be augmented to provide measures that do not fall into any of the predefined categories. Below, we describe selected MesoNet measures reported in each category, as

well as how such measures may be augmented. Prior to describing specific measures, we discuss the general measurement regime, including file naming conventions, applicable to all categories of measures.

3.3.1 General Measurement Regime

Most of the measurements taken by MesoNet are reported as time series, where each data point in each series relates to a designated measurement interval. Thus, a typical measurement file will consist of a list of rows, where each row represents a measurement associated with a specific time interval. The first column in each row records the measurement interval and the remaining columns denote specific measures taken during that interval. Given this approach, one may plot each time series and may also compute statistical summaries (such as the average, median and variance) over all or part of each time series. With the exception of summary measures, MesoNet does not report column headers, so one should consult the descriptions below to understand the contents of each measurement file.

MesoNet uses a convention to name measurement files. The general file naming form is: "_____Run_TimeStamp_____.txt", where the first blank is replaced by the name of the specific measure, the second blank is replaced by the value of exID, and the third blank is replaced by a timestamp taken from the real-time clock of the computer executing the associated simulation. All measurement files for a specific simulation run are placed into a single directory, which is named using the following general form: "Run_TimeStamp_____". For example, if a simulation runs with exID = 0 and a real-time clock value of 34168, then the simulation would generate measurement files in directory "Run0TimeStamp34168". One file in the named directory would be labeled as "ActiveFlowsRun0TimeStamp34168.txt". This particular file reports the temporal evolution of the number of active flows (as discussed below in Sec. 3.3.3). The named directory would also contain additional files reporting other measures, as discussed below.

3.3.2 Summary Measures

MesoNet emits three files containing summary measures. The file that begins with the name "TotalPackets" reports two rows per measurement buffer (recall Sec. 3.2.1). The first row is a header that defines the columns in the second row. The six columns, from left to right, are: (a) the total number of measurement intervals covered by the row, (b) the total number of data packets input to the network, (c) the total number of data packets output from the network, (d) the total number of flows started during the simulation, (e) the total number of flows completed during the simulation and (f) the average number of SYN packets that were required to establish a successful connection. This information gives a general idea of the workload to which the simulated network was subjected.

The file that begins with the name "FlowThroughputsByType" reports two rows per measurement buffer. The first row is a header that defines the columns in the second row. Each row contains six pairs of columns, where each pair is associated with a flow class. The first column in each pair reports the total number of completed flows of the class and the second column reports the average throughput for a completed flow of the class. Flow classes are given in the following order: DD, DF, DN, FF, FN, and NN. These measures give an idea about the general user experience in each flow class.

MesoNet measures flow throughputs as the file size (in packets) divided by the number of time steps required to send the file and then divided by the basicTimeUnit defined for the simulation configuration.

The file that begins with the name "SYNrateByType" reports two rows per measurement buffer. The first row is a header that defines the columns in the second row. Each row contains six columns, where each column gives the average number of SYN packets that were required to establish a successful connection for each flow class. Flow classes are reported in the following order: DD, DF, DN, FF, FN, and NN. These measures give a general idea of the congestion faced by each particular flow class.

3.3.3 Aggregate Measures

MesoNet reports at least 17 aggregate measures depicting spatiotemporal behavior in a simulated network. See Table 3-4 for a summary. Each associated measurement file has the same general format: a series of two-column rows. The first column in a row gives the measurement interval when the measure was recorded and the second column gives the reported measure. Aggregate measurement files contain one row for every measurement interval in a simulation run. Below we describe each aggregate measure.

The file beginning with the name "ActiveFlows" reports the number of flows that were in the process of transferring packets in the network at the end of each measurement interval. This excludes flows that are attempting to establish connections, which are reported in the file starting with "FlowsConnecting".

The file beginning with the name "AverageSRTT" reports the average smoothed round-trip time[5] across all active flows at the end of each measurement interval. The average smoothed round-trip time (SRTT) is influenced by the network diameter and link propagation delays, as well as by queuing delays within each router on the path of each flow. To remove the influence of fixed properties (e.g., network diameter and link propagation delays) one could subtract the average round-trip propagation time of the simulated topology from the SRTT. Doing so would estimate the average queuing delay on network paths.

The file beginning with the name "ConnectionFailures" reports the total number of connection attempts that failed during each measurement interval. One could divide the number of connection failures by the number of active connections plus the number of connection failures to compute a connection-failure rate for each measurement interval.

The file beginning with the name "FlowCongestionWindowOverTime" reports for each measurement interval the average size (in packets) of the congestion window across all active flows. In general, a larger congestion window suggests less network congestion and higher throughputs on flows.

The file beginning with the name "FlowsAboveThreshold" reports for each measurement interval the number of flows that are operating with congestion control regimes different from standard TCP. Most alternate congestion control mechanisms prescribe normal TCP congestion control rules until the congestion window passes a particular threshold value. Thus, this measure reports the number of flows operating past

[5] Smoothed round-trip time (SRTT) is computed continuously for each flow using a weighted average that can assign greater emphasis to either recent or older observations. Thus, average SRTT is the average of the averages computed for each ongoing flow in a network simulation.

the associated threshold. Note that the threshold value is likely to be different for various alternate congestion control mechanisms (see Chapter 5). Flows operating under standard TCP Reno congestion control procedures are reported in the file beginning with "FlowsInNormalCongestionMode". Flows operating within the initial slow-start phase are reported in the file beginning with "FlowsInInitialSlowStart".

Table 3-4. Summary of Aggregate Measures Reported by MesoNet

Aggregate Measure	General Definition
Active Flows	Number of flows that are in the process of transferring data packets
Average SRTT	Average smoothed round-trip time across all active flows in the network
Connection Failures	Number of connection attempts that failed within a measurement interval
Flow Congestion Window	Average congestion-window size across all active flows in the network
Flows Above Threshold	Number of active flows operating with alternate congestion control procedures
Flows Completed	Number of flows completed within a measurement interval
Flows Connecting	Number of flows that are in the process of connection establishment
Flows in Initial Slow Start	Number of active flows that are in the initial slow-start phase
Flows in Normal Congestion Mode	Number of active flows operating with standard TCP Reno congestion control procedures
Loss Rate	Ratio of (packets input – packets output)/packets input for a measurement interval
NAKs	Average number of NAK packets received by each active flow in the network
No Receivers	Number of instances in a measurement interval where a source could not find an available receiver under a chosen access router
Packets In	Number of data packets entering the network during a measurement interval
Packets Out	Number of data packets exiting the network during a measurement interval
Retransmission Rate	Ratio of file size to number of data packets transmitted averaged over each flow completing during a measurement interval
Timeouts	Average number of timeouts recorded by each active flow in the network
Window Increases	Average number of window increases recorded by each active flow in the network

The file beginning with the name "FlowsCompleted" reports the number of flows that were completed during each measurement interval. Since the number of flows completed depends to some extent on the number of active flows, one could compute the flow-completion rate as the ratio of the number of flows completed over the number of active flows plus the number of flows completed. As the flow-completion rate approaches 50 % the simulated network is operating without much congestion. As the flow-completion rate approaches zero the simulated network is quite congested.

The file beginning with the name "LossRate" reports for each measurement interval the ratio of data packets input minus data packets output to data packets input. This gives a rough approximation of the rate at which a simulated network loses packets. The approximation is rough because the network also buffers packets in queues and on backbone links, so packets input in one measurement interval might leave the network in some future interval. Further, the loss rate considers only data packets, while other packet types may also be lost. Over time, the average network loss rate should be reasonably accurate as data points in the time series represent a sequence of rough approximations oscillating around the average.

The file beginning with the name "NAKs" reports the average number of NAK packets received by each flow during each measurement interval. Generally, as the NAK rate increases, the network is becoming increasingly congested. Of course, a congested network may lose an increasing number of NAKs, so severe congestion also exerts downward pressure on the NAK rate. Further, a higher number of active flows cause a given number of NAKs to be prorated over a larger number of flows.

The file beginning with the name "NoReceivers" reports for each interval the number of times a source found no available receiver under an access router selected as the destination for a flow. When a simulation is configured properly and when the destination-selection algorithm is working correctly, this time series should consist of all zeros. This measurement has no value beyond a check that a model is properly configured and uses a reasonable algorithm for selecting destinations.

The file beginning with the name "PacketsIn" records for each measurement interval the total number of data packets injected into a simulated network topology. This measure is also used to compute the loss rate. The number of packets input gives a reasonable picture of the network load in terms of packets per time step. Of course, the actual network load is double the reported number of packets input because data packets stimulate ACK and NAK packets in reply. ACK and NAK packets are not included in the count of input packets. So, for example, a simulation reporting an input of one million packets per time step is actually experiencing a load of two million packets per time step.

The file beginning with the name "PacketsOut" records for each measurement interval the total number of data packets exiting a simulated network topology. This measure is also used to compute the loss rate. As with packets input, packets output does not count ACK and NAK packets. One could compute an average network utilization by summing the rates of all simulated access routers and multiplying that sum by the measurement interval size and then dividing that multiplied sum into twice the number of packets output. This should yield a utilization value between zero and one[6]. For example, assume a measurement interval size of 200 time steps. Then, given the topology shown in Fig. 3-1, and given that the 105 N-class access routers operate at 200 packets per time step (ppts), the 28 F-class access routers operate at 400 ppts and the 6 D-class access routers operate at 2000 ppts, then the entire network can output at most ($105 \times 200 + 28 \times 400 + 6 \times 2000 =$) 44.2×10^3 ppts. (Note that, assuming a basicTimeUnit = 0.001 and a packet size of 1500 bytes, this equates to about 5×10^{11} bits per second.) In one measurement interval then, the simulated network could output at most ($200 \times 44.2 \times 10^3 =$) 8.84×10^6 packets.

[6] Note that since the computation is based on direct average measurement of data packets and an estimate for acknowledgment packets (as twice the number of data packets) the resulting utilization estimate is somewhat crude and can yield a value above one.

Assume that the average number of packets output per measurement interval, computed from a time series of packets output, is 8.15×10^5, then the average, aggregate network utilization can be estimated as $[(8.15 \times 10^5 \times 2) / 8.84 \times 10^6 =] 0.184$.

The file beginning with the name "RetransmissionRate" records the average retransmission rate across all flows completing in each measurement interval. The retransmission rate for a completed flow is computed as the ratio of the number of data packets actually sent on a flow minus the file size (in data packets) to the file size. Data packets sent beyond the file size comprise retransmitted packets arising from lost (data or ACK or NAK) packets or from timeouts. As a reasonable approximation, the average retransmission rate for a network should be somewhere around twice the loss rate (because both data and acknowledgment packets may be lost). In general, increased network congestion leads to increased losses, which lead to an increased rate of retransmissions. A lower rate of retransmissions indicates lower network congestion.

The file beginning with the name "Timeouts" records the average number of timeouts incurred on each active flow in each measurement interval. In MesoNet, timeouts arise most commonly when a data or acknowledgment packet is lost at the end of a data transfer. This occurs because no additional data is sent on a flow to stimulate additional acknowledgments. (Recall that MesoNet does not simulate procedures such as keep-alive acknowledgments.) Less frequently, timeouts occur when a path experiences unexpectedly large queuing delays. More rarely, timeouts might result when a complete congestion window of packets are lost. In general, an increased rate of timeouts suggests an increased level of network congestion.

The file beginning with the name "WindowIncreases" records the average number of times each active flow increases its window during a measurement interval. In general, a higher rate of window increases suggests lower network congestion. Of course, the rate of window increases might also be influenced by network capacity and round-trip times.

3.3.4 Flow-Class Measures

MesoNet reports at least 13 flow-class measures depicting the average temporal behavior of each flow class[7] in a simulated network. See Table 3-5 for a summary. Each associated measurement file has the same general format: a series of seven-column rows. The first column in a row gives the measurement interval when the measure was recorded and the next six columns give the reported measure for each flow class in the following order: DD, DF, DN, FF, FN and NN. Flow-class measurement files contain one row for every measurement interval in a simulation run. Below we describe each flow-class measure.

The file beginning with the name "ActiveFlowsByType" records the number of flows with data transfers in progress in each flow class. This measure gives a good idea of the distribution of flows between various types of access routers across the simulated network topology. The measured distribution of flow classes can be compared with the computed theoretical distribution of flow classes predicted from the simulation configuration. This measure also helps explain the degree of variance observed in temporal throughputs for flows of each class because some flow classes have decidedly fewer active flows than other flow classes. Further, this measure can be used to identify periods during which flows of some particular class were not active. A related file

[7] Chapter 8 introduces some other measures associated with different techniques to classify flows based on various additional characteristics.

("FlowsConnectingByFlowType") reports the number of flows trying to connect in each class.

The file beginning with the name "ConnectionFailuresByType" records the number of connection attempts that failed for each flow class during each measurement interval. This measure can be used to compute a connection-failure rate for each flow class and to compare failure rates across flow classes.

Table 3-5. Summary of Flow-Class Measures Reported by MesoNet

Flow-Class Measure	General Definition
Active Flows by Type	Number of active flows in each flow class
Connection Failures by Type	Number of connection attempts that failed in each flow class
Flow Congestion Window by Type	Average congestion-window size for active flows in each class
Flow Retransmission Rate	Ratio of file size to number of data packets transmitted averaged over each completing flow in each class during a measurement interval
Flows Above Threshold by Flow Type	Number of active flows operating with alternate congestion control procedures in each flow class
Flows Completed by Type	Number of flows completed in each flow class
Flows Connecting by Flow Type	Number of flows of each flow class that are trying to connect
Flows In Initial Slow Start by Flow Type	Number of active flows in each flow class that are operating within initial slow start
Flows In Normal Congestion Mode by Flow Type	Number of active flows in each flow class that are operating under standard TCP Reno congestion control procedures
NAKs by Flow Type	Average number of NAK packets received by each active flow in each flow class
Temporal Flow Throughputs	Average packets per time step output by flow class, divided by the basicTimeUnit
Timeouts by Flow Type	Average number of timeouts recorded by each active flow in each flow class
Window Increases by Flow Type	Average number of window increases recorded by each active flow in each flow class

The file beginning with the name "FlowCongestionWindowByType" records the average congestion window for active flows in each flow class. This measure can be used to compare congestion windows across flows of various classes. Congestion window size is influenced by a complex collection of factors, and may be an interesting measure to study.

The file beginning with the name "FlowRetransmissionRate" records the average retransmission rate for completed flows in each flow class. The measure is computed using the same relationships defined for aggregate retransmission rate.

The file beginning with the name "FlowsAboveThresholdByFlowType" records the number of flows in each flow class that are operating with congestion control

procedures other than standard TCP. The ratio of this measure to the measure of active flows by type reveals the proportion of active flows in each flow class that are operating with a congestion window above the threshold defined for the appropriate congestion control procedures used for each flow. Related files report the number of active flows in each class operating in initial slow-start phase ("FlowsInInitialSlowStartByFlowType") and also operating under standard TCP Reno congestion control procedures ("FlowsInNormalCongestionModeByFlowType").

The file beginning with the name "FlowsCompletedByType" records the number of flows in each flow class that were completed during each measurement interval. This measure may be combined with the measure of active flows by type to compute a flow completion rate for each flow class.

The file beginning with the name "NAKsByFlowType" records the average number of NAKs received by a flow of each class during each measurement interval.

The file beginning with the name "TemporalFlowThroughputs" records the average instantaneous throughput of flows in each flow class. This measure can be divided by the average file size (computed from the simulation configuration) for each flow class to estimate the average time taken by each flow class to transfer an average size file. For example, if a flow class has an average file size of 50 packets (75 Kbytes) and an average throughput of 100 packets (1.2 Mbits) per second, then it would take (50/100 =) 1/2 second to transfer an average size file.

The file beginning with the name "TimeoutsByFlowType" records the average number of timeouts that occur for a flow of each class in each measurement interval. Similarly, the file beginning with the name "WindowIncreasesByFlowType" records the average number of window increases received by a flow of each class in each measurement interval.

3.3.5 Long-Lived Flow Measures

MesoNet reports up to 8 long-lived flow measures depicting the temporal behavior of each long-lived flow configured in a simulated network. See Table 3-6 for a summary. Each associated measurement file has the same general format: a series of multicolumn rows. The first column in a row gives the measurement interval when the measure was recorded and each of the remaining columns report a measure associated with a specific long-lived flow. The recording order corresponds to the order in which long-lived flows were defined in the simulation configuration. Below we describe each measure recorded for long-lived flows.

The file beginning with the name "LongLivedFlowCongestionMode" records the congestion control procedures being used on each long-lived flow at the end of each measurement interval. When a long-lived flow is inactive, the measure reports the value "NONE". When a long-lived flow is active but using standard TCP congestion control procedures, the measure reports the value "NORMAL". Otherwise, the measure reports the specific congestion control procedure in use on each long-lived flow. Valid values for this measure include: "BIC", "COMPOUND", "FAST", "H", "HS" and "SCALABLE". For long-lived flows operating under FAST congestion control procedures a separate measurement file ("LongLivedFlowFASTalpha") reports the temporal evolution of the value of the FAST α parameter (see Chapter 5).

The file beginning with the name "LongLivedFlowCWNDs" reports the size of the congestion window at the end of each measurement interval for each long-lived flow. The file beginning with the name "LongLivedFlowNAKs" records the number of NAKs received by each long-lived flow in each measurement interval. The file beginning with the name "LongLivedFlowTimeouts" records for each long-lived flow the number timeouts experienced in each measurement interval. The file beginning with the name "LongLivedFlowWindowIncreases" records for each long-lived flow the number of increases in the congestion window that occurred in each measurement interval. The file beginning "LongedLivedFlowSRTT" reports the temporal evolution of the smoothed round-trip time for each long-lived flow.

Table 3-6. Summary of Long-Lived Flow Measures Reported by MesoNet

Long-Live Flow Measure	General Definition
Congestion Mode	Congestion control procedures in use on each long-lived flow at the end of each measurement interval
Congestion Window	Size of the current congestion window for each long-lived flow at the end of each measurement interval
FAST Alpha	Value of the FAST α parameter at the end of each measurement interval for each long-lived flow that operates under FAST congestion control procedures
NAKs	Number of NAKs received in each measurement interval for each long-lived flow
SRTT	Value of the SRTT for each long-lived flow at the end of each measurement interval
Throughputs	Average packets per time step output by each long-lived flow, divided by the basicTimeUnit
Timeouts	Number of timeouts recorded in each measurement interval for each long-lived flow
Window Increases	Number of window increases recorded in each measurement interval for each long-lived flow

The file beginning with the name "LongLivedFlowThroughputs" reports for each long-lived flow the average throughput over each measurement interval. Average throughput[8] is computed as the number of ACK and NAK packets sent by a flow's receiver in a measurement interval divided by the measurement interval size and then divided by the basicTimeUnit configured for the simulation.

3.3.6 Per-Router Measures

MesoNet reports measurements associated with each router in the simulated network topology. For the topology shown in Fig. 3-1, this would be 172 routers. MesoNet reports per-router measures in three classes: (a) backbone routers, (b) POP routers and (c) access routers. Six measures are reported for backbone and POP routers, while 12 measures are reported for access routers. This section describes the relevant measures and related measurement files. The description is divided into two parts: (a) measures common to all routers and (b) additional measures recorded only for access routers.

[8] This measure is often referred to as goodput.

3.3.6.1 Measurements Common to All Routers. MesoNet reports six measures for all routers. Table 3-7 gives a summary. Two measures (flows completed and packets forwarded) are aggregate measures summed over all measurement intervals. The remaining measures are time series reported at each measurement interval for each router. The measures are described below, beginning with the aggregate measures.

Files reporting aggregate router measures contain two rows for each measurement buffer. The first row is a header identifying the number of measurement intervals over which the measures were aggregated and the second row contains an aggregate measure for each router. For the topology shown in Fig. 3-1, aggregate measurements for backbone routers would contain 11 columns (routers A through K) while aggregate measurements for POP routers would contain 22 columns (routers A1 through K2) and for access routers would contain 139 columns (routers A1a through K2d).

Table 3-7. Summary of Measures Reported by MesoNet for Each Router

Router Measure	General Definition
Active Flows	The number of flows transiting each router at the end of each measurement interval
Flows Completed	The aggregate number of flows carried by each router over all measurement intervals
Losses	The number of packets discarded by each router in each measurement interval
Queue Length	For each router, the ratio of packets queued to buffer size at the end of each measurement interval
Packets Forwarded	The aggregate number of packets forwarded by each router over all measurement intervals
Utilization	The average utilization for each router over each measurement interval

The file beginning with the name "BackboneRoutersFlowsCompleted" gives the aggregate number of flows that transited each backbone router over all measurement intervals. Similar files, beginning with the names "SubnetRoutersFlowsCompleted" and "LeafRoutersFlowsCompleted", give the same measure for each POP[9] and access router, respectively. The file beginning with the name "BackboneRouterPacketsForwarded" records the aggregate number of packets forwarded by each backbone router over all measurement intervals. Similar files, beginning with the names "SubnetRouterPacketsForwarded" and "LeafRouterPacketsForwarded", report the same measure for each POP and access router.

Files reporting the spatiotemporal evolution of routers contain a time series with one row for each measurement interval. Each row contains multiple columns. The first column gives the measurement interval with which the row is associated and the remaining columns give a measure for each appropriate router, depending upon category. Thus, for the topology shown in Fig. 3-1, time series related to backbone routers would contain rows of 12 columns, while time series related to POP routers would contain rows of 23 columns. Similarly, any time series related to access routers would contain rows of 140 columns.

[9] MesoNet code refers to second-tier routers as Subnet routers rather than POP routers and refers to third-tier routers as Leaf routers rather than access routers.

The file beginning with the name “BackboneRoutersActiveFlows” records the number of active flows transiting each backbone router at the end of each measurement interval. Similarly, files beginning with the names “SubnetRoutersActiveFlows” and “LeafRoutersActiveFlows” report the number of active flows transiting each POP and access router, respectively. The file beginning with the name “BackboneRouterLosses” records the number of packets discarded by each backbone router in each measurement interval. Similar files beginning with the names “SubnetRouterLosses” and “LeafRouterLosses” records drop by each POP and access router.

The file beginning with the name “BackboneRoutersQLength” reports the ratio of packets queued to buffer size for each backbone router at the end of each measurement interval. Similar files, beginning with the names “SubnetRoutersQLength” and “LeafRoutersQLength” report the same ratio for each POP and access router. The file beginning with the name “BackboneRouterUtil” records the ratio of packets forwarded to capacity for each backbone router in each measurement interval. Similar files, beginning with the names “SubnetRouterUtil” and “LeafRouterUtil” record the same ratio for each POP and access router.

3.3.6.2 Measurements Unique to Access Routers. MesoNet reports six additional measures that are recorded only for access routers. These additional measures relate to the activity of flows transiting specific access routers. Table 3-8 gives a summary.

Table 3-8. Summary of Added Measures Reported by MesoNet for Access Routers

Access-Router Measure	General Definition
Connection Failures	The number of failed connection attempts for flows transiting each access router
NAKs	The average number of NAKs on flows transiting each access router
No Receivers	The number of instances when no receivers where available under each router
SYN Rate	The ratio of SYNs sent to first SYNs sent on flows transiting each access router
Timeouts	The average number of timeouts on flows transiting each access router
Window Increases	The average number of window increases on flows transiting each access router

The file beginning with the name “LeafConnectionFailures” reports the number of connection failures during a measurement interval for flows that would have transited each access router. This includes flows where either an intended source or receiver was subordinate to the access router. The file beginning with the name “LeafNAKs” records for each measurement interval the average number of NAKs received on flows transiting each access router. Similar files, beginning with the names “LeafTimeouts” and “LeafWindowIncreases”, report for each measurement the average number of timeouts and congestion-window increases, respectively, on flows transiting each access router. The file beginning with the name “LeafNoReceivers” reports for each measurement interval the number of times each access router could not accommodate a flow because no receiver was available. Finally, the file beginning with the name “LeafSYNrateLeaf” records for the ratio of SYNs sent to first SYNs sent during connection establishment

procedures involving each access router. This file reports the average SYN rates each time a measurement buffer is dumped. Each report is preceded by a header line indicating the number of measurement intervals over which the SYN rate was averaged.

3.3.7 Optional, Link-Level Measures

MesoNet allows an experimenter to select up to one simulated (unidirectional) link to monitor in each router tier. Link selection is controlled by three parameters in the configuration file. Parameter BB_LINK_TO_MONITOR identifies which backbone link in the topology will be monitored. Backbone links are numbered sequentially for a specified topology. For example, given Table 3-1, setting BB_LINK_TO_MONITOR = 1 would monitor the link from backbone router A to B, while setting the parameter to 2 would monitor the link from backbone router B to A. Similarly, assigning a valid POP-router identifier from a topology to POP_LINK_TO_MONITOR will activate monitoring on the incoming link from a parent backbone router to the designated POP router. POP routers are identified with sequentially increasing integers from one (POP router A1 in Fig. 3-1) to the number of POP routers (i.e., the identifier is 22 for POP router K2 in Fig. 3-1). A third parameter, ACCESS_LINK_TO_MONITOR, controls monitoring on an incoming link from a parent router (either backbone or POP) to an access router. The link is specified by assigning ACCESS_LINK_TO_MONITOR a valid access-router identifier. Access routers are identified with sequentially increasing integers from one (access router A1a in Fig. 3-1) to the number of access routers (i.e., the identifier is 139 for access router K2d in Fig. 3-1). By default all link-selection parameters are set to zero, which means that no links are monitored.

When link monitoring is active, MesoNet records the number of packets transiting each monitored link during each measurement interval and writes this information as a time series, where each row contains one two-column observation. The first column identifies the measurement interval and the second column gives the number of packets observed transiting the associated link during the measurement interval. The file beginning with the name “MonitoredBBLink” contains the time series for the specified backbone link. Similar files, beginning with the names “MonitoredSubnetLink” and “MonitoredLeafLink”, hold the time series for the specified POP and access links. Link-monitoring files will not be produced if link monitoring is inactive.

3.3.8 Augmenting Measures

MesoNet can be augmented by an experimenter to make specific measures that are not already incorporated. The purpose of this section is to describe the general approach one should take to accomplish such augmentation. The approach is illustrated by an example involving capturing a traffic-flow matrix. This augmentation is already incorporated into MesoNet as a set of comments. Reviewing these comments should provide further guidance regarding how to augment the measures recorded by the simulation.

To extend measurements made by MesoNet, one must add: (a) an array to hold the measurement of interest, (b) code to write the measurements to disk, (c) code to clear the measurement array and (d) in-line code to make the required measurements at the appropriate points in the model. To illustrate how this might be done, we consider an example where an experimenter decides to monitor traffic flows from selected observation points in a network to each access router.

First, one would define an array to hold the measurement of interests. In this case, the definition is for a three-dimensional array:

```
int inBoundPackets[NUMBER_OBSERVATION_POINTS][LEAF_ROUTERS][M_BUFFERS]
```

used to count packets. The first dimension is the number of observation points; the second dimension is the number of access (leaf) routers; the third dimension is the number of measurement intervals in one measurement buffer. In this case, the number of observation points is defined to equal the number of POP routers plus the number of directly connected access routers. The definition of the number of observation points is applied automatically, as long as the number of POP routers and the number of directly connected access routers is specified in the topology file.

Next, one adds code to the write_measurements( ) procedure. The added code must do four things: (1) define a name for the file in which measurements are recorded, (2) open the file, (3) append the measurements to the file and (4) close the file. The conventions for naming measurement files are to prefix a directory name (Dname) to the file name chosen by the experimenter and to append a time stamp (RTC), followed by the extension ".txt". The values for Dname and RTC are already defined by the program, so the experimenter must use them. Here is an example of the code to define the name for the file to record a traffic-flow matrix:

```
write string=Fname(Dname, RTC) ".//_//FlowMatrix_.txt".
```

This code places the constructed file name into the variable Fname. This must be followed by a statement that opens the file for append. Then one provides code to loop through the three-dimensional array and write out each measurement. In this case, each line written will contain four fields: (1) the number of the observation point where the traffic was observed, (2) the number of the access router to which the traffic was bound, (3) the measurement interval in which the observation was made and (4) the number of packets observed. The measurement-writing code automatically tracks the starting (previousEnd) and ending (currentEnd) measurement intervals for the measurement buffer; thus, the measurement interval is identified by adding the appropriate loop-control variable to the variable previousEnd. (For more details, the reader should see the related source code at the end of the write_measurements( ) procedure.) After writing out the buffer, the file must be closed because the file descriptor is reused for each file that is written.

The experimenter must also provide code in the clear_measurements( ) procedure so that the measurement buffer can be cleared after it is written to disk. The exact nature of the clear code depends on the construction of the measurement array. At the outermost level, the clear procedure loops through the measurement intervals (*i*) in the buffer. Thus, for arrays dimensioned only on time, one can simply add code in this outer loop. The clear procedure also loops through various second (*j*) dimensions (e.g., POP routers, access routers, backbone routers, flow types and long-lived flows). Due to this, one can add code to clear the flow matrix under the loop through access routers. However, the flow matrix has a third (*k*) dimension (observation points), so one must add a loop over the observation points. Inside this innermost loop, one simply sets inBoundPackets[k, j, i] equal to zero.

All that remains is to add code to record the required measurements. This must either be done in-line in elements of the model or in the forever loop within the actions clause of the class StateMonitor. Where to add the measurement code depends on the nature of the measurements. Sample-oriented measurements, taken periodically, should be added to the class StateMonitor, while event-oriented measurements, recorded when they occur, should be added in-line within the appropriate model elements. Recording of the flow matrix requires event-oriented measurements, so code must be added in-line. In this case, we wish to record each packet that successfully reaches an outgoing access router when sent from each source or receiver. This means that we must increment the inBoundPackets array in both sources and receivers. For a source, this requires incrementing the array each time a packet is injected successfully. A source injects packets in four places: (a) upon initially attempting to connect (SYN), (b) upon retrying a connect attempt (SYN), (c) when initially sending a data (DT) packet after becoming connected and (d) when sending each subsequent data (DT) packet. Thus, one must add a line of code to increment the array in each of these places. For a receiver, there is only one place where packets are injected into the network; thus, one must add a line of code at this point to increment the array. (The user can find these five places in the model code by searching for inBoundPackets).

3.4 Tracing Flow Behavior

To support debugging or to enable monitoring of individual behavior of flows, MesoNet provides facilities to print traces associated with selected flows. The model traces only one flow at any given time and randomly selects which flows to trace. When the flow currently being traced is closed, then tracing also ceases for that flow. When a flow turns on and no flow is currently being traced, then the new flow is selected for tracing. (If one knows the identity of a specific flow that should be traced, then the tracing variable may be set directly in the code to ensure that the desired flow is traced.)

In general, the tracing of flow behavior is disabled[10] because copious file writes occur and the model can be slowed significantly or (for large, long runs) can produce massive amounts of trace information. To enable flow tracing, one needs to define a symbol TRACE_TCP. This symbol is already defined in the code; however, by default the symbol is commented out – thus, to activate tracing, one simply must uncomment this symbol. Once tracing is activated, MesoNet will generate two files: (a) one file, whose name begins with “TCPstate”, that records the values of flow state variables at the time of particular events by the source associated with the flow being traced and (b) a second file, whose name begins with “TCPmessages”, that records each packet sent or received by the source associated with the flow being traced.

3.4.1 Tracing Flow States

For each flow being traced, MesoNet records the values of the variables defining the state of the flow’s source. These state variables are recorded each time a significant event occurs. Significant events include: (a) initial congestion window (CWND) established, (b) CWND increased, (c) CWND decreased and (d) timeout. Additional recording is

[10] The user is advised to activate flow tracing only for small, short simulations. Typically, flow tracing is used only for purposes of debugging or verification.

possible by defining a symbol WINDOW_CHECK. When this symbol is defined, state variables will also be recorded each time an ACK is received and a check is made regarding whether or not the CWND should be increased.

Each state recording consists of a single line in the associated file. The line contains 22 state variables. The first variable (Time) gives the time step when the event occurred, while the second variable (Event) identifies the stimulating event. The next six variables describe the flow, giving the source (Tx) and receiver (Rx), the type of congestion control algorithm (Type) used on the flow, the flow class (Flow), the time the flow started (OnAt) and the number of packets (MustDeliver) in the flow. Two additional variables outline the general progress of the flow, including (UnDelivered) how many flow packets remain undelivered (i.e., have not yet been acknowledged by the receiver) and (DTsSent) how many flow packets have been sent. The remaining dozen values give the detailed state of the flow variables, including (Phase) the phase of the flow (e.g., slow start or congestion avoidance), the current size (cwnd) of the congestion window and the current value (ssthresh) of the slow-start threshold. These state variables also include (unSentDTs) the number of packets that can be sent by the source and (unACKed) the number of unacknowledged packets that have been sent. Also provided are the values of four sequence numbers: (a) (nexSEQ#) the next sequence number that may be sent by the source, (b) (HighestACK) the highest sequence number from the receiver, (c) (lastNAK) the value of the source's next sequence number to be sent when the last NAK was processed by the source and (d) (TOseq) the value of the source's next sequence number when the last timeout occurred. The remaining three state variables relate to establishing a timeout period. These variables include the next time step (CRTO) when a timeout will occur, the number of time steps (RTO) that will be added to the current time to establish the time of the next timeout and the latest estimate of the smoothed round-trip time (SRTT) measured on the flow.

3.4.2 Tracing Packets

MesoNet also traces packets as they are sent and received by the source on each flow that is traced. Each packet transmission and reception is recorded on one line of the associated trace file. Each line records five variables from the packet. The recorded variables include: the time step (Time) when the packet was sent or received, the source (Tx) and receiver (Rx) of the flow associated with the packet, the type (Type) of the packet and the sequence number (SQ#) of the packet. Note that packets sent by the receiver will be recorded only if and when they reach the source. On the other hand, packets sent by the source will be recorded when they are sent, regardless of whether or not they reach the intended receiver.

3.5 Notes on Model Construction with SLX

This subsection provides a brief guide to the model constructed using the SLX [84-85] simulation language and development environment. The intent of this section is to guide those who wish to review the model source code. This section is not intended to provide a detailed description of the code[11]. MesoNet is constructed from three SLX files: (a) a

[11] MesoNet source code is freely available from the authors. Note, however, that executing MesoNet requires the SLX run-time environment, which is available commercially from Wolverine Software.

configuration file, (b) a topology file and (c) a file defining behavior of model elements. Each of these files is described below in a separate subsection. The file descriptions are followed by a short discussion of the performance properties of MesoNet.

3.5.1 Configuration File

The configuration file (e.g., MesoNetconfigDE.slx) provides the vehicle for defining configuration parameters, explained in Sec. 3.2. The configuration-file parameters are grouped (using comments) with the same headings as given in the subsections of Sec. 3.2; the parameter names in the source file conform to the parameter names used in those subsections. To support sensitivity analyses and other experiments, the model configuration file may be broken into two parts containing: (a) model parameters that remain fixed across experiment runs and (b) model parameters that change with each run, as guided by an experiment plan. In such cases, the variable portion of the file may be constructed by a configuration generator. This approach was used for the experiments described in subsequent sections of this report. When reviewing the source code associated with these experiments, one will likely find two configuration files for each run, where the parameters described in Sec. 3.2 are divided among the two files depending upon whether the parameters remain fixed or are varied from run to run.

3.5.2 Topology File

The topology file (e.g., MesoNet-AbileneTopologyIIc.slx) defines the layout and characteristics of routers and links under which sources and receivers will be deployed. The information in the topology file is used at the start of a simulation to construct a simulated topology. A topology file begins by defining the number and type of routers included in the topology, as well as the number of backbone links. The file also includes some type definitions to define the classes of routers that may attach to backbone routers and to POP (subnet) routers, as well as the classes of access (leaf) routers included in the topology. For each backbone link, the topology file defines (see array LP_DELAY) the one-way propagation delay (in time steps). These delays are scaled by the value of the deltaX parameter, which may alter the link propagation delays. Since the deltaX parameter is defined in the configuration file, the topology file must be included after the configuration file.

The topology file also contains a 2-D matrix (FORWARDING_LINK) defining the backbone link over which packets should be forwarded when bound between two backbone routers. The first dimension represents the source backbone router and the second dimension represents the destination backbone router. Two auxiliary matrices (SOURCE_BACKBONE_ROUTER and SINK_BACKBONE_ROUTER), which are indexed by backbone link, define the source and sink backbone router associated with each backbone link. This information is used to connect backbone routers and links when generating the topology.

Another 2-D matrix (ROUND_TRIP_DISTANCE) defines the round-trip times used to seed initial estimates of the round-trip delay in each direction between any pair of backbone routers in the topology. The estimates are computed by summing the one-way propagation delays associated with all backbone links transited along the forward and reverse path defined for each route between each pair of backbone routers. A variable (EB_DELAY) defines an estimated buffer delay (in time steps) that is added to the round-

trip propagation delay in order to account for some amount of queuing delay that may be experienced on a route. The seed estimates from ROUND_TRIP_DISTANCE are used by flows as an initial guess for the round-trip time that might be expected on a path. The initial guess for the round-trip time is used to set the initial timeout period for a flow.

Connections of POP and access routers to backbone routers are described by a 2-D matrix (SUBNET_PER). The first dimension represents the backbone routers in the topology. The second dimension represents the class of routers (e.g., POP and directly connected access routers) that may connect to the backbone. The information contained in this matrix is used to generate POP and directly connected access routers under each backbone router in the topology.

Connections of access routers to POP routers are specified by a three-dimensional matrix (LEAF_PER). The first dimension represents a backbone router. The second dimension represents the maximum number of POP (subnet) routers that may exist under any backbone router. The third dimension represents the class of access routers (e.g., fast or normal) that may exist under a POP router. The information contained in this matrix is used to generate (fast and normal) access routers under each POP router in the topology. Several topology files have been defined for use with MesoNet.

3.5.3 Model Behavior File

The main model file defines the behavior of model elements (as discussed previously in Sec. 3.1.1) and the overall behavior of the simulation. The main model file also defines measurement buffers, parameter mappings, auxiliary procedures and sets into which model elements of various types are sorted. Below, we discuss these features of the model in the following categories: (a) model elements, (b) simulation control, and (c) measurement buffers. We do not discuss parameter mappings or sets, which should be obvious from examining the source code.

3.5.3.1 Model Elements. The primary model elements are defined using SLX "active" classes, each of which has an individual behavior defined within an "actions" block. The active classes include: (a) LeafRouter, (b) SubnetRouter, (c) BackboneRouter, (d) BackboneLink, (e) Source and (f) Receiver. The behavior of each of these classes mirrors the description given earlier in Sec. 3.1.1. Each SLX class also includes an "initial" block, which is executed when the class is created. The "initial" block acts as a class constructor, establishing initial conditions. One active class, Source, also contains two methods (state and message) that support flow tracing. (Note that all active classes defined in this model are self-activating because the last statement of the "initial" block activates the class.) One "passive" class, Packet, encompasses the remaining model element. The contents of the Packet class mirror the description given above in Sec. 3.1.3.

3.5.3.2 Simulation Control. The simulation is started and controlled from the SLX "main" procedure, which is located at the end of the source file. The "main" procedure is also supported by a few auxiliary classes and procedures, which are discussed as the need arises. Model execution begins by constructing the timestamp (RTC) and the directory name (Dname) used to identify the model output files. If flow tracing is enabled, the associated output files are also created and opened at this time. Subsequently, the topology is examined, using procedure computeAverageRTT( ), and the associated round-

trip propagation delays are reported. Then the simulated topology is created, starting from the backbone routers downward. Each backbone router creates its own subordinate routers and those subordinate routers create their own children. The backbone links are constructed after the routers. Once the topology is constructed, the average buffer size is computed and reported for each router class (i.e., backbone, POP and access).

Next, the "main" procedure uses procedure createLongLivedFlows( ) to set up any long-lived flows that have been defined in the configuration file. Each long-lived flow is scheduled, using procedure scheduleLongLivedFlow( ), which creates a new source and receiver for the flow and sets initial conditions so that the flow is already connected and ready for data transfer at the desired time. As a final step, the procedure uses the SLX anonymous "fork" construct, which splits the processing into two independent "threads", the calling thread and a forked thread. Upon returning from the procedure, the forked thread waits until the simulation reaches the time step when the long-lived flow should begin and then completes activation of the associated source and receiver.

The main procedure uses the procedure writeConfiguration( ) to generate a file containing the settings of MesoNet parameters for a given run. When adding parameters to MesoNet, one should also insert related parameter-reporting code in procedure writeConfiguration( ). The inserted reporting code should follow the pattern of existing code within the procedure.

Prior to commencing the simulation, the "main" procedure also creates an instance of the StateMonitor class, which periodically makes measurements of the simulated system, manages the measurement buffers, writes measurement data to files and clears measurement buffers. To accomplish some of these operations, the StateMonitor class uses procedures write_measurements( ) and clear_measurments( ). As the last step before starting the simulation, "main" reports the date and time when the simulation started. The simulation commences when the "main" procedure delays itself, using an SLX "advance" statement, for the duration of the configured simulation time. In fact, the delay is slightly longer than the required time in order to permit the final measurement interval to be taken. Upon completing the simulation, the "main" procedure reports the date and time the simulation finished[12]. If appropriate, the flow-tracing files are closed and the "main" procedure terminates.

3.5.3.3 Measurement Buffers. The measurement buffers defined for the simulation appear after the comment line reading "MEASUREMENT INFORMATION". Note that some of the measurement buffers are guarded by #ifdef statements using the symbol SUBNETS. Similarly, one will find measurement statements within the source code also guarded by the same symbol. This permits these measurements to be skipped when a topology is defined without any POP routers.

3.5.4 Performance Properties of MesoNet

Performance of MesoNet is largely influenced by the performance of the SLX simulation compiler and run-time. As we will show, SLX performance is quite good. On the other hand, characteristics of the simulated configuration will also influence both processing time and memory requirements. We address these issues using samples from two experiments described in later chapters (Chapter 4 and Chapter 6). Table 3-9 provides a

[12] Later versions of MesoNet include logic to periodically estimate a projected completion time.

summary of characteristic performance for MesoNet when used to conduct the two experiments. Both experiments adopt the topology presented earlier (recall Sec. 3.1.2).

In a sensitivity analysis experiment, MesoNet was used to simulate 20 minutes of network evolution with an average of about 30×10^3 sources and 160×10^3 receivers. In these simulations, backbone routers operated at either 4.8 Gbps or 9.6 Gbps (depending on the configuration). A typical run required simulating about 10 million flows and processing around 1.2 billion packets. At any given time, about 10×10^3 flows were active and around 65×10^3 packets where in transit across the network. For a simulation of this scale, MesoNet required about 5.7 hours of processing time and around 166 Mbytes of memory.

Table 3-9. Characteristic Performance for MesoNet in Two Experiments

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
Simulated Minutes	20	25
Sources (avg.)	28.81×10^3	22.63×10^4
Receivers (avg.)	160.79×10^3	167.52×10^4
Total Flows (avg.)	971.557×10^4	874.4×10^5
Total Packets (avg.)	607.814×10^6	554.833×10^7
Active Flows (avg.)	9.991×10^3	32.194×10^3
Packets in Transit (est. avg.)	63.291×10^3	350.949×10^3
CPU Hours (avg.)	5.7	6.7×10^1
Memory in Mbytes (avg.)	166	119.2×10^1

In a comparison-robustness experiment, MesoNet was used to simulate 25 minutes of network operation with an average of about 225×10^3 sources and 1.7 million receivers. In these simulations, backbone routers operated at either 96 Gbps or 192 Gbps (depending on the configuration). A typical run required simulating over 85 million flows and processing aver 11 billion packets. At any given time, about 32×10^3 flows were active and around 350×10^3 packets where in transit across the network. For a simulation of this scale, MesoNet required about 67 hours of processing time and around 1.2 Gbytes of memory.

The comparison-robustness experiment increased the simulated system size by about an order of magnitude and ran the simulation for 25 % more simulated time, as compared with the sensitivity-analysis experiment. Thus, one might expect resource requirements to grow on the order of 12.5 times. The actual processing requirements grew by 11.75 times, which is within range of the estimate. (This nearly linear growth[13] in processing time may be attributed to the excellence of the SLX compiler and run-time environment.) The actual memory requirements increased only sevenfold. This increase was lower than the expected tenfold increase. The smaller than expected increase may be attributed to changes in the measurement strategy adopted between the two simulations. Further details about processing requirements and memory requirements are provided below.

[13] In subsequent experiments (see Chapter 9 vs. Chapter 8), simulating much larger networks with much higher router speeds for one hour of network operation, we found a 10-fold increase in network size and speed led to a 16-fold increase in processing time. We attribute this to a substantial increase in the size of the event lists that SLX needed to manage.

Table 3-10. Processing Requirements for MesoNet in Two Experiments

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
CPU Time (s) per simulated flow-minute	0.17	0.49
CPU Time (us) per simulated packet	33	44

3.5.4.1 Processing Requirements. As shown in Table 3-10, for the sensitivity-analysis experiment, MesoNet required an average of 170 milliseconds of CPU (central-processing unit) time to process a simulated flow-minute, while requiring 490 milliseconds per flow-minute under the comparison-robustness experiment. Table 3-10 also shows that the processing time per packet increased (by 33%) from the sensitivity-analysis experiment to the comparison-robustness experiment. The increase in per-packet processing time may be attributed to the increase in the size of simulation state (e.g., event lists) that SLX needed to process under the comparison-robustness experiment. The increase in processing times exhibited between the two experiments is quite reasonable.

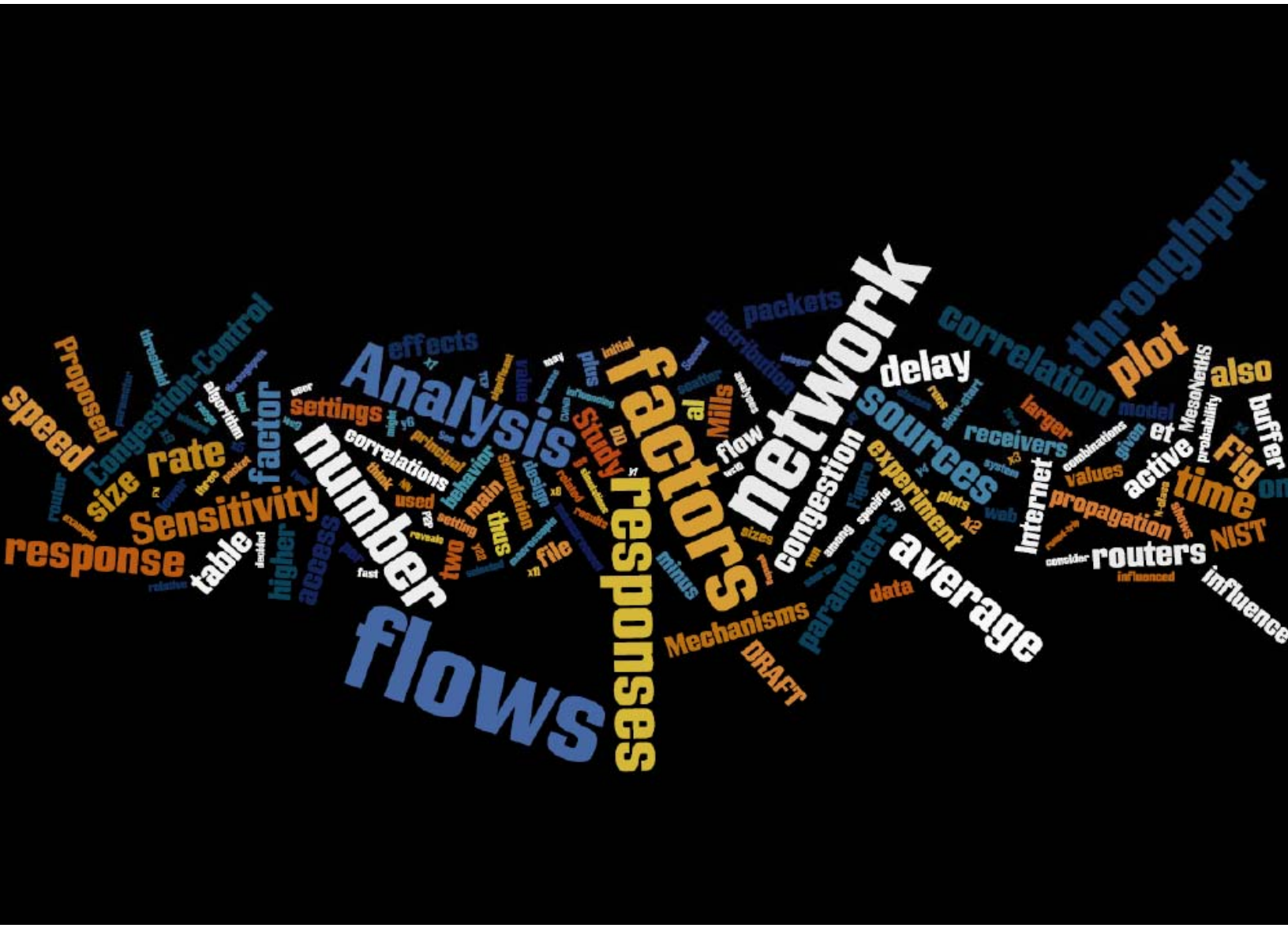
Table 3-11. Memory Requirements (Mbytes) for MesoNet in Two Experiments

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
Sources	16.8	100.6
Receivers	20.6	214.4
Sets and Membership	25.5	256.9
Simulation Processing State	26.7	251.6
Measurement Buffers	40.5	10.1
Other Memory	38.3	299.1

3.5.4.2 Memory Requirements. Table 3-11 shows the average allocation of memory by SLX for each of the two experiments. The memory allocated to sources and receivers reflects the number of each of these objects in the simulation. Sets and set membership includes memory associated with static model categories, as well as packet queues within routers, links, sources and receivers. Simulation processing state encompasses memory allocated to active objects, as needed to manage time evolution of the simulation. Measurement buffers are allocated as directed by the configuration file. The remaining memory is associated with the SLX run-time, with routers and links and with packets transiting the simulated network.

As shown in Table 3-11, and as expected, memory requirements generally expand by about an order of magnitude across the board. The exception is memory allocated for measurement buffers, which decreases for the comparison-robustness experiment to ¼ the size required for the sensitivity-analysis experiment. This occurs because the sensitivity-analysis experiment allocates measurement buffers to cover 6×10^3 measurement intervals, while the comparison-robustness experiment allocates memory for only 1.5×10^3 measurement intervals. As a tradeoff, the measurement buffers must be written to disk five times during each run of the comparison-robustness experiment but only once at the end of the sensitivity-analysis experiment.

Chapter 4 – Sensitivity Analysis of MesoNet



4 Sensitivity Analysis of MesoNet

This section discusses a sensitivity analysis of MesoNet, along with related correlation and principal components analyses. Sensitivity analysis [94] varies settings of a model's input parameters and assesses resulting changes in model outputs. Correlation analysis [90] examines the way in which two outputs vary with relation to each other when exposed to the same conditions. Principal components analysis generates orthogonal linear combinations of weighted measures that account for variance in model outputs. Here, we conduct a sensitivity analysis to understand the behavior of MesoNet and to discover the most significant model inputs that influence model response. To assess relationships between model inputs and outputs we use a 10-step graphical analysis technique developed at NIST. In addition to allowing analysis of input-output relationships, exercising a model over a wide range of its parameter space helps to reveal software implementation errors, which can be corrected prior to applying the model in particular studies. We use correlation and principal components analyses to identify significant aspects of MesoNet behavior. In general, application of correlation and principal components analyses can help to reduce the number of responses that must be used in subsequent statistical analyses. The results of our analyses serve to validate that MesoNet reasonably represents the macroscopic behavior of a network of TCP flows. The results of our analyses also help to answer some questions raised in the literature regarding the applicability of particular findings from small-scale simulations to a larger network.

The current practice of network modeling omits the use of sensitivity analyses, despite the fact that network researchers understand the benefits of such analyses [70, 72]. Why is this so? Most network simulators [76-83] are quite detailed, involving hundreds of parameters with potential settings that can range over many values. Running such simulations with large topologies and billions of TCP flows can be a daunting task, requiring substantial computational resources. In addition, configuring the parameter settings in such simulations can be time-consuming and tedious. Sensitivity analysis requires running a simulation through many combinations of settings. Thus, configuring and computing the required combinations for a detailed model with a large parameter space is infeasible.

Recently, two groups of researchers developed hybrid models [71, 73] that aim to reduce the computational requirements and range of parameters necessary to simulate TCP flows in reasonably large topologies. MesoNet was motivated by the same aims: establishing a new class of network models that can simulate many flows operating over a large network topology, while maintaining reasonable configurability and computational requirements. For example, MesoNet has on the order of 20 fundamental parameters and, depending on specific parameter settings, can simulate tens of billions of flows in days or weeks of processing time on commercial servers using x86-compatiable chips with cycle speeds of 2.6 GHz to 3.66 GHz. Thus, it becomes possible to contemplate conducting sensitivity analyses for this new class of network models.

Still, 20 parameters, each with a large possible range (n) of values, can suggest a large space of (n^{20}) combinations to consider. To circumvent such a problem, experiment designs used in many scientific disciplines have long adopted an approach where the range of values for system parameters is limited to a small number, typically two or three,

referred to by experiment designers as levels. As explained previously in Chapter 2, this approach, when applied to MesoNet, could limit the number of combinations to 2^{20} (just over a million). Still, running a million experiments would prove challenging when each experiment requires several days of processing time. Of course, individual combinations could be spread across independent processors to reduce the latency before all combinations are completed. For example, if 2^5 (32) processors were available, then 2^{20} simulations could be reduced to only 2^{15} serial executions. However, if each simulation required two days, then these simulations would still take 2^{16} days to complete. No one is willing to wait 180 years to collect data for a sensitivity analysis. Adding 32 additional processors to conduct the simulations would reduce the latency to around 90 years. Each additional 32 processors would cut the time further. Perhaps one day soon computation servers will offer 2^{16} processors in an affordable package. Such a computation engine would allow us to complete 2^{20} MesoNet simulations in about one month. In the meantime, we must adopt another approach to solve the problem.

Many scientific disciplines face situations where the number of desired experiments (even when considering only two levels per parameter) is unaffordable due to issues of cost or time. The best available practice in such situations is to use orthogonal fractional factorial (OFF) experiment designs [89] tailored to provide the maximum possible information from an affordable number of experiments. For example, if we could afford to run only 2^8 (256) MesoNet simulations, then we would use a $2^{20\text{-}12}$ OFF experiment design. Such a design would select 256 combinations of parameter settings that allow us to probe the parameter space in a balanced and orthogonal form. A balanced experiment design means that all combinations of parameter settings will be given an equal number of observations. An orthogonal experiment design means that observations will be spread equally throughout the space of possible parameter combinations. The properties of balance and orthogonality yield significant benefits when conducting statistical and graphical analyses of experiment data. In addition, properly selected combinations of parameters will limit the amount of confounding that arises when analyzing experiment data. When confounding arises a particular observed effect cannot be clearly attributed to a single factor or interaction of factors. Sometimes, domain knowledge can be used to resolve the uncertainty from confounding; however, one should strive to create an experiment design that eliminates confounding among at least the main effects[1] and as many two-parameter and three-parameter interactions[2] as possible. OFF experiment designs can be combined to good effect with a 10-step graphical analysis technique used in many scientific studies conducted at NIST. This 10-step technique is explained in Appendix D using detailed examples drawn from this sensitivity analysis.

The remainder of this chapter is organized into eight sections. Sec. 4.1 outlines our method for experiment design and analysis. Sec. 4.2 describes the specific experiment design used for the sensitivity analysis of MesoNet. Sec. 4.3 discusses the execution of the simulations and the data collection techniques. Sec. 4.4 presents a correlation analysis of 22 responses collected from each of the experiment executions.

[1] Main effects are changes in model response that can be attributed to changes in individual model parameters (or input factors).

[2] Interactions occur when changes in model responses can be attributed to simultaneous changes in multiple (e.g., two or three) model parameters.

Sec. 4.5 uses principal components analysis (PCA) as an alternate means to investigate relationships among the responses. Section 4.6 details the sensitivity analysis of MesoNet. In Sec. 4.7 we consider the effects of buffer sizing on network behavior. We conclude in Sec. 4.8.

4.1 Method

We use a method that involves five main elements. First, we use 2-level orthogonal fractional factorial experiment design (Sec. 4.1.1) to yield maximum information using the available computing resources. Second, we select candidate responses (Sec. 4.1.2) to analyze. Third, we employ correlation analysis and clustering (Sec. 4.1.3) along with principal components analysis (Sec. 4.1.4) to identify significant behaviors represented within the candidate responses. Third, we apply a 10-step graphical analysis (Sec. 4.1.5 and Appendix D) to provide insight into the main input parameters (or factors) driving the behavior of the simulation model. In addition, we augment our analyses with various exploratory plots (e.g., Sec. 4.1.6) designed to shed light on specific questions of interest. Below, we elaborate on these elements.

4.1.1 Experiment Design

We consider our model in the form of following equation: $[y_1, y_2, \ldots y_M] = f(x_1, x_2, \ldots x_N)$, which represents the model as a function transforming its N inputs (factors) into M outputs (responses). Designing an experiment consists of four main steps. First, we identify the N factors (model parameters) whose influence on system behavior we would like to investigate. Second, we select the number (L) of levels and then the settings ($s_1, s_2, \ldots s_L$) for each level of each factor ($x_1s_1, x_1s_2, \ldots x_Ns_L$). Third, we specify the combinations of factor settings that we intend to simulate. Fourth, we identify the M responses we are interested in investigating. We discuss each of these steps in turn.

4.1.1.1 Identify Factors. At a maximum, the factors include all parameters associated with a model of interest. Of course, this can be quite a large number, so one may wish to limit the specific parameters to investigate. Some parameters might specify control details, such as the number or granularity of measurement intervals and the seeds of random number generators. Typically, these may be fixed to specific values during a sensitivity analysis. Fixed parameters are not factors to be investigated in a set of experiments.

If the number of factors is still too large, other reduction steps may be adopted. For example, one may fix various factors and conduct sensitivity analyses with a limited number of runs. Repeating this process with various groupings of factors may identify some parameters as having limited influence on system behavior, at least for the range of settings envisioned for a particular experiment. Parameters that appear to have limited influence can be fixed during a sensitivity analysis that investigates more significant parameters. Domain expertise may also be applied to select various parameters to fix. One should exercise care in fixing particular parameters because some important elements of system behavior could be missed. Once parameters have been classified as fixed or variable for a given set of experiments, the variable parameters become the N factors for the experiment.

4.1.1.2 Select Number of Levels and Level Settings for Factors. Selecting the number of levels for an experiment determines the maximum number of combinations (L^N) that will be investigated. The most common practice in engineering experiments is to specify 2-level ($L = 2$) designs, which yield 2^N as the maximum number of combinations. 2^N designs result in nice properties of balance and orthogonality when OFF designs are used to reduce the number of combinations in a particular experiment. For this reason, we adopt $L = 2$ in our sensitivity analysis.

Given $L = 2$, one needs to select values for each of the N factors at each of two levels. This mapping of levels to factors yields specific parameter values to be used in a set of experiment executions. The two levels are typically encoded as a plus (+1) level and a minus (-1) level. This form of encoding simplifies many mathematical transformations that are applied during experiment design and data analysis. By convention, the larger value of a setting is assigned to the +1 level and the smaller value is assigned to the -1 level.[3]

Selecting the specific settings for the +1 and -1 levels of each factor is a key step that relies on domain expertise of an experimenter. Little general guidance exists because specific domains of investigation vary widely. In general, settings should be selected so as to be both realistic in the domain and also to stimulate the system sufficiently to reveal differences in response. When experiments are done using computers, preliminary simulations can be used to probe for the effects of varying specific parameters. No matter what settings are chosen, the analysis method relies upon an assumption that responses vary monotonically over the range of settings investigated. In cases where behaviors are non-monotonic, analysis of experiment data could completely miss significant and important behaviors. Further, the conclusions from data analysis are limited to (i.e., robust over) the range of settings investigated. For this reason, it is often prudent to run a second sensitivity analysis using different level settings to confirm conclusions from an initial sensitivity analysis. As discussed in Appendix C, we adopt this measure of prudence in our sensitivity analysis. Later, we plan to conduct a more complete sensitivity analysis with $N = 20$, covering the entire MesoNet parameter space.

4.1.1.3 Select Specific Combinations to Simulate. Ideally, one would run a full factorial experiment that simulates all 2^N possible combinations of level settings and factors. Often, though, executing 2^N runs would be unaffordable. For example, we selected ($N =$) 11 factors for our sensitivity analysis. A full factorial experiment would require 2^{11} (2048) runs. We could spread those runs over 16 processors, but each run requires between four and 10 hours of processing. We estimated that running a full-factorial experiment would require about 32 days of computing time plus overhead associated with managing the process. Such overhead includes configuring and monitoring simulations, collecting and summarizing data and recovering from various hardware and software

[3] Note that due to an encoding error in the design of our sensitivity analysis we inadvertently encoded higher network speed (our factor x2) as the -1 level and slower network speed as the +1 level. Unfortunately, this can lead to confusion when viewing some of the related plots. Despite this potential for confusion, the encoding approach works fine even when larger values are assigned to the -1 setting and smaller values to the +1 setting. Correcting this in our situation would require rerunning the related experiments, which would prove too costly.

failures that might arise. This illustrates that there may be other justified reasons for reducing the number of experiment runs even when one can afford the processing cost.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
-1	-1	-1	-1	-1	-1	-1	+1	-1	+1	+1
+1	-1	-1	-1	-1	-1	-1	-1	+1	+1	-1
-1	+1	-1	-1	-1	-1	-1	-1	+1	-1	+1
+1	+1	-1	-1	-1	-1	-1	+1	-1	-1	-1
-1	-1	+1	-1	-1	-1	+1	-1	-1	+1	+1
+1	-1	+1	-1	-1	-1	+1	+1	+1	+1	-1
-1	+1	+1	-1	-1	-1	+1	+1	+1	-1	+1
+1	+1	+1	-1	-1	-1	+1	-1	-1	-1	-1
-1	-1	-1	+1	-1	-1	+1	-1	-1	-1	-1
+1	-1	-1	+1	-1	-1	+1	+1	+1	-1	+1
-1	+1	-1	+1	-1	-1	+1	+1	+1	+1	-1
+1	+1	-1	+1	-1	-1	+1	-1	-1	+1	+1
-1	-1	+1	+1	-1	-1	-1	+1	-1	-1	-1
+1	-1	+1	+1	-1	-1	-1	-1	+1	-1	+1
-1	+1	+1	+1	-1	-1	-1	-1	+1	+1	-1
+1	+1	+1	+1	-1	-1	-1	+1	-1	+1	+1
-1	-1	-1	-1	+1	-1	+1	+1	-1	-1	-1
+1	-1	-1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	-1	+1	-1	+1	-1	+1	+1	-1
+1	+1	-1	-1	+1	-1	+1	+1	-1	+1	+1
-1	-1	+1	-1	+1	-1	-1	-1	-1	-1	-1
+1	-1	+1	-1	+1	-1	-1	+1	+1	-1	+1
-1	+1	+1	-1	+1	-1	-1	+1	+1	+1	-1
+1	+1	+1	-1	+1	-1	-1	-1	-1	+1	+1
-1	-1	-1	+1	+1	-1	-1	-1	-1	+1	+1
+1	-1	-1	+1	+1	-1	-1	+1	+1	+1	-1
-1	+1	-1	+1	+1	-1	-1	+1	+1	-1	+1
+1	+1	-1	+1	+1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	-1	+1	+1	-1	+1	+1
+1	-1	+1	+1	+1	-1	+1	-1	+1	+1	-1
-1	+1	+1	+1	+1	-1	+1	-1	+1	-1	+1
+1	+1	+1	+1	+1	-1	+1	+1	-1	-1	-1
-1	-1	-1	-1	-1	+1	-1	+1	+1	-1	-1
+1	-1	-1	-1	-1	+1	-1	-1	-1	-1	+1
-1	+1	-1	-1	-1	+1	-1	-1	-1	+1	-1
+1	+1	-1	-1	-1	+1	-1	+1	+1	+1	+1
-1	-1	+1	-1	-1	+1	+1	-1	+1	-1	-1
+1	-1	+1	-1	-1	+1	+1	+1	-1	+1	-1
-1	+1	+1	-1	-1	+1	+1	+1	-1	+1	-1
+1	+1	+1	-1	-1	+1	+1	-1	+1	+1	+1
-1	-1	-1	+1	-1	+1	+1	-1	+1	+1	+1
+1	-1	-1	+1	-1	+1	+1	+1	-1	+1	-1
-1	+1	-1	+1	-1	+1	+1	+1	-1	-1	+1
+1	+1	-1	+1	-1	+1	+1	-1	+1	-1	-1
-1	-1	+1	+1	-1	+1	-1	+1	+1	+1	+1
+1	-1	+1	+1	-1	+1	-1	-1	-1	+1	-1
-1	+1	+1	+1	-1	+1	-1	-1	-1	-1	+1
+1	+1	+1	+1	-1	+1	-1	+1	+1	-1	-1
-1	-1	-1	-1	+1	+1	+1	+1	+1	+1	+1
+1	-1	-1	-1	+1	+1	+1	-1	-1	+1	-1
-1	+1	-1	-1	+1	+1	+1	-1	-1	-1	+1
+1	+1	-1	-1	+1	+1	+1	+1	+1	-1	-1
-1	-1	+1	-1	+1	+1	-1	-1	+1	+1	+1
+1	-1	+1	-1	+1	+1	-1	+1	-1	+1	-1
-1	+1	+1	-1	+1	+1	-1	+1	-1	-1	+1
+1	+1	+1	-1	+1	+1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1	+1	-1	-1
+1	-1	-1	+1	+1	+1	-1	+1	-1	-1	+1
-1	+1	-1	+1	+1	+1	-1	+1	-1	+1	-1
+1	+1	-1	+1	+1	+1	-1	-1	+1	+1	+1
-1	-1	+1	+1	+1	+1	+1	+1	+1	-1	-1

Figure 4-1. Encoding Template for a 2^{11-5} Orthogonal Fractional Factorial Experiment Design

After estimating the time requirements for conducting a full-factorial experiment, an investigator must decide on the number of affordable runs. In making this decision, one should also consider the confounding effects that would arise for a particular choice.

For example, suppose we decided to limit the number of affordable runs to 64 instead of 2048. We would then need to select a subset of (2^6) combinations from among the complete set of 2^{11}. Experiment design theory [89] labels such a design as a 2^{11-5} (= 2^6) design. Experiment design theory also specifies which 64 combinations to select and reveals the resulting confounding structure for the experiment.

Fig. 4-1, taken from Dataplot [92], a software package available from NIST, shows the +1/-1 encoding of a 2^{11-5} OFF design as a matrix. Each row in the matrix represents a specific experiment run. Each cell in a given row specifies the level setting to be used for a designated factor (x1 through x11). Thus, having previously assigned +1/-1 level settings for each factor, an experimenter need only map level settings according to this table to create the specific combination of experiment parameters for each run.

Experiment design theory also specifies the precise confounding structure associated with this experiment design. A 2^{11-5} OFF design has no confounding of main effects with two-factor interactions, which is a desired property of an experiment design. Some main effects are confounded with some three-factor and higher interactions, but most systems are not driven by such higher interactions. From this we conclude that a 2^{11-5} OFF design would yield significant information for our sensitivity analysis.

A reasoning process such as outlined above should be used when selecting specific combinations to simulate. Of course, the reasoning process must be tailored to the specific number of factors and affordable runs. Experiment design theory provides appropriate algorithms to generate designs and determine associated confounding structures for any bounds of interest. The NIST Dataplot software [92] also provides encoded experiment designs and confounding structures for a range of typical OFF designs, as documented by Hunter and Box [89].

4.1.1.4 Select Responses to Examine. Often simulation models can measure system response through tens to hundreds of outputs, which might represent aspects of fewer significant underlying model behaviors. Usually, experimenters select a subset of model outputs to analyze because considering all available responses proves too time consuming, too costly or computationally infeasible. MesoNet, for example, can monitor the time-varying average aggregate behavior of the network for about 20 responses, can report about 6 time-varying properties for every router in a topology and can measure average throughputs experienced by users in six topologically determined flow classes. Summarizing and analyzing all of this data would prove time-consuming.

When choosing a subset of simulation outputs, experimenters may select outputs in a fashion that overemphasizes particular behaviors. These mistakes become particularly salient during careful exploration of a model's parameter space, where experimenters seek to understand the response of a model to changes in input parameters. Overweighting significant model behaviors can yield misleading conclusions, thus some method is required to identify precisely the model outputs that correspond to each significant behavior. Fodor [93] describes this mathematically as a dimension reduction problem: "given the p-dimensional random variable $x = (x_1,\ldots, x_p)^T$, find a lower dimensional representation, $s = (s_1,\ldots, s_k)^T$ with $k \leq p$, that captures the content in the original data, according to some criterion." Fodor goes on to survey numerous linear and non-linear techniques that may be applied to reduce the dimension of high-dimensional data sets. Adopting any of these techniques would provide a principled approach that

experimenters could use to identify significant model behaviors from a large collection of model output data. Of course, one wonders whether some techniques are superior to others. Fodor identifies principal components analysis (PCA) as the best (in terms of mean-square error) linear dimension reduction technique. In our analysis we use two techniques to reduce dimension in MesoNet response data. We use PCA and we also combine correlation analysis and clustering (CAC). Applying both techniques allows us to compare and contrast their findings, which provides additional information about MesoNet behavior.

4.1.2 Candidate Responses

We chose to examine a set of 22 MesoNet responses to which we applied PCA and CAC to identify lower dimensional response spaces representing the most significant model behaviors. This information could help us validate our model and could also help us to reduce the number of responses to analyze in subsequent experiments. We selected our 22 candidate responses from among the measurements (see Sec. 3.3) provided by MesoNet. We selected responses in two classes: (a) responses that depict macroscopic behavior of the network and (b) responses that indicate user experience for various flow classes. We discuss these response classes in turn.

4.1.2.1 Responses Characterizing Macroscopic Network Behavior. We chose 12 fundamental responses to characterize macroscopic network behavior and we augmented those with four derived responses in order to investigate how well the fundamental responses represented the intended information. Table 4-1 lists the responses we used to characterize macroscopic behavior. MesoNet records each response as a value associated with each measurement interval, providing a time series for each response. To compute the fundamental responses that we analyzed, we discarded data from the first 3000 of 6000 measurement intervals recorded. We then averaged the remaining data (from the second 3000 intervals) to obtain a mean value for each response. To compute a derived response, we mathematically manipulated some combination of fundamental responses, sometimes including a factor setting. The details are given as appropriate in Table 4-1.

We tracked the number of active flows (y1) over time and used that number to indicate the general amount of user activity in the network. Because more potential sources might lead to more active flows, we chose also to consider (y2) what proportions of potential flows were represented by the active flows. In this way, we could investigate whether the number of possible flows was a key determinant in the number of active flows, or whether the number of active flows was driven primarily by other factors. We measured separately the number of data packets entering (y3) and leaving (y4) the network because we wanted to understand what relationship, if any, exists between the rate at which packets are injected into the network and the number of active flows. Given the rate of packets entering and leaving the network, we could also measure the loss rate (y5), which should give us some rough indication of the amount of network congestion.

While the rate of data packets leaving the network gives us some idea of aggregate throughput, we were also interested in investigating the ability of the network to complete flows (y6), which could be combined with the number of active flows to yield a flow-completion rate (y7). Since we implemented TCP connection establishment (explained in Chapter 5) procedures, congestion could lead connection establishment to

fail. We measured the number of connection failures (y8) and also related the failures to the number of active flows (y1) to create a connection-failure rate (y9).

Table 4-1. Responses Characterizing Macroscopic Network Behavior

Response	Definition
y1	Active Flows – flows attempting to transfer data
y2	Proportion of potential flows that were active: Active Flows/All Sources
y3	Data packets entering the network per measurement interval
y4	Data packets leaving the network per measurement interval
y5	Loss Rate: y4/(y3+y4)
y6	Flows Completed per measurement interval
y7	Flow-Completion Rate: y6/(y6+y1)
y8	Connection Failures per measurement interval
y9	Connection-Failure Rate: y8/(y8+y1)
y10	Retransmission Rate
y11	Congestion Window per Flow
y12	Window Increases per Flow per measurement interval
y13	Negative Acknowledgments per Flow per measurement interval
y14	Timeouts per Flow per measurement interval
y15	Smoothed Round-Trip Time
y16	Relative queuing delay: y15/(x1x41)

For active flows, we were interested in understanding the average level of congestion. We suspected that several measures of flow congestion should be correlated. We measured the average retransmission rate (y10) for flows, which we postulated should be about twice the loss rate. We also measured the average congestion window per flow (y11) – larger congestion windows indicate that flows should be receiving better throughputs. In addition to the congestion window size, we chose to measure the average number of window increases (y12) received per flow during each measurement interval. To determine to what extent retransmissions arose from indicated losses vs. timeouts, we measured the number of negative acknowledgments (y13) and number of timeouts (y14) per flow.

Finally, we were interested in monitoring smoothed round-trip time (y15), which might provide some indication of congestion. We also wanted to see how changing buffer sizes influenced round-trip time. We computed a relative queuing delay (y16) by factoring out propagation delay from the smoothed round-trip time. We computed y16 because we wished to discover if there would be any differences in the pattern between smoothed round-trip time and queuing delay.

4.1.2.2 Responses Characterizing User Experience. Aside from aggregate network behavior, we were interested in exploring the throughputs received for the six possible flow classes allowed by MesoNet. This required monitoring six additional responses, as shown in Table 4-2. Here, the measure gives average instantaneous throughput for a flow in each class, so the metric captures the throughput for active flows rather than flows that have finished. As with the aggregate measures, we computed the average value for each flow class over the final 3000 measurement intervals of each simulation run.

Table 4-2. Responses Characterizing Instantaneous Throughput for Active Flows by Flow Class

Response	Definition
y17	Average Throughput for Active DD Flows
y18	Average Throughput for Active DF Flows
y19	Average Throughput for Active DN Flows
y20	Average Throughput for Active FF Flows
y21	Average Throughput for Active FN Flows
y22	Average Throughput for Active NN Flows

We chose to examine the throughput of the various flow classes in order to determine whether or not different factors affect the throughput of flows transiting different types of access routers. We collected separate throughput data for flows that completed and for flows that completed in each flow class. For purposes of our sensitivity analysis, we decided not to analyze the throughput data for completed flows.

4.1.3 Correlation Analysis and Clustering

As part of our analysis we wish to investigate the sensitivity of various model responses to model inputs. Of course, we are also interested in learning relationships among the responses. A reasonable hypothesis might be that correlated responses are influenced by the same model inputs. Further, clustering correlated responses into significant model behaviors might allow us to reduce the number of responses analyzed in future experiments. To determine relationships among responses we conduct a correlation analysis using the techniques described in this section. First, we generate scatter plots among all response pairs. Second, we compute correlations among each pair of responses. Third, we combine the selected scatter plots and correlation values into a single visualization. The combined visual can be ordered using several techniques to reveal correlation groupings. Finally, we select a correlation threshold above which we wish to consider correlations, and then generate an ordered index-index plot to highlight correlation groups and to help select specific responses for further study. We explain these four steps below. To aid our explanation, we use designators for various responses. The designators are y*N*, where y denotes a response and *N* denotes the number of the response. Here, *N* may range from 1 to 22 to correspond with the 22 candidate responses described in Sec. 4.1.2.

4.1.3.1 Y-Y Scatter Plots. Scatter plots of each pair of responses can visually reveal linear correlations and can also suggest structure beyond correlation. Fig. 4-2 shows a sample scatter plot between two responses from our sensitivity analysis. The abscissa gives values for response y22 (average instantaneous throughput among typical flows) and the ordinate gives values for response y7 (flow-completion rate). Perhaps unsurprisingly, the scatter plot reveals a positive linear correlation among the two responses. Higher throughput for typical flows, which are most numerous, leads to higher flow-completion rate. Perhaps surprisingly, the scatter plot also reveals a bifurcation in correlation structure. Attributable to the properties of our OFF experiment design, the scatter plot can be augmented to reveal the cause underlying this bifurcation. We discuss the use of other exploratory plots and analyses below in Sec. 4.1.6.

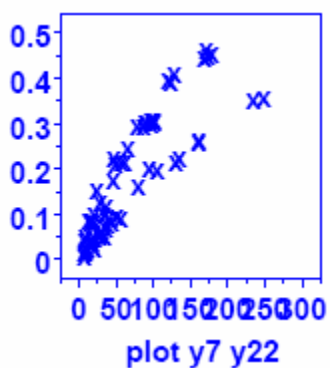


Figure 4-2. Enlargement of Sample Scatter Plot of Response y7 vs. y2 – y axis gives the flow completion rate (y7) as a proportion (ranging from 0 to 0.5, where each tick mark represents 0.05) and x axis gives average goodput of NN flows (y22) in packets per second (ranging from 0 to 300, where each tick mark represents 25 packets/second)

4.1.3.2 Correlation Computations. We also compute correlations among all pairs of responses generated from our sensitivity analysis. We compute the signed values, which separate positive and negative correlations, and the absolute values, which allow us to order correlations by magnitude. Fig. 4-3 provides a sample table of correlations, ordered by magnitude, where magnitude $\geq$ 0.9. The table consists of four columns: (a) absolute value of the correlation between a pair of responses (Y*i* and Y*j*), (b) the signed value of the correlation, (c) the identifier (*i*) of the first response in the pair and (d) the identifier (*j*) of the second response in the pair. Here, two subgroups are shown: (1) correlations $\geq$ 0.95 and (2) correlations $\geq$ 0.9 and < 0.95. In this particular sample, all correlations are positive.

|Corr(Yi,Yj)|	Corr(Yi,Yj)	i	j
0.9950	0.9950	5	10
0.9888	0.9888	19	22
0.9868	0.9868	3	4
0.9834	0.9834	18	20
0.9813	0.9813	19	21
0.9781	0.9781	21	22
0.9449	0.9449	12	22
0.9402	0.9402	8	9
0.9333	0.9333	9	10
0.9317	0.9317	12	21
0.9307	0.9307	13	14
0.9221	0.9221	5	9
0.9211	0.9211	1	2
0.9168	0.9168	12	19
0.9093	0.9093	8	10
0.9031	0.9031	7	12

Figure 4-3. Sample (and Partial) Table of Correlations among Response Pairs

We also plot a histogram (see Figure 4-4) of the absolute values of all pairs of correlations. This gives a concise view of the distribution of correlations. The histogram can help us select a threshold above which to consider the correlations. Fig. 4-4, for example, suggests that correlations greater than about 0.65 should be considered because there is a notable change above that value, appearing as a separate sub-distribution centered on a different mode. This sub-distribution includes around 40 of the 231 correlation pairs computed.

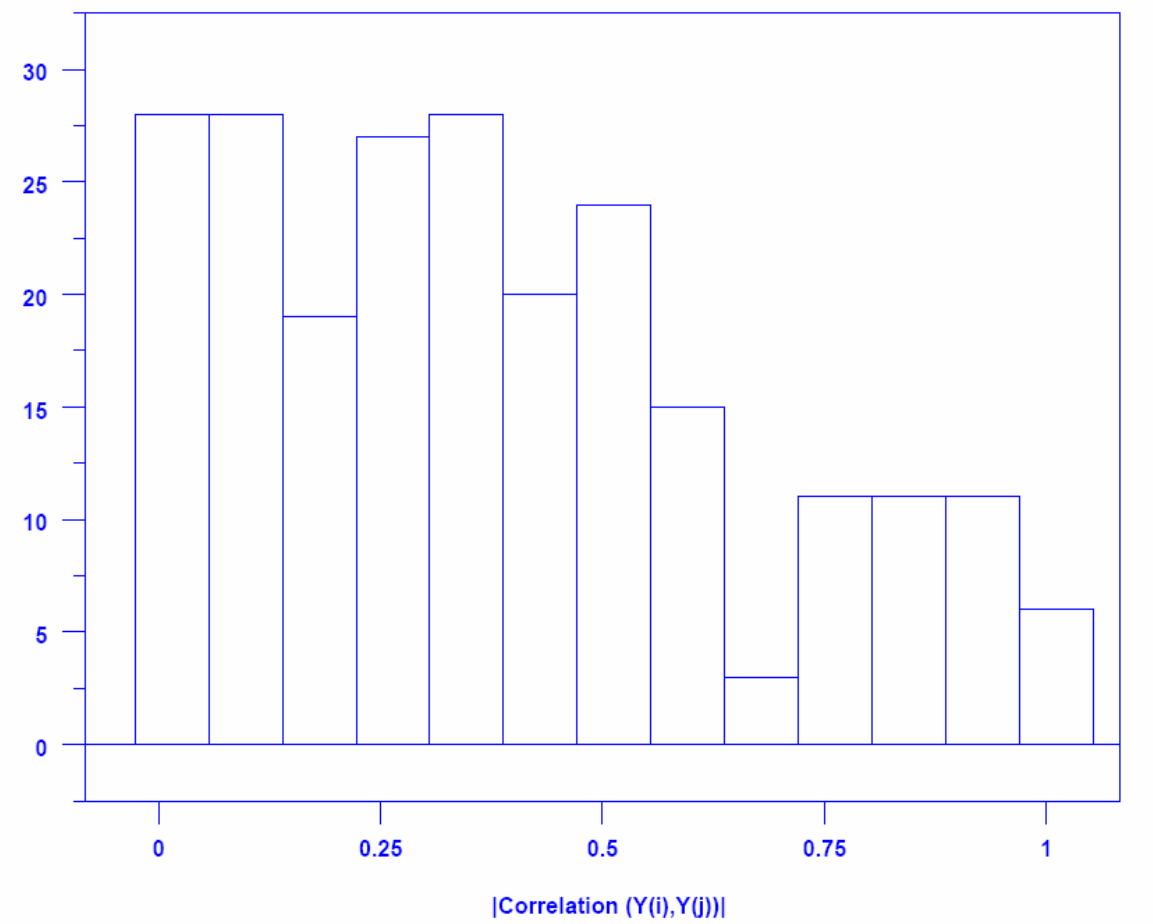


Figure 4-4. Sample Histogram of Correlation Magnitudes among Response Pairs (x axis depicts correlation strength divided into 13 bins, where each bin spans a range of size ~0.075 and y axis gives the count, or frequency, of correlation pairs appearing in each bin)

4.1.3.3 Combined Matrix Visualization. We can combine the response scatter plots and computed correlation values into a matrix visualization providing a concise view of all relevant information. Further, we can use color to highlight various correlation groupings. Fig. 4-5 gives a 6-x-6 subset taken from our complete matrix for all 22 responses.

The diagonal of the matrix identifies a particular response associated with each column and row. The scatter plots are displayed to the right and above the diagonal and the associated correlation values (multiplied by 100, rounded and truncated) are displayed to the left and below. For example, consider the response y3 (data packets input per time unit), which is third on the diagonal in Fig. 4-5. The scatter plot in the cell directly to the right of y3 and above y4 (data packets output per time unit) depicts the linear correlation

between y3 and y4. The cell directly below y3 and to the left of y4 reads 99, which is the associated correlation value. Not surprisingly, the correlation is positive and quite high. Similarly, the scatter plot related to y2 (proportion of flows that are active) vs. y3 is shown in the cell directly above y3 and to the right of y2. The related correlation value (37) is given in the cell immediately below y2 and to the left of y3. Perhaps the weakness of this correlation is surprising. Other scatter plots and correlation values may be located similarly. For example, the scatter plot in the cell in the upper right-hand corner depicts y1 (number of active flows) vs. y6 (flows completed per time unit) and the related correlation value (-6) appears in the cell in the lower left-hand corner of the matrix. While the negative direction of the y1-y6 correlation is not surprising, the lack of correlation might be unexpected.

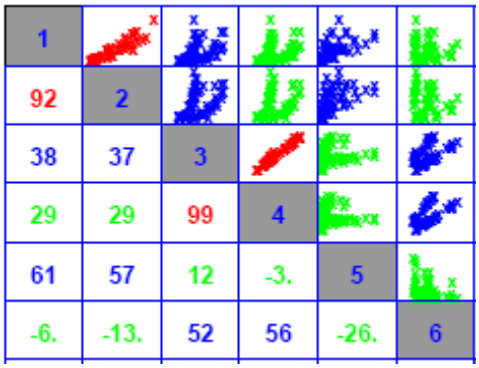


Figure 4-5. Sample 6-x-6 Subset from a Combined Matrix of Scatter Plots and Correlation Values

Thresholds may be selected for coloring the scatter plots and correlation values in the combined matrix visualization. In Fig. 4-5 we chose three colors, based on the magnitude of the correlation values. For correlation magnitudes 80 and above, we colored the related cells red. We colored cells blue for correlation magnitudes below 80 and greater than or equal to 30. The green cells represent correlation magnitudes below 30. After coloring, one can scan the matrix to visually group correlations by their strengths. The diagonal of the colored matrix may also be reordered, along with the related scatter plots and correlation values. Such reordering may readily indentify correlation groupings. For example, Fig. 4-5 could be reordered by descending mean, median or maximum correlation of each given response with all other responses. Later, in Sec. 4.3, we order our matrix by descending mean correlation, which nicely groups correlations among response pairs.

4.1.3.4 Index-Index Plot. Fig. 4-6 shows an index-index plot involving all 22 responses from our sensitivity analysis. Guided by Fig. 4-4, we display only correlations with magnitudes above 0.65. The x and y axes in Fig. 4-6 both list all 22 responses in order of numerically increasing designator (N = 1 to 22). Then a grid is formed. A point is placed at each grid intersection when the magnitude of the correlation between the related pair of responses exceeds 0.65. In Sec. 4.3, we use this index-index plot but we reorder the axes in a different form. The resulting correlation groups, not obvious in Fig. 4-6, become quite apparent after the axes are reordered (for example, see Fig. 4-22).

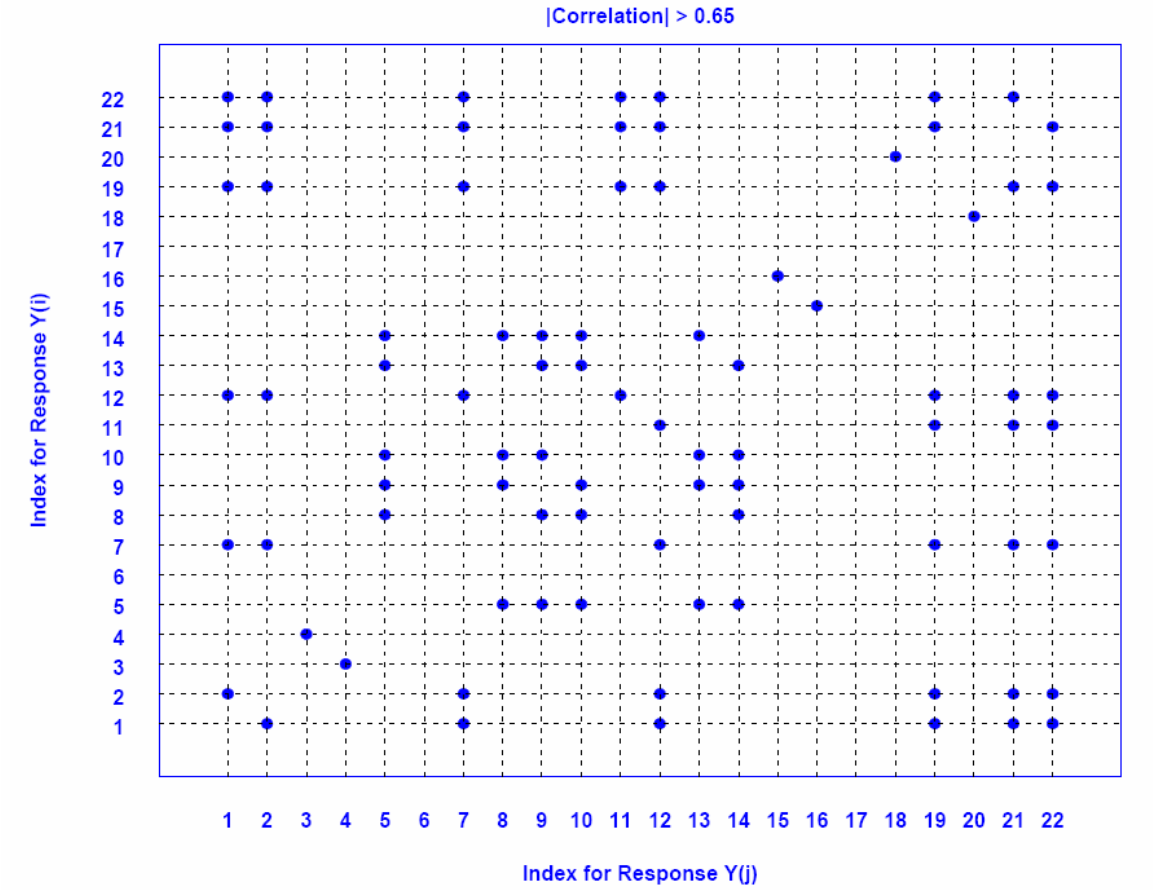


Figure 4-6. Index-Index Plot Identifying Response Pairs with Correlation Magnitude above 0.65

4.1.4 Principal Components Analysis

Principal Components Analysis (PCA) provides another approach to identify significant behaviors in model response data. PCA aims to reduce the dimensionality of model responses by finding orthogonal linear combinations (i.e., principal components, or PCs) of the responses that account for the largest variance. In essence, PCA identifies as many PCs as there are responses, with each PC being orthogonal to the others and with PC1 accounting for the largest variance in the data and PC2 second largest and so on to PC*n*, where *n* is the total number of responses. For many sets of responses, the first several PCs account for most of the variance in the data, and thus those PCs represent the most significant model behaviors.

In our case we have 64 samples (recall Fig. 4-1) for each of 22 response variables (recall Tables 4-1 and 4-2). Since variance depends upon the scale of each response, we must first normalize each response to have a mean of zero and a standard deviation of one. This can be done for a given response by subtracting the mean of the 64 samples from the response and then dividing by the standard deviation of the 64 samples. Such normalization will place all responses into comparable units.

In our application, each PC consists of a 22 dimensional weight vector representing the linear weighted combination of response variables necessary to generate

the PC. For example, Fig. 4-7 depicts a graphical representation of the weight vector for the first PC (PC1) from one of our PCAs. The figure depicts the 22 response variables on the x axis, while the y axis gives the (positive or negative) weighting. A horizontal line denotes zero weight. Given a weight vector for a PC, it is often customary in heuristic interpretation to suppress consideration of the low-weighted variables. In Fig. 4-7 for example, we might choose to suppress the following variables: y3, y4, y6, y15, y16, y17, y18 and y20. It is also customary to differentiate between "average" weights and "contrasting" weights, though such differentiation is not warranted in Fig. 4-7.

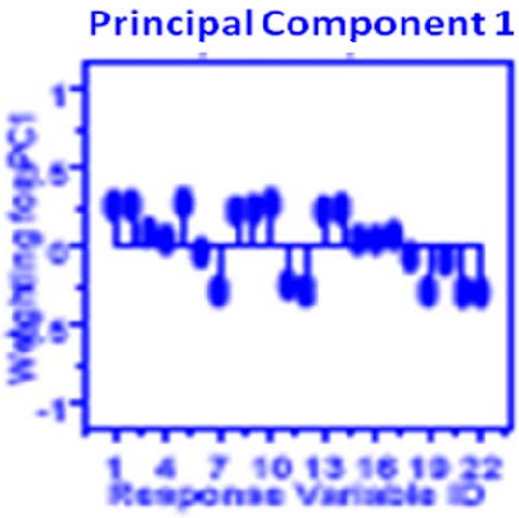


Figure 4-7. Enlargement of a Sample Weight Vector for First Principal Component (x axis identifies the response variable, ranging from 1 to 22 and y axis identifies weight, ranging from -1 to +1)

For each PC we can plot a histogram of the 64 values using appropriate components to represent the variance accounted for by the PC. For example, Fig. 4-8 gives the histogram corresponding to PC1. The x axis divides the standard deviation over the 64 values into appropriately sized bins and the y axis gives the count of values that fall into each bin. Above the plot we give the standard deviation accounted for by the PC. Given an entire set of such histograms, we can determine the relative variance accounted for by each PC by summing the standard deviations and then dividing each by that sum. For example, Fig. 4-23 recounts the 22 histograms representing each PC in one of our PCAs. In that case, the first four PCs account for about 86 % of the variance in the data.

4.1.5 10-Step Graphical Analysis of Selected Responses

Once we select the specific responses to examine, we can subject them to a 10-step graphical analysis regime developed at NIST. Each analysis step produces a different type of plot intended to reveal information about model responses. In this section we simply introduce the intent of each plot type, as shown in Table 4-3, which lists each of the ten plots and provides a summary of the purpose of each plot. In Appendix D we give detailed examples and explanation of each plot type. Here we introduce in detail only the main effects plot, which proved most insightful for purposes of our sensitivity analysis.

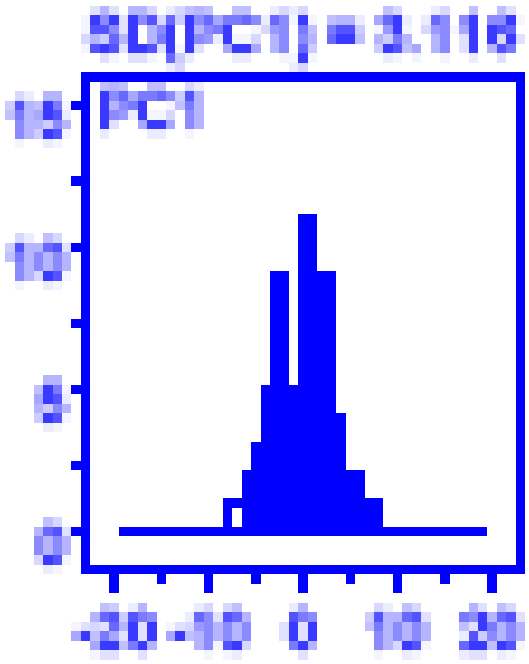


Figure 4-8. Enlargement of a Sample Histogram for First Principal Component (x axis identifies bins of normalized component values ranging from -20 to +20 and y axis the count of values within each bin). Above the plot is the standard deviation in the data accounted for by the Principal Component.

We illustrate the main effects plot using response variable y11, average congestion window (CWND) size. The transmission control protocol (TCP) manages a congestion window variable that represents the number of packets that can be sent prior to receipt of an acknowledgment. The larger the congestion window, the more packets that can be sent per unit time and thus the greater will be the transmission rate. For that reason, a network with a high average congestion window (y11) will be able to transmit more packets than a network with a lower average congestion window. In general, a congestion window is reduced when packets are lost, usually due to congestion. Lowering the congestion window slows the rate of packet transmissions in the network and thus should reduce congestion. Subsequent to a reduction, TCP allows the congestion window to increase linearly and so the rate of packet transmissions in the network should also increase. Once the transmission rate becomes too high, packets are lost and congestion windows are reduced and the rate of transmission slows and so on. Thus the average congestion window size might be used to represent the level of congestion in a network.

Fig. 4-9 gives a sample main effects plot, which is the most essential plot to identify the factors and settings driving a system's response. The x axis identifies each of 11 MesoNet parameters and the y axis gives the mean response. For each parameter the plot gives two means: (1) when the parameter is set to -1 value and (2) when set to the +1 value. Fig. 4-9 shows that the mean CWND size was about under 8.5 packets when network speed (X1) was high (-) and was about 4 packets under low network speed (+). For each parameter, a line connects the two means to indicate direction and magnitude of the effect when changing the parameter from its -1 to +1 value. Two numbers are reported just above each parameter label. The top number gives the effect in raw terms

(e.g., CWND size of 4.33 fewer packets under lower network speed) and the bottom number gives the change relative to (i.e., as a % of) the mean response, which is about 6.2 packets in Fig. 4-9 (i.e., the 4.33 packet change in CWND size is 70 % of the mean, which is called the relative effect). The plot also gives the number of parameters (k = 11) and observations (n = 64).

Table 4-3. Identity and Purpose of 10 Plots in the 10-Step Graphical Analysis (For sample and explanation of each plot see Appendix D)

Plot	Purpose
Ordered Data Plot	Reveal how combinations of parameter settings influence response
Multi-factor Scatter Plot	Reveal influence of individual parameter levels on response distribution
Main Effects Plot (see Fig. 4-8)	Reveal individual parameters having greatest influence on response
Interaction Effects Matrix	Reveal degree of influence of parameter pairs on response
Block Plot	Test robustness of statistically significant parameters in light of secondary or nuisance factors
Youden Plot	Reveal parameters and parameter pairs with greatest influence on response
|Effects| Plot	Reveal magnitude of a change in response due to specific parameters and parameter interactions
Half-Normal Probability Plot of |Effects|	Separate influential parameters and parameter interactions from those that are not influential
Cumulative Residual SD Plot	Provide information sufficient to construct a linear model to represent response data
Contour Plot	Suggest how alterations in parameter settings could influence system response in predictable directions.

Fig. 4-9 reveals that the most influential factor in determining CWND is network speed (70 % of mean) followed by three closely grouped factors: buffer-sizing algorithm (54 %), initial slow-start threshold and think time (53 % each). The distribution of sources also has a significant (50 %) influence. Notice that the plot reveals a smaller number of sources and receivers (x8 = -) leads to a (1.7 packet) larger average CWND than a larger number. A domain expert will understand that fewer sources sharing the same network mean that each source may transmit faster, which is reflected in a larger CWND. Thus, the main effects plot clearly reveals the nature of the influence of the factors and settings on the response.

In thinking about the main effects, an experimenter with domain knowledge might be quite pleased with the meaning of these results regarding the validity of the model. Fewer, simultaneously active, flows (x5 = +, x8 = - and x9 = -), higher network speeds

(x2 = -) together with more buffers (x3 = +) should permit higher CWND. Under these circumstances, the ability to increase the CWND to a higher threshold via initial slow-start (x11 = +) should also lead to higher CWND, because CWND increases faster during initial slow start.

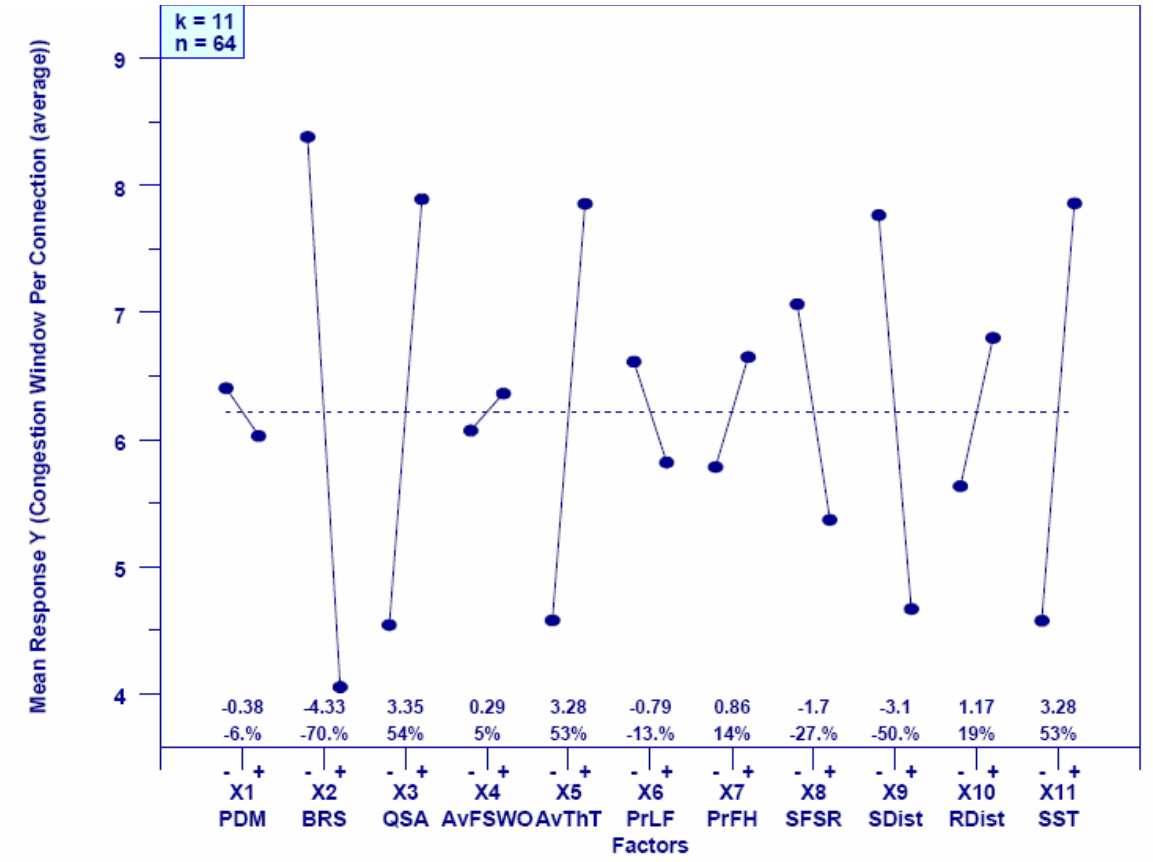


Figure 4-9. Sample Main Effects Plot for Response y11, average congestion window size, x axis lists 11 model parameters with a – and + value for each parameter, y axis gives average congestion window (CWND) size in packets, two average CWND sizes are given for each parameter, one size when the parameter is set to its – level and one size when the parameter is set to its + level, and a line connects the pair of average sizes for each parameter. Dashed line is the overall average CWND size (about 6.2 packets)

In Appendix D we illustrate the application of the entire 10-step graphical analysis technique to analyze model parameters influencing CWND size. An experimenter might also apply the 10-step graphical analysis technique to examine influences on principal components. We give an example of this technique in Sec. 4.6.2.

4.1.6 Other Exploratory Plots and Analyses

Using an orthogonal fractional factorial (OFF) design opens the possibility for a range of exploratory plots and analyses to supplement the correlation and clustering analysis, the principal components analysis and the 10-step graphical analysis presented so far. For example, bifurcations in response-response scatter plots can be explored by altering the scatter plot symbols to reflect factor settings. As a sample, recall Fig. 4-2, a scatter plot of y7 vs. y22, which revealed a bifurcation. One means to explore the underlying reason for

the bifurcation is to plot points using symbols, e.g., - when associated with minus settings for each factor, and as +, when associated with plus settings. Fig. 4-10 illustrates twelve scatter-plots for y7 vs. y22. The first plot, upper left-hand corner, repeats the scatter plot from Fig. 4-2. The remaining plots encode the plus (in blue) and minus (in red) settings responsible for the responses given each of the 11 factors (x1 through x11).

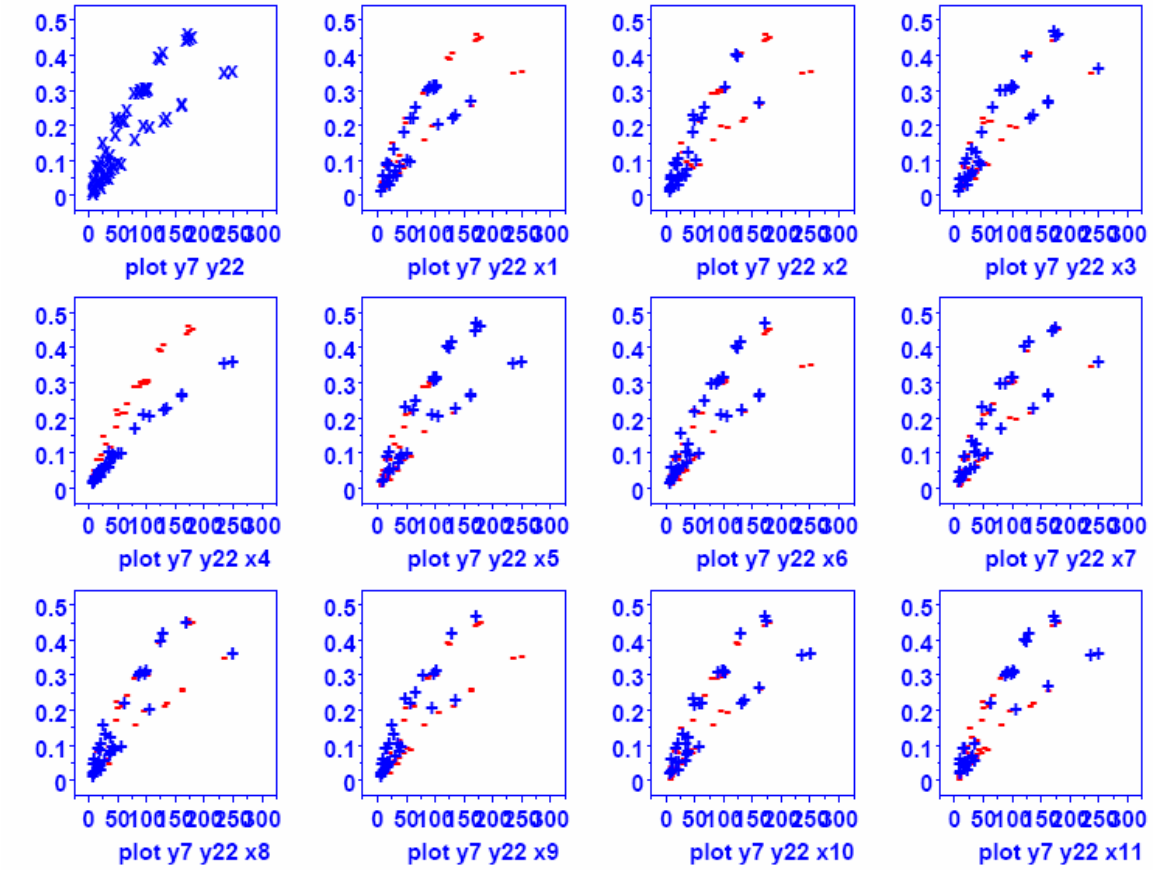


Figure 4-10. Sample Y-Y-X plot for Responses y7 and y22 – configuration of each plot is as explained for Fig. 4-2 – x axis is goodput in packets/second on NN flows ranging from 0 to 300 and y axis is flow completion rate ranging from 0 to 0.5

Examining Fig. 4-10, one can appreciate that factor x4 (average file size) is responsible for the bifurcation. Shorter file sizes result in higher completion rates (y7) and yet lead to lower average throughputs for typical flows (y22). Thinking this through reveals a sensible explanation. Shorter files spend a higher percentage of their transfer in TCP slow start, during which throughputs are lower. On the other hand, shorter files are generally transferred more quickly because they involve fewer packets. Since shorter files are transferred more quickly, more flows are completed per unit of time, so the flow completion rate is higher. Longer files spend a higher percentage of their transfer beyond TCP slow start, during which throughputs are higher. On the other hand, longer files require transferring more packets. Since it takes longer to transfer more packets, fewer flows are completed per unit time. Thus, the explanation for the bifurcation, as revealed in the y7-y22-x4 plot in Fig. 4-10, matches an explanation that appears reasonable to a

domain expert. Such analysis and reasoning can help to verify a model's correctness, or to reveal flaws.

Additional exploratory plots and analyses are also possible. For example, one can combine factor settings to create additional conditions and then compare the relative effects of varying each of the combined factor settings on the ordering of selected responses. More specifically, one could combine the 2-level settings for factors x1 through x3 (propagation delay, network speed and buffer size) to create $2^3 = 8$ conditions, and then examine the relative influence of varying each of the factors on selected responses. We use such an approach in Sec. 4.7 to explore the relative effects on system response due to changing network speed, propagation delay and buffer size. We defer a more detailed explanation of the technique to Sec. 4.7.

4.2 Experiment Design for MesoNet Sensitivity Analysis

This section outlines the experiment design used for the MesoNet sensitivity analysis, and explains the rationale underlying the design. The design consists of a 2^{11-5} orthogonal fractional factorial (OFF) design, which requires 64 simulation runs. We compared the results of the simulation runs across 22 responses (described in Sec. 4.1.2). Below, we summarize MesoNet parameters and we identify the 11 parameters chosen as factors in our OFF design. We then explain the levels, and related settings, chosen for each factor. We summarize the 64 specific combinations simulated.

4.2.1 MesoNet Factors

For this sensitivity analysis, MesoNet parameters may be divided into six general categories: (1) simulation-control parameters, (2) parameters controlling user behavior, (3) parameters adapting the characteristics of the network, (4) parameters altering the properties of sources and receivers, (5) parameters controlling the startup pattern of sources and (6) parameters related to TCP operation. We describe the specific parameters in each category. Parameter descriptions may identify a parameter's type as an integer or a float. In such cases, one should assume that an integer may take on values between -2^{31} and $+2^{31}$. A float may take on values in the range of 1.797^{-308} to 1.797^{+308}. The range of values for any other parameter types will be given explicitly. For more detail on these parameters see Sec. 3.2.

As we discuss the MesoNet parameters, we identify (highlighted in blue bold) which were chosen as factors for our sensitivity analysis and we give our reasoning. In each table, we also give (highlighted in red) the fixed values assigned to the excluded parameters. At the end of the section, we recap the parameters included as factors in the sensitivity analysis.

4.2.1.1 Simulation Control Parameters. Simulation control is affected by six parameters, as defined in Table 4-4. The sensitivity analysis will not consider the response of MesoNet to variations in simulation-control parameters. Thus, five of these parameters will simply be fixed (to the values shown in Table 4-4) across all experiment runs. The maximum propagation delay in our experiments will be around 200 time steps, which we select for our basic measurement interval duration. We set our fundamental time-step duration to 1 millisecond, so each measurement interval captures about 200 milliseconds (i.e., five measurement intervals cover one second). We run each simulation for 6000

measurement intervals, which is (6000/5 =) 1200 seconds (20 minutes). We set our random number seed to a fixed value for all simulations because we are interested in capturing changes due to parameter variations, and not variations due to randomness. We vary the run number from 1 to 64 to identify the particular configuration of factors used in specific experiments.

Table 4-4. Simulation-Control Parameters

Parameter	Definition	Type
P1	Number of time steps in a measurement interval (**200**)	Integer
P2	Number of measurement intervals (**6000**)	Integer
P3	Number of measurement intervals in a measurement buffer (**6000**)	Integer
P4	Run number (**1 to 64**, signifying a combination of factors)	Integer
P5	Random number seed (**200000**)	Integer
P6	Duration of each time step (**0.001 s**)	Float

4.2.1.2 Parameters Controlling User Behavior. Eight parameters, shown in Table 4-5, determine how individual users (sources) behave over the course of a simulation. The MesoNet sensitivity analysis considers only typical Web traffic, so parameters (P12-P14) dealing with jumbo file transfers will be assigned fixed values (see Table 4-5) that cause them to be deactivated. The five remaining parameters (P7-P11) are candidates to include as factors in the experiments.

Table 4-5. Parameters Controlling User Behavior

Parameter	Definition	Type
P7	Shape parameter for the distribution of web-object sizes (**1.5**)	Float
P8	**Average size (in packets) of web objects**	**Integer**
P9	**Average think time (in time steps) between web clicks**	**Integer**
P10	**Probability a user decides to download a larger document**	**Float**
P11	Factor by which web-object size is multiplied if it is a larger document (**10**)	Integer
P12	Proportion of simulation time that elapses before jumbo file transfers begin (**1.0**)	Float
P13	Proportion of simulation time that elapses before jumbo file transfers end (**1.0**)	Float
P14	Factor by which web-object size is multiplied if it is a jumbo file (**100**)	Integer

We decided to fix the value of parameters P7 and P11, so we selected only three parameters to control user behavior during the sensitivity analysis. We chose to fix P7 (= 1.5) because experiments with P7 set to 1.2 and 1.5 revealed little difference in results. We chose to fix P11 (= 10) because varying the size of Web objects (P8) will also implicitly vary the size of larger documents. Further, preliminary sensitivity analyses with P11 set to either 5 or 10 showed little influence on the results.

4.2.1.3 Parameters Adapting Network Characteristics. Nine parameters, shown in Table 4-6, may be varied to adapt characteristics of a network topology defined for a MesoNet simulation. We decided to vary only three of these parameters during the sensitivity

analysis. We wanted to be able to vary propagation delay, network speed and buffer sizes. Parameter P15 can alter the base propagation delays defined in a network topology. Network speed can be influenced by six parameters (P16-P21). We chose only to vary the backbone speed (P17), which has the effect of varying the speeds of the other routers because their speeds are expressed in terms of backbone-router speed. Thus, even though we fix parameters P18-P21 to the values given in Table 4-6, the speeds of the associated routers vary as we vary P17. We chose not to vary the speedup of backbone routers (P16 is fixed to 1) because our anticipated scenario of simulated Web traffic was unlikely to overwhelm the backbone routers with traffic.

Table 4-6. Parameters Adapting Network Characteristics

Parameter	Definition	Type
P15	**Factor by which to multiply basic propagation delays defined within a simulated topology**	**Float**
P16	Multiplier used to speed up backbone routers (**1**)	Integer
P17	**Backbone router speed (in packets per time step)**	**Integer**
P18	Divisor used to reduce the speed of POP routers relative to backbone routers (**4**)	Integer
P19	Divisor used to reduce the speed of access routers relative to POP routers (**10**)	Integer
P20	Multiplier used to increase the speed of directly connected access routers over typical access routers (**10**)	Integer
P21	Multiplier used to increase the speed of fast access routers over typical access routers (**2**)	Integer
P22	**Identification of a specific buffer-sizing algorithm to adopt**	**Integer (1 to 3)**
P23	Multiplier used to increase or reduce buffer sizes as computed by the algorithm selected by parameter P22 (**1.0**)	Float

Buffer sizes can be varied by choosing among several algorithms to calculate buffers in each router. This choice is controlled by P22, which we varied for our sensitivity analysis. Another parameter, P23, may be used to refine buffer sizing, either increasing or decreasing the basic buffer sizes computed by a chosen algorithm. For our sensitivity analysis, we decided to stick with the choice among alternate algorithms, so we fixed the value of P23 to 1.0.

4.2.1.4 Parameters Altering Properties of Sources and Receivers. Nine parameters, shown in Table 4-7, control the properties of sources and receivers within the model. Controllable properties include: the network interface speeds of hosts on which sources and receivers operate, the relative number of sources and receivers and the distribution of sources and receivers within the network topology.

In our sensitivity analysis, we are interested in examining the effects on responses as the number and speeds of sources and receivers is changed and as the distribution of sources and receivers is altered in the topology. This requires varying the six parameters (P26, P28-P32) highlighted in Table 4-7. We decided there is no need to vary the speeds of either basic or fast hosts, so we simply fix the speed of each (P24 = 1 and P25 = 8). Varying the probability a host is fast and the number of sources and receivers in the network should provide sufficient variation in the number of fast and slow hosts in the

network. Similarly, we decided not to vary the base number of sources (P27 = 100) under an access router because the number of sources and receivers are determined by multiplying the base number by a scaling factor (P28). Thus, varying P28 achieves sufficient variability among the number of sources and receivers in a topology.

Table 4-7. Parameters Altering Properties of Sources and Receivers

Parameter	Definition	Type
P24	Speed (packets per time step) of basic host (**1**)	Integer
P25	Speed (packets per time step) of fast host (**8**)	Integer
P26	**Probability a host is fast**	**Float**
P27	Base number of sources under an access router (**100**)	Integer
P28	**Multiplier by which to scale the base number of sources**	**Float**
P29	**Probability source is located under a typical access router**	**Float**
P30	**Probability source is located under a fast access router**	**Float**
P31	**Probability receiver is located under a typical access router**	**Float**
P32	**Probability receiver is located under a fast access router**	**Float**

MesoNet permits the distribution of sources and receivers to be varied by reallocating some sources and receivers among the three classes of access router (normal, fast and directly connected). Two parameters (P29 and P30) control the allocation of sources (note that the probability a source is allocated to a directly connected access router is equal to 1 – P29 – P30). Similarly, two parameters (P31 and P32) control the allocation of receivers. We chose to combine the three probabilities associated with sources into a single factor and also to combine the three probabilities associated with receivers into a single factor. Thus, the six highlighted parameters in Table 4-7 will comprise only four factors in our sensitivity analysis.

4.2.1.5 Parameters Controlling Source Startup Pattern. Sources are started randomly in stages: some portion start in the ON state, some portion enter the ON state after about 1/3 of the average think time, some portion enter the ON state after about 2/3 of the average think time and the remaining sources enter the ON state after about the average think time. This startup pattern is controlled by three parameters (P33-P35) as shown in Table 4-8. Subtracting the value of these three parameters from one reveals that half of the sources start after about the average think time: 1 – 0.25 – 0.08 – 0.17 = 0.50.

Table 4-8. Parameters Controlling Source Startup Pattern

Parameter	Definition	Type
P33	Portion of sources that start ON (**0.25**)	Float
P34	Portion of sources that come ON after about 1/3 average think time (**0.08**)	Float
P35	Portion of sources that come ON after about 2/3 average think time (**0.17**)	Float

For two reasons, we decided not to vary parameters controlling source startup pattern. First, we discard the first half of our observations and consider only the second half. Thus, the influence of startup pattern should not be evident in the data. Second, we conducted preliminary sensitivity analyses where we varied the startup pattern, along

with other parameters, and found that such variations had no influence on the long-term results.

4.2.1.6 Parameters Related to TCP Operation. MesoNet includes only three parameters, given in Table 4-9, controlling the operation of standard TCP. Given lack of widespread agreement on the choice of initial slow-start threshold for TCP, we were interested in exploring the influence of the threshold on network performance. We decided to fix the other two parameters: initial congestion window (P36 = 2) and threshold (P38 = 100) for switching from exponential slow-start increase to logarithmic increase.

Table 4-9. Parameters Related to TCP Operation

Parameter	Definition	Type
P36	Initial TCP congestion window (**2**)	Integer
P37	**Initial slow-start threshold**	**Integer**
P38	Threshold for switching from exponential to logarithm slow-start (**100**)	Integer

Parameter P38 influences slow-start operation only if the value of the initial slow-start threshold (P37) exceeds the value of P38. Assuming this condition, the congestion window begins at the value of P36 and then increases exponentially with each round-trip time until reaching the value of P38, after which the congestion window increases logarithmically until reaching P37 and then linearly. Assuming that P37 < P38, the congestion window increases exponentially until reaching P37 and then linearly. Of course, under either assumption, whenever a loss is encountered, slow-start is abandoned and the congestion window increases linearly when standard TCP is being simulated.

4.2.1.7 Summary of Factors Selected for Sensitivity Analysis. Table 4-10 recaps the eleven factors selected for the sensitivity analysis and the relationship of those factors to MesoNet parameters. Parameters not included in Table 4-10 are assigned fixed values, as indicated in Tables 4-4 through 4-9.

Table 4-10. Recap of Sensitivity Analysis Factors and Mapping to MesoNet Parameters

	Factor	Definition	MesoNet Parameter(s)
Network Factors	x1	Propagation delay	P15
	x2	Network speed	P17
	x3	Buffer sizing	P22
User Factors	x4	Average file size for web pages	P8
	x5	Average think time between web clicks	P9
	x6	Probability a user opts to transfer a larger file	P10
Source & Receiver Factors	x7	Probability a source or receiver is on a fast host	P26
	x8	Scaling factor for number of sources & receivers	P28
	x9	Distribution of sources	P29 & P30
	x10	Distribution of receivers	P31 & P32
Protocol Factors	x11	Initial TCP slow-start threshold	P37

As Table 4-10 demonstrates, the sensitivity analysis is designed to consider the influence of four main classes of factors: (1) network factors, (2) user factors, (3) factors affecting sources and receivers and (4) protocol factors. The factors are fairly balanced with three or four in each category, except for a single protocol factor. In general, the three protocol-related factors might have been fixed, but the inclusion of the initial slow-start threshold as a factor was driven by a specific question, about which the related literature [6, 7, 10] indicates there is no widespread agreement. Now that values have been assigned to 25 fixed parameters, it remains to select the number of levels and settings for the 11 factors identified in Table 4-10. We address that topic next.

4.2.2 Number of Levels and Settings for MesoNet Factors

Adopting the convention of a two-level experiment allows us to produce the kind of OFF designs often used in engineering studies [88, 89, 95] and to benefit from the positive effects such designs have on related analysis techniques. For this reason, we decided to choose two levels for each factor in our sensitivity analysis. Of course, doing so limits our conclusions to the range of settings chosen for our (robustness) factors. Even here we are assuming that the system behaves monotonically in the range between any two settings. If we have reason to believe that behavior is non-monotonic between particular settings, then we should not select such settings for our sensitivity analysis. To extend confidence in the findings produced by our sensitivity analysis, we should explore different specific values for our settings and see whether or not our conclusions also hold. We adopted this supplementary exploration. Here, we focus on our initial sensitivity analysis. We present our supplementary sensitivity analysis in Appendix C.

In choosing specific settings for our two levels (plus and minus) of each factor, we were guided by a desire to complete our 64 experiment runs within a week or so of computing time. We had already determined that computation time in our model was influenced by the number of packets that need to be processed during a simulation. Given that we had decided to fix our simulation to a 20-minute period of network operation, this meant that the computational requirements of our model would be driven largely by the number of sources and the network speed. For this reason, we chose to restrict our simulated network to a few tens of thousands of potential sources and to restrict our network speed to about 10 Gbps in the backbone. Increasing the number of potential sources and the network speed would increase our computational requirements. We decided that increasing the number of sources and the network speed would not be necessary for our sensitivity analysis. Of course, we verified this decision by using more sources and higher backbone speeds in Appendix C.

4.2.2.1 Two-Level Factor Settings. Table 4-11 presents settings chosen for the plus and minus levels for all eleven factors in the sensitivity analysis. Next, we discuss the reasons underlying our choices and the ramifications for the related simulations.

4.2.2.2 Rationale for (and Ramifications of) Network Factor Settings. The topology used in our experiments (recall Fig. 3-1) has defined link propagation delays (recall Table 3-1) that lead to specified minimum round-trip times on designated routes (see Table 3-2). We decided to assign one setting (x1 = 1) to indicate the propagation delays defined in this topology and a second setting (x1 = 2) that doubles those propagation delays. With the

minus setting, paths in the topology average a round-trip propagation delay of 41 time steps and a maximum round-trip propagation delay of 100 time steps. This is consistent with a network spanning the United States. When the plus setting is used, average and maximum propagation delays increase to 81 and 200 time steps, respectively. The increased propagation delays are consistent with a network that spans from the west coast of Asia across the United States and into Europe. Note that the setting for propagation delay also influences buffer sizes because the average round-trip propagation delay makes up the *RTT* component of the buffer-sizing algorithms.

Table 4-11. Two-Level Settings for Each of 11 Factors in Sensitivity Analysis

	Factor	Plus	Minus	Parameter Mapping
Network Factors	x1	2	1	+(P15 = 2) or –(P15 = 1)
	x2[4]	400 p/ms	800 p/ms	+(P17 = 400) or –(P17 = 800)
	x3	RTTxC	*RTT*x*C*/SQRT(*n*)	+(P22 = 1) or +(P22 = 2)
User Factors	x4	100 packets	50 packets	+(P8 = 100) or –(P8 = 50)
	x5	5000 ms	2000 ms	+(P9 = 5000) or –(P9 = 2000)
	x6[5]	0.01	0.02	+(P10 = 0.01) or –(P10 = 0.02)
Source & Receiver Factors	x7[6]	0.2	0.4	+(P26 = 0.2) or –(P26 = 0.4)
	x8	3	2	+(P28 = 3) or –(P28 = 2)
	x9	P2P	WEB	+(P29 = 0.33 and P30 = 0.33) or –(P29 = 0.13 and P30 = 0.53)
	x10	P2P	WEB	+(P31 = 0.33 and P32 = 0.33) or –(P31 = 0.5 and P32 = 0.25)
Protocol Factors	x11	1.07x10[9] packets	43 packets	+(P37 = 1.07x10[9]) or –(P37 = 43)

For network speed, we chose to consider a backbone operating near 10 Gbps. Thus, we chose 800 p/ms (packets per millisecond – 8 x 10^5 packets per second) as the top speed of our backbone routers. (8 x 10^5 packets per second x 12 x 10^3 bits per packet = 9.6 Gbps). Of course, modern backbone routers operate at many times this speed; however, we were interested in keeping our simulation time within reason, while still providing some level of load to the simulated network. We chose to define our slower network speed as half our higher speed; thus, we chose 400 packets per millisecond, which equates to a 4.8 Gbps backbone. Note that the choice of backbone router speed determines the choice of router speeds for the other five router types, as shown in Table 4-12. In addition, router speeds influence buffer size because router speed equates to the capacity (*C*) component of the buffer-sizing algorithm.

For buffer sizes, we chose two algorithms. One algorithm, *RTT*x*C*, instantiates the conventional wisdom [40] regarding how to select buffer sizes to match the expected round-trip time of routes transiting the router and also the capacity of links attached to the router. The second algorithm, *RTT*x*C*/SQRT(*n*), incorporates an alternate proposal suggesting that one can reduce buffer capacity proportional to the square root of the expected number of flows transiting a router. In the paper proposing the second algorithm

[4] Unfortunately, we coded an increased network speed under the minus setting (and a lower network speed under the plus setting). The reader should bear this in mind when interpreting the results in following sections. Changing this coding would necessitate rerunning the experiment, which would be rather costly.

[5] We also coded this setting incorrectly. Fortunately, this factor doesn't have a large influence on model response, so it does not become confusing in the discussion.

[37], it was left as future work to assess the influence of this algorithm in a large network. This open research question motivated us to include the second buffer-sizing algorithm as an alternative to the typical algorithm.

Table 4-12. Relationship among the Speed of Backbone Routers and Other Router Types (all values given in packets per millisecond)

Router Type	Plus	Minus
Backbone	400	800
POP	100	200
Typical Access	10	20
Fast Access	20	40
Directly Connected Access	100	200

4.2.2.3 Rationale for (and Ramifications of) User Factor Settings. Defining user behavior required selecting three parameters: average file size, average think time and likelihood of downloading a larger file. These parameters are meant to characterize Web users who click from Web page to Web page and occasionally download a picture or a paper or a music file. Previous research [33-36] has established that Internet file sizes exhibit a long-tailed distribution that can be approximated with a Pareto distribution with a shape parameter below 2. We adopted this approach. On the other hand, we need to select an average for the distribution (factor x4). We chose 100 packets (100 packets x 1500 bytes per packet = 1.5×10^5 bytes per Web page) as a reasonable size for typical Web pages. We decided to also consider Web pages at half that size: 50 packets (7.5×10^4 bytes).

The think time (x5) between Web clicks could be chosen in two different ways. One way is to imagine how long a user typically dwells on a page, while perusing it. Another way is to choose times to obtain a desired load of active users on the network. We took this second approach. A more heavily loaded network would be represented by sources that clicked on a Web link every 2000 milliseconds (x5 = 2 seconds), while we modeled a more lightly loaded network through sources that clicked on a Web link every 5000 milliseconds (x5 = 5 seconds). Of course, this factor interacts with the number of potential users. Many potential users clicking very often create a heavier load and fewer potential users clicking less often create a lighter load. And combinations would fall in between. Note that a heavily loaded network would require users to take longer to transfer their files and thus would mean that users might not be able to arrive for additional transfers quickly because they are slower with ongoing transfers. This implies that there is some dependency-based feedback inherent in the model. Such feedback is probably congruent with the same type of feedback inherent in real networks. The overall effect of this technique for modeling network traffic is not clear, but one must bound the number of simulated users in some fashion.

The probability for a user to decide to download a larger document (x6) represents the possibility that, after looking at a Web page, the user decides to download a paper or a photo or some other document that is larger than a typical Web page. Since we set that file size multiplier to a fixed value (10), a user will download files with an average size of 1.5 Mbytes (x4 = 100) or 750 Kbytes (x4 = 50). As with normal Web objects, these larger documents will be distributed according to a Pareto distribution, which gives a long tail. Lacking concrete measurements, we chose to imagine that a user might download a larger document once in every 100 clicks, so we could set x6 = 0.01.

We also decided to consider the situation where a user downloads a document twice as often (x6 = 0.02), or twice in every 100 clicks.

4.2.2.4 Rationale for (and Ramifications of) Source & Receiver Factor Settings. Defining parameters for sources and receivers required deciding how fast each source or receiver could operate, determining how many sources and receivers existed in the topology, and also indicating the distribution of sources and receivers. These decisions influenced the number of potential active flows and the probability of flows between various classes of access router. We begin by noting that computers connected to the Internet are in transition from slower speed connections (e.g., 100 Mbps) to higher speed connections (e.g., 1 Gbps), so sources and receivers operate on computers with different network-connection speeds. To reflect this, we decided to experiment with two different mixes of computer speeds: 20 % fast computers (x7 = 0.20) and 40 % fast computers (x7 = 0.40).

We began by fixing the base number of sources (P27) under each access router to 100. Since each router has on average four times as many receivers as sources, the base number of receivers becomes 400. We decided to investigate two scaling factors for the number of sources and receivers; we set the scaling factor to either two (x8 = 2) or three (x8 = 3). A scaling factor of two implies that each access router will have around 200 sources and 800 receivers, while a scaling factor of three implies that each access router will have about 300 sources and 1200 receivers. Thus, the total number of potential sources in the network will vary from around 18.56×10^3 to 41.7×10^3 and the total number of potential receivers will vary from around 111.2×10^3 to 219.6×10^3.

The average number of sources and receivers under each access router (and also total sources and receivers in the network) will be further adjusted through the distribution pattern assigned to sources (x9) and receivers (x10). The combination of distribution patterns will also affect the number of sources and receivers under each access router and throughout the network. (Sec. 3.2.4 explains the specific relationships that determine the resulting distribution of sources and receivers.)

To recap, given a specified base number of sources and receivers, a scaling factor and a distributional pattern for sources and for receivers, MesoNet populates the network topology with a specified number of sources and receivers and distributes those sources and receivers in the required proportion under each class of access router: normal (N-class[6]) routers, fast (F-class) routers and directly connected (D-class) routers. Table 4-13 shows the resulting distribution of sources for each combination of relevant factors (x8, x9 and x10) used in our sensitivity analysis. Table 4-14 shows the resulting distribution of receivers. A given distribution of sources and receivers also leads to a particular apportioning of flows among the three classes of access router, as shown in Table 4-15.

As the tables indicate, the distributional factors (x9 and x10) control the probability that flows go between specific combinations of access router classes: directly connected to directly connected (DD), directly connected to fast (DF), directly connected to normal (DN), fast to fast (FF), fast to normal (FN) and normal to normal (NN). The scale factor (x8) coupled with the fixed base sources parameter (P27) determines the number of potential active flows (which is also the number of sources).

[6] We continue our convention of color coding designators for access-router classes to match the colors used in Fig. 3-1.

Table 4-13. Relation between Factors and Number and Distribution of Sources

x8	x9	x10	Total Sources	% under D Routers	% under F Routers	% under N Routers
2	P2P	P2P	27.8×10^3	4.32	20.14	75.54
3	P2P	P2P	41.7×10^3	4.32	20.14	75.54
2	WEB	WEB	18.56×10^3	6.46	48.27	45.25
3	WEB	WEB	27.84×10^3	6.46	48.27	45.25
2	P2P	WEB	27.8×10^3	4.32	20.14	75.54
3	P2P	WEB	41.7×10^3	4.32	20.14	75.54
2	WEB	P2P	18.56×10^3	6.46	48.27	45.25
3	WEB	P2P	27.84×10^3	6.46	48.27	45.25

Table 4-14. Relation between Factors and Number and Distribution of Receivers

x8	x9	x10	Total Receivers	% under D Routers	% under F Routers	% under N Routers
2	P2P	P2P	111.2×10^3	4.32	20.14	75.54
3	P2P	P2P	166.8×10^3	4.32	20.14	75.54
2	WEB	WEB	146.4×10^3	2.45	11.47	86.06
3	WEB	WEB	219.6×10^3	2.45	11.47	86.06
2	P2P	WEB	146.4×10^3	2.45	11.47	86.06
3	P2P	WEB	219.6×10^3	2.45	11.47	86.06
2	WEB	P2P	111.2×10^3	4.32	20.14	75.54
3	WEB	P2P	166.8×10^3	4.32	20.14	75.54

Table 4-15. Relation between Factors and Distribution of Flow Classes

x8	x9	x10	% DD Flows	% DF Flows	% DN Flows	% FF Flows	% FN Flows	% NN Flows
2	P2P	P2P	0.186	1.74	6.52	4.05	30.43	57.06
3	P2P	P2P	0.186	1.74	6.52	4.05	30.43	57.06
2	WEB	WEB	0.159	1.92	6.67	5.53	46.74	38.95
3	WEB	WEB	0.159	1.92	6.67	5.53	46.74	38.95
2	P2P	WEB	0.106	0.99	5.57	2.31	26.00	65.01
3	P2P	WEB	0.106	0.99	5.57	2.31	26.00	65.01
2	WEB	P2P	0.279	3.38	6.83	9.72	45.58	34.18
3	WEB	P2P	0.279	3.38	6.83	9.72	45.58	34.18

One final note: the number and distribution of sources and receivers also influences the determination of router buffer sizes when using the *RTT*x*C*/SQRT(*n*) algorithm. The *RTT*x*C* algorithm computes buffer sizes based on multiplying the average round-trip propagation delay in the network by the capacity of each router. Table 4-16 shows the results for this algorithm when using the factor values adopted in this sensitivity analysis. When switching to the *RTT*x*C*/SQRT(*n*) algorithm, the values in

Table 4-16 are divided by the estimated average number of active flows expected to transit each router. This estimate depends on the number and distribution of sources and receivers throughout the topology. In general, using the *RTT*x*C*/SQRT(*n*) algorithm reduces buffers within routers by one or two orders of magnitude.

Table 4-16. Buffers for Combinations of Round-Trip Propagation Delay (x1) and Capacity (x2)

x1	x2	Backbone Router Buffers (avg.)	POP Router Buffers (avg.)	Access Router Buffers (avg.)
1	400	16.277×10^3	4.070×10^3	647
2	400	32.553×10^3	8.139×10^3	1.294×10^3
1	800	32.553×10^3	8.139×10^3	1.294×10^3
2	800	65.106×10^3	16.277×10^3	2.588×10^3

4.2.2.5 Rationale for (and Ramifications of) Protocol Factor Settings. After investigating the literature, we came to the realization that there is no consensus value to use for the initial TCP slow-start threshold. Some authors [4] suggest using the receive window provided by a corresponding TCP entity. Some authors [10] suggest picking a small value. Some authors [6] suggest picking a very large number. A colleague, Mark Carson (personal communication, November 12, 2008) indicated that some operating systems select this value based upon characteristics of the local network card. Given this general lack of consensus, we decided to include the initial TCP slow-start threshold as a factor (x11) in our sensitivity analysis. We decided there were two main schools of thought about choosing a value: choose a small value and choose a large value. To represent the small-value school of thought, we chose (x11 =) 43 packets, which was recommended by Stevens [10]. To represent the large-value school of thought, we chose an arbitrarily large value of (x11 =) 1.07×10^9 packets, as suggested by Fall [6]. We also adopted the recommendation of Floyd [7], where a flow increases its sending rate exponentially up to a congestion window of 100 and then logarithmically until a higher threshold is reached or loss encountered. The rationale for choosing a large value derives from the purpose of initial slow-start: to quickly determine how fast a source may send on a given path. Choosing a small value could lead a flow to switch to a linear increase prior to achieving its maximum transmission rate, so a flow might end before maximum rate is achieved. We decided to see what difference the choice of initial TCP slow-start threshold would make given our other factors and parameter settings.

4.2.3 Specific Combinations Simulated

Given 11 factors, each with two possible levels, a full factorial experiment would require (2^{11} =) 2048 simulation runs. Assuming an average run takes about 8.5 processor hours, conducting all these simulation runs would require 17.408×10^3 processor hours. If we split these among 24 processors, we could complete the work in about 725 hours – or 30 days. We preferred to be able to complete our simulations within a week, so we adopted a 2^{11-5} orthogonal fractional factorial (OFF) design that required only 64 simulation runs. The design can be found in Fig. 4-1. To generate our parameterized runs, we set our fixed factors to the values indicated in Tables 4-8 through 4-9 and then we generated 64 configuration files that varied the factors (x1 to x11) as instructed by Fig. 4-1 – taking

from Table 4-11 the PLUS values to substitute for the +1 designators in Fig. 4-1and the MINUS values to substitute for the -1 designators in Fig. 4-1.

4.3 Experiment Execution

The experiment plan required 64 simulation runs, each simulating a different combination of factor settings (recall Fig. 4-1). We had 28 physical processors[7] on which we could run our experiments, so we could conduct simulations in parallel. However, we were sharing these processors with other projects, so we could not always use all of the available processors. Below, we give a brief discussion of the resource requirements for the simulations and then we recount our approach to data collection and summarization.

4.3.1 Resource Requirements for Simulations

Table 4-17 reports the characteristics of the 28 processors available for our sensitivity analysis. Since MesoNet is implemented in SLX, each of the processors had access to an SLX simulation environment. SLX comes in two varieties: one configured to run in a 32-bit address space and one configured to run in a 64-bit address space. Some of the available processors were configured with a 64-bit operating system, which could support both the 32-bit and 64-bit versions of SLX. We chose to run all our simulations using the 32-bit version of SLX. We made this choice because our simulations could easily fit within a 32-bit address space and 32-bit simulation runs faster than 64-bit simulation. This is true largely because 64-bit simulation requires the use of 64-bit arithmetic when manipulating pointers that address simulation objects. Also 64-bit simulation requires more memory than 32-bit simulation because of the doubling of size for address pointers. For these reasons, 64-bit simulation should be reserved for situations where the size of the simulation cannot be contained within a 32-bit address space.

Table 4-17. Characteristics of Processors Executing Simulation Runs

Node	Physical Processors	Speed (GHz)	Hyperthreaded	Memory (GB)	Operating System
ws7	4	3.66	Yes	20	Windows Server 2003 R2 x64 Edition SP2
ws8	4	3.66	Yes	20	Windows Server 2003 R2 x64 Edition SP2
ws9	8	2.6	No	32	Windows Server 2003 R2 x64 Edition SP2
ws10	8	2.6	No	32	Windows Server 2003 R2 x64 Edition SP2
DT	4	3.2	No	3	Windows XP SP2

We executed the simulations in three rounds (runs 1-35, runs 36-50 and runs 51-64) over about one week. All simulation runs required a similar amount of memory: on the order of 120 Mbytes. On the other hand, simulation runs required varying amounts of

[7] Hyperthreading™ was enabled on 8 of these physical processors. Hyperthreading creates two independent logical threads on a single physical processor. With hyperthreading the number of available logical processors totaled (28 + 4 x 2 =) 36. On hyperthreaded processors, our simulations ran at (or below) half the speed that was possible without using hyperthreading. The reader should take this into account when interpreting the execution time requirements given in Table 4-18.

processor time, depending on the specific combination of factors and on the specific node used to execute the simulation. Table 4-18 recounts the execution time used for each simulation run. Executing all 64 runs required a total of 537.6 hours of processing time, which amounts to 8.4 hours on average per run. However, due to the fact that ws7 and ws8 used hyperthreading, this figure is somewhat misleading.

Table 4-18. Execution Time (Hours) Required for Each Simulation Run

Run	Node	Time	Run	Node	Time	Run	Node	Time	Run	Node	Time
1	ws9	7.7	17	ws7	13.8	33	DT	10.7	49	DT	6.8
2	ws9	6.2	18	ws7	12.2	34	DT	6.5	50	DT	2.4
3	ws9	3.8	19	ws7	11.5	35	DT	5.2	51	ws9	2
4	ws9	4.3	20	ws7	12.9	36	ws9	4.5	52	ws9	3.6
5	ws9	4.9	21	ws7	8.8	37	ws9	7.3	53	ws9	2.8
6	ws9	9.2	22	ws7	15.6	38	ws9	6.9	54	ws9	3.1
7	ws9	5.1	23	ws7	15.4	39	ws10	5.7	55	ws9	3
8	ws9	4.1	24	ws7	8.4	40	ws9	4.9	56	ws9	3.2
9	ws10	7.5	25	ws8	16.7	41	ws9	8.2	57	ws9	5.7
10	ws10	8.8	26	ws8	24.6	42	ws10	8.3	58	ws10	5.6
11	ws10	6.1	27	ws8	19.1	43	ws10	4.9	59	ws10	5
12	ws10	4.3	28	ws8	16.4	44	ws9	4.8	60	ws10	4.1
13	ws10	10.2	29	ws8	24.7	45	ws10	9.2	61	ws10	7.5
14	ws10	8.5	30	ws8	22.5	46	ws9	8.2	62	ws10	4
15	ws10	5.6	31	ws8	19	47	ws10	5.1	63	ws10	3.8
16	ws10	5.1	32	ws8	19.9	48	DT	6.8	64	ws10	4.9

Considering the processing time required for runs on individual nodes, runs on ws9 averaged 5.2 hours, runs on ws10 averaged 6.2 hours, runs on DT averaged 6.4 hours, runs on ws7 averaged 12.3 hours and runs on ws8 averaged 20.4 hours. Grouping nodes into those that were not hyperthreaded (ws9, ws10 and DT) and those that were hyperthreaded (ws7 and ws8), we found that the hyperthreaded nodes required an average of 16.3 hours per run, while the non-hyperthreaded nodes required an average of 5.8 hours per run. Thus, the hyperthreaded nodes took an average of 2.8 times longer than the non-hyperthreaded nodes to execute a simulation run. This suggests that the hyperthreaded processors ran at about 36 % the speed of the non-hyperthreaded processors. Of course, to gauge the effects due to hyperthreading alone, one must account for the fact that the processor speeds of the hyperthreaded nodes were different than the processor speeds of the non-hyperthreaded nodes.

Given that the processor speed of ws7 (and ws8) is 3.66 GHz, one would expect hyperthreading to provide half the processing speed, or (3.66/2 =) 1.83 GHz, to each logical thread. Thus, one might expect that it would take (2.6/1.83 =) 1.42 times longer to run simulations on ws7 (and ws8) than on ws9 (and ws10). We found that on average it took 2.8 times longer to run simulations on the hyperthreaded nodes. These findings do not provide a complete characterization of differences between hyperthreaded and non-hyperthreaded operations. First, we did not run the same workload on both types of processors, as we split various experiment configurations among the processors. Second, the hyperthreaded processors employed chip architectures (Intel Xenon MP) different from some of the non-hyperthreaded processors (ws9 and ws10 used AMD Opteron 8218 and DT used Intel Xenon).

4.3.2 Data Collection and Summarization

MesoNet records response data as time series. This allows monitoring response changes over time. For example, Fig. 4-11 shows the time series for the number of active flows (response y1) during run 64 of our sensitivity analysis. As shown, the time series reports the number of active flows (y axis) at the end of each of measurement interval for each of 6000 measurement intervals (x axis) recorded during the simulation run. To facilitate our analyses, we summarize each response to an average value for each run. As illustrated in Fig. 4-11, we do this by discarding the first half of the data (measurement intervals 1 to 3000) and then computing the average value for the remaining data (measurement intervals 3001-6000). As illustrated in Fig. 4-11, discarding the first 3000 measurement intervals eliminates transient startup effects and enables us to retain behavior representative of the model operating in steady-state. In this case, for run 64, the mean value of y1 over measurement intervals 3001-6000 is 214.676 x 10^2 flows.

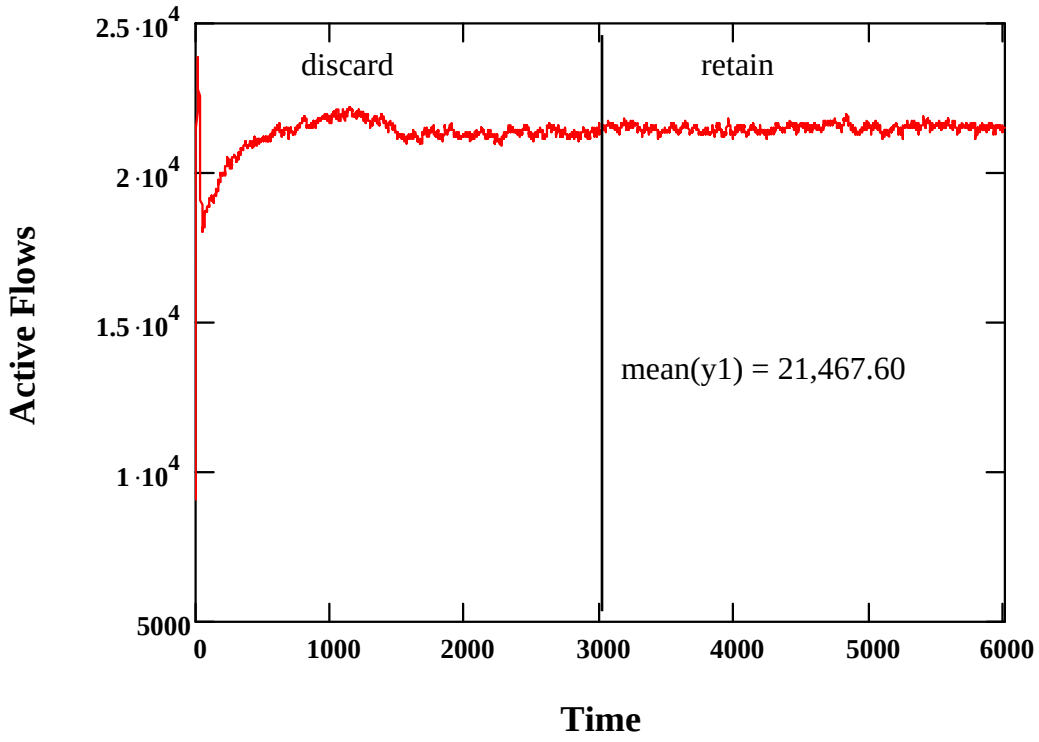


Figure 4-11. Example Illustrating the Technique used to Summarize System Responses (x axis gives the number of active flows and y axis gives time in 200 ms intervals)

We conduct such a summarization for all 22 responses under each of the 64 conditions and collect the summarizations into a table, as shown in Fig. 4-12. For example, the value placed in the cell for response y1 and run 64 in Fig. 4-12 is the value we computed in Fig. 4-11. The summarization table forms the basis for all of our analyses.

Run	y1	y2	…	y21	y22
1	4680.619	0.168126	…	92.034	89.785
2	6654.512	0.239371	…	72.596	57.738
3	9431.405	0.339259	…	29.569	13.963
4	11565.81	0.415439	…	23.427	19.882
…	…	…	…	…	…
61	10319.55	0.247471	…	87.969	41.573
62	1738.469	0.093668	…	159.298	161.602
63	1783.509	0.096094	…	148.395	161.36
64	21467.6	0.514811	…	26.159	9.981

Figure 4-12. Sample Data Summarization: 22 Responses for each of 64 Simulation Runs

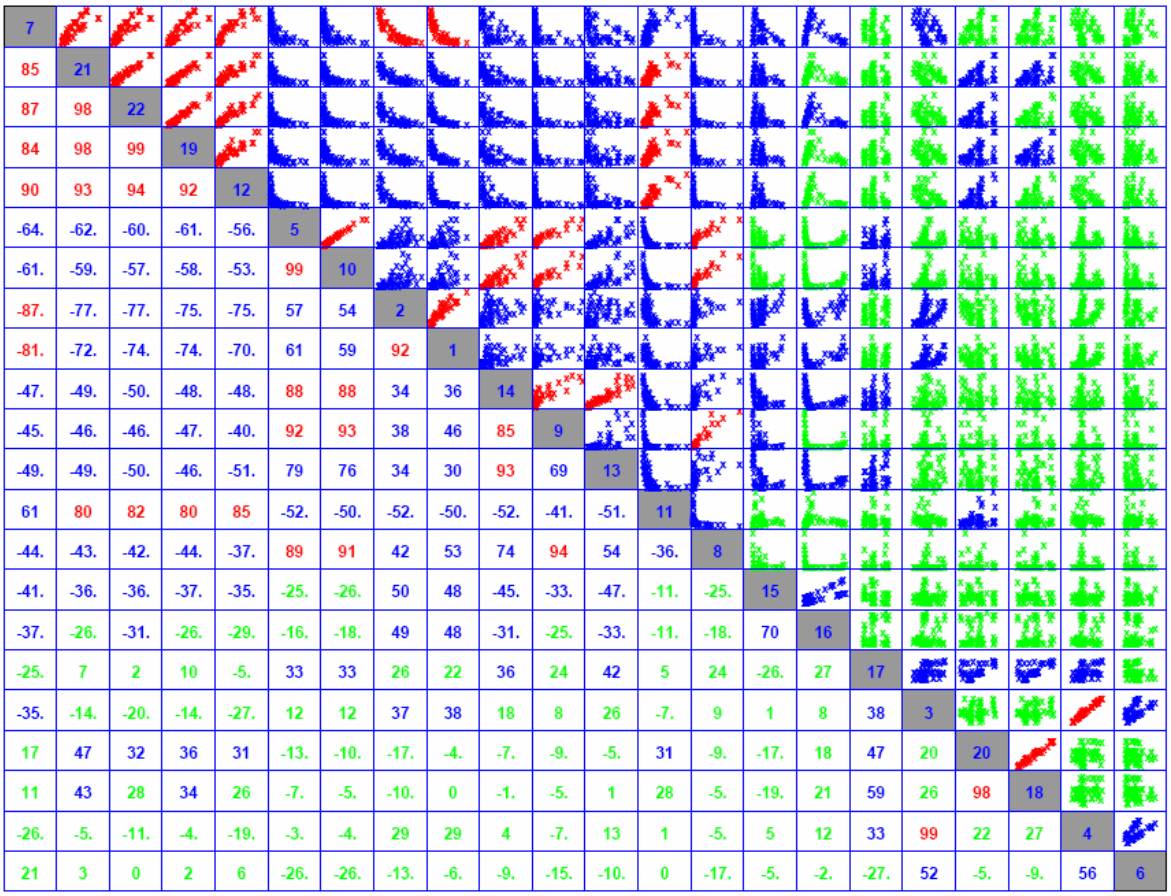


Figure 4-13. Combined Matrix of Scatter Plots and Correlation Values for 22 Responses

4.4 Correlation Analysis and Clustering

Given 64 average values (one per run) for 22 responses, correlation analysis investigates the degree to which pairs of responses are correlated. Recall that Tables 4-1 and 4-2 identify the 22 responses. We begin by generating a scatter plot and computing the correlation for each pair of responses. Then we plot (in Fig. 4-13) the results as a combined matrix of scatter plots and correlation values. We order the diagonal by decreasing average correlation for each response with the 21 other responses. The highest average correlation is for response y7 and the lowest is for response y6. Correlations of .8 and above are colored red, correlations between .3 and .79 are colored blue and correlations below .3 are colored green.

Fig. 4-13 reveals some correlation groupings. For example, responses y7, y21, y22, y19, y12, y11, y1 and y2 show mutual correlations. Responses y5, y10, y14, y8 and y9 also exhibit mutual correlations. Strong correlations appear between selected pairs of responses: y21 and y22; y22 and y19; y5 and y10; y1 and y2; y8 and y9; y13 and y14; y18 and y20; y3 and y4. These mutual correlations suggest that it should prove feasible to reduce the number of responses examined from 22 to some lower dimension. On the other hand, a few responses (e.g., y6 and y17) appear largely uncorrelated with other responses.

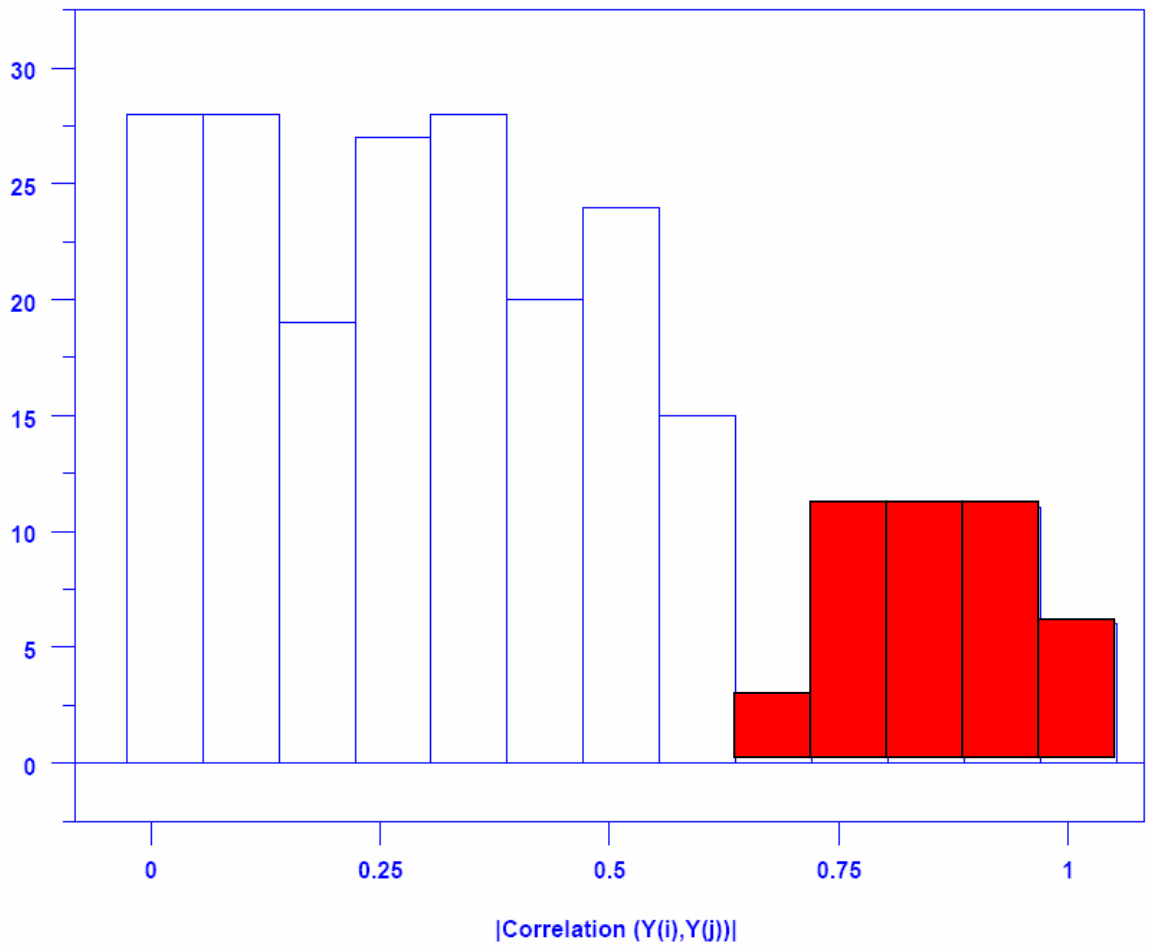


Figure 4-14. Frequency Distribution of the Absolute Value of Correlations for All Pairs of Responses

To identify particular correlation groups, we need to select a threshold for the absolute value of correlations such that above that threshold we will consider correlations sufficiently strong to warrant inclusion in further analyses, while we will discard correlations below that threshold. To help identify a reasonable threshold, we plot (in Fig. 4-14) a frequency distribution of the absolute values of all correlation pairs.

In Fig. 4-14, we emphasize (in red) the range of correlations that appear most significant because there is a notable change above that value, appearing as a separate sub-distribution centered on a different mode. The range of correlations emphasized in Fig. 4-14 run from about 0.65 to 1.0, so we decided to use correlations whose absolute value exceeds 0.65. We discard correlations with lower absolute values. For the correlations retained, we produced an index-index plot (recall Fig. 4-6). In Fig. 4-15 we reorder the indices from Fig. 4-6 (on both the ordinate and abscissa) by increasing total number of variables exhibiting above threshold correlation with the designated variable. Where the count of mutual correlations is the same, our order is arbitrary. We begin with responses y6 and y17, which have no retained correlations. For those responses, we order y17 first because it has only one mutual correlation > 0.5, while y6 has two such correlations – thus, y17 is somewhat less correlated with other responses than is y6.

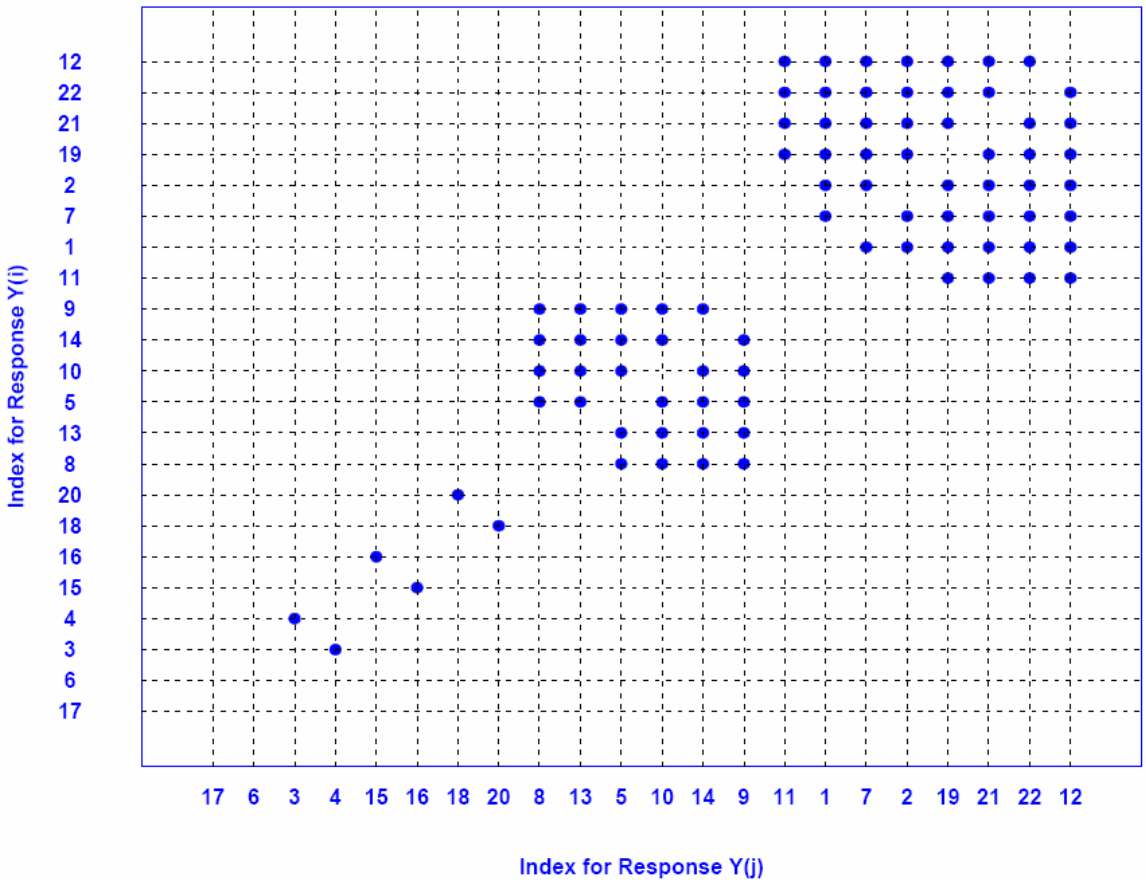


Figure 4-15. Index-Index Plot for Correlation Pairs where |Correlation (Y_i, Y_j)| > 0.65

Fig. 4-15 identifies seven clear correlation groups: y17 (no correlations); y6 (no correlations); y3 and y4 (pair-wise correlation); y15 and y16 (pair-wise correlation); y18

and y20 (pair-wise correlation); y8, y13, y5, y10, y14 and y9 (28 mutual correlations); y11, y1, y7, y2, y19, y21, y22 and y12 (50 mutual correlations). This suggests that we can characterize system response through seven, rather than 22, responses. Next, we address the issue of whether or not the seven correlation groupings make sense from the perspective of the network simulation model. We also discuss what information may be conveyed by lack of correlation. We begin our discussion with the three mutually correlated pairs and then consider the group with 28 mutual correlations, followed by the group with 50 mutual correlations. We close by considering the two uncorrelated responses.

The rate of data packets injected into the network (y3) is highly correlated (0.99) with the rate of packets leaving (y4). This strong correlation is expected because packets must enter the network before they can exit and the rate of entry and exit should be balanced (unless many packets are lost within the network). Perhaps more surprising is the fact that the rate of packets entering and exiting the network is not strongly correlated with any other responses. The closest correlation (around 0.5) is with the rate of flow completions (y6), which is largely uncorrelated with any other responses. One might expect correlation between the number of active flows (y1) and the number of packets entering and leaving the network, but this is not the case. From this, we conclude that the rate of packets flowing through the network is influenced by factors different from those influencing the number of active flows. Thus, our sensitivity analysis needs to consider either the rate of packets entering or leaving the network but not both.

The (SRTT) smoothed round-trip time (y15) and relative queuing delay (y16) are somewhat correlated (0.7). This makes sense because the relative queuing delay is computed by transforming the SRTT. The correlation is not particularly strong because the relative queuing delay factors out the propagation delay and gives enhanced weight to time spent in buffers. Buffer size has a greater influence on y16, while that influence is somewhat diluted (by propagation delay) in y15. The low strength of the correlation suggests that our sensitivity analysis should consider both y15 and y16. On the other hand, the reasons underlying the correlation suggest that perhaps we could use only y15, which captures influences due to both propagation delay and queuing delay.

The average instantaneous throughput for DF flows (y18) is strongly correlated (0.98) with the throughput for FF flows (y20). This reflects the fact that throughput is constrained by the capabilities of the slower of the two access-router classes over which such flows transit. This strong correlation implies that we need only consider one of these two responses for our sensitivity analysis.

The next correlation group in Fig. 4-15 consists of 28 mutual correlations among six responses: loss rate (y5); connection failures (y8) and connection-failure rate (y9); retransmission rate (y10); negative acknowledgment rate (y13) and timeout rate (y14). Most of these mutual correlations exceed 0.8. The correlations among these responses appear reasonable because packet losses have numerous consequences: negative acknowledgments or timeouts, connection failures and retransmissions. The strongest correlation (0.99) exists between loss rate and retransmission rate. In fact, since both data packets and acknowledgments may be lost, one would expect the retransmission rate to be about twice the loss rate. The (0.89) correlation between loss rate and connection failures is lower because connection attempts are retried; three connection attempts must be lost before a connection fails. The (0.79) correlation between loss rate and negative

acknowledgment rate (as seen by sources) is lower because negative acknowledgments may also be lost; losses push up the rate at which receivers send negative acknowledgments but also increase probability that negative acknowledgments are lost. When acknowledgments (negative or positive) are lost, the rate of timeouts increases, so there is a higher correlation (0.88) between loss rate and timeouts. The six responses in this correlation group are measures of packet losses and the ensuing consequences for the network. Our sensitivity analysis need only consider one of these responses, such as retransmission rate (y10), which reflects both packet losses and the packets resent to recover from losses.

The final correlation group consists of 50 mutual correlations among eight responses: active flows (y1) and proportion of possible flows that are active (y2); flow-completion rate (y7); average congestion window (y11) and window-increase rate (y12); and average instantaneous throughput for DN (y19), FN (y21) and NN (y22) flows. Most correlations, which are negative, stem from sharing network resources. Increasing active flows leads to decreases in flow-completion rate, the average congestion window, window increase rate and instantaneous throughput for flows transiting normal access routers. Flows (DN, FN and NN) transiting normal access routers are most numerous; sharing access routers affects the throughput of these flows. As the number of flows transiting an access router increases, each flow receives a lower share of the bandwidth and so will receive lower throughput. Lower throughput implies smaller congestion windows. Smaller congestion windows imply a slower rate of window increases. More active connections also imply a lower rate of connection completion. Note, however, that the (-0.5) correlation between active connections and average congestion window is not strong enough to be included in this correlation group. Stronger correlations exist between average congestion window and flow throughputs (about 0.8) and window increase rate (0.85). This suggests that congestion window size is influenced by factors not solely related to the number of active connections. In fact, in Sec. 4.1.5 we showed that congestion window size is influenced by network speed, buffer-sizing algorithm and initial slow-start threshold, as well as by factors that influence the number of active connections. For our sensitivity analysis we can select one response (such as y22) to reflect the degree of sharing among common network resources. We should probably also include the number of active flows in order to investigate what factors influence the need to share resources.

The two remaining responses, flows completed (y6) and average instantaneous throughput for DD flows (y17), are uncorrelated with other responses. Apparently, the number of flows completed is driven by factors different from the factors driving other responses. The reason for this is not obvious. Throughput for DD flows is also driven by factors different from the factors influencing throughput for other flow classes. The reason for this appears straightforward. First, DD flows are relatively few in number, when compared with other flow classes. Second, DD flows cross high-speed access routers that are connected directly to backbone routers, so DD flows see less contention for bandwidth on the ingress and egress paths of the network. Since y6 and y17 are uncorrelated with other responses, we must include them in our sensitivity analysis.

To recap, Table 4-19 identifies the responses we chose to investigate during our sensitivity analysis. Correlation analysis suggested that we could characterize system response using only seven of 22 responses. We decided to include an eighth response: the

number of active flows (y1). This added response allowed us to consider which factors lead to an increased number of active flows, a main influence on the degree of resource sharing required within a network. Two responses deal with the aggregate throughput of packets (y4) and flows (y6). One response (y10) reflects the degree and consequences of packet losses. One response (y15) mirrors the degree of network delay. The remaining responses gauge throughput for flows constrained by transiting directly connected (y17), fast (y20) or normal (y22) access routers.

Table 4-19. Responses Selected for Investigation in Sensitivity Analysis

Response	Definition
y1	Average number of active flows
y4	Average number of packet output per measurement interval
y6	Average number of flows completed per measurement interval
y10	Average retransmission rate
y15	Average smoothed round-trip time
y17	Average instantaneous throughput for DD flows
y20	Average instantaneous throughput for FF flows
y22	Average instantaneous throughput for NN flows

4.5 Principal Components Analysis

Principal components analysis (PCA) is an alternative (or complementary) technique often used to assess the covariance structure of a set responses [96]. In this section, we describe the findings of a PCA applied to the 22 responses from our sensitivity analysis simulation runs. As the first step in the PCA, we transform our data responses into a standardized form by subtracting the mean value (over all 64 conditions) from each response to yield (22 x 64 =) 1408 normalized data points, as discussed previously in Sec. 4.1.4. In this way, all responses are placed on an equivalent scale with respect to variance around the mean. Next, we find a weight vector that yields the maximum possible variance (or standard deviation), subject to the constraint that the sum of all weights (with each weight squared) is equal to one. We repeat this process, possibly up to the total number of responses, and each time require the weights selected to be orthogonal to the weights used in previous steps. Using this technique we are looking for the largest sources of variation in different directions through the data with each step. Each different direction through the data is considered a principal component. The amount of variation accounted for diminishes with each principal component considered. At some point, most of the variance will be accounted for and one could stop the analysis.

For example, consider Figure 4-16, which displays the results of a PCA for the 22 responses from our sensitivity simulations. The detailed layout of each sub-plot was explained previously in Sec. 4.1.4 (recall Fig. 4-8). The upper left-hand plot depicts the standard deviation (SD) across all normalized responses (y1 through y22). The remaining 22 plots show the standard deviation accounted for by each of 22 principal components in decreasing order of magnitude. We note that most (about 86 %) of the variation in the data is accounted for by the first four principal components. Next, we plot the weights associated with each response in each of the first four principal components. Figure 4-17 shows this information. The detailed layout of each sub-plot was explained previously in Sec. 4.1.4 (recall Fig. 4-7).

The results of the PCA suggest that the behavior of our model can be represented with as few as four (statistically uncorrelated) responses, instead of the seven responses suggested by our correlation analysis. Further, these four principal components, or PCs, are linear combinations of many regular responses. Extracting responses from each PC in Fig. 4-17 using heuristics mentioned in Sec. 4.1.4, we can group responses by principal component, as shown in Tables 4-20 through 4-23.

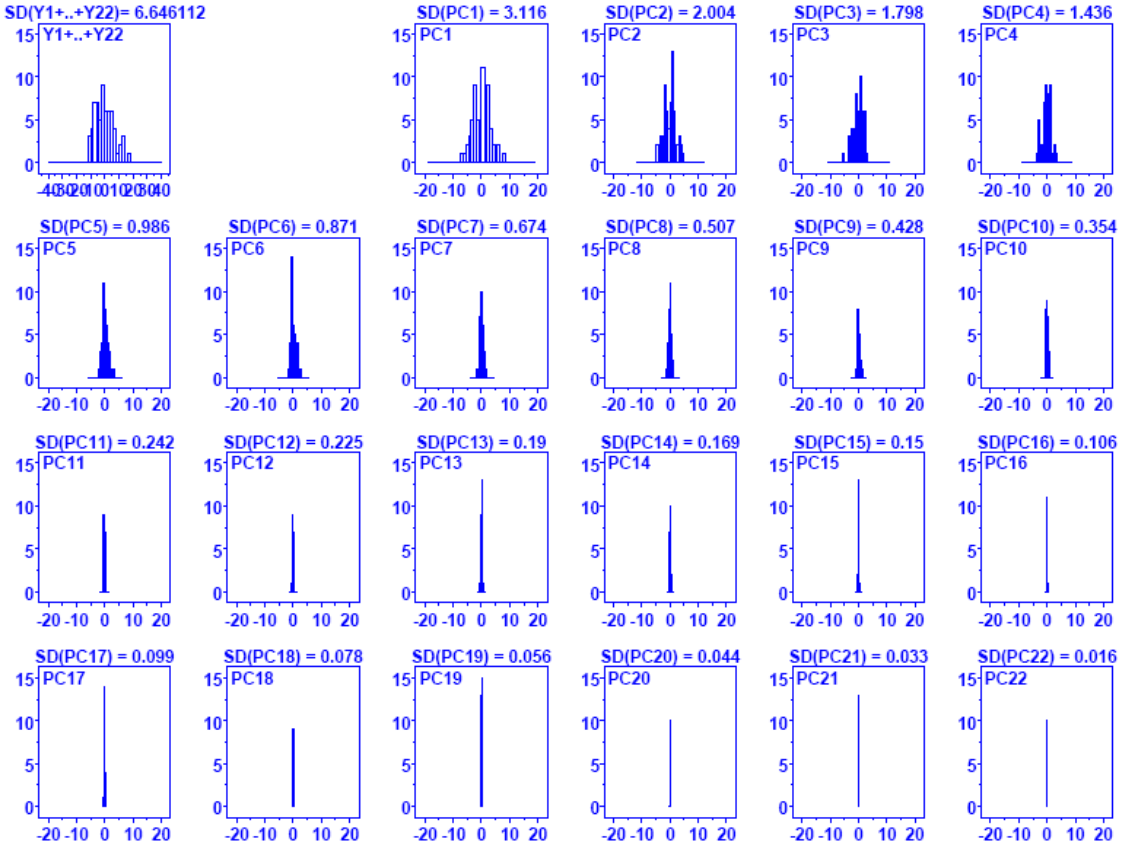


Figure 4-16. Histograms for 22 Principal Components (x axis of each sub-plot identifies bins of normalized component values ranging from -20 to +20 and y axis the count of values within each bin). Above each sub-plot is the standard deviation in the data accounted for by the Principal Component. The first sub-plot gives the distribution of the normalized responses.

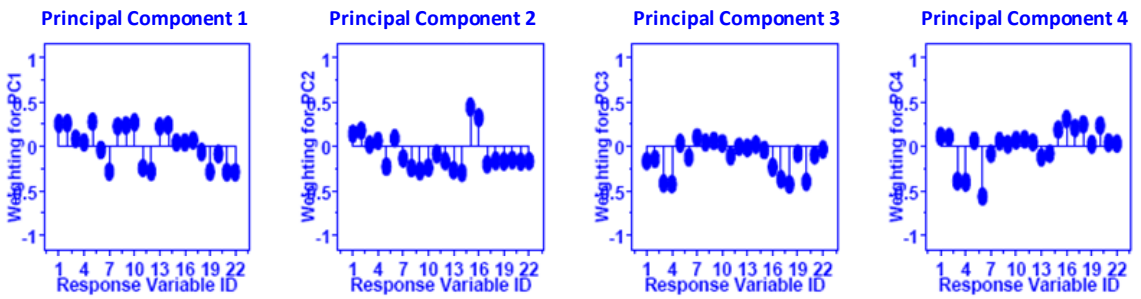


Figure 4-17. Weight Vectors for the First Four Principal Components

The first principal component (Table 4-20) combines two response groups identified in the correlation analysis. One group represents the effects of resource sharing and congestion on the throughput of flows that transit typical access routers. Such flows are most numerous in any given simulation. The second group represents the level of congestion present in the network. Congestion occurs most often at access routers. Thus, the PCA finds that the largest source of variance in the 22 responses arises from the level of congestion at access routers in the simulated network.

Table 4-20 Responses Composing Principal Component One

Correlation Cluster	Response	Definition
Effects of Resource Sharing and Congestion on Throughput in Flows Transiting Typical Access Routers	y1	Average number of active flows
	y2	Proportion of possible flows that are active
	y7	Flow-completion rate
	y11	Average congestion window
	y12	Window-increase rate
	y19	Average instantaneous throughput for DN flows
	y21	Average instantaneous throughput for FN flows
	y22	Average instantaneous throughput for NN flows
Overall Network Congestion	y5	Loss rate
	y8	Connection failures
	y9	Connection-failure rate
	y10	Retransmission rate
	y13	Negative-acknowledgment rate
	y14	Timeout rate

Table 4-21. Responses Composing Principal Component Two

Response	Definition
y15	Smoothed round-trip time
y16	Relative queuing delay

The second principal component (Table 4-21) corresponds to a pair of responses (y15 and y16) grouped together by the correlation analysis. These responses represent the level of delay within the network.

Table 4-22. Responses Composing Principal Component Three

Correlation Cluster	Response	Definition
Packet Throughput	y3	Packets input
	y4	Packets output
DD-flow Throughput	y17	Average instantaneous throughput for DD flows
DF- & FF-flow Throughput	y18	Average instantaneous throughput for DF flows
	y20	Average instantaneous throughput for FF flows

The third principal component (Table 4-22) unites three separate groupings found in the correlation analysis. One group represents the number of data packets flowing in and out of the network, which correlation analysis suggested were not strongly correlated with other responses. Note, though, that there were moderate correlations with throughput

on faster flows (y17, y18 and y20) and with the number of flows completing (y6). In fact, the PCA assigns similar weights for y3 and y4 in both principal components three and four. (For this reason, we also include y3 and y4 in the grouping associated with principal component four.) Responses relating to throughputs for flows transiting only fast and directly-connected routers were grouped together by the principal components analysis, while correlation analysis separated these responses. Principal component three also seems to include the effects of the higher throughput flows on packets flowing into and out of the network.

The fourth principal component represents the ability of the network to complete flows. Included in this component is the association with packets entering and leaving the network. If called upon to place y3 and y4 into only a single principal component, we would choose to place them into PC4. On the other hand, as shown in Table 4-23, PC4 unites two separate groupings found in the correlation analysis.

Table 4-23. Responses Composing Principal Component Four

Correlation Cluster	Response	Definition
Packet Throughput	y3	Packets input
	y4	Packets output
Flow Throughput	y6	Flows completed per measurement interval

The principal components analysis both confirms the findings of the correlation analysis and also provides additional information. For example, the PCA groups together the symptoms and effects of congestion. This appears sensible. The PCA also reveals a connection between packets in and out and two other groupings: throughput on high-throughput flows and the number of connections completed. The correlation analysis hinted at these connections. The PCA suggests that throughput on DD flows should be grouped together with throughput on DF and FF flows; the correlation analysis indicated that DD flows should be studied separately. We will use findings from both the correlation and principal components analyses as we investigate the sensitivity of model responses to input parameters.

4.6 Sensitivity Analysis

In this section, we use the experiment design, the model responses and the results of the correlation and principal components analyses to assess the sensitivity of MesoNet to changes in eleven input factors. We begin by exploring how model inputs affect the eight responses identified by our correlation analysis (recall Table 4-19). Subsequently, we consider how the four main principal components (recall Tables 4-20 through 4-23) vary with changes in input factors.

4.6.1 Sensitivity Analysis Guided by Correlation Analysis

We begin by exploring how model inputs affect three, congestion-related responses: number of active flows (y1), retransmission rate (y10) and average instantaneous throughput for NN flows (y22). Subsequently, we consider the five remaining responses in the following order: average smoothed round-trip time (y15), rate of data packets output (y4), number of flows completed per measurement interval (y6) and average instantaneous throughput for DD flows (y17) and for FF flows (y20).

4.6.1.1 Congestion-Related Responses. For the topology and experiment design we adopted, flows transiting through the slowest (N-class) access routers were most numerous. For this reason, congestion tends to occur most often in N-class access routers, which affects the throughput of flows transiting such routers. The affected flows include DN, FN and NN flows, which our analysis showed to be significantly correlated. We selected NN flows as a representative flow class to consider. The throughput experienced by NN flows is likely to be affected by the number of active flows transiting N-class access routers and by the retransmission rate on those flows. Since flows transiting N-class access routers are most numerous, macroscopic measures of the number of active flows and the retransmission rate network-wide should be indicative of the level of congestion experienced by NN flows. For these reasons, we decided to consider y1, y10 and y22 as a related set of responses.

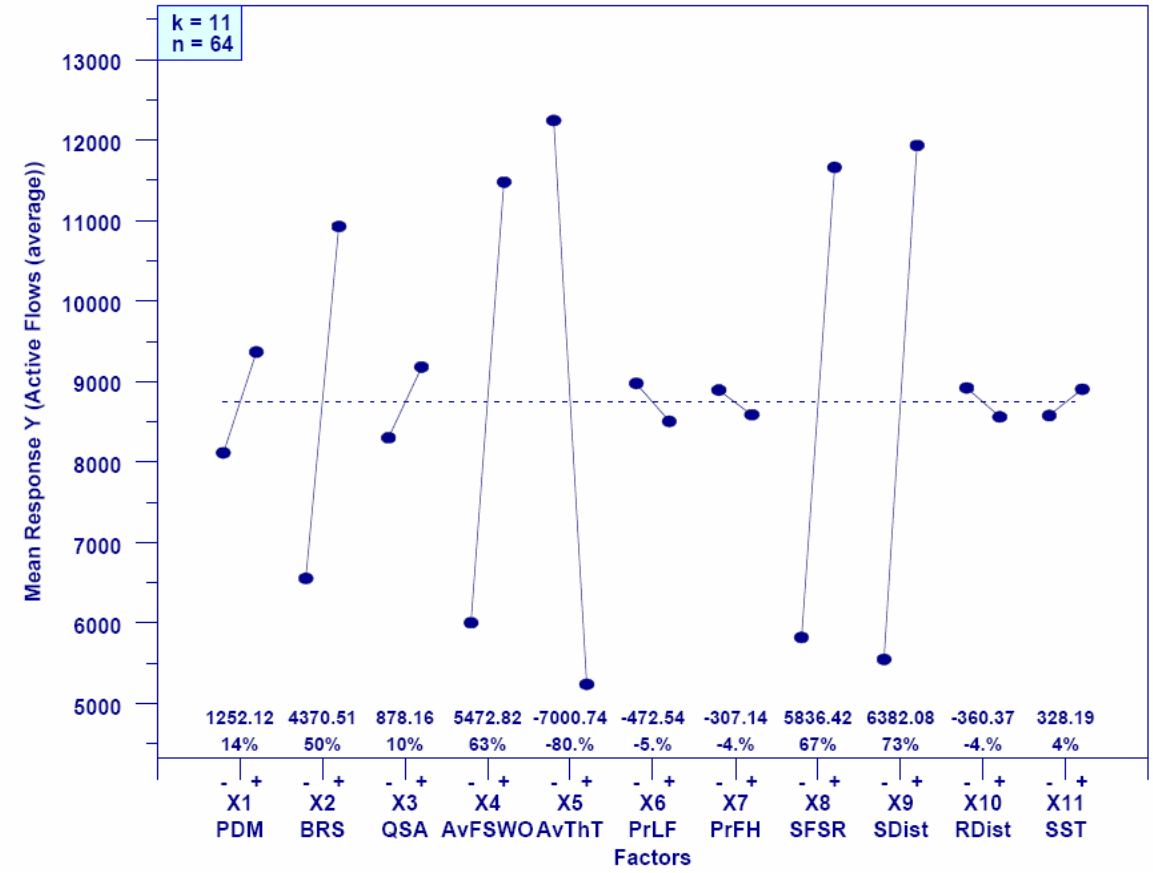


Figure 4-18. Main-Effects Plot for Response y1 (Average Number of Active Flows)

We decided to first examine factors that influence the number of active flows in the network, since the number of active flows is likely to affect congestion. Fig. 4-18 gives the main-effects plot highlighting factors influencing the number of active flows. The main factors appear to fall into three categories: (a) number of sources underneath N-class access routers, (b) idle interval for those sources and (c) duration for which flows remain active. The think time of sources (x5) is the main influence on the number of active flows. The shorter the think time the more often sources become active and

attempt to transfer files (i.e., sequences of packets). Naturally, the more sources that exist under N-class access routers, the greater will be the effects of shorter think time. The number of sources under N-class access routers is influenced by two factors: the base number (x8) of sources used to populate the topology and the distribution (x9) of those sources. The plus setting for x9 increases the probability flows will exist between sources and receivers under N-class access routers. This setting gives the network a bit of a peer-to-peer (P2P) character.

For flows active between N-class access routers, the longer it takes for flows to complete, the more likely the number of active flows will increase. There is a bit of reinforcement at work here. The more active flows that transit a given access router, the lower will be the throughput of each flow and the longer it will take for each flow to finish transferring its packets. Thus, the higher the arrival rate of flows (i.e., the lower the source think time) the larger the number of active flows. Two other factors have significant influence on the time taken to complete flows. The first factor is the average file size (x4). Larger files take longer to transfer because more packets must be relayed and acknowledged. The second factor is network speed (x2): a slower network (plus setting) will take longer to transfer files of any particular size. Fig. 4-18 reveals this complex collection of related and reinforcing influences on the number of active flows.

Some other plots (not reproduced here) from the ten-step analysis also reveal interactions between number and distribution of sources (x8/x9), file size and distribution of sources (x4/x9) and think time and distribution of sources (x5/x9). These interactions make sense given the discussion contained in the previous paragraph. The effects from these factor interactions are much less significant than the main factors alone. In fact, the analysis of all 22 responses reveals that MesoNet simulations are driven by main factors and not by interactions among factors.

Congestion at N-class access routers could certainly lead to packet losses, which would stimulate retransmissions and cause flows to take longer to complete because the required number of packet transmissions would increase. Given this reasoning, one would expect many of the same factors influencing the number of active flows to also influence the retransmission rate. Fig. 4-19 displays the main-effects plot for network-wide retransmission rate (y10). Comparing Fig. 4-18 and Fig. 4-19 one can certainly see significant overlap in the main factors: number (x8) and distribution of sources (x9), average file size (x4) and think time (x5) and network speed (x2). In fact, the same settings for these factors that lead to increased number of active flows also lead to increased retransmission rate. The main difference is that retransmission rate is influenced most significantly (and equally) by network speed (x2) and buffer sizing algorithm (x3). Fewer buffers (minus setting) and lower network speed (plus setting) lead to increased probability of packet losses, which stimulate retransmissions.

The fact that buffer size was not so important with respect to the number of active flows reflects TCP congestion control. Given a larger number of flows, the TCP congestion control mechanism reacts to losses by adjusting flow sending rate: slowing packet transmissions, which leads to lower throughputs but also mitigates packet losses. Fig. 4-19 shows that mitigating packet losses becomes more difficult when buffer sizes are severely restricted. One would expect the main effects influencing retransmission rate to be identical to the main effects influencing loss rate (y5). Our review of the main-effects plot (not reproduced here) for loss rate confirms this expectation.

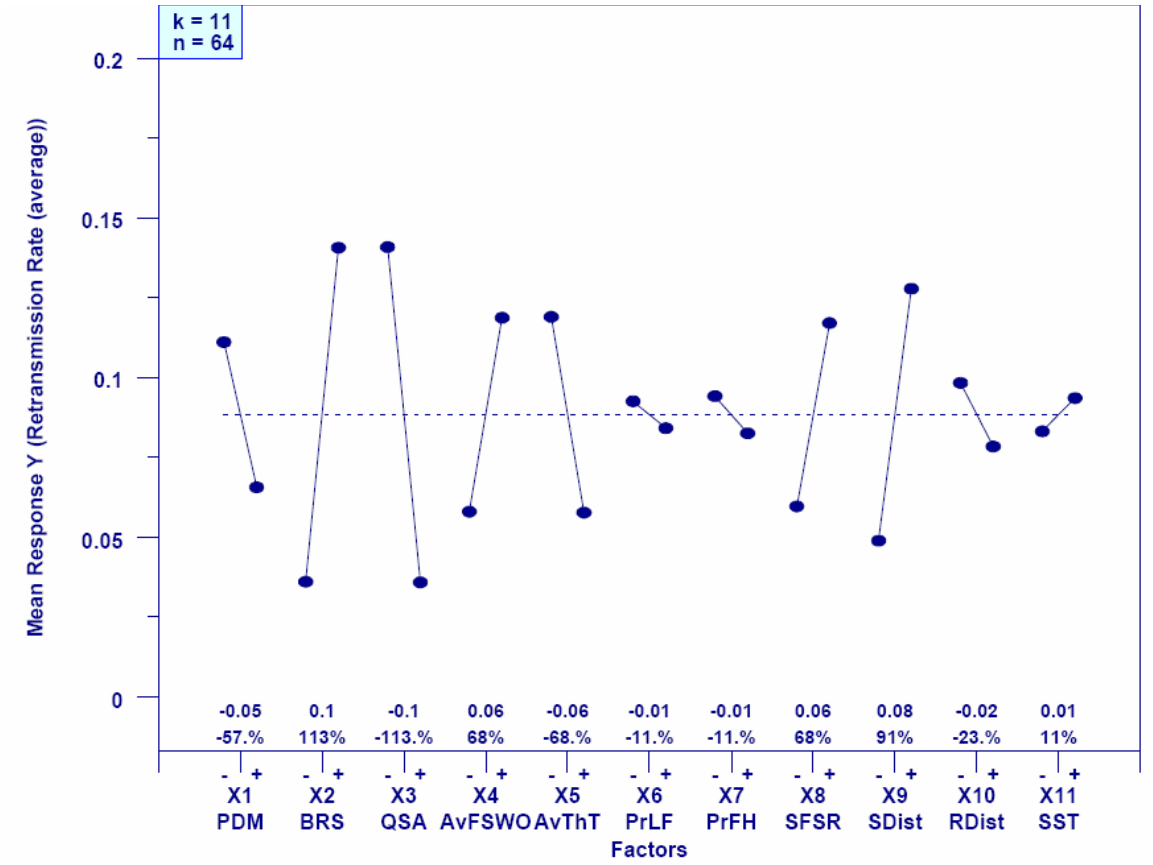


Figure 4-19. Main-Effects Plot for Response y10 (Average Retransmission Rate)

Given the analysis related to number of active flows and retransmission rate, one would expect throughput[8] on NN flows to be driven primarily by a relationship between available bandwidth (network speed) and number of active flows. The main-effects plot, Fig. 4-20, supports this expectation. Factors leading to fewer active flows include a lower number of sources (x8 minus) and a distribution that leads to fewer NN flows (x9 minus), as well as longer think time (x5 plus). Setting x9 to minus increases the probability that sources under N-class access routers will exchange data with receivers under F-class access routers, which gives the network a bit of a Web-centric character. With fewer active NN flows and higher network speed (x2 minus), the throughput achieved by NN flows is higher; under reverse conditions the throughput is lower.

Fig. 4-20 also reveals some subtle, although less significant, effects. Shorter propagation delay (x1 minus) yields higher throughput. This occurs because sources receive feedback more quickly and timeout values remain lower. Perhaps unexpectedly, throughput is higher when file sizes are smaller (x4 minus). This appears related to reducing the number of active flows, as flows complete more quickly when fewer packets must be transferred. Finally, larger buffer sizes (x3 plus) lead to higher throughput. This appears due to experiencing fewer losses, which requires fewer retransmissions and timeouts.

[8] Note that, though we use the term *throughput* when discussing flow classes, what we actually measure is often referred to as *goodput*; thus, retransmissions are not considered to be throughput.

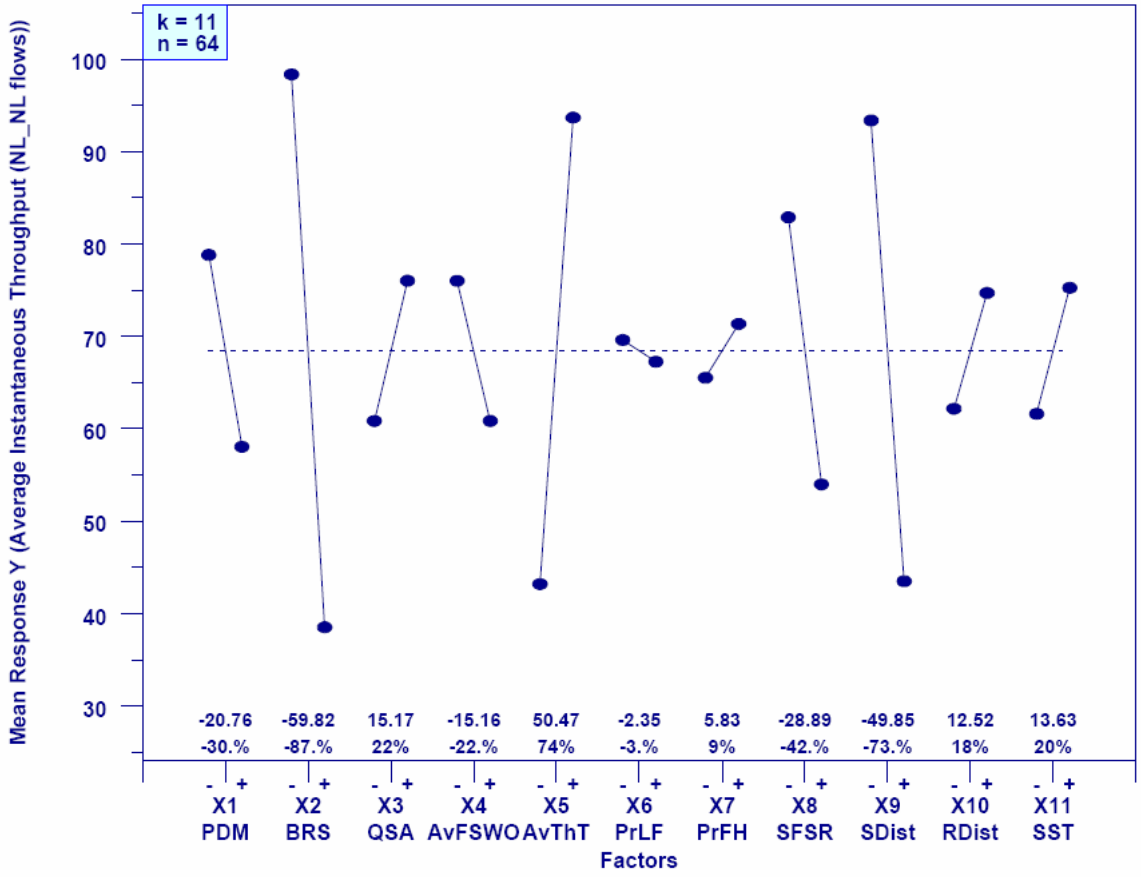


Figure 4-20. Main-Effects Plot for Response y22 (Average Instantaneous Throughput for NN Flows)

4.6.1.2 Delay-Related Responses. We selected average, smoothed, round-trip time (SRTT) as the response (y15) reflecting changes in network delay. Fig. 4-21 gives the related main-effects plot, which reveals that buffer-sizing algorithm and propagation delay are the main factors influencing SRTT. This makes eminent sense: higher propagation delay (x1 plus) and larger buffer sizes (x3 plus) lead directly to increase in SRTT. Larger buffer sizes permit bigger queues of packets, which increases queuing delay. Fig. 4-21 also reveals some minor effects, which suggest that congestion influences SRTT. This makes sense: more congestion leads to more packets residing in the bigger buffers.

4.6.1.3 Responses Related to Macroscopic Throughput. To represent the macroscopic throughput of the network, we selected two responses: data packets output per interval (y4) and flows completed per interval (y6). The first response represents the rate at which packets are flowing through the network, while the second response represents the rate at which flows are being completed by the network. We begin by considering the rate of packet output.

Fig. 4-22 reveals that the main influence on the rate of packet output is network speed: higher network speed (x2 minus) means a greater rate of packet output. This

stands to reason in a network with a sufficient number of active flows. The combination of shorter think times (x5 minus) and more sources (x8 plus) leads to an increase in the number of flows and the higher network speed implies that each flow can transmit faster, so the aggregate rate of packet output should be greater under these circumstances. File size is another factor significantly affecting the rate of packet output. Larger file sizes (x4 plus) lead to greater throughputs because a smaller portion of the transfer occurs during slow-start, the transfer phase during which a flow's congestion window is lowest. Flows transferring with a larger congestion window achieve higher throughput, which helps to increase the aggregate network throughput.

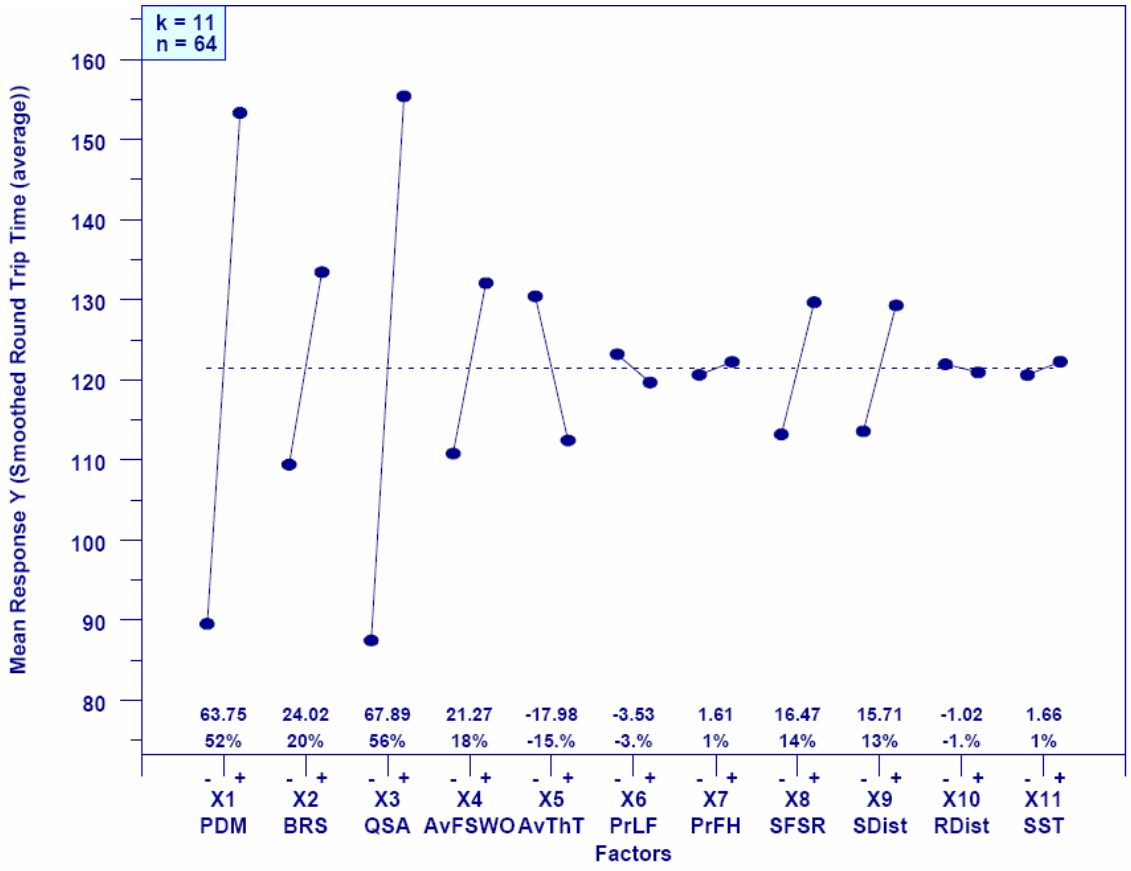


Figure 4-21. Main-Effects Plot for Response y15 (Average Smoothed Round-Trip Time)

As shown in Fig. 4-23, with one major exception, the story regarding the rate of flow completions is quite similar to the story regarding the rate of packet outputs. A sufficient number of connections (x5 minus and x8 plus) combined with higher network speed (x2 minus) contributes to a higher rate of flow completion. The exception involves file size (x4). In the case of packets output, larger file sizes (x4 plus) led to higher throughputs and thus to more packets output. On the contrary, for flows completed, smaller file size led to a higher completion rate. This stands to reason; smaller flows will be completed sooner. The sooner flows can be completed, the more flows can be completed per unit of time.

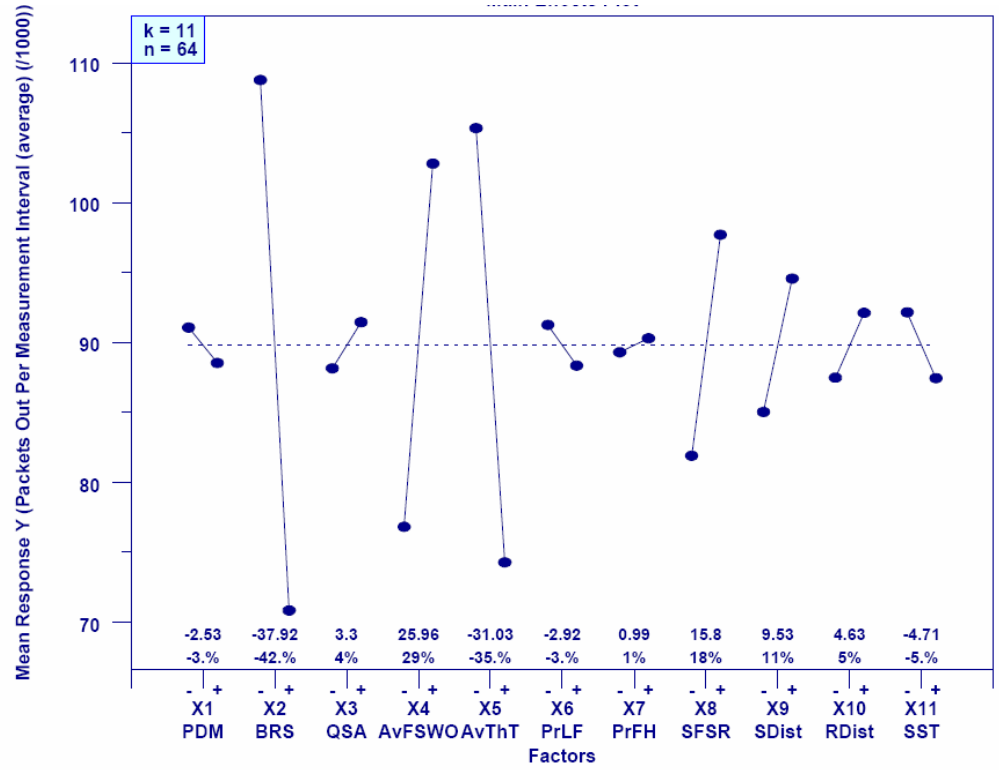


Figure 4-22. Main-Effects Plot for Response y4 (Average Packets Output per Measurement Interval)

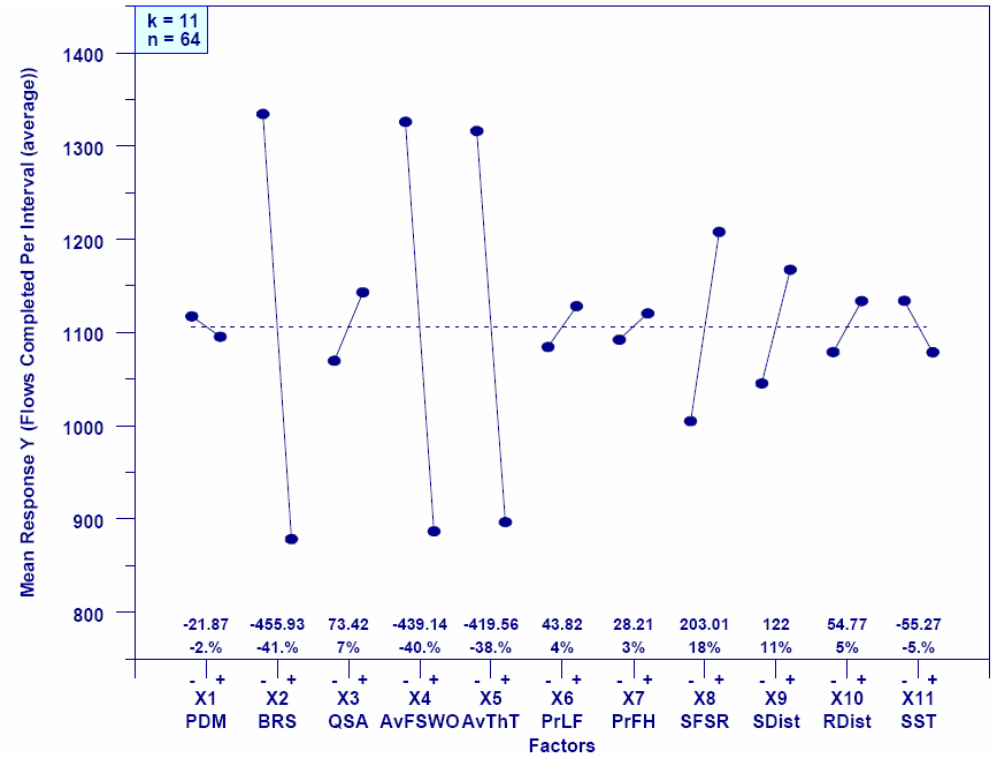


Figure 4-23. Main-Effects Plot for Response y6 (Flows Completed per Measurement Interval)

4.6.1.4 Responses Related to Advantaged Flow Classes. The final two responses we investigate represent throughputs achieved over advantaged flow classes, which are flows that transit between sources and receivers located under directly-connected and fast access routers. We examine the average instantaneous throughput of DD flows (y17) and FF flows (y20).

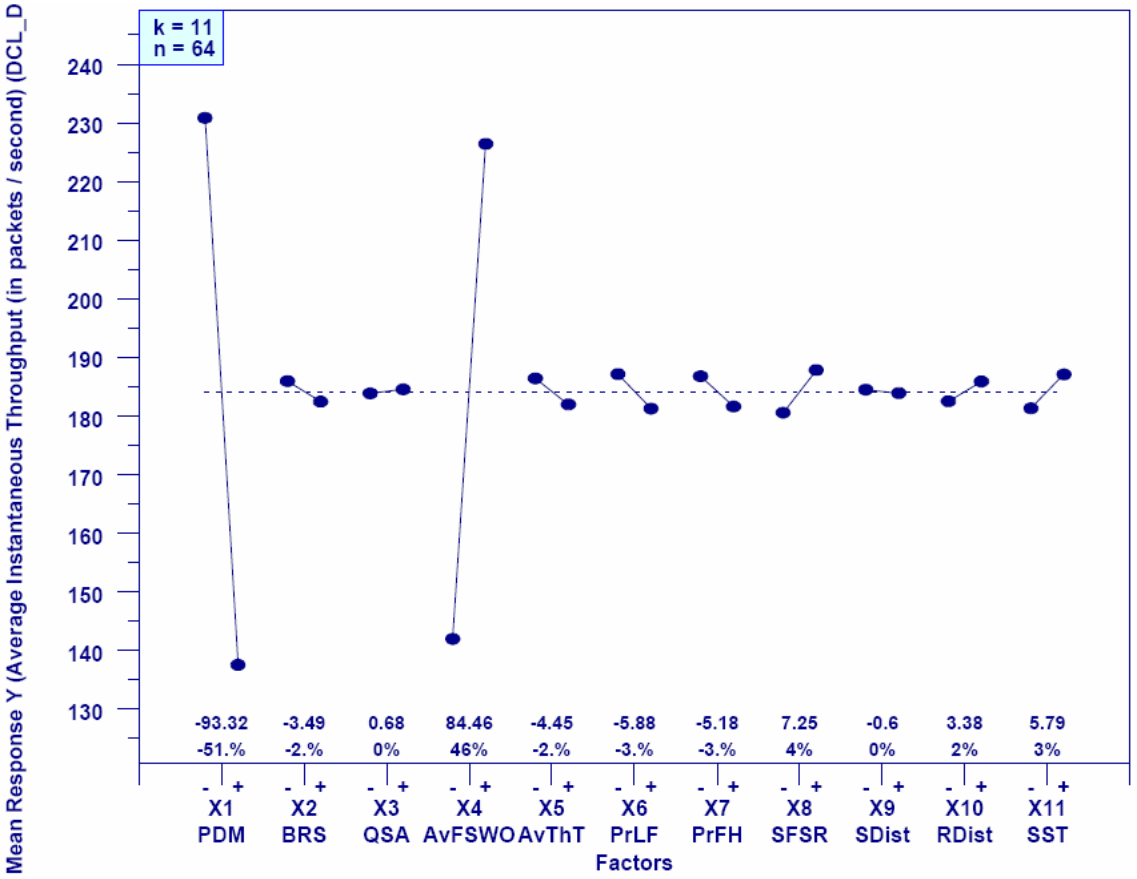


Figure 4-24. Main-Effects Plot for Response y17 (Average Instantaneous Throughput of DD Flows)

Each DD flow transits across a pair of D-class access routers, which are directly connected to backbone routers. D-class access routers are comparable in speed to POP routers, which are 10 times faster than N-class access routers. Given these factors, DD flows are the most advantaged in the simulation and should be able to achieve highest throughputs under the traffic scenario adopted for the sensitivity analysis. Few factors should impede the throughput of DD flows. Fig. 4-24 reveals that the throughput of DD flows is influenced by only two factors: propagation delay (x1) and file size (x4). This makes sense. Shorter propagation delay (x1 minus) permits faster feedback on DD flows, which allows the congestion window to increase more quickly. The rate of feedback is most important during the initial slow-start phase, where the congestion window starts at a small size but doubles with each acknowledgment received. The influence of file size is also clear. Larger file sizes (x4 plus) allow more of the packets in a file to be transferred after the flow reaches its peak sending rate. Smaller file sizes (x4 minus) imply that more of the packets in a file will be sent early in the slow-start phase, when a flow is building

up toward its peak sending rate. Throughput early in slow-start will be much smaller than throughput after a flow reaches its peak rate.

Two other classes of advantaged flows are those where a source or receiver is under an F-class access router and its correspondent is under either an F-class or D-class access router. These flows comprise the following classes: DF flows and FF flows. The throughput achievable on these flows is constrained by the F-class access routers, which operate at twice the speed of N-class access routers. DF and FF flows are less advantaged than DD flows. We use the throughput on FF flows (response y20) to represent both classes.

Fig. 4-25 shows the main effects influencing throughput on FF flows. FF flows are influenced by a more complex mix of factors than DD flows. The significance of propagation delay (x1) and file size (x4) are two clear common factors among all advantaged flow classes. Shorter propagation delay means quicker feedback, which leads to faster increase in the congestion window for flows that are not impeded by congestion. Larger file sizes allow more of a flow's packets to be transferred at a higher sending rate. Less advantaged (DN, FN and NN) flows are influenced mainly by congestion, so propagation delay has less effect on those flows.

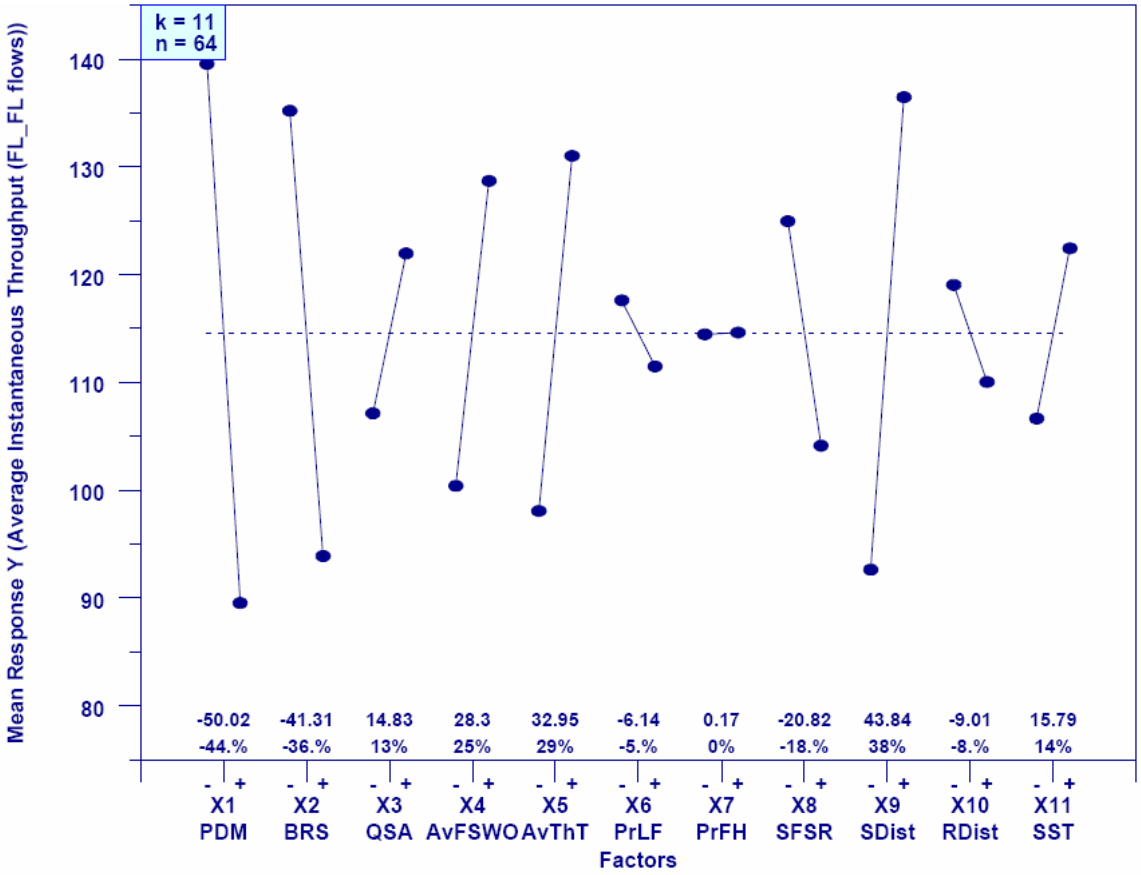


Figure 4-25. Main-Effects Plot for Response y20 (Average Instantaneous Throughput of FF Flows)

Unlike DD flows, FF flows can face some congestion because selected source distributions lead to higher numbers of FN flows. Specifically, a source distribution (x9 minus) that gives the network a Web-centric characteristic leads to more FN flows, which

compete for throughput with FF and DF flows. In addition, lower average think time (x5 minus) leads to more active flows that can compete for throughput. Under these circumstances, higher network speed (x2 minus) allows competing flows to achieve higher throughputs. The influence of all these factors is evident in Fig. 4-25.

Our investigation of throughput reveals three general categories of flows. Throughput in one category, which includes the most numerous (DN, FN and NN) flows, is influenced mainly by congestion and network speed. Throughput in a second category, which includes only the most advantaged and least numerous DD flows, is influenced mainly by propagation delay and file size. Throughput in the remaining category (DF and FF flows) is influenced by a combination of the factors influencing the other two categories.

4.6.2 Sensitivity Analysis Guided by Principal Components Analysis

In this section we examine the sensitivity of the principal components (PCs) to variations in model inputs. Recall from our PCA (Sec. 4.5) that we identified four main principal components accounting for most variation in the model's 22 responses. We viewed these PCs, summarized in Table 4-24, as groupings of responses representing different aspects of the model's behavior. The reader may note a correspondence between the groupings by principal component and the groupings used (in Sec. 4.6.1) to describe the sensitivity analysis of responses guided by correlation analysis. Below, we report the results of applying main-effects analysis to the PCs identified in Table 4-24.

Table 4-24. Definition of Major Principal Components in Model Response

Principal Component	Responses in Principal Component
Congestion (PC1)	y1, y2, y5, y7, y10, y11, y12, y13, y14, y19, y21, y22
Delay (PC2)	y15, y16
Throughput for Advantaged Flows (PC3)	y17, y18, y20
Macroscopic Throughput (PC4)	y3, y4, y6

4.6.2.1 Congestion. Given that PC1 represents the effects of network congestion, one would expect significant congruence between factors affecting PC1 and factors affecting responses driven by congestion. Previously, we analyzed three congestion-related responses: number of active flows (y1), retransmission rate (y10) and average instantaneous throughput for NN flows (y22). We also noted that loss rate (y5) and retransmission rate were related closely. Analysis of PC1 should show that the same factors influence PC1 as influence responses y1, y5, y10 and y22. Fig. 4-26 displays the main-effects plot for PC1. The key factors influencing PC1, in order of significance, include: network speed (x2), think time (x5), distribution (x9) and number (x8) of sources, file size (x4) and buffer size (x3). This set of factors is also the union of factors most significantly driving responses y1, y10 and y22. Further, given insights from our previous analyses, we conclude that Fig. 4-26 illustrates the following factors induce network congestion and its consequences: slower network speed (x2 plus), smaller buffer sizes (x3 minus), larger file size (x4 plus), shorter think time (x5 minus), more sources (x8 plus) and a P2P-like distribution of sources (x9 plus). Reversing these factors eases network congestion and its consequences.

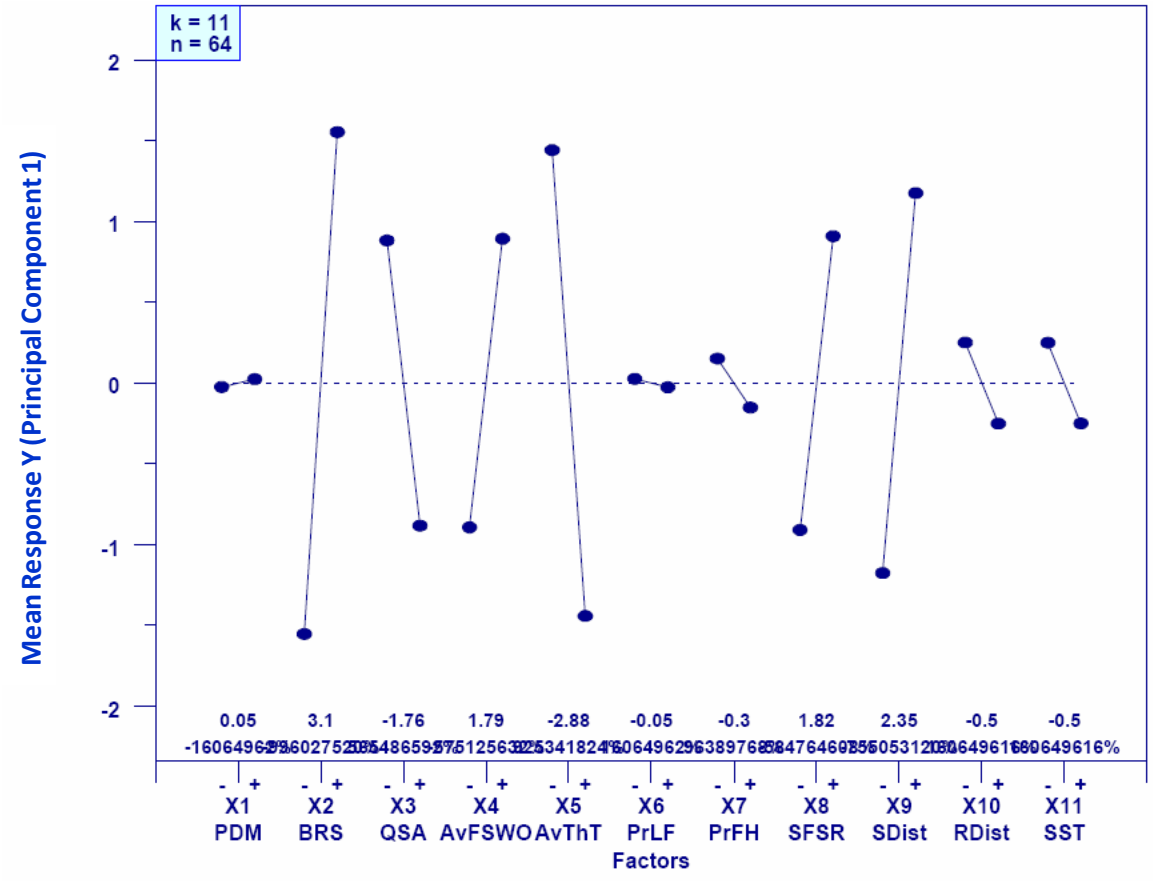


Figure 4-26. Main-Effects Plot for PC1 (Network Congestion) – the % computations in Main-Effects plots for PCs appear unreadable (and meaningless) because PC values are normalized to a 0 mean. The reader may simply use the magnitude and slope of the lines connecting the – and + level of each of the 11 factors to provide information that may be compared with previous Main-Effects plots applied to individual responses rather than PCs, which are linear combinations of many individual responses

The reasons that these factors modulate network congestion have already been explained in Sec. 4.6.1.1. Our analysis suggests that factors modulating network congestion will influence all responses grouped under PC1. For example, increasing network congestion lowers the average congestion window (y11), decreases the rate of congestion window increases (y12), increases the rate of negative acknowledgments (y13) and timeouts (y14) and reduces average throughput for DN (y19), FN (y21) and NN (y22) flows. For the traffic scenario adopted, the macroscopic pattern of network congestion relates primarily to the most numerous types of flows, which transit the most numerous N-class access routers. Other factors influence less numerous, more advantaged flows, as discussed below when considering PC3.

4.6.2.2 Delay. Given that PC2 represents effects on network delay, one would expect significant congruence between factors affecting PC2 and factors affecting responses driven by delay. Previously, we analyzed one delay-related response: average smoothed round-trip time – SRTT (y15). SRTT was driven primarily by two factors: buffer size (x3) and propagation delay (x1). Note that SRTT is significantly correlated (0.70) with

relative queuing delay (y16), which is driven mainly by one factor: buffer size (x3). One would expect PC2 to be driven by the same factors that drive SRTT and relative queuing delay. Fig. 4-27 depicts the main-effects plot for PC2. The plot shows that PC2 is mainly influenced by two factors: buffer size and propagation delay. Fig. 4-27 also reveals a minor influence of think time (x5). Interpreting Fig. 4-27 shows that average network delay increases with increases in propagation delay (x1 plus) and buffer size (x2 plus). Further, Fig. 4-27 suggests that decreasing think time (x5 minus) tends to increase delay; this is likely due to the fact that more flows are active simultaneously, which helps to fill the larger available buffer space.

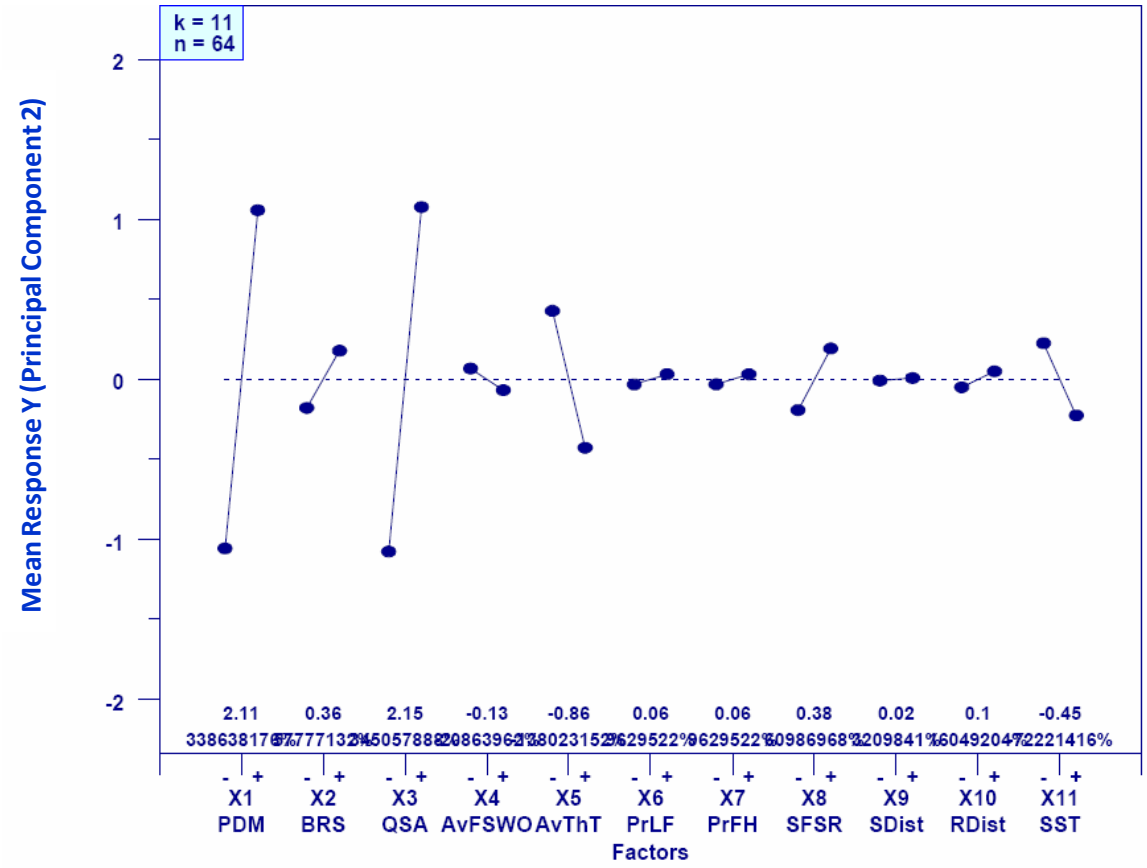


Figure 4-27. Main-Effects Plot for PC2 (Network Delay)

These results indicate that, for the traffic scenario used here, network delay is largely orthogonal to network congestion. Why might this be so? Under congestion, the TCP congestion control mechanism causes flows to reduce their sending rate. This adapts the flow of packets into the network in accordance with perceived congestion. The feedback rate for the congestion control mechanism depends largely on network delay, which is due to two factors: propagation delay and queuing delay. Propagation delay is modulated by the distance packets must travel, and queuing delay is modulated by the size of buffers in network routers. Congestion cannot affect the distance that packets must travel. When buffers are small, congestion cannot affect the queuing delay because queues will be small. Only when buffers are large can the degree of congestion influence

network delay, but in this case TCP congestion control reacts to reduce the rate of traffic entering the network, which tends to limit the number of packets in the network. For these reasons, results suggesting lack of correlation between network congestion and delay appear reasonable.

4.6.2.3 Throughput for Advantaged Flows. Given that PC3 represents the effects on throughput for advantaged flows, one would expect significant congruence between factors affecting PC3 and factors affecting throughput for DD (y17), DF (y18) and FF (y20) flows. Previously, we analyzed factors influencing throughput on DD and FF flows. Factors influencing throughput on DF flows (not included in this report) are identical to the factors influencing FF flow throughput. The main factors influencing throughput for advantaged flows include: propagation delay (x1), file size (x4), network speed (x2), distribution of sources (x9) and think time (x5). Review of Fig. 4-28 shows that the same factors influence PC3. PC3 is also driven to some extent by buffer size (x3), which was not a significant factor in the previous analysis of throughput for DD or FF flows.

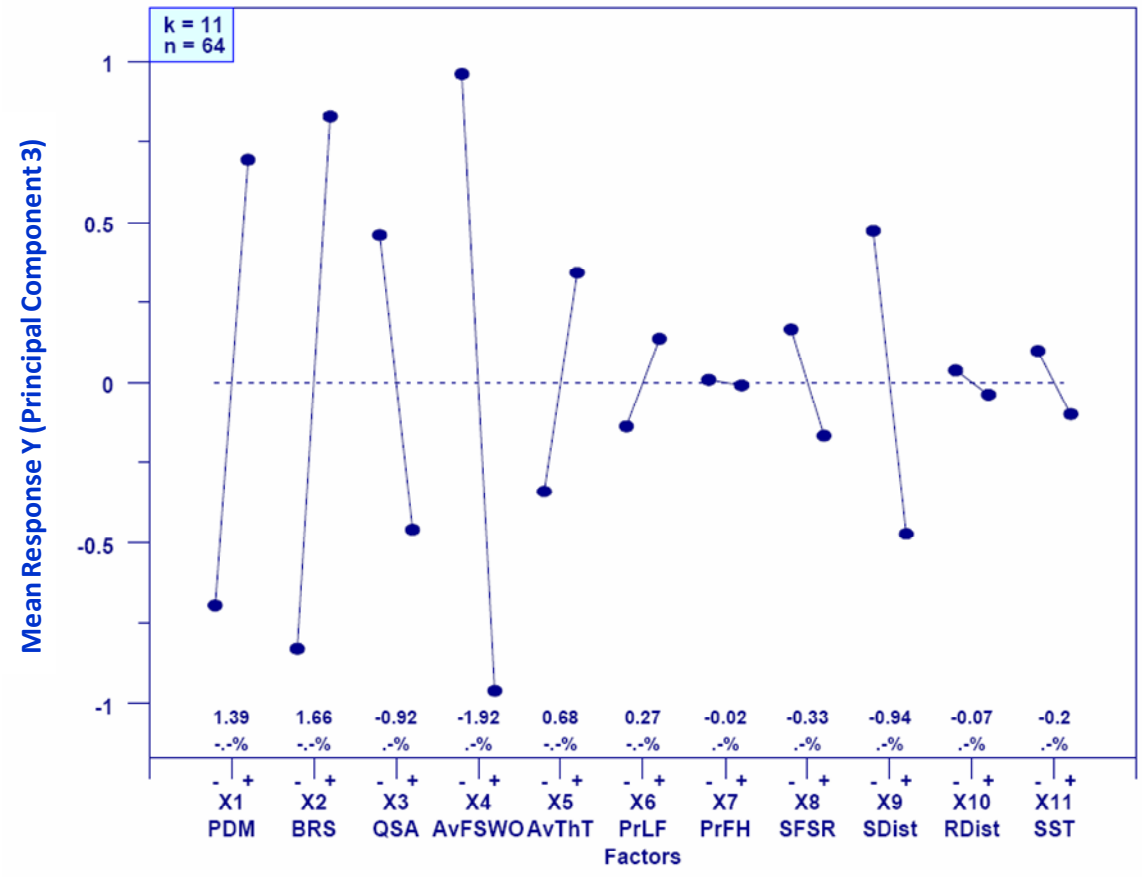


Figure 4-28. Main-Effects Plot for PC3 (Throughput for Advantaged Flows)

One could interpret Fig. 4-28 as depicting lower throughput above the zero line and higher throughput below the zero line. Using this interpretation, Fig. 4-28 indicates that higher throughput on advantaged flows results from: larger file sizes (x4 plus),

higher network speed (x2 minus), shorter propagation delay (x1 minus), and more P2P-like network traffic (x9 plus). Reversing the settings for these factors would lead to lower throughputs. These findings are consistent with our previous analysis of the factors influencing throughput for DD (y17) and FF (y20) flows.

One somewhat new piece of information is revealed by Fig. 4-28: the influence of buffer size on throughput for advantaged flows. In our previous analyses, buffer size had a more modest influence on throughput. The influence that was present indicated that smaller buffers led to lower throughputs and larger buffers led to higher throughputs. This seems to make sense because small buffers lead to increased losses, which lead to increased retransmissions, which lead to longer file transfer times, which results in lower throughputs. Fig. 4-28 shows the same influence, so the interpretation of PC3 remains consistent with the earlier results for DD and FF flows.

4.6.2.4 Macroscopic Throughput. Given that PC4 represents effects on macroscopic (network-wide) throughput, one would expect significant congruence between factors affecting PC4 and factors influencing the rate of data packets leaving the network (y4) and the flow-completion rate (y6). Recall, though, that one factor, file size (x4), had the opposite influence on y6 and y4. With this information, we should be able to determine which aspect of macroscopic throughput is represented by PC4. Sec. 4.5 suggested that PC4 represents macroscopic throughput of flow completions.

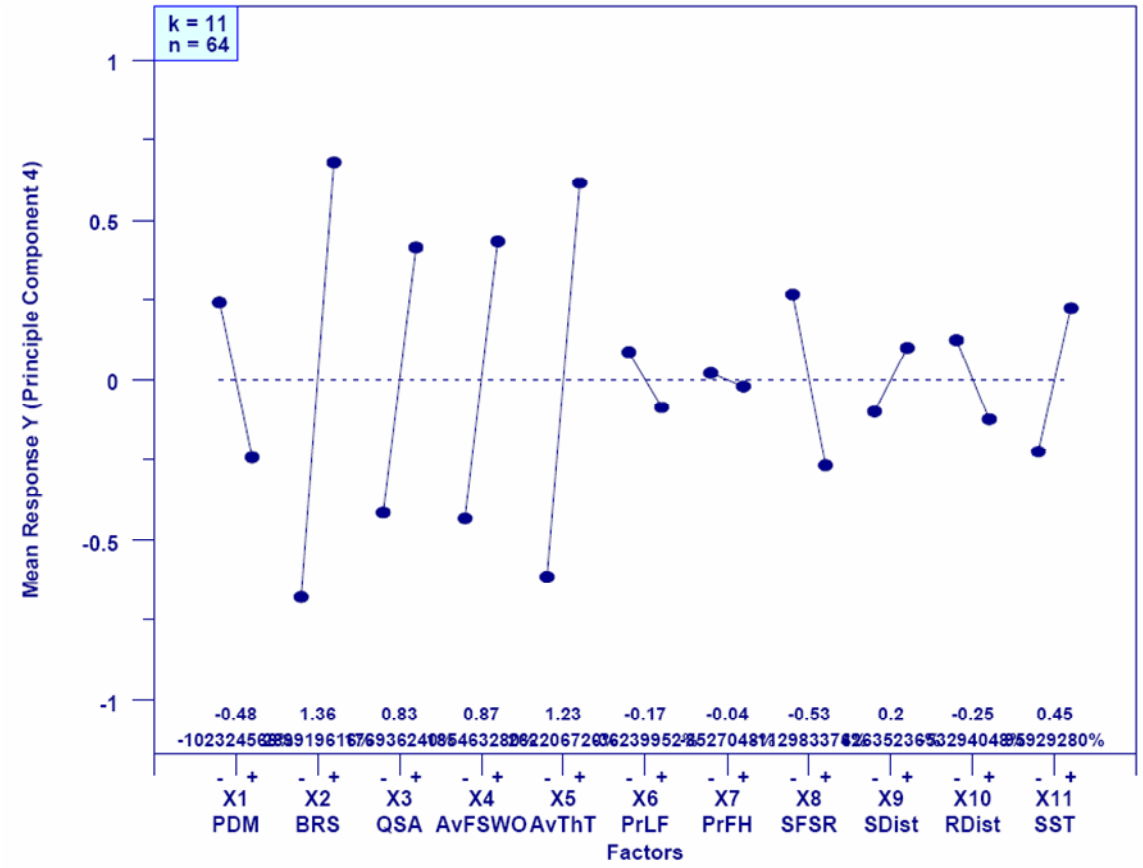


Figure 4-29. Main-Effects Plot for PC4 (Macroscopic Throughput)

Fig. 4-29 displays the main-effects plot for PC4. The primary factors influencing PC4, in order of significance, include: network speed (x2), think time (x5) and file size (x4). Interpreting Fig. 4-29 suggests that higher network speed (x2 minus) and shorter think time (x5 minus) increase macroscopic throughput.[9] These findings are consistent with the factors influencing both the rate of packet output (y4) and the rate of flow completion (y6). Fig. 4-29 shows that smaller file size (x4 minus) causes variation in the same direction as higher network speed and shorter think time. From this, we conclude that PC4 represents macroscopic throughput of flow completions. This agrees with the previous PCA.

4.6.3 Summary of Findings from the Sensitivity Analysis

Results from the sensitivity, correlation and principal components analyses increased our confidence in MesoNet. The behavioral patterns and relationships revealed by the sensitivity analysis aligned with our expectations. Further, the sensitivity analysis provided significant insight into model operation. We review our key findings here. We begin with a summary of the main aspects of model behavior. Second, we characterize the major factors influencing model behavior. Third, we identify and discuss factors that appear to have little influence on model behavior.

4.6.3.1 Main Aspects of Model Behavior. The sensitivity analysis revealed the model to have six main behavioral aspects: congestion, delay, throughout for DD flows, throughput for DF and FF flows, packet throughput and flow completion throughput. We discuss each of these in turn.

Congestion. The largest behavioral aspect of the model relates to macroscopic congestion, which occurs primarily in the slowest (N-class) access routers. In the topology employed for the sensitivity analysis, most users (represented as model sources) accessed the network through the (105) N-class routers. This is analogous to business and home users who connect to a network via limited bandwidth links. Higher network tiers (represented in the model by 22 POP and 11 backbone routers) typically operate at speeds sufficient to support traffic entering the network from the access tier. The topology used in the sensitivity analysis reflects this fact of network design. The model's heterogeneous topology allowed selected (D-class and F-class) access routers to operate at higher speeds. (Twenty-eight) F-class access routers represented larger businesses that might support Web sites, which could be accessed by many users, most of whom connect to the network through N-class routers. (Six) D-class access routers represented research institutions and very large corporations that connect directly to the network backbone.

The net result of this topology is that most active flows transit N-class access routers because most users reside underneath such routers. These flows include NN, FN and DN flows. Since these flows are most numerous, their behavior tends to drive macroscopic congestion, which occurs at the network edge (i.e., in the access tier). Of course, this is also due in part to the homogeneous Web-like traffic model employed during the sensitivity analysis. Regardless of traffic model, one should expect network congestion to arise primarily at the access tier because transit networks are continuously

[9] Note that PCA involves normalizing and transforming responses to a scale differing from the scales of the original responses. This means that interpretation of the main-effects plots for principal components must be aided by context provided from previous interpretation of main-effects influencing particular responses.

monitored by network providers and the bandwidth in the POP and backbone tiers is provisioned to meet traffic demands from the access tier.

Throughput available to individual NN, FN and DN flows is constrained by the bandwidth of N-class routers. This means that lower network speed (x2 plus) will reduce flow throughputs, while higher speed (x2 minus) will increase flow throughputs. Further, the more flows (y1) that transit an N-class router, the lower will be the individual throughputs for each flow. This behavior is revealed by the related response variables: y19, y21 and y22. The number of active flows transiting an access router is influenced primarily by three factors: number (x8) and distribution (x9) of sources and average idle time (x5) between source transfers. Increasing the number of sources leads to increased congestion within a fixed topology of N-class access routers. In the sensitivity analysis, the FN to NN ratio (in number of flows) shifts depending on the distribution of sources. With the plus setting for x9 the pattern of flows takes on a P2P-like characteristic, where the FN to NN ratio decreases. With the minus setting the flow pattern adopts a Web-centric characteristic, where the FN to NN ratio increases. Since NN flows take slightly longer to complete than FN flows, the P2P pattern tends to result in more flows transiting N-class routers at any given instant. And, of course, the shorter the idle time between transfers the more sources will arrive in any given period.

Network congestion also influences macroscopic responses, including: loss (y5) and retransmission (y10) rates, congestion window (y11) and its rate of increase (y12), and rate of negative acknowledgments (y13) and timeouts (y14). As with flow throughputs, these responses can be primarily attributed to the relative number of flows simultaneously transiting N-class access routers, as well as to the speed of N-class routers. The existence of fewer simultaneous flows, combined with higher speed, creates a better experience for individual flows and less congestion at the network edge.

To summarize, congestion occurs at the network edge. The primary effects of congestion are due to flows transiting N-class access routers. Higher speed N-class routers mitigate congestion to some extent. The macroscopic effects of congestion are due to the most numerous flow types: NN, FN and DN flows.

Delay. Network delay, measured as smoothed round-trip time (SRTT), is influenced by two main factors: propagation delay (x1) and buffer-sizing algorithm (x3). As one would expect, longer propagation delay and larger buffer sizes lead to increased SRTT (y15). Further, relative queuing delay (y16) is driven only by buffer size. These relationships are as expected. Perhaps unexpected is that network delay is largely uncorrelated with congestion. We attribute this to the fact that the TCP congestion control mechanism responds to network congestion by slowing the rate of packets injected into the network and thus limiting the number of packets that might otherwise be sitting in network buffers.

Throughput for DD Flows. For DD flows, both the source and receiver reside under D-class access routers, which connect directly to backbone routers and operate at the same speed as POP routers. Further, the number of simultaneously active DD flows is typically quite small, relative to other flow classes. Given these facts, DD flows should be able to achieve throughputs constrained only by the minimum of the speeds of the source and receiver. The sensitivity analysis revealed that, unique among flow classes, throughput of DD flows is influenced by only two factors: propagation delay (x1) and file size (x4).

A DD flow must transit through TCP slow-start before reaching its maximum achievable throughput. The time taken to reach maximum throughput then depends upon the feedback rate on the flow. The feedback rate is determined mainly by propagation delay. Longer propagation delay lengthens time taken to achieve maximum throughput. Further, for larger files, more packets may be transferred at maximum throughput, so average throughput is higher. For the traffic patterns used in the sensitivity analysis, no other factors influenced throughput of DD flows. For this reason, DD flows must be given separate consideration from other flow classes.

Throughput for DF and FF Flows. DF and FF flows could potentially achieve maximum throughputs constrained only by the minimum of the speeds of the source and receiver, but some other factors can interfere. For example, DF and FF flows compete for throughput with FN flows, which might be smaller or larger in number, depending upon various factors discussed previously. When the relative number of FN flows is smaller, then DF and FF throughputs are influenced mainly by propagation delay and file size, as is the case for DD flows. When the relative number of FN flows is larger, then DF and FF throughputs are influenced more by the factors that influence throughput of FN flows.

Packet Throughput. Packet throughput (y4) is influenced primarily by network speed (x2), idle time of sources (x5) and file size (x4). When network speed is faster (x2 minus), flow congestion windows are larger, and so flows can send more packets per unit of time. When idle time is smaller (x5 minus), more flows tend to be active, which means more flows are injecting packets into the network. Finally, when file sizes are larger (x4 plus), then more flows are operating at their maximum achievable throughputs, so more packets are being injected into the network. The higher the network speed and the more packets being injected into the network, the greater the number of packets leaving the network. These factors determine packet throughput. Of course, when there are many active flows and lower network speed, then congestion increases and the TCP congestion control mechanism slows the rate of packets entering the network, which also slows the rate of packets exiting the network.

Flow-Completion Throughput. Flow-completion throughput (y6) is also influenced primarily by network speed, idle time of sources and file size. In this case, however, smaller file sizes (x4 minus) lead to shorter file-transfer times, which increases the number of flows completed in a given time period. Of course, since each file transfer spends a lower proportion of its duration at maximum achievable throughput, the number of packets injected will be lower than if the file size is smaller. Thus, to some extent, variations in file size lead to a tradeoff between packet throughput and flow-completion throughput.

4.6.3.2 Major Factors Influencing Model Behavior. Based on the results of the sensitivity analysis, we can identify the major factors influencing the behavior of MesoNet. We consider the results of the analysis by a domain expert and also the PCA. We begin with the results from a domain expert, which are based on six main behavioral characteristics, as identified in the preceding section.

We use one response to represent each characteristic: packet throughput (y4), flow-completion throughput (y6), congestion (y10), delay (y15) and throughput of DD (y17) and FF (y20) flows. Table 4-25 shows the result of a rank analysis, where the relative influence of each factor on each of the six responses is assigned a rank from one

(most influential) to 11 (least influential) based upon the degree to which the factor altered the response when moving from a plus to a minus setting. The average rank is computed for each factor, and then the average rank is converted into an integer ranking based on ordering the factors from most (one) to least (11) influential. The table shows that network speed (x2) is the most influential factor, followed by file size (x4) and think time (x5). Next is number of sources (x8), followed by propagation delay (x1) and distribution of sources (x9). Buffer-sizing algorithm (x3) ranks seventh. The remaining factors lag: initial slow-start threshold (x11), distribution of receivers (x10), probability that a flow transfers a larger document (x6) and then probability that a host is fast (x7).

Table 4-26 gives a similar analysis based on the top four principal components identified by the PCA. The PCA squeezes out some redundancy included in the analysis conducted by the domain expert. In particular, the domain expert separated throughput for advantaged flows into two responses (y17 and y20) based upon an observation that different factors drove the two responses. In addition, the domain expert noted that packet throughput (y4) was driven in two different directions, depending upon file size (x4); thus, decided to retain packet throughput as a separate response. The PCA amalgamated y17 and y20 (in PC3) and also combined y4 with y6 (in PC4).

Table 4-25. Rank Analysis based on Domain Expertise

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
y4	9.5	1	8	3	2	9.5	11	4	5	6.5	6.5
y6	11	1	6	2	3	9	10	4	5	7.5	7.5
y10	7	1.5	1.5	5	5	10	10	5	3	8	10
y15	2	3	1	4	5	8	10	6	7	10	10
y17	1	8	10.5	2	8	5	5	3	10.5	8	5
y20	1	3	8	5	4	10	11	6	2	9	7
Average Rank	5.25	2.92	5.83	3.50	4.50	8.58	9.50	4.67	5.42	8.17	7.67
Ordinal Rank	5	1	7	2	3	10	11	4	6	9	8

Table 4-26. Rank Analysis based on Principal Components Analysis

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
PC1	10.5	1	6	5	2	10.5	9	4	3	7.5	7.5
PC2	2	6	1	7	3	8.5	8.5	4	11	10	5
PC3	3	2	5	1	6	8	10.5	7	4	9	10.5
PC4	5	1	4	3	2	10	11	7	9	8	6
Average Rank	5.13	2.50	4.00	4.00	3.25	9.25	9.75	5.50	6.75	8.63	7.25
Ordinal Rank	5	1	4	4	2	10	11	6	7	9	8

Comparing Tables 4-25 and 4-26 reveals similarities (in the main) and a few differences. Both tables rank factors x6, x7, x10 and x11 as not very influential on system response. Both tables identify network speed (x2) as the main factor driving response and both tables also rank file size (x4) and think time (x5) as significant factors. Both tables also agree on the relative influence of propagation delay (x1). The PCA suggests that buffer size is fairly influential, while the domain expert finds buffer size to be less significant. When the redundancies (y17 and y4) are removed from Table 4-25, the significance of buffer size is comparable for both analyses. Overall, the analyses are quite consistent.

4.6.3.3 Factors Exhibiting Little Influence on Model Behavior. The sensitivity analysis, whether based on domain expertise or principal components, shows that system response is little influenced by four factors: probability of transferring a larger file (x6), probability that a source is on a fast host (x7), distribution of receivers (x9) and initial slow-start threshold. The experiment design varied the probability of transferring a (10x) larger file from 0.01 (x6 plus) to 0.02 (x6 minus). Apparently, the difference in these probabilities was small enough that the system response was not influenced. Results might have proven different if the higher probability were increased, but this would imply that a lower proportion of file transfers were reserved for simple Web browsing activities.

The probability that a source is on a fast host (x7) makes little difference in system response because most sources existed underneath N-class access routers, which had limited bandwidth to share among those sources. Apparently, due to multiplexing with other sources, most sources were seldom able to realize their maximum potential transmission rate.

The distribution of receivers (x10) had little influence on results because no matter the setting, most receivers resided under N-class access routers. The proportion of receivers under N-class access routers varied from 76 % (x10 plus) to 86 % (x10 minus). The distribution of receivers had more bearing on the number of receivers under D-class access routers (4 % and 2 % for x10 plus and minus, respectively) and also under F-class access routers (20 % and 12 % for x10 plus and minus, respectively).

The initial slow-start threshold (x11) had significant influence on only one response: average congestion window size (y11). Absent losses, the congestion window on a flow can become very high even though flow throughput will be limited by the minimum of the maximum speeds of the source and receiver. Setting a high initial slow-start threshold (x11 plus) allows flows to increase their congestion window very quickly to a large size. Setting a lower initial slow-start threshold (x11 minus) permits quick increase to a small size and then linear increase afterward. Thus, when congestion is light, using a large threshold for initial slow-start permitted the average congestion window size to become much larger, even though there was little influence on flow throughput.

4.7 Exploring Effects of Buffer Sizing

As we explained in Sec 4.1.6, the experiment design approach we used can facilitate additional exploratory analyses that were not necessarily planned at the time the experiment was undertaken. Here, we demonstrate this feature of our approach by using model response data to investigate the importance of buffer sizing relative to network speed and propagation delay. Our experiment design used two algorithms for buffer sizing. One algorithm, which is recommended practice [40], sets buffer size by multiplying average estimated round-trip time by capacity. The second algorithm, suggested by McKeown and colleagues [37], divides buffer size computed from the first algorithm by the square root of the expected number of flows. This second algorithm requires much less buffer space in network routers. McKeown and colleagues conducted an analytical study and empirical experiment that found similar performance when using either buffer sizing algorithm. The McKeown study, which was limited to a small number of flows transiting a few routers, suggested that network providers could deliver

reasonable performance while requiring much less memory in routers. The study left for future work consideration of the effects of the alternate buffer-sizing algorithm in a network-wide context. Our sensitivity analysis was not intended explicitly to study detailed effects of buffer-sizing algorithms, but the experiment design used does provide information that could shed some light on the topic.

In this section, we use the results from our sensitivity experiment to explore the effects of buffer-sizing algorithm on overall network behavior and user experience. Our goal is to develop evidence relevant to the findings of the McKeown study. First, we consider the effects that choice of buffer-sizing algorithm has on smoothed round-trip time (SRTT) and on relative queuing delay. In previous discussions, we showed that buffer-sizing algorithm influenced both these aspects of network delay. Here, we look a bit more explicitly at the data. Second, we conduct a rank analysis based on selected responses chosen to characterize overall network behavior and user experience.

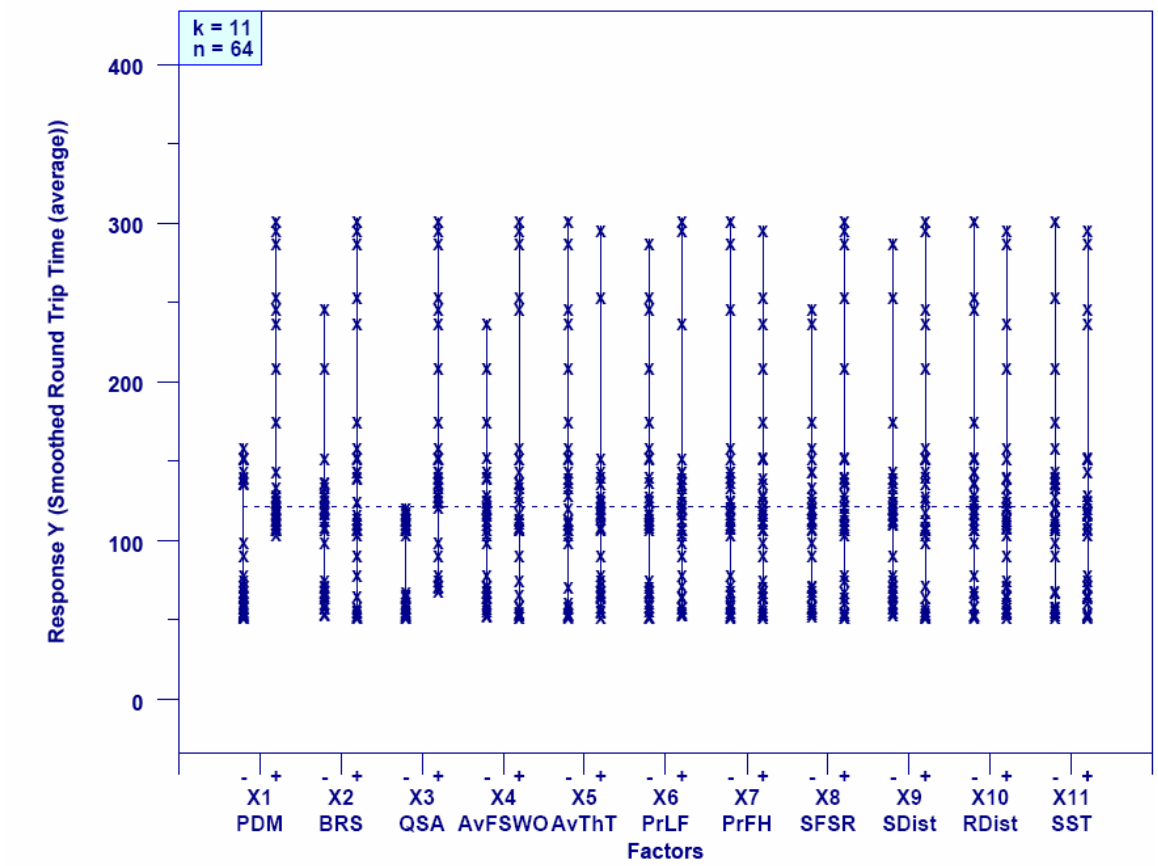


Figure 4-30. Multi-factor Scatter Plot of Smoothed Round Trip Time (y15) for each of the 11 Experiment Factors

4.7.1 Effects on Delay Variation. Fig. 4-30 presents a multi-factor scatter plot (explained in Appendix D.2) for SRTT. Fig. 4-31 shows a similar plot for relative queuing delay. Each plot depicts how the related response varies with the two settings of each of the 11 factors used in our experiments. For the current discussion we are interested in the buffer-sizing algorithm (factor x3). Fig. 4-30 shows that the choice of buffer-sizing algorithm shifts the pattern of SRTT. The McKeown algorithm restricts variation in SRTT. This

occurs because buffer sizes are much smaller and thus queuing delays must also be smaller.

Fig. 4-31 illustrates clearly that reduced buffer size restricts the range of queuing delay that packets experience. Reducing variation in queuing delay within a network leads to more predictable SRTT and also to faster feedback regarding congestion. These traits might be considered valuable for selected networks and applications. On the other hand, one wonders whether more predictable delay might come at the cost of worsening behavior in other aspects of the network. The McKeown study suggested that smaller buffer size would not detract from user experience. We investigate this question next.

4.7.2 Effects on Other Aspects of Network Behavior. In this section, we consider the relative influence of propagation delay (x1), network speed (x2) and buffer-sizing algorithm (x3) on selected responses, chosen to represent macroscopic network behavior and user experience. To represent macroscopic behavior, we use packet throughput (y4), flow-completion throughput (y6), retransmission rate (y10) and relative queuing delay (y16). To represent user experience, we use average throughput from three different flow classes: DD flows (y17), FF flows (y20) and NN flows (Y22). We aim to determine which of the three factors (x1, x2 or x3) has largest influence on the combined responses.

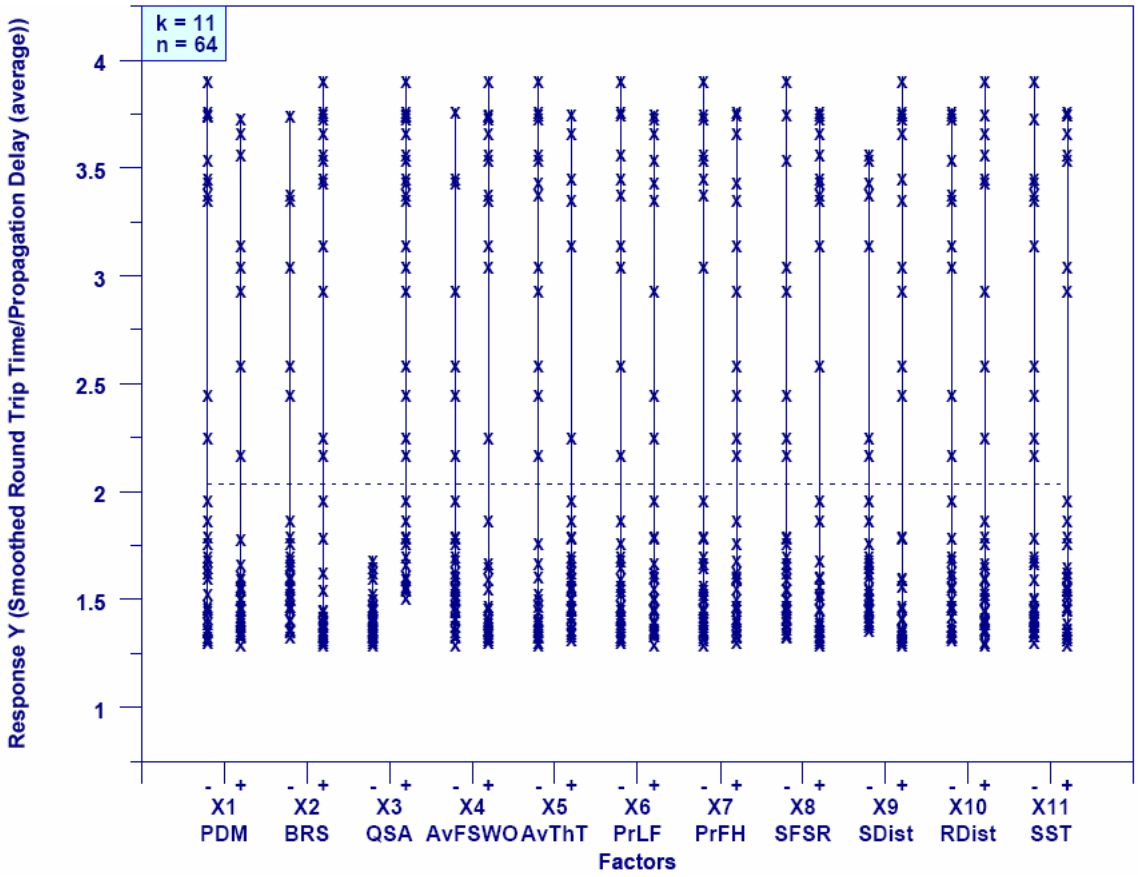


Figure 4-31. Multi-factor Scatter Plot of Relative Queuing Delay (y16) for Each Experiment Factor

We use a rank analysis to study the effects of our chosen factors on our selected responses. In this particular analysis, we elected to use a larger number to indicate higher

rank and a smaller number to indicate lower rank. We begin by combining our three factors into a condition that can be assigned one of eight settings, as illustrated in Table 4-27. Next, we compute the average value for each of our responses under each condition. Table 4-28 displays the results of this averaging.

Table 4-27. Mapping of Factor Settings to Eight Conditions
(M means the – level of a factor and P means the + level of a factor)

		Values		
Condition	Factor Settings x1:x2:x3	Propagation Delay Multiplier	Backbone Router Speed	Buffer Sizing Algorithm
C1	M:M:M	1	800	*RTT*x*C*/SQRT(*n*)
C2	P:M:M	2	800	*RTT*x*C*/SQRT(*n*)
C3	M:P:M	1	400	*RTT*x*C*/SQRT(*n*)
C4	P:P:M	2	400	*RTT*x*C*/SQRT(*n*)
C5	M:M:P	1	800	*RTT*x*C*
C6	P:M:P	2	800	*RTT*x*C*
C7	M:P:P	1	400	*RTT*x*C*
C8	P:P:P	2	400	*RTT*x*C*

Table 4-28. Average Response Values for Each Condition

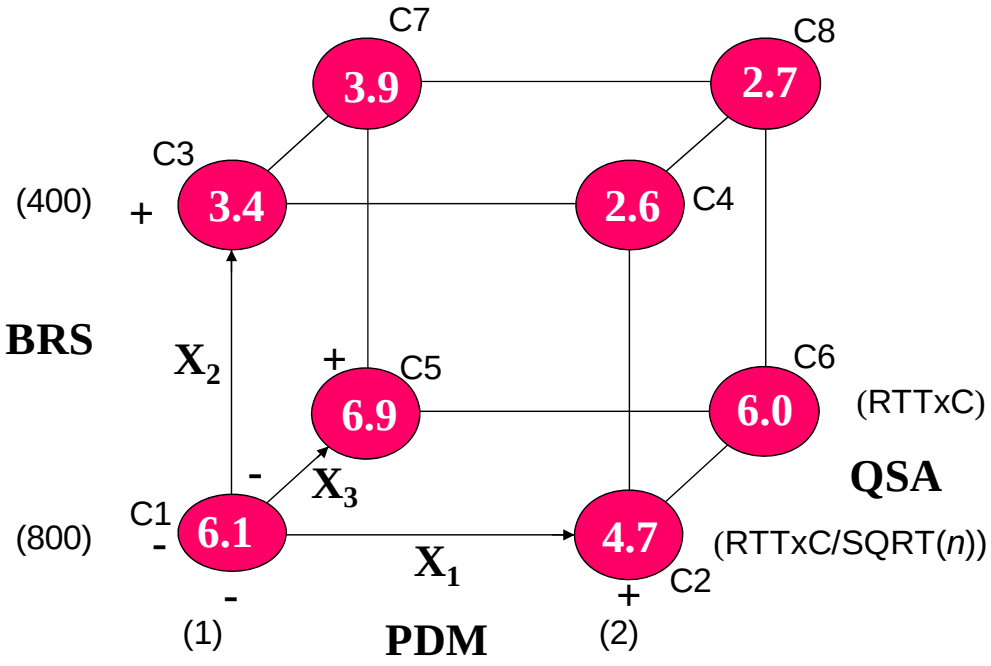
	Response						
Condition	y4	y6	y10	y16	y17	y20	y22
C1	109 863.71	1 384.55	0.07	1.53	229.88	167.12	107.06
C2	104 049.61	1 218.14	0.05	1.45	138.02	97.66	70.59
C3	68 721.38	803.31	0.27	1.41	229.82	89.81	37.48
C4	69 996.51	872.54	0.17	1.39	137.65	73.92	28.25
C5	111 195.23	1 324.93	0.02	2.52	237.65	169.29	119.39
C6	109 949.76	1 409.62	0	1.91	138.17	106.73	96.33
C7	74 509.45	956.32	0.08	3.27	226.01	131.99	51.3
C8	70 170.53	881.34	0.04	2.83	136.26	79.82	37.04

Using the average responses from Table 4-28, we next rank each condition from high (8) to low (1) for each response, based on the appropriate ordering criteria. For retransmission rate (y10) and relative queuing delay (y16) a lower value would be ranked higher. For the other five responses in Table 4-28 a higher value would be ranked higher. After ranking the conditions with respect to each response, we compute an average ranking. The results of our ranking are shown in Table 4-29.

We can assign the average rank for each condition to the vertex of a cube, where each vertex represents a specific combination of settings for propagation delay, network speed and buffer size. Fig. 4-32 shows the cube corresponding to Table 4-29. Moving along the edges among the vertices on the cube allows us to determine changes in ranking attributable to each factor. The change in each factor (x1, x2 and x3) across all conditions is represented by a set of four different edges from among the 12 edges contained in the cube. We extract the relevant changes in ranking and display them in Table 4-30.

Table 4-29. Ranking for Each Condition vs. Each Response

	Condition							
Response	C1	C2	C3	C4	C5	C6	C7	C8
y4	6	5	1	2	8	7	4	3
y6	7	5	1	2	6	8	4	3
y10	4	5	1	2	7	8	3	6
y16	5	6	7	8	3	4	1	2
y17	7	3	6	2	8	4	5	1
y20	7	4	5	1	8	5	6	2
y22	7	5	3	1	8	6	4	2
Average Rank	6.1	4.7	3.4	2.6	6.9	6.0	3.9	2.7



4-32. Average Condition Ranking Displayed on Vertices of a Cube

Table 4-30. Changes in Ranking Attributable to Each Factor

	Propagation Delay (x1)	Network Speed (x2)	Buffer Sizing (x3)
Edge 1	1.4	2.7	0.8
Edge 2	0.8	2.1	0.5
Edge 3	0.9	3	1.3
Edge 4	1.2	3.3	0.1

Interpreting Table 4-30 we see that changing network speed has the largest effect on the responses we selected. Changing propagation delay has the second largest effect. Changing buffer-sizing algorithm has the smallest effect. Further, Fig. 4-31 shows that changing from fewer to more buffers has a larger effect when network speed is high and propagation delay is long. This makes intuitive sense because more packets could

potentially be inside the network when speed and propagation delay increases, so a higher proportion of the increased buffers would likely be occupied.

While our examination of the effect of buffer-sizing algorithm should not be considered definitive, the results extracted from our sensitivity experiment tend to support the findings of McKeown and colleagues. For the topology and traffic patterns used in our study, reducing buffer sizes by the square root of the expected number of active flows transiting each router had little overall effect on macroscopic network behavior and user experience. On the other hand, we found that reducing buffer sizes can markedly restrict the range of variation in queuing delay and thus in round-trip times. Reducing variance in round-trip times allows faster feedback on losses and permits the TCP flows to adapt more quickly, which could offset some of the losses that might otherwise occur due to reducing the number of buffers.

4.8 Conclusions

We described a method for conducting sensitivity analyses for simulations of large, complex systems, such as communications networks, computing grids and service-oriented architectures. The method included: orthogonal fractional factorial (OFF) design of two-level experiments, correlation and principal components analyses and a ten-step graphical analysis. We applied the method to gain an understanding of MesoNet (described in Chapter 3). Correlation and principal components analyses revealed the main dimensions of MesoNet behavior to include: (1) congestion, (2) delay, (3) throughput of advantaged flows and (4) aggregate rate of flow completions.

Sensitivity analysis identified the main factors influencing each aspect of MesoNet behavior. Congestion is influenced primarily by network speed, number and distribution of sources and average idle time for sources. Delay is influenced primarily by buffer size and propagation delay. Advantaged flows come in two categories: DD flows and FF (and DF) flows. Throughput for DD flows is influenced by two factors: propagation delay and file size. While propagation delay and file size prove influential, throughput for FF flows is also affected by network speed and distribution and idle time of sources. These additional factors reflect situations where increased numbers of FN flows compete with DF and FF flows. The aggregate rate of flow completions is influenced by network speed, source idle times and file size. Using rank analysis, we found the order of overall influence exerted by key factors. From more to less influential, the overall influence of factors was ordered as follows: network speed, file size and idle time, number of sources, propagation delay, distribution of sources and buffer size.

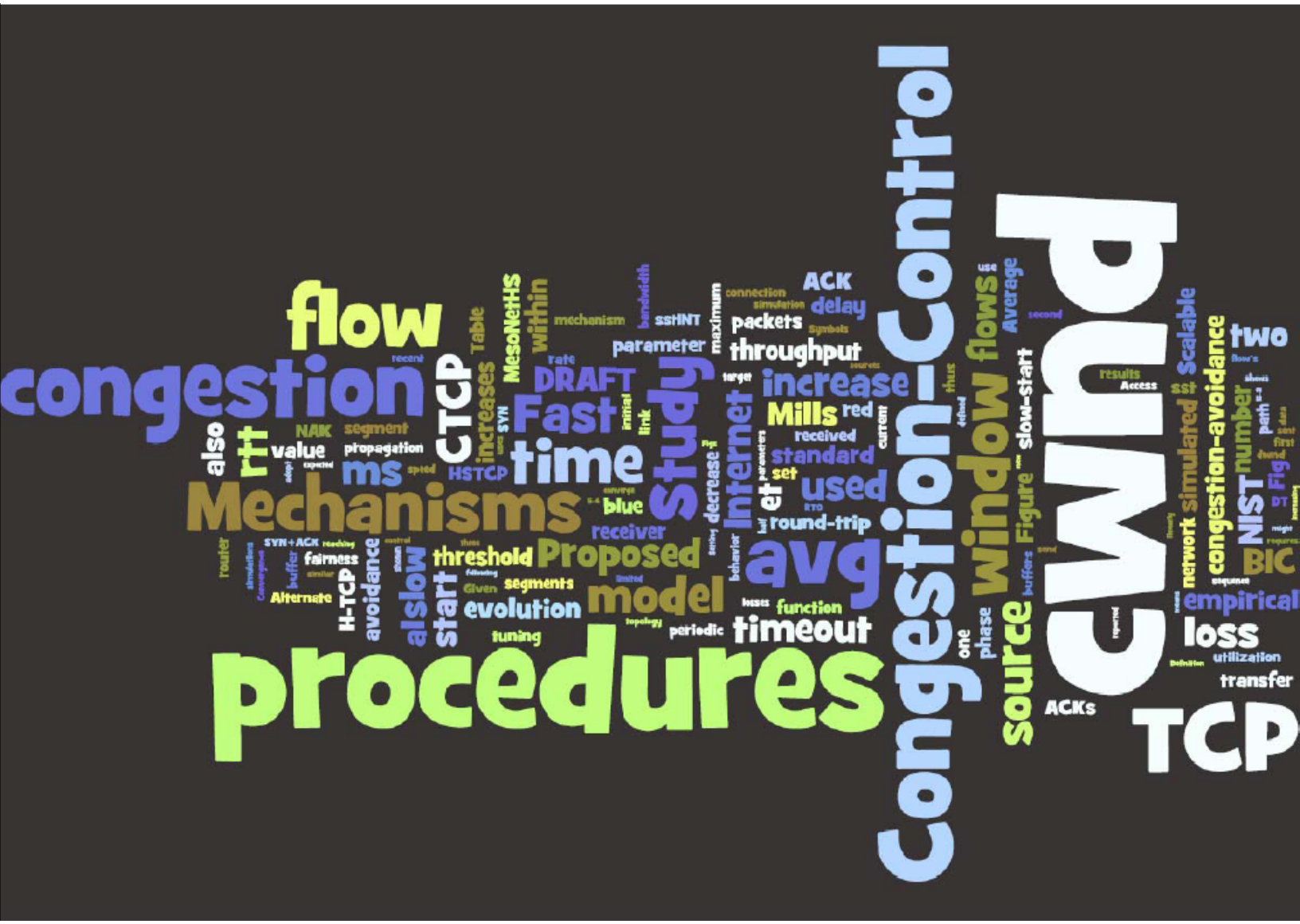
Sensitivity analysis also identified four factors that had little influence on the behavior of MesoNet. These factors included: probability of electing to download a larger file, probability of sources and receivers residing on fast hosts, distribution of receivers and initial slow-start threshold.

We extended our analysis to investigate explicitly the relative influence of network speed, propagation delay and buffer size on overall behavior of the model. We found that network speed had greatest influence and buffer size had least influence. We also showed that very small buffer sizes restrict the range of variance in smoothed round-trip times. Further, we found that buffer size has greater influence on model behavior when network speed and propagation delay are larger.

We also conducted a second sensitivity analysis (see Appendix C) that used the same 2^{11-5} OFF experiment design template documented in Sec. 4.1, but that changed the specific values assigned to each of the 11 experiment factors so that network size and speed were increased and so that the distances between the – and + levels for each parameter were expanded. The sensitivity analysis documented in Appendix C found the same main factors driving MesoNet behavior as we documented in Chapter 4, though the influence of propagation delay was increased due to the expanded distance between the values chosen for the two levels. In general, the network simulated in Appendix C exhibited lower correlations because overall congestion was lower. This was also reflected in changes in principal components, which became more difficult to interpret.

Overall, analyses conducted on MesoNet increase our confidence in the model's correctness and reasonableness. The fundamental characteristics of the model and the topology, as investigated here and in Appendix C, provide a reasonable basis for comparing the effects of alternate congestion control algorithms on macroscopic network behavior and user experience. In the next section, we discuss the congestion control algorithms we will study and we show how we modeled those algorithms.

Chapter 5 – Modeling Congestion Control Algorithms



5 Modeling Alternate Congestion Control Mechanisms

The fundamental design of the Internet protocol suite [3] assumes that network elements, such as routers, are relatively simple – receiving, buffering and forwarding packets among connected links and dropping packets when buffers are insufficient to accommodate arriving packets. Under this assumption, computers connected to the Internet must implement decision algorithms to pace the rate at which packets are injected into the network. Such decision algorithms, known typically as congestion control mechanisms, are implemented independently by each source with the goal of achieving a satisfactory network-wide outcome and a fair distribution of resources to all active sources. In the current state-of-the-practice, congestion control mechanisms are implemented as part of the transmission control protocol (TCP) [8-10] that operates within every computer attached to the global Internet. While TCP congestion control procedures have proven quite successful [2] at achieving desired properties, numerous researchers [46-51, 64] have postulated potential changes in relationships among bandwidth and propagation delay as the speed of network links increases toward 10s and 100s of gigabits per second (Gbps). Under such envisioned circumstances, researchers predict that TCP congestion control procedures will prove insufficient, leading to substantial underutilization in network resources and preventing end users from achieving high transfer rates, potentially reaching or surpassing 1 Gbps. These predictions have stimulated researchers to propose alternate congestion control mechanisms [52-61] that might achieve higher network utilization and better user performance as network speeds increase.

As part of proposing alternate congestion control mechanisms, researchers typically model, simulate and implement prototypes and then explore how candidate congestion control mechanisms might affect the Internet and its users. Given the increasing number of proposals, interest is growing [62-68] in developing procedures to fairly and effectively evaluate the properties of the proposals. A similar motive underlies the work reported in the current study, where our approach is to simulate proposed congestion control mechanisms within a reasonably large network that can support $O(10^5)$ active flows simultaneously. To illustrate our methodology, we have chosen to investigate six proposed alternate congestion control mechanisms [52-54, 58, 60-61], which have been simulated and studied empirically at smaller scales.

In this chapter of our study, we introduce the basic concepts underlying TCP congestion control and we explain the changes to those procedures that are proposed by six different research teams. Other research teams [55-57, 59] have also proposed changes to TCP congestion control procedures. We chose to examine only six proposals in order to limit our study, which focuses on methods for conducting evaluations rather than on an exhaustive consideration of all published proposals. We selected five specific proposals because a recent study by Li, Leith and Shorten [67] reports empirical results from prototype implementations included within Linux. This enables us to validate our simulations of the proposals against the reported empirical measurements. We chose a sixth alternate congestion control mechanism, Compound TCP [58], or CTCP, because it has been proposed by researchers at Microsoft and, thus, may be available in the future within a large number of computers attached to the Internet. Further, there are some recent empirical results [66] against which we can validate our model of CTCP. The methodology we define in our study can be applied to additional proposals for alternate congestion control mechanisms.

The remainder of this chapter is organized into five major topics. We begin (in Sec. 5.1) by introducing TCP congestion control and then (in Sec. 5.2) define the procedures adopted by six, selected, alternate proposals for congestion control in the Internet. Next (in Sec. 5.3), we describe how we model congestion control procedures within MesoNet. In Sec. 5.4, we describe the test configuration used to verify that we correctly model each congestion control mechanism. We then present simulation results showing evolution of congestion windows over time for each congestion control mechanism that we model. The information reported in this section sets the stage for us to consider (in Chapters 6 through 9) whether proposed alternate congestion control procedures might change macroscopic network behavior or user experience.

5.1 TCP Congestion Control

A typical TCP flow evolves through three phases: connection, transfer and close. For purposes of congestion control, we limit our discussion to the connection and transfer phases. Fig. 5-1 gives a high-level view of these two phases. During the connection phase, a source attempts to establish contact with an intended receiver. Inability to establish contact results in a connection failure, which prevents data from flowing between source and receiver; thus, connection establishment procedures provide one form of congestion control implemented by TCP. During the transfer phase, a source sends data (in the form of segments) on the flow until the required number has been received successfully. A receiver signals receipt of data segments by sending acknowledgments (ACKs) to the source. By sending duplicate acknowledgments, a receiver may also indicate failure to receive specific segments, which the source must then retransmit. Further, a sender may fail to receive acknowledgments, which requires the sender to raise a timeout and to retransmit unacknowledged data. During the transfer phase, congestion control procedures determine when a source may send data segments to a receiver. The resulting series of segments is known as a flow.

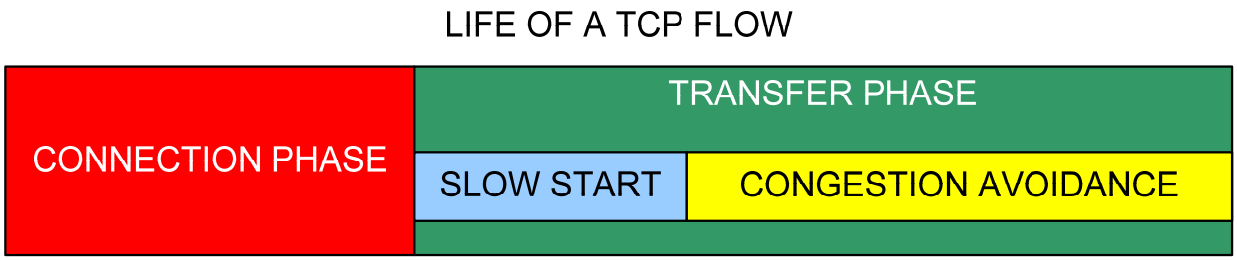


Figure 5-1. Main Phases and Congestion Control Procedures in the Life of a TCP Flow (The Six Alternative Congestion Control Mechanisms in this Study Change the Congestion Avoidance Regime Only)

TCP flows consist of a series of data segments (or packets) sent from a source to a receiver, along with a corresponding stream of acknowledgment packets flowing in the reverse direction. At any given time, a source may send a prescribed number of packets (known as the congestion window, or *cwnd*) prior to receiving an acknowledgment. Thus, the size of the *cwnd* controls the rate of packet transmission on a flow. Using TCP congestion control procedures, a source increases a flow's *cwnd* exponentially from a small initial value until either a loss is detected or until the *cwnd* reaches a threshold, known as the initial slow-start threshold, or *sst*. If the *sst* is reached, the source subsequently increases the *cwnd* more slowly, at a linear rate. If a packet is lost, then the *cwnd* is reduced in half and then increased linearly until another packet is lost after which the *cwnd* is reduced in half again and so on. The resulting saw-tooth pattern in

the *cwnd* (see Fig. 5-7) induces a corresponding variation in the rate of transmission on a flow. TCP congestion control procedures require that sources use dynamic measurement of losses on a path to discover how many packets per second may be transmitted on a given flow. In addition, as we show later, these dynamic measurements allow TCP sources to adapt flow transmission rate as path characteristics change. The main goal of TCP congestion control is to allow all flows transiting shared paths to obtain an equal (fair) share of any available transmission capacity, while also allowing flows to increase or decrease transmission rate to achieve full utilization of path capacity. In particular, when flows are added to a path already supporting several flows in equilibrium with fair transmission rates, TCP congestion control procedures aim to achieve a new equilibrium, where all flows achieve fair (but lower) transmission rates. Similarly, when flows are dropped from a path in equilibrium, TCP congestion control aims to achieve a new equilibrium where all flows achieve fair (but higher) transmission rates. We define the responsiveness of TCP congestion control procedures as the time taken to achieve a new equilibrium by dynamically adjusting the congestion windows on flows sharing a common network path.

Dynamic adjustment of the congestion window happens only during the transfer phase, which includes two regimes: slow start and congestion avoidance. Slow start occurs when a source is uncertain about the transmission rate that might be achieved on a TCP flow. For this reason, after establishing a connection, a source begins the transfer phase using slow-start procedures. A source also adopts slow-start procedures after a timeout. Slow-start begins by sending data at a slow rate but then increases that rate quickly (e.g., exponentially) as ACKs arrive from the receiver. Once a source has a better idea about an achievable transmission rate, slow-start procedures are abandoned in favor of congestion avoidance procedures, which attempt to increase the sending rate more slowly (e.g., linearly). Thus, congestion control procedures during the transfer phase have three basic purposes: (1) find an achievable transfer rate on a flow; (2) maintain the achievable transfer rate if possible; (3) attempt to increase the achievable transfer rate. Proposals for revising TCP congestion control procedures target mainly congestion avoidance procedures within the transfer phase.

Below, we describe the procedures used in our model for connection establishment and slow start. Then we outline our model of standard (i.e., Reno) TCP congestion avoidance procedures. Subsequently, in Sec. 5.2, we describe our model of congestion avoidance procedures for each of the six alternate congestion control mechanisms that we simulate.

5.1.1 Connection Phase

Typically, establishing a TCP connection (or flow) requires a three-way handshake involving a connection-request (SYN) segment sent from a source to a receiver, followed by a connection-confirm (SYN+ACK) segment sent from a receiver to a source and then ending with an ACK segment sent from a source to a receiver. Our model simulates connection establishment as a two-way handshake – SYN followed by SYN+ACK – because TCP allows ACKs to be piggybacked on data (DT) segments. This implies that the first DT sent from a source to receiver during the transfer phase may also double as the final segment of connection establishment.

Of course, congestion may lead to lost SYN or SYN+ACK segments, so a source must implement error detection and recovery procedures, which typically involve retransmitting SYN segments. In our model, we simulate such procedures while adopting default parameters typically used in TCP implementations within the Microsoft Windows® family of operating systems. We take this decision because many computers connected to the Internet use the

Microsoft implementation of TCP. Fig. 5-2 illustrates schematically our connection establishment model, showing one possible scenario leading to connection failure.

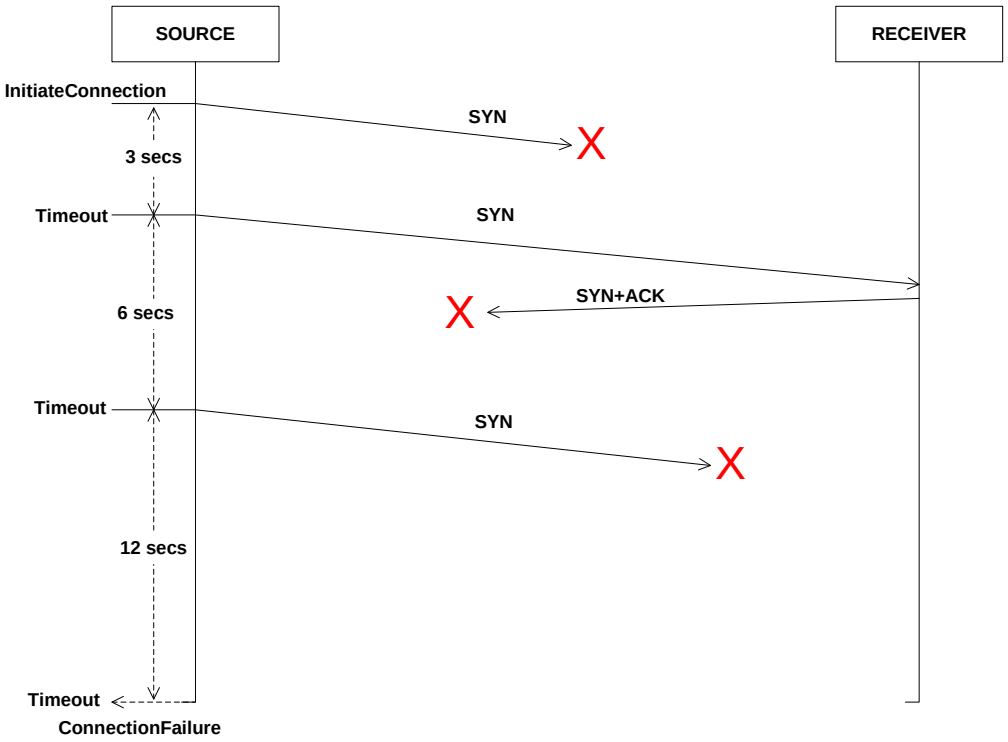


Figure 5-2. TCP Connection Establishment Procedures Leading to Connection Failure

During connection establishment a source first sends a SYN segment and waits for a period of time (3 s in Fig. 5-2) for a SYN+ACK. If no SYN+ACK segment arrives, the source sends a second SYN and waits for a longer period of time (6 s in Fig. 5-2). If no SYN+ACK segment arrives, the source sends a third SYN and waits for a longer period of time (12 s in Fig. 5-2). This cycle repeats until the maximum number of SYNs (3 in Fig. 5-2) has been sent or until a SYN+ACK segment arrives. If no SYN+ACK segment arrives after the maximum number of SYNs is sent, then (as shown in Fig. 5-2) TCP raises a connection-failure signal. For the parameters we adopt, connection failures occur after 21 s without receipt of a SYN+ACK after the first SYN is sent. Arrival of a SYN+ACK segment during this timeout period (as illustrated in Fig. 5-3) results in successful connection establishment. Fig. 5-2 and Fig. 5-3 show several losses that can require retransmission of SYN and SYN+ACK segments.

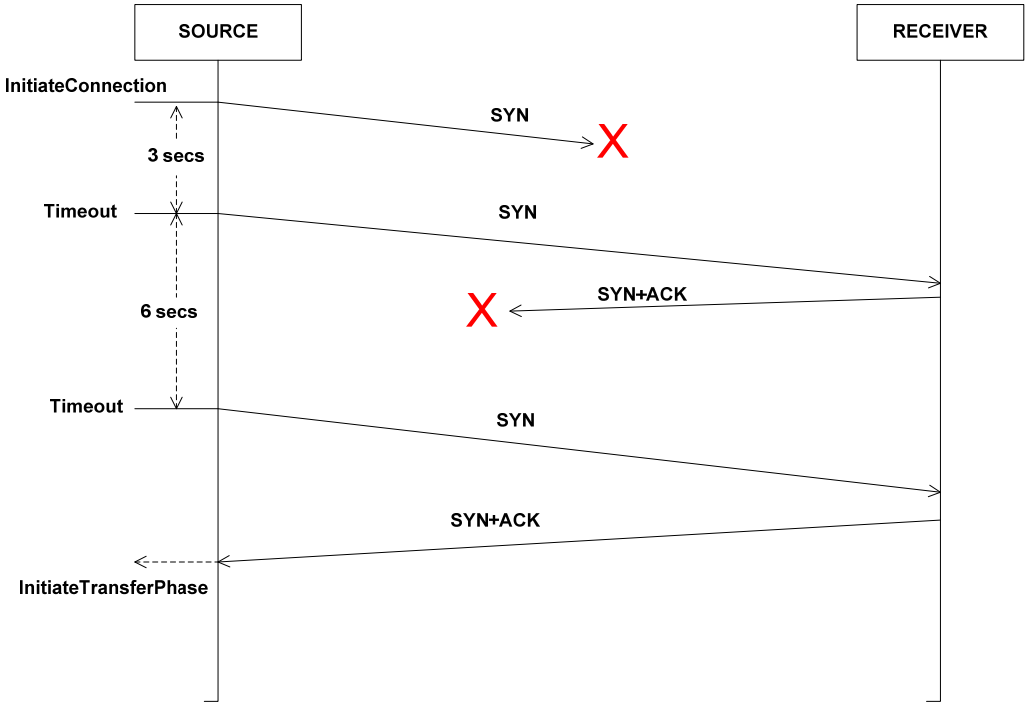


Figure 5-3. TCP Connection Establishment Procedures Leading to Initiation of the Transfer Phase

Our model of connection establishment procedures uses the variables identified and defined in Table 5-1.

Table 5-1. Definition of Symbols Used to Model Connection Establishment Procedures

Symbol	Definition
syn_{INT}	Timeout interval (sec) for initial SYN
syn_{MAX}	Maximum number of SYNs to send
syn_{SENT}	Number of SYNs that have been sent
syn_{TO}	Timeout (sec) for current SYN
time	Current time

Initiation of the connection phase entails the following steps by a source.

$$\textbf{InitiateConnection} \equiv \left| \begin{array}{l} syn_{MAX} \leftarrow 3 \\ syn_{INT} \leftarrow 3\ s \\ syn_{TO} \leftarrow \textbf{time} + syn_{INT} \\ syn_{SENT} \leftarrow 1 \\ \textbf{send}(\text{SYN}) \end{array} \right. \quad (1)$$

Upon a timeout, a source implements the following procedures, which amount to an exponential back-off in the timeout period until the maximum number of SYN segments have been sent.

$$\begin{array}{l}\textbf{Timeout} \equiv \left| \begin{array}{l} \textit{if}\ \ syn_{SENT} < syn_{MAX} \\ \quad \left| \begin{array}{l} syn_{INT} \leftarrow 2 \times syn_{INT} \\ syn_{TO} \leftarrow \textbf{time} + syn_{INT} \\ syn_{SENT} \leftarrow syn_{SENT} + 1 \\ \textbf{send}(\textbf{SYN}) \end{array} \right. \\ \textbf{signal}(\textbf{ConnectionFailure}) \quad \textit{otherwise} \end{array} \right. \end{array} \tag{2}$$

If a SYN+ACK segment is received prior to connection failure, then the source initiates the transfer phase, which is discussed next in two parts: slow start and congestion avoidance.

$$\textbf{SYN} + \textbf{ACKreceived} \equiv \textbf{InitiateTransferPhase} \tag{3}$$

5.1.2 Transfer Phase – Slow Start

During the transfer phase a TCP flow establishes and adjusts a congestion window (*cwnd*) and slow start threshold (*sst*), which requires introducing and defining some additional symbols, as shown in Table 5-2. Our model permits two forms of slow-start: (a) standard TCP slow start or (b) limited slow start [7]. We explain each of these in turn.

Table 5-2. Definition of Symbols Used to Model Slow-Start Procedures

Symbol	Definition
cwnd	Current congestion window in number of packets
$cwnd_{INT}$	Initial congestion window (we use $cwnd_{INT}$ = 2 packets)
sst	Current slow-start threshold in number of packets
sst_{MAX}	Threshold (in packets) to switch from exponential to logarithmic increase (varies with experiment)
sst_{INT}	Threshold (in packets) to terminate initial slow start (varies with experiment)

5.1.2.1 Standard Slow Start. Upon entering slow start, a TCP flow adopts a small value ($cwnd_{INT}$) for the congestion window (*cwnd*). During standard slow start, a flow then increases *cwnd* exponentially as ACKs are received until reaching an initial slow-start threshold (sst_{INT}). After the congestion window reaches sst_{INT} (or upon a loss) the flow enters a congestion avoidance regime. In our model, a flow initiates slow start with the following procedures.

$$\textbf{InitiateTransferPhase} \equiv \left| \begin{array}{l} cwnd \leftarrow cwnd_{INT} \\ sst \leftarrow sst_{INT} \end{array} \right. \quad (4)$$

5.1.2.2 Limited Slow Start. During limited slow start, a flow increases *cwnd* exponentially as ACKs are received until reaching a maximum slow-start threshold (sst_{MAX}). After the congestion window reaches sst_{MAX} the flow increases *cwnd* logarithmically with each ACK received until reaching sst_{INT}. After the congestion window reaches sst_{INT} (or upon a loss) the flow enters a congestion avoidance regime.

Our model distinguishes standard slow start from limited slow start based on the relationship between sst_{MAX} and sst_{INT}. Limited slow-start procedures are used if $sst_{MAX} < sst_{INT}$. Otherwise, standard slow-start procedures are used. These conventions are specified through the combination of configuration parameters for sst_{MAX} and sst_{INT}, initialization procedures (4) and the following procedures upon receiving an ACK.

$$\textbf{ACK} \wedge (cwnd < sst) \equiv \left| \begin{array}{l} cwnd \leftarrow cwnd + 1 \quad if \quad cwnd < sst_{MAX} \\ cwnd \leftarrow cwnd + \dfrac{1}{\left(\dfrac{cwnd}{0.5 \times sst_{MAX}} \right)} \quad otherwise \end{array} \right. \quad (5)$$

5.1.2.3 Setting Slow-Start Threshold. The literature indicates no widespread agreement on what value should be chosen for sst_{INT}. Some authors [6] recommend setting sst_{INT} to an arbitrarily large value, which implies that initial slow start will continue until a flow experiences its first loss or timeout. Other authors [10] recommend setting sst_{INT} to a small value, which means that slow start might terminate before a flow has determined its available bandwidth, so the maximum available bandwidth might not be achieved before the flow terminates (depending on the number of data segments in the flow). Mark Carson (personal communication, November 12, 2008) indicated that Linux sets sst_{INT} selectively based upon properties maintained by the device driver for the network interface. In addition, some [4] suggest using the advertised receiver window (*rwnd*) returned from a receiver to set sst_{INT}. The *rwnd* indicates the number of packets that fit in a receiver's buffer.

Given such varying suggestions, we included sst_{INT} as a configuration parameter of our model. This allows sst_{INT} to be set to large and small values, as desired. Our model does not support setting sst_{INT} variably based on properties of the network interface. Our model does not simulate a receiver's *rwnd*, so setting sst_{INT} based on that value is not supported.

5.1.3 Transfer Phase – Congestion Avoidance

In our model of TCP Reno, congestion avoidance, which begins once $cwnd \geq sst$, increases the congestion window linearly, at the rate of one packet per round-trip time. The increase accrues fractionally as ACKs are received. When the receiver signals a loss, the congestion window is cut in half. Upon a timeout, the slow-start threshold is set to half the congestion window and the congestion window is set to its initial value. Below we specify the procedures used by a source to increase *cwnd* on receipt of each ACK, to decrease *cwnd* upon each signaled loss and to decrease *cwnd* and *sst* at each timeout. (As explained in Sec. 5.2, alternative congestion control procedures replace the standard TCP increase and decrease procedures.)

5.1.3.1 Increase Congestion Window after Acknowledgment. For each ACK received within each round-trip time up until a loss or timeout, a TCP source increases its congestion window by a fraction, using the following procedures.

$$\mathbf{ACK} \equiv cwnd \leftarrow cwnd + \frac{1}{cwnd} \tag{6}$$

An actual TCP implementation will use *cwnd* as part of a decision function to determine its send window (*swnd*). The decision function is *swnd* = **min**(*cwnd, rwnd*). Since our model does not simulate *rwnd*, a source's *swnd* is always equal to its *cwnd*. This means the sending rate of sources in our model will be constrained by network congestion rather than by local policies within receivers.

5.1.3.2 Decrease Congestion Window after Signaled Loss. When a receiver signals a loss within a given round-trip time, a TCP source reduces its *cwnd* by half. In real TCP implementations, a loss is signaled by receiving three consecutive, duplicate ACKs. This convention was designed to accommodate cases where DTs are delivered out of order by the network. Reordering DTs can lead to duplicate ACKs even though a DT was not lost, so a TCP source defers any decision that a DT was lost until three duplicate ACKs arrive in sequence. Modern router vendors strive to ensure that packets are not reordered on a given flow [41-43], but some researchers [38-39, 45] have reported cases where packets are reordered within a router. Our simulation model permits packets to be lost, but not reordered. For this reason, our sources detect explicit losses upon receiving a single duplicate ACK, which we model as a negative acknowledgment (NAK).

Sources in our model reduce a flow's *cwnd* once in a round-trip time when a loss is signaled by a receiver. The reduction rule follows.

$$\mathbf{Loss} \equiv \left| \begin{array}{l} cwnd \leftarrow \frac{cwnd}{2} \\ sst \leftarrow cwnd \end{array} \right. \tag{7}$$

The *sst* is reset to the *cwnd* so that the flow continues in congestion avoidance rather than reentering slow start.

5.1.3.3 Decrease Slow-Start Threshold and Reset Congestion Window after Timeout. A source encounters a timeout when no ACKs or NAKs have been received on a flow for the duration of a retransmission timeout (RTO). The RTO for a flow is maintained to be no less than twice and no greater than 32 times round-trip propagation delay between a source and receiver. Regardless of congestion control mechanism, our model implements a single set of procedures for maintaining and increasing RTO. Upon initiation of a flow's transfer phase, RTO is set to twice the round-trip propagation delay. Upon receipt of an ACK or NAK, a flow's RTO is set to the maximum of 1.5 times the measured, smoothed round-trip time (SRTT) or twice the round-trip propagation delay. With each timeout the RTO is doubled, which leads to an exponential back-off, up to the maximum RTO.

Occurrence of a timeout indicates a significant interruption in the path between a source and receiver. For this reason, our model adopts a conservative strategy in responding to timeouts. The *sst* is reduced using the reduction rules required by the specific congestion control

mechanism in use for the flow. In addition, the *cwnd* is reset to its initial value. This implies that the flow will reenter slow start and then use rapid increase procedures until *cwnd* $\geq$ *sst*. Subsequently, the flow returns to congestion avoidance procedures. Our model for TCP Reno uses the following timeout procedures.

$$\text{Timeout} \equiv \left| \begin{array}{l} sst \leftarrow \max\left(\frac{cwnd}{2}, cwnd_{INT}\right) \\ cwnd \leftarrow cwnd_{INT} \end{array} \right. \tag{8}$$

5.1.3.4 Combined Effects of Slow Start and Congestion Avoidance. To appreciate the combined effects of slow start and congestion avoidance, as implemented in our model of TCP Reno, we consider some schematic graphs of the temporal changes of the *cwnd* for hypothetical flows. Fig. 5-4 depicts changes in *cwnd* assuming the use of standard slow start, with both sst_{MAX} and sst_{INT} set to 128 packets. The *cwnd* increases exponentially in slow start until it reaches sst_{INT}. Subsequently, congestion avoidance commences and the *cwnd* increases linearly. Just after the *cwnd* reaches 150 (time 30 s), a loss occurs and the *cwnd* is reduced to 75. The *cwnd* then increases linearly until it reaches 100 (time 55 s). At about time 63 s the source experiences a timeout and the *cwnd* is reduced to its initial value (2). At the same time *sst* is set to 50 (half the value of the *cwnd* when the timeout occurred). As ACKs resume the *cwnd* increases exponentially (in slow start) to 50 (the value of *sst*) after which the flow returns to congestion avoidance and the *cwnd* increases linearly.

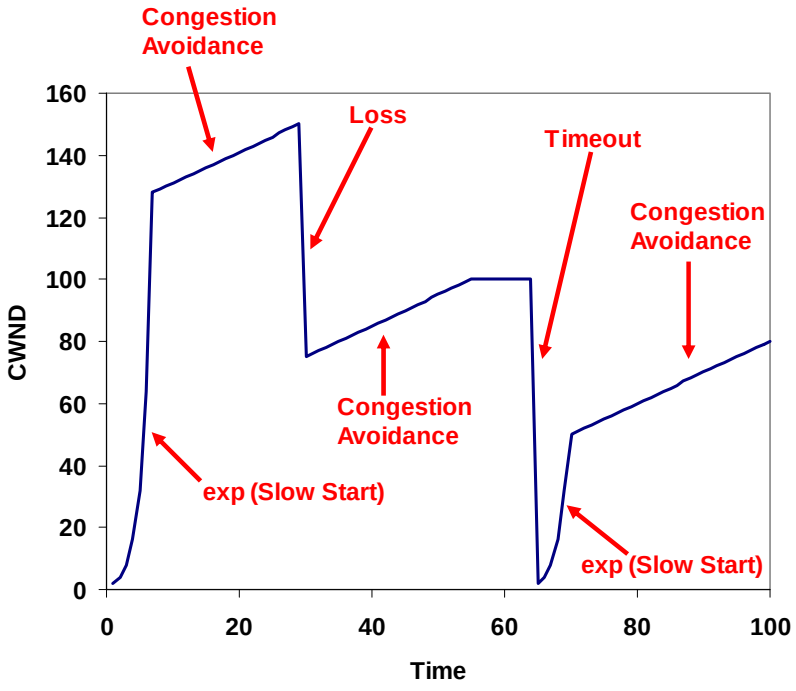


Figure 5-4. Sample Change in Congestion Window (packets) over Time (secs) under Standard Slow Start and Congestion Avoidance

Fig. 5-5 displays the same scenario as Fig. 5-4, with the exception that standard slow start is replaced by limited slow start, with sst_{MAX} = 16 and sst_{INT} = 128. Here, the *cwnd* increases exponentially until reaching 16 and then increases logarithmically until reaching 128. Subsequently, *cwnd* increases linearly in congestion avoidance until a loss occurs, just after the *cwnd* reaches 150 packets (about time 42 s). After the loss, the *cwnd* drops in half (to 75) and then increases linearly until reaching 100. At about time 80 s the source experiences a timeout and the *cwnd* is reduced to 2, while *sst* is reset to 50 (half the value of the *cwnd* when the timeout occurred). When ACKs resume the *cwnd* increases exponentially to 16 and then logarithmically to 50 (the value of *sst*) from which the *cwnd* increases linearly as the flow returns to congestion avoidance.

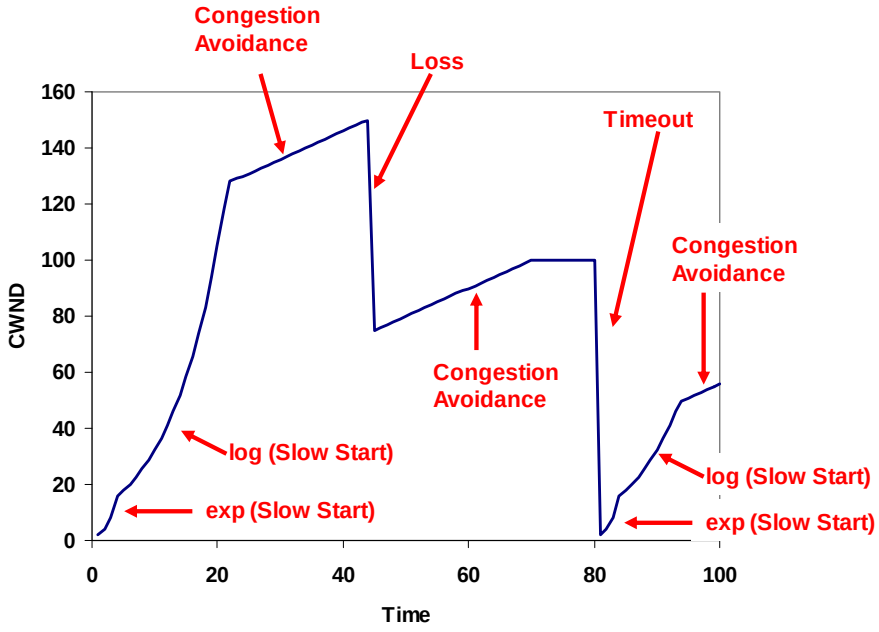


Figure 5-5. Sample Change in Congestion Window (packets) over Time (secs) under Limited Slow Start and Congestion Avoidance

5.2 Congestion Avoidance Procedures for Six Alternate Congestion Control Mechanisms

In this section, we describe congestion avoidance procedures defined by six proposed alternate congestion control mechanisms: Binary Increase Congestion control (BIC) [61], Compound TCP (CTCP) [58], Fast Active Queue Management (AQM) Scalable TCP (FAST) [60], High-Speed TCP (HSTCP) [52], Hamilton TCP (H-TCP) [54] and Scalable TCP [53]. For each congestion control mechanism, we specify increase procedures taken upon receipt of an ACK, decrease procedures used upon explicit notification of a loss and timeout procedures. In addition, three of the congestion control mechanisms (CTCP, FAST and H-TCP) require periodic actions, which we also specify.

All but two (FAST and H-TCP) of the alternate congestion control mechanisms define a threshold, such that when the congestion window is below the threshold then normal TCP congestion avoidance procedures are used. This means that the alternate congestion avoidance procedures will be invoked only when a flow's congestion window surpasses the threshold.

Further, whenever a congestion window passes and then falls below the threshold, normal TCP congestion avoidance procedures will be resumed. Alternate procedures will be reactivated when the congestion window again passes the threshold. H-TCP also uses an activation threshold, but defined in terms of elapsed time since the most recent loss on a flow. FAST does not use a threshold, so the alternate congestion avoidance procedures are always applied for FAST flows. When appropriate, we specify the threshold values associated with each alternate congestion control mechanism.

5.2.1 BIC

The congestion avoidance procedures used by BIC aim to make aggressive increases in the *cwnd* when the current *cwnd* is far from a target and smaller increases as the current *cwnd* nears the target. BIC determines the target by conducting a binary search within some range around the current *cwnd*. When the target falls beyond the search range, BIC increases the *cwnd* additively by a fixed increment and then reinitiates the binary search within the new range. Implementing this behavior requires rather complex logic, so BIC procedures for congestion avoidance tend to be somewhat elaborate. The resulting *cwnd* evolution for BIC reflects its complexity – reproducing a function that appears to change in a pattern resembling a human heartbeat.

Table 5-3. Symbols and Definitions Used to Model BIC Congestion Avoidance Procedures

Symbol	Definition
β_B	Multiplicative back-off factor for loss (BIC parameter β = 0.8)
B_B	Parameter for computing increase in congestion window (BIC parameter B = 4)
Δ_B	Candidate numerator for increase in congestion window
LW_B	Low-window threshold (LW_B = 14 packets) for applying BIC procedures
MIN_B	Variable to track minimum value for computing BIC target window
MAX_B	Variable to track maximum value for computing BIC target window
$PREV_B$	Variable to track previous MAX_B
σ_B	Parameter for computing increase in congestion window (BIC parameter σ = 20)
$SMAX_B$	Threshold to begin rapid increase in congestion window (BIC parameter S_{max} = 32)
SS_B	Boolean indicating whether the flow is in BIC slow start (**true**) or not (**false**)
SST_B	BIC slow-start target
SSW_B	BIC slow-start congestion window
TGT_B	BIC target window (BIC variable w_1)

Specifying BIC congestion avoidance procedures requires some additional symbols, as identified and defined in Table 5-3. Where applicable, the table denotes parameter values used within our model. We adopt parameter values matching default values reported in the empirical study by Li, Leith and Shorten [67] of prototype implementations for congestion control mechanisms.

A given BIC flow uses normal TCP congestion avoidance procedures or the alternate BIC procedures, defined below, depending on the relationship between *cwnd* size and a low-window parameter (LW_B = 14 packets).

$$\textbf{SelectProcedures} \equiv \begin{cases} \textbf{TCPcongestionAvoidance} & \textit{if } cwnd < LW_B \\ \textbf{BICcongestionAvoidance} & \textit{otherwise} \end{cases}$$

5.2.1.1 Increase Procedures. The window increase procedures (9) used by BIC differ depending upon whether the current *cwnd* is below or beyond a previously determined maximum *cwnd*. As the *cwnd* passes the previously determined maximum, BIC invokes slow-start procedures (these differ from TCP slow start) that increase the *cwnd* until the *cwnd* approaches a new maximum (set to twice the previous maximum). As the *cwnd* nears the new maximum, BIC slow-start procedures are abandoned and the *cwnd* is increased using a binary search. Once the new maximum is exceeded, then BIC reenters its slow-start procedures. Thus, during congestion avoidance, BIC increase procedures alternate between BIC slow-start and binary search.

$$\textbf{ACK} \equiv \begin{cases} \textit{if } SS_B = \textbf{false} \\ \quad \Delta_B \leftarrow \dfrac{(TGT_B - cwnd)}{B_B} \\ \quad cwnd \leftarrow cwnd + \dfrac{f_\alpha(\Delta_B, cwnd, TGT_B)}{cwnd} \\ \quad \textit{if } (cwnd < MAX_B) \\ \quad\quad MIN_B \leftarrow cwnd \\ \quad\quad TGT_B \leftarrow \dfrac{(MAX_B + MIN_B)}{2} \\ \quad \textit{otherwise} \\ \quad\quad SS_B \leftarrow \textbf{true} \\ \quad\quad MAX_B \leftarrow cwnd \times 2 \\ \quad\quad SSW_B \leftarrow 1 \\ \quad\quad SST_B \leftarrow cwnd + 1 \\ \textit{otherwise} \\ \quad cwnd \leftarrow cwnd + \dfrac{SSW_B}{cwnd} \\ \quad \textit{if } cwnd \geq SST_B \\ \quad\quad SSW_B \leftarrow SSW_B \times \dfrac{B_B}{(B_B - 1)} \\ \quad\quad SST_B \leftarrow cwnd + SSW_B \\ \quad \textit{if } SSW_B \geq MAX_B \\ \quad\quad SS_B \leftarrow \textbf{false} \\ \quad\quad TGT_B \leftarrow \dfrac{(MAX_B + MIN_B)}{2} \end{cases} \tag{9}$$

with

$$f_\alpha(\Delta_B, cwnd, TGT_B) \equiv \begin{cases} \frac{B_B}{\sigma_B} & \text{if } (\Delta_B < 1 \wedge cwnd < TGT_B) \vee (TGT_B \le cwnd < TGT_B + B_B) \\ \Delta_B & \text{if } (1 < \Delta_B < SMAX_B) \wedge cwnd < TGT_B \\ \frac{TGT_B}{(B_B - 1)} & \text{if } B_B < cwnd - TGT_B < SMAX_B \times (B_B - 1) \\ SMAX_B & \text{otherwise} \end{cases} \quad (10)$$

During the BIC binary search, the *cwnd* is increased more quickly when further from the current midpoint and less quickly as it nears the midpoint. The rules controlling the increase pattern are encoded within a function (10).

5.2.1.2 Decrease Procedures. Upon notification of a loss (11), BIC sets the maximum *cwnd* for the binary search to the current *cwnd* and then multiplicatively decreases the congestion window. If the loss followed closely behind a previous loss, then BIC also multiplicatively decreases the maximum *cwnd* used for the binary search. In addition, BIC abandons its slow-start procedures to ensure a new binary search commences within the reduced range.

$$\textbf{Loss} \equiv \begin{cases} MAX_B \leftarrow \frac{(1+\beta_B)}{2} \times cwnd & \text{if } cwnd < PREV_B \\ MAX_B \leftarrow cwnd & \text{otherwise} \\ PREV_B \leftarrow MAX_B & \\ TGT_B \leftarrow 0 & \\ SS_B \leftarrow \textbf{false} & \\ cwnd \leftarrow \beta_B \times cwnd & \\ sst \leftarrow cwnd & \end{cases} \quad (11)$$

5.2.1.3 Timeout Procedures. For BIC timeout procedures (12) we adopt logic similar to that used for a loss, except that the *sst* is set to half the *cwnd* and the *cwnd* is set to its initial value ($cwnd_{INT}$). This ensures that standard (or limited) slow-start is used until the flow's *cwnd* reaches the new *sst*. After that, BIC congestion avoidance procedures resume.

$$\textbf{Timeout} \equiv \begin{cases} MAX_B \leftarrow \frac{(1+\beta_B)}{2} \times cwnd & \text{if } cwnd < PREV_B \\ MAX_B \leftarrow cwnd & \text{otherwise} \\ PREV_B \leftarrow MAX_B & \\ TGT_B \leftarrow 0 & \\ SS_B \leftarrow \textbf{false} & \\ sst \leftarrow \textbf{max}\left(\frac{cwnd}{2}, cwnd_{INT}\right) & \\ cwnd \leftarrow cwnd_{INT} & \end{cases} \quad (12)$$

5.2.2 CTCP

Compound TCP, or CTCP, augments the congestion window with a second component, called the delay window (*dwnd*). (See Table 5-4 for a complete listing of symbols and parameter settings used to specify CTCP behavior.) The *dwnd* is added to the *cwnd* to establish the actual send window used for CTCP flows. CTCP defines rules for increasing *dwnd* aggressively when a flow is underutilizing the available transmission rate and also defines rules for reducing *dwnd* as a flow's transmission rate nears the available bandwidth. Upon detection of congestion, either through explicit losses or timeouts, CTCP reduces the delay window toward zero.

Table 5-4. Symbols and Definitions Used to Model CTCP Congestion Avoidance Procedures

Symbol	Definition
α_C	Window increase (α_C = 0.125) weight for CTCP
A_C	Actual throughput ($cwnd/SRTT_C$) experienced on CTCP flow
β_C	Window decrease (β_C = 0.5) weight for CTCP
CD_C	Boolean denoting whether early congestion has been detected (**true**) or not (**false**)
γ_C	CTCP gamma threshold (γ_C = 30) for detecting early congestion
D_C	Difference between expected and actual throughput experienced on CTCP flow
dwnd	CTCP delay window
E_C	Expected throughput ($cwnd/minRTT_C$) on CTCP flow
ζ_C	CTCP zeta parameter (ζ_C = 0.1) defining reduction speed in delay window
k_C	Exponent (k_C = 0.8) for CTCP window-increase procedures
LW_C	Low-window threshold (LW_C = 41) for applying CTCP procedures
$minRTT_C$	Minimum round-trip time experienced on CTCP flow
$SRTT_C$	Average Smoothed Round-Trip Time experienced on CTCP flow

CTCP procedures update the *dwnd* periodically, typically once per round-trip time. As a flow's transmission rate nears equilibrium around some estimated available bandwidth, CTCP tends to cause the send window to oscillate by exponentially increasing the *dwnd* when the estimated number of packets queued for a flow falls below a threshold (γ_C = 30) and then linearly decreasing *dwnd* when the estimated number of queued packets exceeds the threshold. On the other hand, when the transmission rate is increasing on a flow, CTCP exponentially increases the *dwnd* without exerting a countervailing linear decrease. Consequently, the CTCP send window can reach a large size relatively quickly when a transmission path exhibits no congestion.

As shown below (13), CTCP uses normal TCP congestion avoidance procedures for adjusting the *cwnd* whenever the *cwnd* is below a threshold (LW_C = 41 packets). This means that the *dwnd* is used only when the *cwnd* is sufficiently large.

$$\textbf{SelectProcedures} \equiv \begin{cases} \textbf{TCPcongestionAvoidance} & \textit{if } cwnd < LW_C \\ \textbf{CTCPcongestionAvoidance} & \textit{otherwise} \end{cases} \tag{13}$$

5.2.2.1 Increase Procedures. CTCP increases (14) the *cwnd* fractionally with each ACK received in a round-trip time without a loss. The increased *cwnd* is then added to the current *dwnd*. As with other congestion avoidance procedures, CTCP suspends increases after a loss in a round-trip time until an ACK arrives associated with a subsequent round-trip. The increase procedures adopted by CTCP consider both the *cwnd* and the *dwnd*, as follows.

$$\textbf{ACK} \equiv \begin{cases} cwnd \leftarrow cwnd + \dfrac{1}{(cwnd + dwnd)} \\ cwnd \leftarrow cwnd + dwnd \end{cases} \tag{14}$$

5.2.2.2 Decrease Procedures. Upon a loss, CTCP decreases the *cwnd* by half and then notes that congestion was detected. As with other congestion avoidance procedures, the *sst* is reset to the *cwnd* in order to ensure the flow remains in congestion avoidance. During the next periodic update cycle, CTCP will act on the loss notification by reducing the *dwnd*. Equation (15) specifies the precise decrease procedures used by CTCP upon an explicit loss.

$$\textbf{Loss} \equiv \begin{cases} cwnd \leftarrow \dfrac{cwnd}{2} + dwnd \\ CD_C \leftarrow \textbf{true} \\ sst \leftarrow cwnd \end{cases} \tag{15}$$

5.2.2.3 Timeout Procedures. Given a timeout, CTCP adopts the same procedures used by TCP and then augments those procedures by resetting *dwnd* to zero and noting that congestion was detected. The precise procedures follow.

$$\textbf{Timeout} \equiv \begin{cases} sst \leftarrow max\left(\dfrac{cwnd}{2}, cwnd_{INT}\right) \\ cwnd \leftarrow cwnd_{INT} \\ dwnd \leftarrow \textbf{0} \\ CD_C \leftarrow \textbf{true} \end{cases} \tag{16}$$

5.2.2.4 Periodic Procedures. The remaining CTCP procedures are used periodically, every round-trip time, to update the *dwnd*. The update procedures (17) depend upon a number of parameters. We adopt the recommended settings for those parameters, as shown in Table 5-4.

$$\begin{aligned}
\textbf{every}(SRTT_C) \equiv \;& E_C \leftarrow \frac{cwnd}{minRTT_C} \\
& A_C \leftarrow \frac{cwnd}{SRTT_C} \\
& D_C \leftarrow (E_C - A_C) \times minRTT_C \\
& \textit{if}\;\; CD_C = \textbf{true} \\
& \quad dwnd \leftarrow \textbf{min}\left[\textbf{0}, cwnd \times (\textbf{1} - \beta_C) - \frac{cwnd}{\textbf{2}}\right] \\
& \quad CD_C \leftarrow \textbf{false} \\
& dwnd \leftarrow dwnd + \textbf{min}\left(\textbf{0}, \alpha_C \times cwnd^{k_C} - \textbf{1}\right) \;\; \textit{if}\;\; CD_C = \textbf{false} \wedge D_C < \gamma_C \\
& dwnd \leftarrow \textbf{min}[\textbf{0}, dwnd - (\zeta_C \times D_C)] \;\; \textit{otherwise} \\
& cwnd \leftarrow \textbf{max}(\textbf{int_max}, cwnd + dwnd)
\end{aligned} \tag{17}$$

5.2.3 FAST

FAST TCP adopts a fundamentally different approach from the other congestion control mechanisms considered in this study. First, FAST aims to achieve an equilibrium *cwnd* that does not change during the life of a flow, while other congestion control mechanisms lead to an oscillating *cwnd*. Second, FAST updates the *cwnd* based mainly on measured changes in queuing delay, using loss signals only when congestion prevents reaching a lossless equilibrium. Third, FAST does not resort to standard TCP congestion avoidance procedures; instead, FAST uses its own procedures at all times during congestion avoidance. FAST adopts these approaches based on the idea that queuing delay can be measured quite frequently and thus accurately, while packet losses are rare events that provide insufficient information to estimate loss probability on a given flow.

Explaining FAST congestion avoidance procedures requires numerous parameters and variables, listed and defined in Table 5-5. In addition to procedures associated with *cwnd* increase on ACKs and decrease on losses and timeouts, FAST requires a periodic procedure to determine a target *cwnd* ($Tcwnd_F$). FAST also defines optional, periodic procedures for tuning a parameter (α_F), which determines how many packets a flow attempts to keep queued between a source and receiver. These optional, α-tuning procedures require two periodic processes: one to estimate flow throughput and one to adjust α_F based on changes in flow throughput.

5.2.3.1 Increase Procedures. FAST uses periodic procedures (explained below in Sec. 5.2.3.4) to determine a target congestion window ($Tcwnd_F$) and then increases or decreases *cwnd* as needed to reach $Tcwnd_F$. FAST does not move *cwnd* to $Tcwnd_F$ in one step, but instead paces the rate of increase to reflect that expected number of ACKs arriving on a given flow within each round-trip time. Our model uses the following procedures (18) for adjusting *cwnd* with each arriving ACK.

$$\begin{aligned}
\textbf{ACK} \equiv \;& cwnd \leftarrow \frac{Tcwnd_F - cwnd}{acksRTT_F} \;\; \textit{if} \;\; Tcwnd_F > cwnd \wedge acksRTT_F > \textbf{0} \\
& cwnd \leftarrow Tcwnd_F \;\; \textit{if} \;\; Tcwnd_F < cwnd
\end{aligned} \tag{18}$$

5.2.3.2 Decrease Procedures. Upon an explicit loss for FAST, we reduce (19) *cwnd* by half and assign the reduced value to both the $Tcwnd_F$ and the *sst*. These actions provide a new, lower

basis from which FAST can begin increasing the *cwnd* and also ensure that the flow remains in congestion avoidance.

$$\textbf{Loss} \equiv \left| \begin{array}{l} cwnd \leftarrow \frac{cwnd}{2} \\ Tcwnd_F \leftarrow cwnd \\ sst \leftarrow cwnd \end{array} \right. \quad (19)$$

Table 5-5. Symbols and Definitions Used to Model FAST Congestion Avoidance Procedures

Symbol	Definition
$acksRTT_F$	Count of ACKs received on FAST flow during the most recent $SRTT_F$
α_F	Current α parameter
AD_F	Default α parameter setting (AD_F = 200) when α-tuning disabled
AT_F	Boolean indicating whether α-tuning is enabled (**true**) or disabled (**false**)
$A1_F$	First α parameter setting ($A1_F$ = 8 packets)
$A2_F$	Second α parameter setting ($A2_F$ = 20 packets)
$A3_F$	Third α parameter setting ($A3_F$ = 200 packets)
Bk_F	Current average throughput on FAST flow
γ_F	Weight (γ_F = 0.5) of recent information when updating $Tcwnd_F$
$minRTT_F$	Minimum round-trip time experienced on FAST flow
$M0M1_F$	Set $\alpha = A2_F$ when $\alpha = A1_F$ and throughput passes $M0M1_F$ (= 1500 ppms)
$M1M0_F$	Set $\alpha = A1_F$ when $\alpha = A2_F$ and throughput passes $M1M0_F$ (= 1250 ppms)
$M1M2_F$	Set $\alpha = A3_F$ when $\alpha = A2_F$ and throughput passes $M1M2_F$ (= 15,000 ppms)
$M2M1_F$	Set $\alpha = A2_F$ when $\alpha = A3_F$ and throughput passes $M2M1_F$ (= 12,500 ppms)
$SRTT_F$	Average Smoothed Round-Trip Time experienced on FAST flow
T_F	Weight (T_F = 0.5) to assign to most recent throughput sample when computing Bk_F
$Tcwnd_F$	Current target congestion window
UA_F	Periodicity (UA_H = 200 s) for updating α parameter when α-tuning enabled
UW_F	Periodicity (UW_H = 20 ms) for updating $Tcwnd_F$

5.2.3.3 Timeout Procedures. For a FAST timeout, we adopt procedures (20) analogous to those used with other congestion control mechanisms. We set the *sst* to half the *cwnd* and then set *cwnd* and $Tcwnd_F$ to the initial congestion window ($cwnd_{INT}$) and recommence slow start.

$$\textbf{Timeout} \equiv \left| \begin{array}{l} sst \leftarrow \max\left(\frac{cwnd}{2}, cwnd_{INT}\right) \\ cwnd \leftarrow cwnd_{INT} \\ Tcwnd_F \leftarrow cwnd \end{array} \right. \tag{20}$$

5.2.3.4 Periodic Procedures. FAST defines one mandatory periodic process (21) to update the target congestion window ($Tcwnd_F$) for the flow every UW_F (= 20 ms, here). A parameter (γ_F) determines how much weight is placed on the previous *cwnd* and how much weight is given to recent information. FAST procedures prevent the new target *cwnd* from being more than twice the current *cwnd*.

$$\textbf{every}(UW_F) \equiv Tcwnd_F \leftarrow \min\left[2 \times cwnd, (1-\gamma_F) \times cwnd + \gamma_F \times \left(\frac{minRTT_F}{SRTT_F} \times cwnd + \alpha_F\right)\right] \tag{21}$$

The α_F parameter may be fixed or tuned. If α-tuning is enabled, the following procedures (22) are executed every UA_F (= 200 s, here).

$$\textbf{every}(UA_F) \;\; \textit{if} \;\; AT_F = \textbf{true} \equiv \left| \begin{array}{l} \alpha_F \leftarrow A2_F \;\; \textit{if} \;\; Bk_F \geq M0M1_F \wedge \alpha_F = A1_F \\ \alpha_F \leftarrow A1_F \;\; \textit{if} \;\; Bk_F \leq M1M0_F \wedge \alpha_F = A2_F \\ \alpha_F \leftarrow A3_F \;\; \textit{if} \;\; Bk_F \geq M1M2_F \wedge \alpha_F = A2_F \\ \alpha_F \leftarrow A2_F \;\; \textit{if} \;\; Bk_F \leq M2M1_F \wedge \alpha_F = A3_F \end{array} \right. \tag{22}$$

The various parameters associated with α-tuning are defined in Table 5-5. Estimated flow throughput (Bk_F) is a variable used in the α-tuning procedures. To estimate Bk_F, the following procedures (23) are used each round-trip time.

$$\textbf{every}(SRTT_F) \equiv \left| \begin{array}{l} Bk_F \leftarrow T_F \times Bk_F + (1-T_F) \times \frac{acksRTT_F}{SRTT_F} \\ acksRTT_F \leftarrow 0 \end{array} \right. \tag{23}$$

5.2.4 HSTCP

High Speed TCP (HSTCP) modifies standard TCP congestion control procedures in order to achieve high transmission rates (e.g., 10 Gbps) when network conditions permit, while maintaining comparable performance to standard TCP when a network path exhibits moderate to heavy congestion. HSTCP retains the fundamental additive-increase and multiplicative-decrease (AIMD) strategy adopted by standard TCP, but HSTCP alters the AIMD parameters to become a function of congestion window size. The altered AIMD functions result in more aggressive increases and less aggressive decreases at larger window sizes. Below a low-window threshold (LW_{HS}) HSTCP adopts standard TCP congestion-avoidance procedures.

$$\textbf{SelectProcedures} \equiv \left| \begin{array}{l} \textbf{TCPcongestionAvoidance} \;\; \textit{if} \;\; cwnd < LW_{HS} \\ \textbf{HSTCPcongestionAvoidance} \;\; \textit{otherwise} \end{array} \right. \tag{24}$$

Table 5-6 identifies and defines symbols used below when explaining HSTCP congestion-avoidance procedures.

Table 5-6. Symbols and Definitions Used to Model HSTCP Congestion Avoidance Procedures

Symbol	Definition
HW_{HS}	High-window threshold (HW_{HS} = 83,000) for HSTCP procedures
LW_{HS}	Low-window threshold (LW_{HS} = 31) for applying HSTCP procedures
R_{HS}	Decrease congestion window by this percentage (R_{HS} = 0.1) after loss above LW_{HS}

5.2.4.1 Increase Procedures. HSTCP increases the *cwnd* additively upon receiving each ACK in a round-trip time until a loss is detected. The increase procedures (25) appear quite similar to standard TCP increase procedures, except that the numerator for the increase is a function of *cwnd* size.

$$\text{ACK} \equiv cwnd \leftarrow cwnd + \frac{f_\alpha(cwnd)}{cwnd} \tag{25}$$

Function $f_\alpha(c)$, defined below (26), returns the increase numerator that will yield the desired packet drop rate for a given window *c*. Function $f_\alpha(c)$ uses a subsidiary function, $g_\beta(c)$, defined below (27). Function $g_\beta(c)$ is also used to determine the multiplicative-decrease parameter applied on losses and timeouts.

$$f_\alpha(c) \equiv c^2 \times \frac{0.078}{c^{1.2}} \times 2 \times \frac{g_\beta(c)}{2 - g_\beta(c)} + 0.5 \tag{26}$$

$$g_\beta(c) \equiv (R_{HS} - 0.5) \times \frac{\log_{10}(c) - \log_{10}(LW_{HS})}{\log_{10}(HW_{HS}) - \log_{10}(LW_{HS})} + 0.5 \tag{27}$$

5.2.4.2 Decrease Procedures. Upon detecting an explicit loss, HSTCP reduces the *cwnd* by a multiplicative factor that is a function of the *cwnd*. The specific procedures, which use function $g_\beta(c)$, are given below.

$$\text{Loss} \equiv \begin{cases} cwnd \leftarrow \left[1 - g_\beta(cwnd)\right] \times cwnd \\ sst \leftarrow cwnd \end{cases} \tag{28}$$

5.2.4.3 Timeout Procedures. For a timeout (29), HSTCP sets *sst* to the reduced *cwnd* and then resets the *cwnd* to its initial value. This enables slow-start procedures up to the new *sst*, after which congestion avoidance resumes.

$$\textbf{Timeout} \equiv \begin{vmatrix} cwnd \leftarrow \left[1 - g_{\beta}(cwnd)\right] \times cwnd \\ sst \leftarrow \textbf{max}\left(cwnd, cwnd_{INT}\right) \\ cwnd \leftarrow cwnd_{INT} \end{vmatrix} \quad (29)$$

5.2.5 H-TCP

H-TCP differs from other congestion avoidance procedures in two main aspects. First, H-TCP determines the numerator of the *cwnd* increase as a function of elapsed time since the most recent packet loss. The increase is scaled by the round-trip time experienced on a path in order to compensate for differences in feedback delay. The motive is to give larger increases in *cwnd* during periods of low network congestion, so a flow could reach higher transmission rates more quickly on uncongested, high-bandwidth, long-delay paths. H-TCP adopts standard TCP increase procedures for a specified time after each loss. Second, H-TCP implements an adaptive back-off procedure to determine the multiplicative decrease in *cwnd* after a loss. The back-off factor is varied based on estimating the queuing delay on a path. The motive is to prevent senders from backing off too much after packet losses. H-TCP adopts standard TCP decrease procedures when flow throughput has changed by more than a specified amount since the most recent loss. To monitor changes in flow throughput, H-TCP requires a periodic process to measure average throughput. Table 5-7 identifies and defines parameters and variables used below to explain H-TCP congestion avoidance procedures.

Table 5-7. Symbols and Definitions Used to Model H-TCP Congestion Avoidance Procedures

Symbol	Definition
$acksRTT_H$	Count of ACKs received on H-TCP flow during the most recent U_H
β_H	Most recent computed percentage (initially $\beta_H = 0.5$) *cwnd* residual on a loss
$Bk1_H$	Average throughput on H-TCP flow at time of most recent loss
Bk_H	Current average throughput on H-TCP flow
Δ_H	Time elapsed since the most recent loss
ΔB_H	Percentage throughput increase ($\Delta B_H = 0.2$) for selecting
ΔL_H	Use normal TCP procedures until $\Delta_H > \Delta L_H$, where $\Delta L_H = 1$ s
G_H	Maximum percentage ($G_H = 0.8$) *cwnd* reduction on a loss
$maxRTT_H$	Maximum round-trip time experienced on H-TCP flow
$minRTT_H$	Minimum round-trip time experienced on H-TCP flow
T_H	Weight ($T_H = 0.5$) to assign to most recent throughput sample when computing Bk_H
U_H	Periodicity ($U_H = 250$ ms) for updating throughput estimate for H-TCP flow

5.2.5.1 Increase Procedures. For each ACK received without a loss in a round-trip time, H-TCP increases *cwnd* by a fraction of the *cwnd*. The numerator of the increase fraction is a function (30) of the most recently computed multiplicative-decrease parameter (β_H) and the elapsed time (Δ_H) since the most recent loss.

$$\text{ACK} \equiv cwnd \leftarrow cwnd + \frac{2 \times (1 - \beta_H) \times f_\alpha(\Delta_H)}{cwnd} \tag{30}$$

Function $f_\alpha(\Delta_H)$ returns (31) one if standard TCP increase procedures are in effect; otherwise it returns a value exhibiting a quadratic increase with increasing Δ_H. The quadratic increase (32) is scaled by the minimum round-trip time measured on the flow.

$$f_\alpha(\Delta_H) \equiv \begin{cases} 1 & if \quad \Delta_H \le \Delta L_H \\ \max\left[h_\alpha(\Delta_H) \cdot minRTT_H, 1\right] & otherwise \end{cases} \tag{31}$$

$$h_\alpha(\Delta_H) \equiv 1 + 10 \times (\Delta_H - \Delta L_H) + 0.25 \times (\Delta_H - \Delta L_H)^2 \tag{32}$$

5.2.5.2 Decrease Procedures. Upon an explicit loss, H-TCP reduces (33) the *cwnd* by a fraction, computed as a function (34) of changing throughput. In addition, H-TCP records the average flow throughput at the time of the loss.

$$\text{Loss} \equiv \begin{cases} \beta_H \leftarrow g_\beta\left(\left|\frac{Bk_H - Bk1_H}{Bk1_H}\right|\right) \\ cwnd \leftarrow cwnd \times \beta_H \\ Bk1_H \leftarrow Bk_H \end{cases} \tag{33}$$

Given default parameters, the H-TCP algorithm varies the back-off fraction between 0.5 and 0.8. The lower value (larger reduction) is adopted whenever measured throughput (Bk_H) has changed by a significant percentage (ΔB_H) since the most recent loss, which suggests that the flow is undergoing some disturbance or transition. Less significant change in throughput indicates that the flow is nearer to stability. Stable flows are reduced by a fraction reflecting the estimated queuing delay as a proportion of estimated propagation delay. The residual *cwnd* is capped by parameter G_H (= 0.8).

$$g_\beta(B) \equiv \begin{cases} 0.5 & if \quad B > \Delta B_H \\ \min\left(\frac{minRTT_H}{maxRTT_H}, G_H\right) & otherwise \end{cases} \tag{34}$$

5.2.5.3 Timeout Procedures. For a timeout, we adopt procedures (35) that mirror the rules H-TCP uses for an explicit loss with significant change in flow throughput. This amounts to reducing the *sst* to half the *cwnd*, recording the flow's average throughput and setting β_H = 0.5. We also reset the *cwnd* to its initial value, which reinitiates slow start.

$$\text{Timeout} \equiv \begin{cases} \beta_H \leftarrow 0.5 \\ Bk1_H \leftarrow Bk_H \\ sst \leftarrow \max\left(\frac{cwnd}{2}, cwnd_{INT}\right) \\ cwnd \leftarrow cwnd_{INT} \end{cases} \tag{35}$$

5.2.5.4 Periodic Procedures. To support recording and monitoring of flow throughput, H-TCP requires a periodic process (36) to estimate average throughput. In our model, estimated throughput is updated every U_H (= 250 ms, here).

$$\textbf{every}(U_H) \equiv \left| \begin{array}{l} Bk_H \leftarrow T_H \times \dfrac{acksRTT_H}{U_H} + (1 - T_H) \times Bk_H \\ acksRTT_H \end{array} \right. \quad (36)$$

5.2.6 SCALABLE TCP

Scalable TCP adopts a simple, fixed-increase rule aimed at allowing a flow to increase its congestion window more quickly than would be the case with standard TCP. In addition, Scalable TCP defines a decrease rule that limits a flow to a fixed multiplicative decrease that is recommended to be much less than the 50% decrease used by standard TCP. The Scalable TCP rules are defined in an additive-increase, multiplicative-decrease (AIMD) form, but the rules actually amount to a multiplicative-increase, multiplicative-decrease (MIMD) regime. Researchers have found [1] that MIMD algorithms are not guaranteed to converge to fair bandwidth sharing in drop-tail networks, such as the Internet. Empirical measurements by Li, Leith and Shorten [67] have also shown that failure to converge is a property of Scalable TCP. Below, we describe Scalable TCP procedures for increase on ACK, decrease on explicit loss and decrease on timeout. Our description uses the symbols and definitions shown in Table 5-8.

Table 5-8. Symbols and Definitions Used to Model Scalable TCP Congestion Avoidance Procedures

Symbol	Definition
α_S	Increase (α_S = 0.01) applied by Scalable TCP on each ACK
β_S	Percentage residual *cwnd* (β_S = 0.875) applied by Scalable TCP on each loss
LW_S	Low-window threshold (LW_S = 16 packets) for applying Scalable TCP procedures

Scalable TCP includes (37) a low-window threshold (LW_S) that ensures standard TCP procedures for congestion avoidance are followed when the *cwnd* is small. Scalable TCP congestion avoidance procedures are used only when the *cwnd* is above the threshold.

$$\textbf{SelectProcedures} \equiv \left| \begin{array}{ll} \textbf{TCPcongestionAvoidance} & \textit{if } cwnd < LW_S \\ \textbf{ScalableTCPcongestionAvoidance} & \textit{otherwise} \end{array} \right. \quad (37)$$

5.2.6.1 Increase Procedures. Upon receiving each ACK within a round-trip time without a congestion signal Scalable TCP increases (38) the *cwnd* by a fixed value α_S (= 0.01).

$$\textbf{ACK} \equiv cwnd \leftarrow cwnd + \alpha_S \quad (38)$$

5.2.6.2 Decrease Procedures. Upon receiving an explicit loss notification, Scalable TCP reduces (39) the *cwnd* by a fixed percentage. Here, the reduction amounts to (1 - β_S =) 0.125. We also set *sst* to the new, lower *cwnd* to ensure the flow remains in congestion avoidance.

$$\textbf{Loss} \equiv \left| \begin{array}{l} cwnd \leftarrow cwnd \times \beta_S \\ sst \leftarrow cwnd \end{array} \right. \tag{39}$$

5.2.6.3 Timeout Procedures. For a timeout we define procedures (40) that require Scalable TCP to set the *sst* to the reduced *cwnd* and then reset the *cwnd* to its initial value. This means that, like the other congestion control mechanisms we model, Scalable TCP will reenter slow start until the *cwnd* passes the *sst* and then return to congestion avoidance.

$$\textbf{Timeout} \equiv \left| \begin{array}{l} cwnd \leftarrow cwnd \times \beta_S \\ sst \leftarrow \textbf{max}(cwnd, cwnd_{INT}) \\ cwnd \leftarrow cwnd_{INT} \end{array} \right. \tag{40}$$

5.3 Modeling the Transfer Phase in MesoNet

The section explains the key ideas underlying our model for the transfer phase of a flow. We begin by explaining how our model simulates data transfer procedures in general and then concentrate separately on slow start and congestion avoidance. Previously in Sec. 5.2, we explained the detailed slow-start and congestion avoidance procedures for individual congestion control mechanisms. Here, we focus on the common approach used by MesoNet to model the transfer phase across all congestion control mechanisms.

5.3.1 General Data Transfer Procedures

We adopt a simplified model of data transfer procedures in order to simulate fundamental aspects of congestion control without incurring the detailed complexity of TCP implementations. A simplified model permits simulating reasonably large, fast networks for suitable time durations on standard computing hardware without incurring excessive costs in processing time and memory use. Our simplified model retains key properties that enable us to compare and contrast various congestion control mechanisms under a wide range of network conditions.

During the data transfer phase for each flow, a simulated source transfers a randomly selected number (*flowDTs*) of DT segments. Each DT segment is assigned a sequence number; the first segment is number one and the sequence number increases by one for each subsequent segment. A flow's receiver, then, expects to receive DT segments in sequence, where each segment has a sequence number one greater than the most recently received segment. When the sequence number is as expected, the receiver sends an ACK back to the source. When the sequence number is higher than expected, the receiver sends a NAK back to the source. The ACK or NAK is numbered with the next expected sequence number. This simplification, which ignores the possibility for reordered segments, is feasible because MesoNet allows packets to be discarded but does not permit packet reordering. Absent reordering, our model of data transfer procedures may omit features such as duplicate ACKs and selective ACKs.

A source in our model expects to receive a stream of ACKs and NAKs from a flow's receiver. Since a receiver sends an ACK or NAK only upon receiving a DT segment, a source can simply count each received ACK and NAK as evidence that one DT segment has been delivered successfully to the receiver. When the source has received one ACK or NAK for each segment comprising the flow, then the data transfer is finished and the source can terminate the flow. Of course, each NAK received by a source also causes the number of DT segments sent on the flow to increase by one (a retransmission) because the NAK indicates that one DT segment was lost and, thus, was not counted as successfully delivered. Receiving a NAK also stimulates a source to take remedial action. In our model, receipt of a NAK activates a source's loss procedures on a flow.

ACK and NAK segments may also be lost on a flow. This means that each lost ACK or NAK will not be counted by the source. For this reason, each lost ACK and NAK will also increase by one (a retransmission) the number of DT segments that must be sent by the source in order to receive a sufficient number of ACKs and NAKs. Further, if no ACKs or NAKs are received by a source for a retransmission timeout (RTO) period, then a source must also take remedial action. In our model, expiration of a RTO activates a source's timeout procedures on a flow.

Whenever a NAK is received or a timeout occurs, a source notes the next sequence number that it intends to send on the flow. This enables the source to ignore window increase and decrease procedures for all subsequent ACKs and NAKs that arrive with lower sequence numbers. This technique ensures that window increase procedures are abandoned in a round-trip time after a loss or timeout. The technique also ensures that window decrease procedures are activated only once within a round-trip time.

The remaining elements of our general data transfer model concern controlling the ability of a source to transmit a DT. A source maintains a flow *cwnd* using the procedures described earlier (Sec. 5.2). A source also knows the sequence number (*nextSeq*) for the next DT segment it intends to send and the highest sequence number (*highSeq*) received in an ACK or NAK. With this information, a source can compute (41) the number of unacknowledged DT segments (*unAckedDTs*) and thus the number of DT segments it is permitted to send (*unsentDTs*).

$$\begin{aligned} unAckedDTs &\leftarrow nextSeq - highSeq \\ unsentDTs &\leftarrow cwnd - unAckedDTs \end{aligned} \tag{41}$$

Equation 41 reveals the self-clocking nature of TCP flows. Two conditions enable a source to send a DT segment: arrival of an ACK or NAK or increase in the congestion window. In our model, a timeout causes *highSeq* to be set to *nextSeq*, which means that *unsentDTs* will equal *cwnd*. Recall that we also reset *cwnd* to its initial value upon a timeout.

One last detail must be explained. The procedures in equation 41 can cause extra DTs to be sent at the end of a flow. To prevent this, our model computes (42) the difference (*residualDTs*) between the number of DTs (*flowDTs*) comprising a flow and unacknowledged DTs (*unAckedDTs*). The number of DTs that can be sent (*unsentDTs*) is then set to the minimum of the segments allowed by the congestion window or the segments required to complete the flow.

$$\begin{aligned} residualDTs &\leftarrow flowDTs - unAckedDTs \\ unsentDTs &\leftarrow \mathbf{min}(unsentDTs, residualDTs) \end{aligned} \tag{42}$$

Data transfer procedures are distributed across elements of MesoNet. Sources manage sending of DT segments and also react to flow timeouts. Access routers process incoming ACKs and NAKs on behalf of sources and also process incoming DTs on behalf of receivers. When processing incoming ACKs and NAKs an access router updates the *cwnd* as required by the congestion control mechanism in use on the related flow. When processing incoming DTs an access router determines whether a receiver needs to send an ACK or NAK and then queues that assignment for the receiver.

5.3.2 Slow Start

Slow-start procedures are invoked within a simulated access router upon receipt of an ACK whenever the *cwnd* is below the *sst*. If the *cwnd* is below sst_{MAX}, then standard slow-start procedures are used to increase the *cwnd* at an exponential rate; otherwise, limited slow-start procedures are used to increase the *cwnd* at a logarithmic pace.

5.3.3 Congestion Avoidance

Congestion avoidance procedures are invoked within a simulated access router upon receipt of any qualified NAK, and for qualified ACKs where the *cwnd* equals or exceeds the *sst*. Selected congestion control mechanisms also require periodic procedures, which our model implements within simulated access routers. We implement timeout procedures within simulated sources.

5.3.3.1 Acknowledgement Procedures. MesoNet assigns one congestion control mechanism to each simulated source, which represents a computer attached to the network and running a particular version of TCP. This means that the particular congestion control mechanism in operation on a simulated flow will be determined by the congestion control mechanism used by the flow's source. Upon receipt of a qualified ACK a simulated access router selects the appropriate window increase procedures for the flow as a function of the congestion control mechanism (*tcpType*) used by the simulated source. Qualified ACKs include all ACKs received within a round-trip time prior to a congestion signal.

5.3.3.2 Negative Acknowledgement Procedures. Upon receipt of a qualified NAK a simulated access router selects the appropriate window decrease procedures for the flow as a function of the *tcpType* used by the simulated source. Qualified NAKs include the first NAK received within any given round-trip time for a flow.

5.3.3.3 Periodic Procedures. In general, MesoNet activates periodic procedures only after a flow passes initial slow start. Periodic procedures that estimate throughput are always active during a flow's transfer phase. MesoNet implements periodic procedures in a somewhat approximate form. Specifically, periodic procedures are invoked within a simulated access router only when an ACK or NAK has been received and provided that sufficient time has elapsed. Further, the timer is reset only after invoking the related procedures. Thus, MesoNet does not invoke periodic procedures on a precisely rigid schedule, as might be stimulated by a timer. Periodic procedures can be invoked regardless of whether an ACK or NAK is qualified to stimulate increase or decrease procedures for a flow.

5.3.3.4 Timeout Procedures. MesoNet invokes timeout procedures within a simulated source when a flow's RTO expires. A source's RTO is reset within a simulated access router whenever

any ACK or NAK arrives for the source. Upon expiration of the RTO, a source selects the appropriate timeout procedures for the flow as a function of the source's *tcpType*.

5.4 Verifying Simulated Congestion Control Mechanisms

To verify the behavior of congestion control mechanisms simulated within MesoNet, we defined a test configuration similar to that used in the Li, Leith and Shorten study [67] of congestion control mechanisms implemented in Linux. We also adopted parameters used in that study and then simulated similar scenarios and recorded temporal changes in the congestion window. Below, we give our simulated *cwnd* graphs and compare the behavior of our simulated congestion control mechanisms to findings reported by Li, Leith and Shorten. First, we describe the test configuration adopted to produce the reported *cwnd* graphs.

We defined a dumbbell topology, shown in Fig. 5-6, similar to the dumbbell topology used by Li, Leith and Shorten. The topology in Fig. 5-6 is annotated with key parameter values used to generate the results presented below. The topology consists of two sources that attach to the same access router. Each source can transmit DTs to one of a pair of receivers that attach to the second access router in the topology. Li, Leith and Shorten place a *dummynet* router between the sources and receivers and use that router to control propagation delay, bottleneck speed and buffer provisioning on the network path between the sources and receivers. Our simulations use MesoNet facilities to control path characteristics.

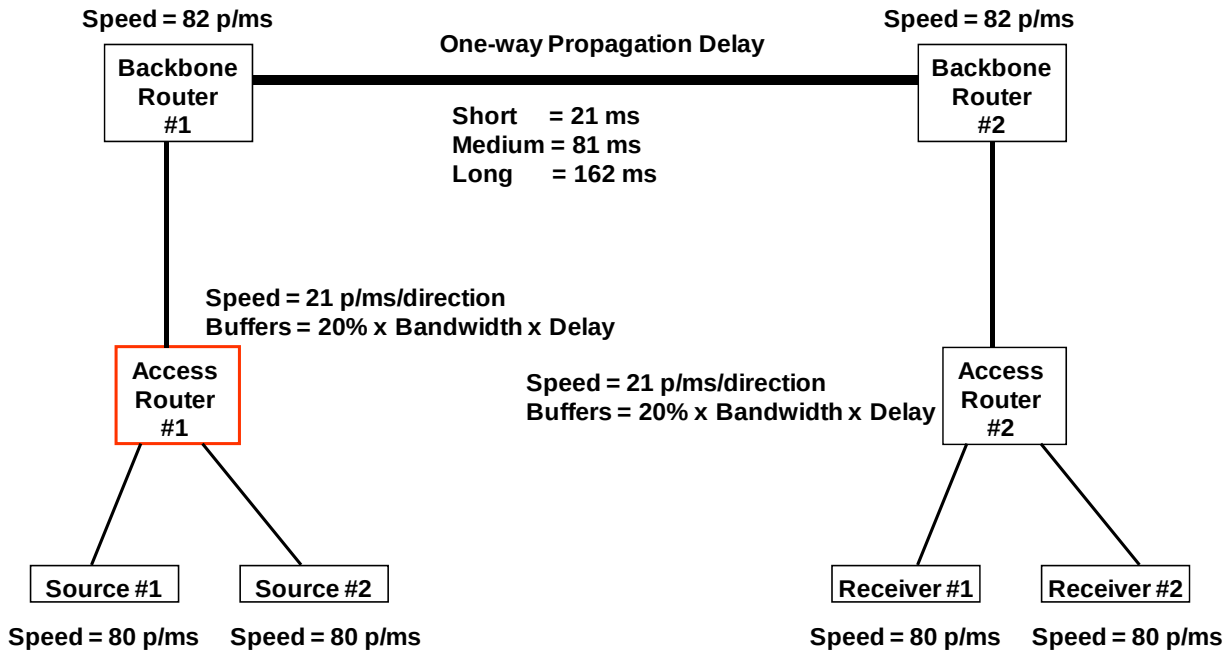


Figure 5-6. Simulated Dumbbell Topology for MesoNet Verification Experiments

Access Router #1, highlighted in red in Fig. 5-6, simulates the bottleneck bandwidth for the path. Here, the bottleneck speed is set to 21 p/ms (packets/millisecond), which amounts to 21 p/ms x 1000 ms/sec x 12000 bits/packet = 252 million bits per second (Mbps), assuming 1500-byte packets. This approximates a 250 Mbps bottleneck link used in the empirical study. Note that the sources, receivers and backbone routers are configured with speeds exceeding the

bottleneck access routers. The sources and receivers are capable of transmitting at 960 Mbps, which is close to 1 billion bits per second (Gbps). Similarly the backbone routers can transmit at 984 Mbps.

The propagation delay of the network path in Fig. 5-6 is controlled by the one-way propagation delay of the backbone link. Round-trip propagation delay will be twice the one-way propagation delay. For our simulations with the dumbbell topology we used three different one-way delays (21 ms, 81 ms and 162 ms) to match the short (42 ms), medium (162 ms) and long (324 ms) round-trip propagation delays used by Li, Leith and Shorten.

We configured MesoNet to provision buffers for each router sufficient to accommodate the bandwidth-delay product. MesoNet also includes a parameter that can adjust the number of provisioned buffers. Here, we reduce the buffers to be 20 % of the number required by the bandwidth-delay product. This matches the buffer provisioning used for several scenarios reported in the study by Li, Leith and Shorten.

Given the topology and parameters from Fig. 5-6, we simulated congestion control mechanisms under a scenario lasting 1000 s, where one source begins sending data immediately and the second source delays (250 s) and then starts to send data. For each congestion control mechanism we use limited slow-start, with sst_{MAX} = 100 and sst_{INT} = 2^{32}/2.We repeat the scenario three times for each congestion control mechanism, varying the round-trip propagation delay (*rtt*) from short, to medium, to long with each repetition. We record and graph (on the y axis) the time-varying *cwnd* (in packets) for each scenario. The maximum value (in packets) of the y axis on each graph scales with *rtt*: 1200 at 42 ms, 4500 for 162 ms and 9000 for 324 ms. The x axis in all graphs is denominated in 100 ms units. All graphs also include the average overall *cwnd* when one flow is transmitting and when both flows are transmitting. When both flows are transmitting, each graph also displays the average *cwnd* for each flow.

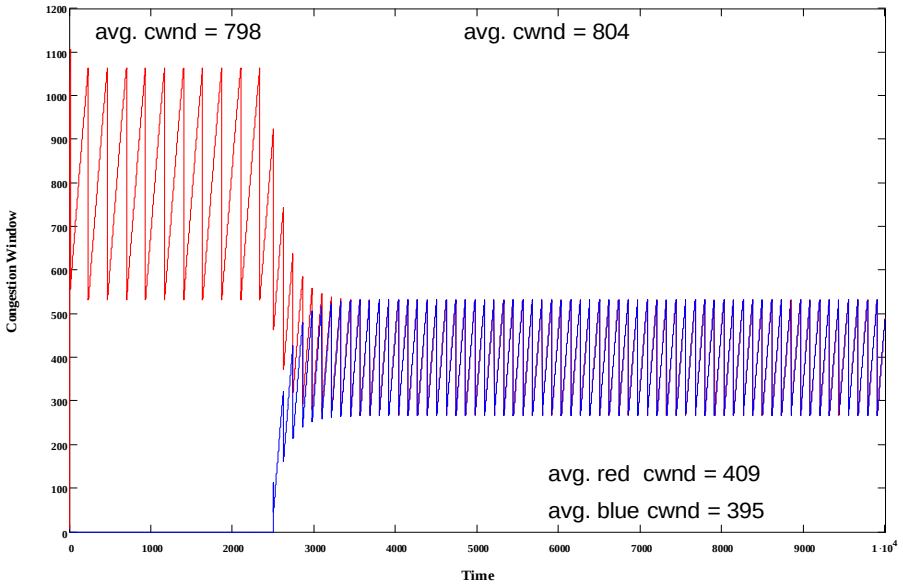


Figure 5-7. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two TCP Flows (*rtt* = 42 ms)

5.4.1 Standard TCP Congestion Control Model

Fig. 5-7 graphs *cwnd* evolution for standard TCP congestion control under a short propagation delay. The graph shows the expected behavior of the TCP *cwnd*, which (after initial slow start

ends with a loss just after the *cwnd* passes 1100) oscillates in a saw-tooth pattern between a *cwnd* of 550 and 1050 (average about 800). The bandwidth-delay product is (42 ms x 21p/ms =) 882 packets, so an average *cwnd* of 800 seems appropriate. After the second flow begins (at time 250 s) it takes between 50 and 100 s for the *cwnd* of the two flows to converge to a similar value (average around 400 packets). Convergence to a similar *cwnd* means the two flows will receive fairly equal average throughputs. This property of convergence to fairness is a hallmark trait of TCP congestion control.

The next scenario, displayed in Fig. 5-8, begins to show why many researchers believe standard TCP congestion control procedures are ill-suited to high-speed, long-delay environments. Here, the 162 ms round-trip propagation delay (*rtt*) suggests a *cwnd* of (162 ms x 21p/ms =) 3402 packets. The first flow reaches (and then exceeds) that value during slow start, which ends with a loss (at *cwnd* = 4200) early in the flow. After the loss, TCP reduces the *cwnd* in half (to 2100) and then TCP enters its congestion avoidance regime. Increasing the *cwnd* with standard TCP congestion avoidance procedures requires about 150 s for the flow to reach its peak window. Thus, absent other activities, the single flow would oscillate in throughput over periods of about 150 s.

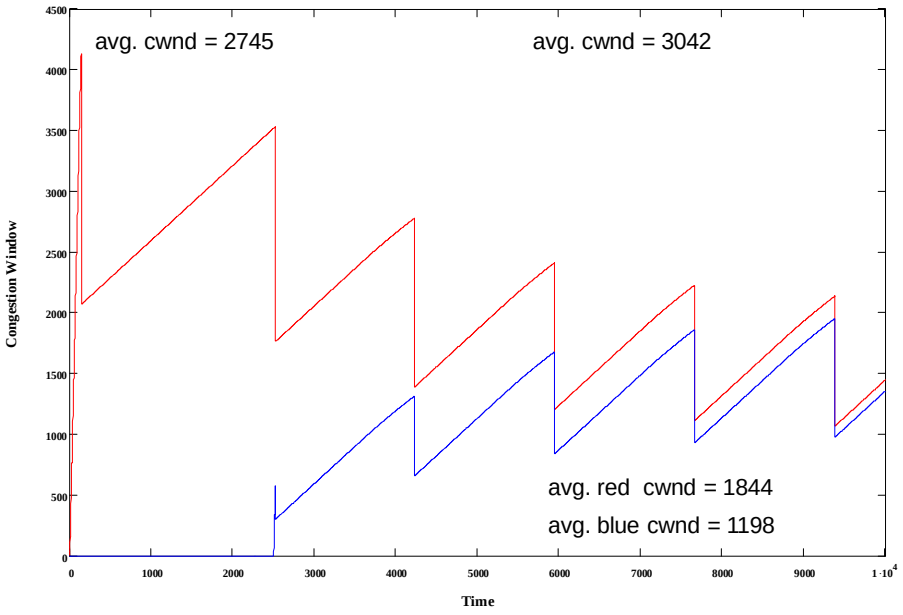


Figure 5-8. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two TCP Flows (*rtt* = 162 ms)

Once the second flow commences, the *cwnd* of the two flows begin to converge; however, the lengthy propagation delay slows the increase in *cwnd* and the rate of convergence. In fact, the two flows in Fig. 5-8 have not fully converged even after 750 s. The situation becomes worse when *rtt* becomes even longer, as shown in Fig. 5-9. Further, increasing the network speed would increase the bandwidth-delay product and worsen the delay in recovering from packet losses and converging to fair throughputs.

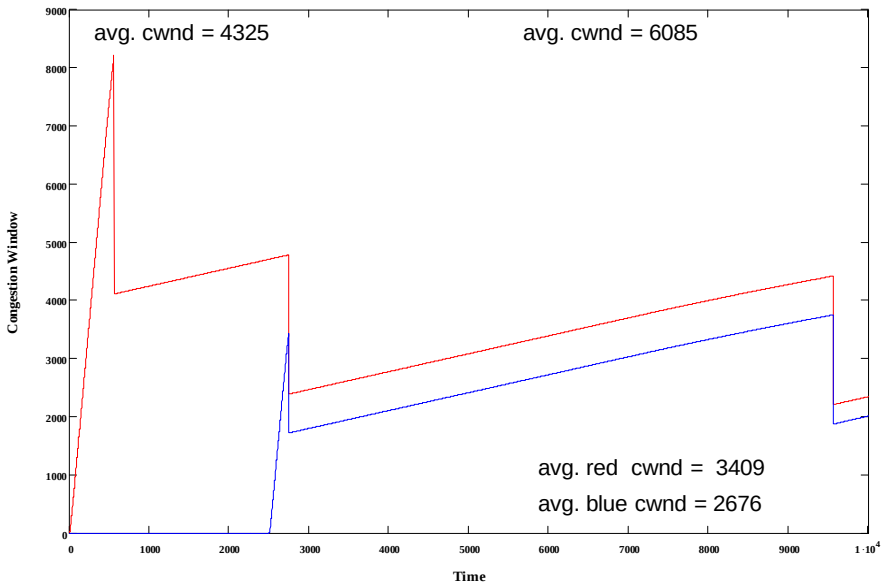


Figure 5-9. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two TCP Flows (*rtt* = 324 ms)

5.4.2 Behavior of BIC Congestion Control Model

Next, we subject BIC congestion control to the same three scenarios under which we simulated standard TCP. The resulting *cwnd* evolutions are shown in Figs. 5-10 through 5-12. The graphs display the heartbeat-like pattern of BIC *cwnd* evolution, as seen in the empirical study by Li, Leith and Shorten. Note that BIC congestion avoidance shows small improvement in convergence time for scenarios with short and medium propagation delays. At the long propagation delay, BIC exhibits significantly less fairness in bandwidth allocation than standard TCP. These findings are consistent with findings by Li, Leith and Shorten.

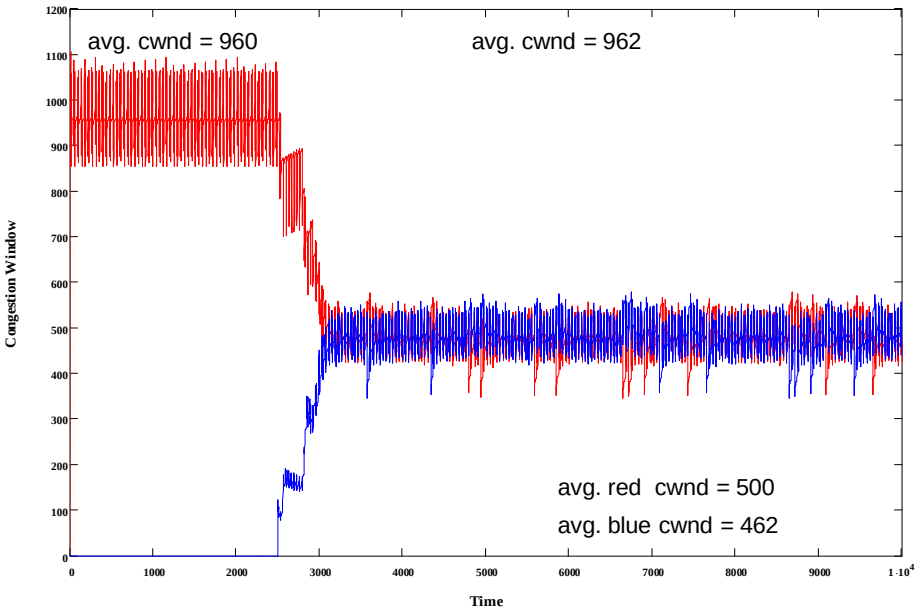


Figure 5-10. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two BIC Flows (*rtt* = 42 ms)

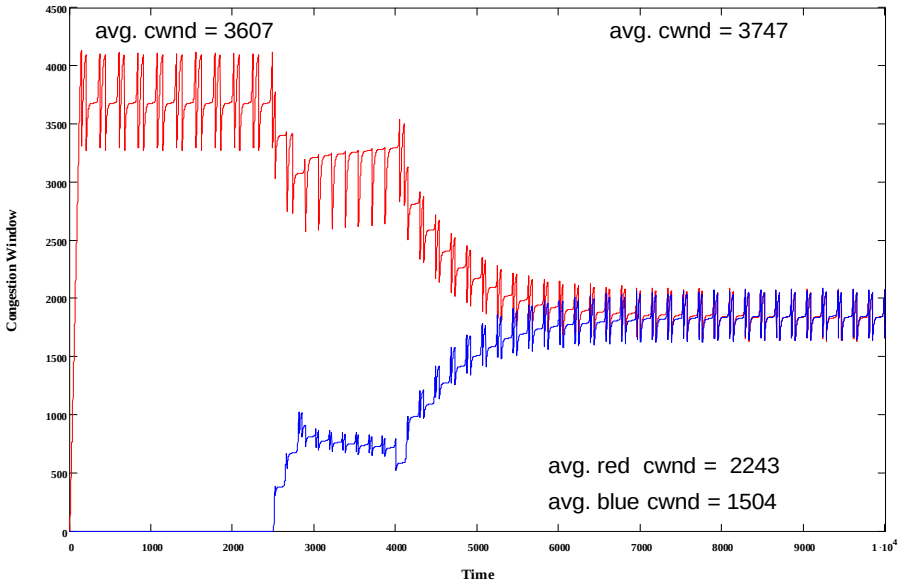


Figure 5-11. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two BIC Flows (*rtt* = 162 ms)

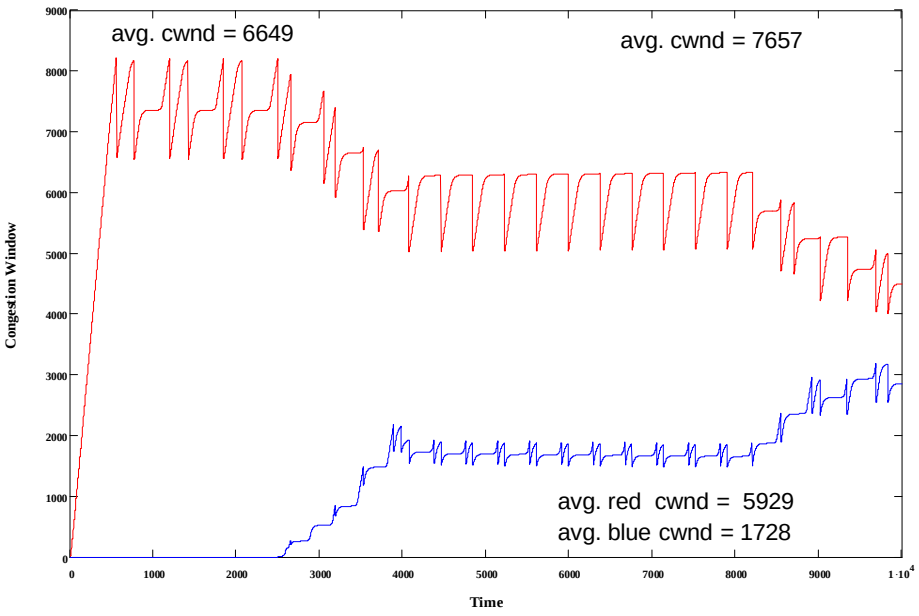


Figure 5-12. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two BIC Flows (*rtt* = 324 ms)

5.4.3 Behavior of CTCP Congestion Control Model

The empirical study by Li, Leith and Shorten did not include CTCP, so in verifying the behavior of CTCP we must compare our simulations to results from a later empirical study by Leith, Andrew, Quetchenbach, Shorten and Lavi [66]. Unfortunately, the later study did not use the same parameters and scenarios used by Li, Leith and Shorten. For that reason, comparing our CTCP simulation results to empirical results is not quite as direct as for the other congestion control mechanisms. We can compare the pattern of *cwnd* evolutions between the simulations

and the empirical results of Leith, Andrew, Quetchenbach, Shorten and Lavi and we can compare their CTCP-related findings with the findings from our simulation.

Figs. 5-13 through 5-15 show *cwnd* evolution for CTCP under our scenario with short, medium and long *rtt*. CTCP exhibits a distinctive pattern of *cwnd* evolution, which becomes evident once the second flow starts in Fig. 5-13 and 5-14. This pattern is also evident in one of the CTCP *cwnd* graphs shown by Leith, Andrew, Quetchenbach, Shorten and Lavi. They also report that the time taken by CTCP to recover from a loss, as well as the convergence time when a second flow begins, is similar to standard TCP. Further, they find that convergence time scales linearly with bandwidth-delay product. The MesoNet simulation of CTCP exhibits the same properties, as shown in the *cwnd* graphs below.

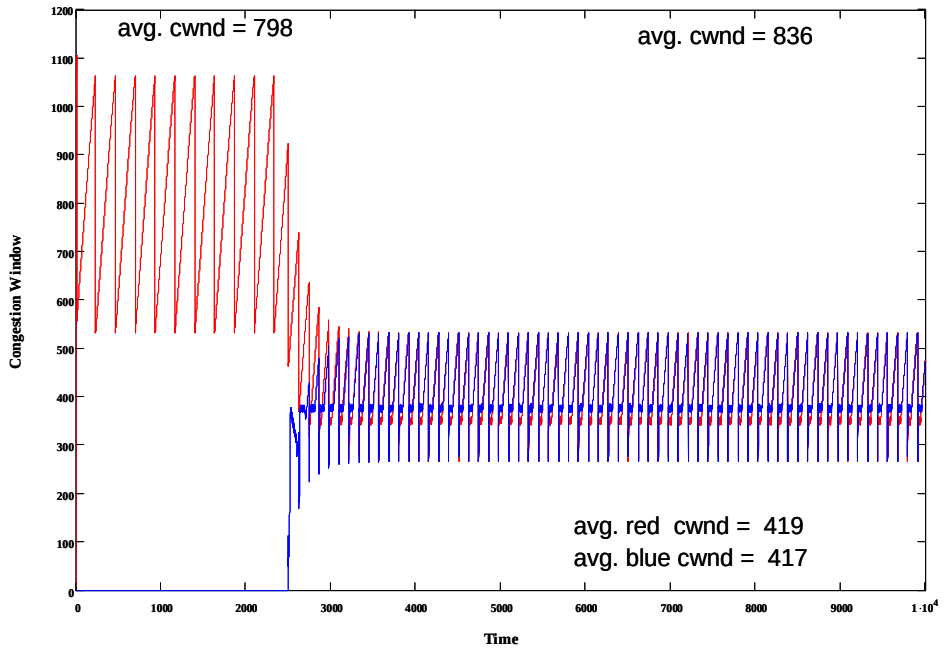


Figure 5-13. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two CTCP Flows (*rtt* = 42 ms)

Leith, Andrew, Quetchenbach, Shorten and Lavi report that CTCP exhibits similar *rtt* fairness to TCP Reno, but when buffers are smaller CTCP has slightly better *rtt* fairness. MesoNet simulations also show a similar fairness between CTCP and standard TCP, but CTCP had a slight edge in *rtt* fairness for the case of medium propagation delay (*rtt* = 162 ms). Leith, Andrew, Quetchenbach, Shorten and Lavi find that link utilizations can be low and responsiveness can be sluggish for CTCP. The potential sluggishness of CTCP responsiveness is also evident in Figs. 5-14 and 5-15. In Sec. 5.4.8, we report more about fairness, as well as link and buffer utilization, among all congestion control mechanisms that we simulated.

5.4.4 Behavior of FAST Congestion Control Model

The FAST congestion control algorithm includes α–tuning as an option, which complicates the verification of the FAST simulation within MesoNet. Li, Leith and Shorten [67] report results for FAST with α–tuning enabled. The designers of FAST indicate [60] that α–tuning is no longer used routinely within FAST implementations. Instead, the designers suggest fixing α_F to a value suitable for expected network conditions. Of course, the designers recognize that fixing α_F is not

a general solution and list α–tuning as an open issue. In some empirical studies, the designers of FAST set α_F = 200. We report simulation results for FAST under three different configurations: α–tuning enabled (Figs. 5-16 through 5-18), α_F = 80 (Figs. 5-19 through 5-21) and α_F = 200 (Figs. 5-22 through 5-24).

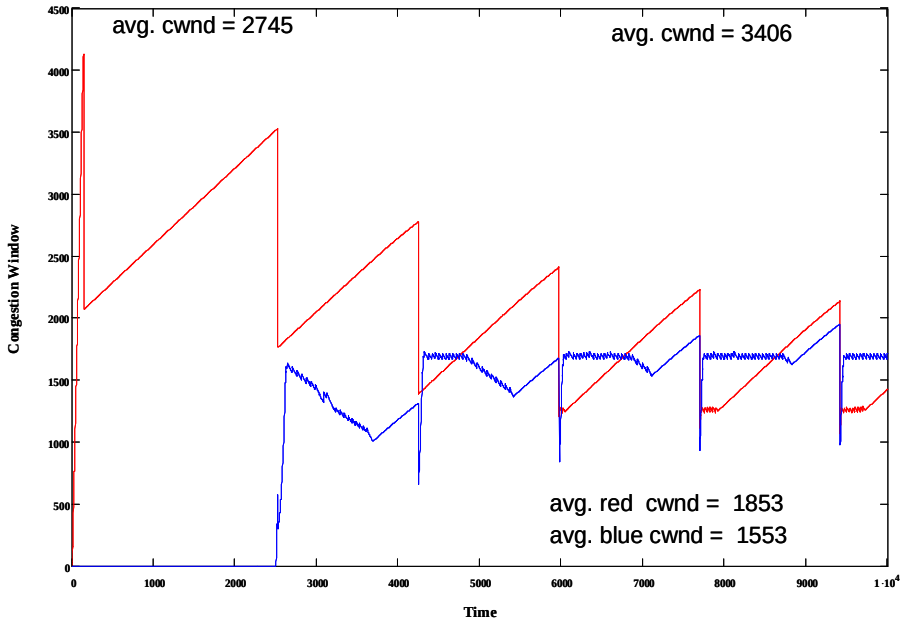


Figure 5-14. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two CTCP Flows (*rtt* = 162 ms)

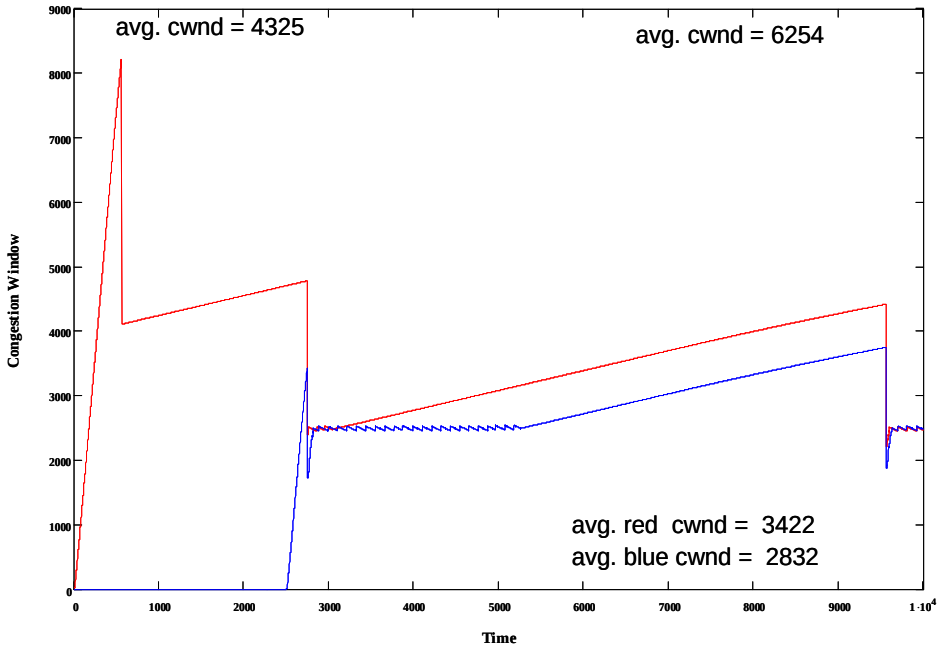


Figure 5-15. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two CTCP Flows (*rtt* = 324 ms)

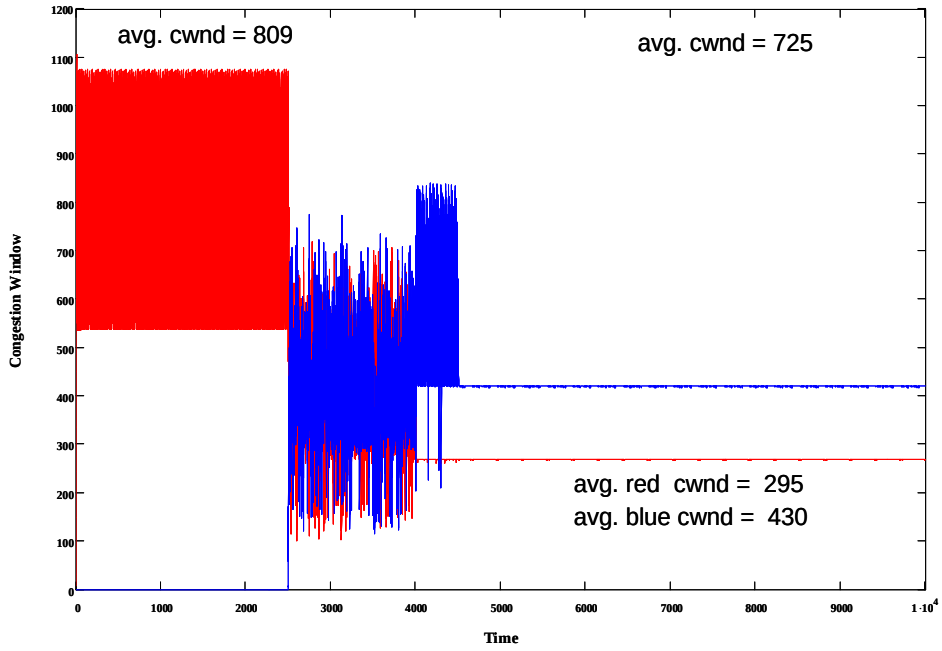


Figure 5-16. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (*α*–tuning enabled, *rtt* = 42 ms)

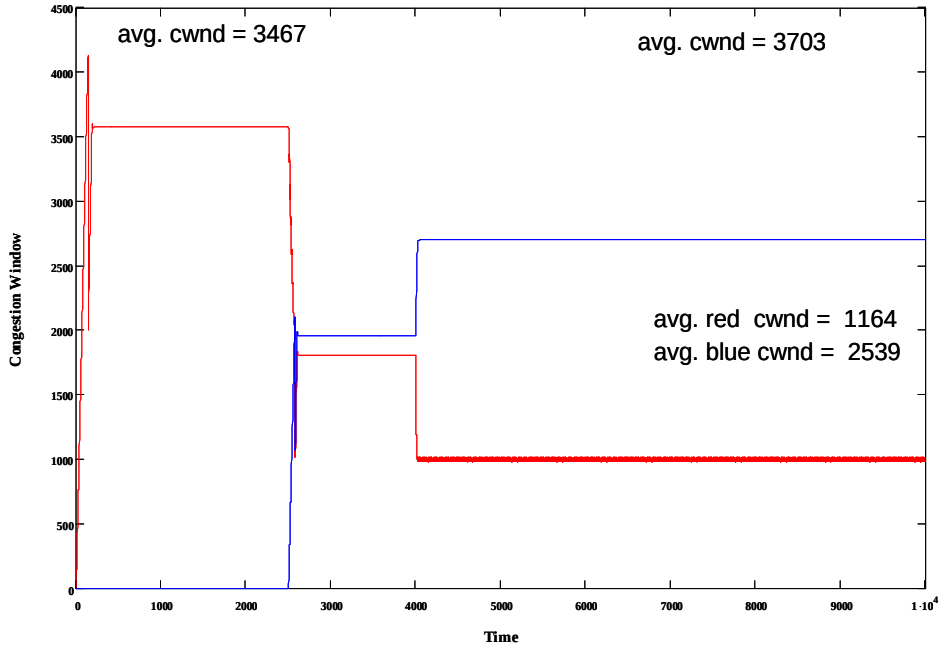


Figure 5-17. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (*α*–tuning enabled, *rtt* = 162 ms)

Li, Leith and Shorten report two main findings regarding FAST with *α*–tuning enabled. First, FAST can converge to fair bandwidth allocation and then diverge to unfair allocation. The MesoNet simulation shows this trait in Figs. 5-16 to 5-18. Second, when the network path has insufficient buffers to sustain $\alpha_F/2$ queued packets per flow, then *cwnd* oscillates as FAST floods the buffers with too many packets, leading to substantial packet losses. The FAST designers

report this same tendency to oscillate when buffers are insufficient. The MesoNet simulation shows this oscillatory behavior in Fig. 5-16, for each flow prior to reaching equilibrium, which becomes possible once α–tuning reduces α_F from its initial value (200) to 20. Fig. 5-22 also shows this oscillatory behavior for α_F = 200, which prevents either flow from ever achieving equilibrium.

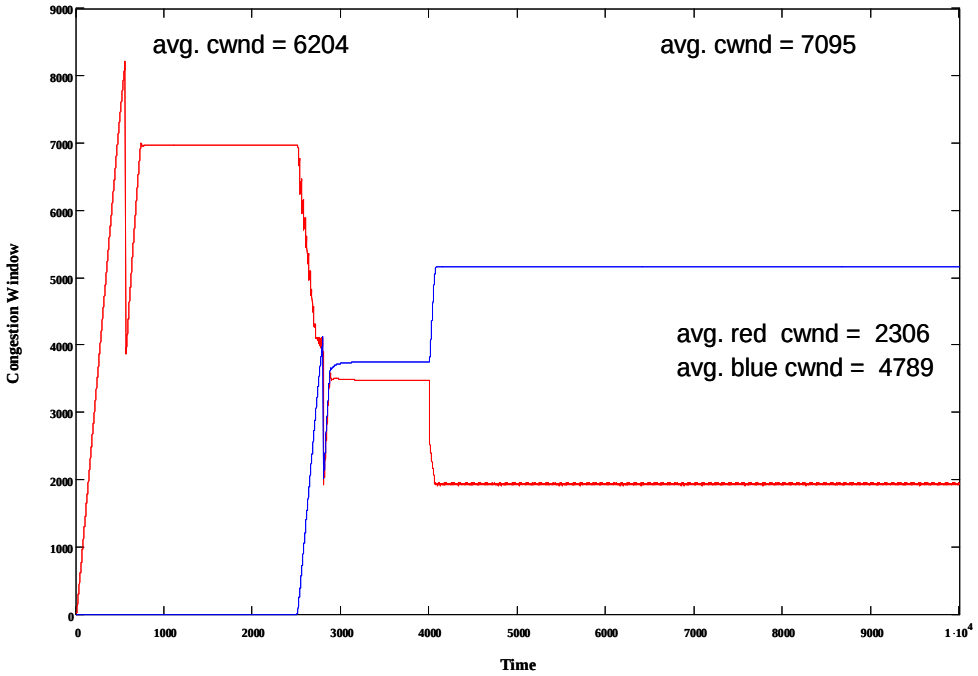


Figure 5-18. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α–tuning enabled, *rtt* = 324 ms)

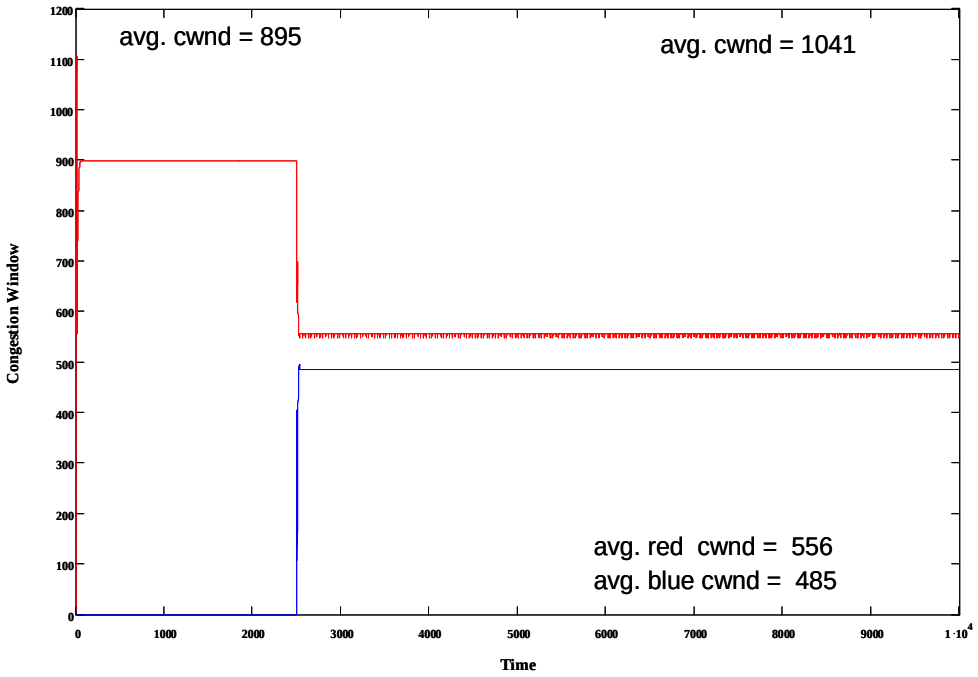


Figure 5-19. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 80, *rtt* = 42 ms)

Li, Leith and Shorten also report that FAST has the fastest convergence time among the congestion control mechanisms compared. Of course, they note the issue of divergence must be taken into account. For all MesoNet simulations where FAST converges to equilibrium the convergence time is very fast.

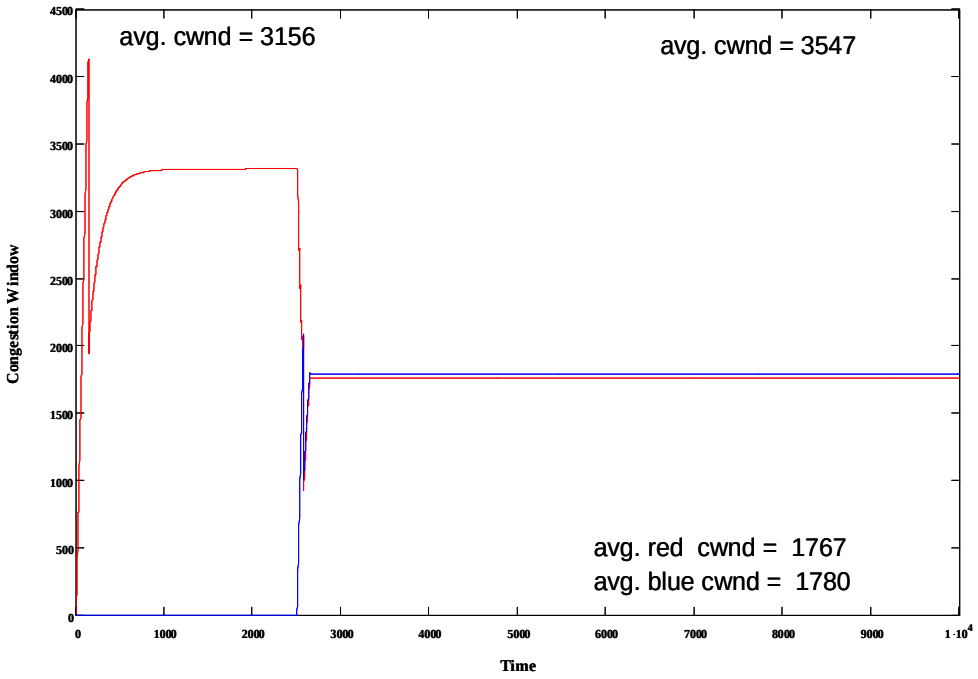


Figure 5-20. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 80, *rtt* = 162 ms)

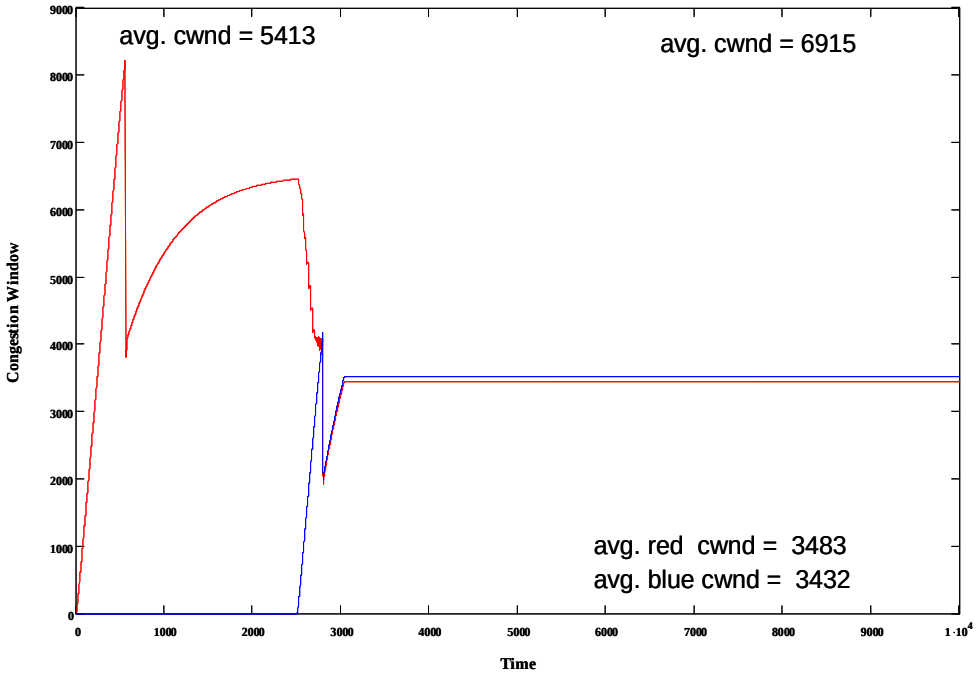


Figure 5-21. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 80, *rtt* = 324 ms)

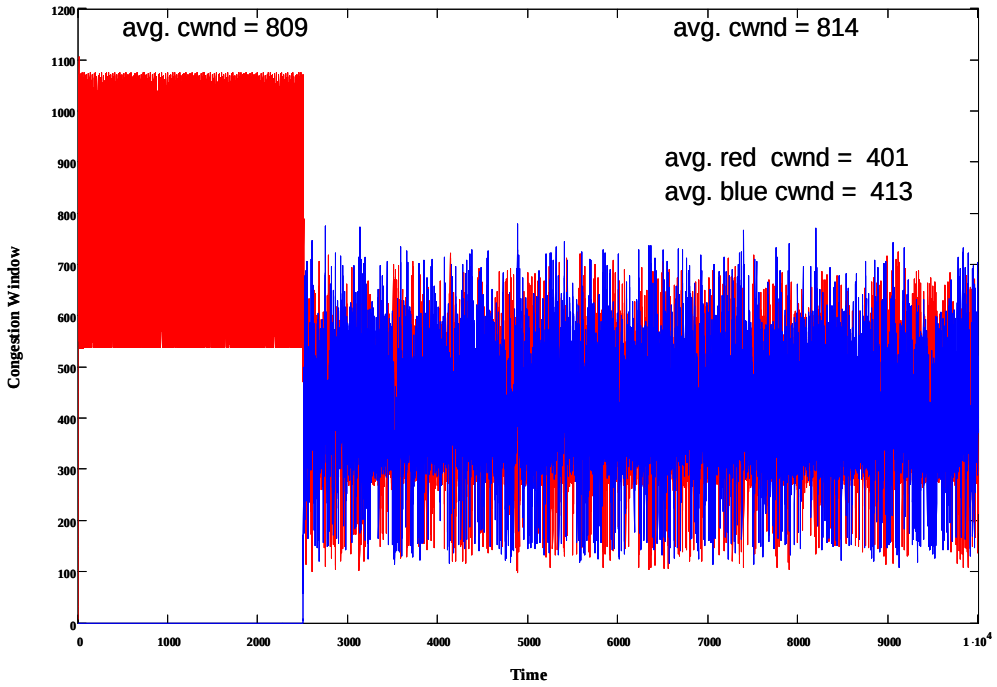


Figure 5-22. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 200, *rtt* = 42 ms)

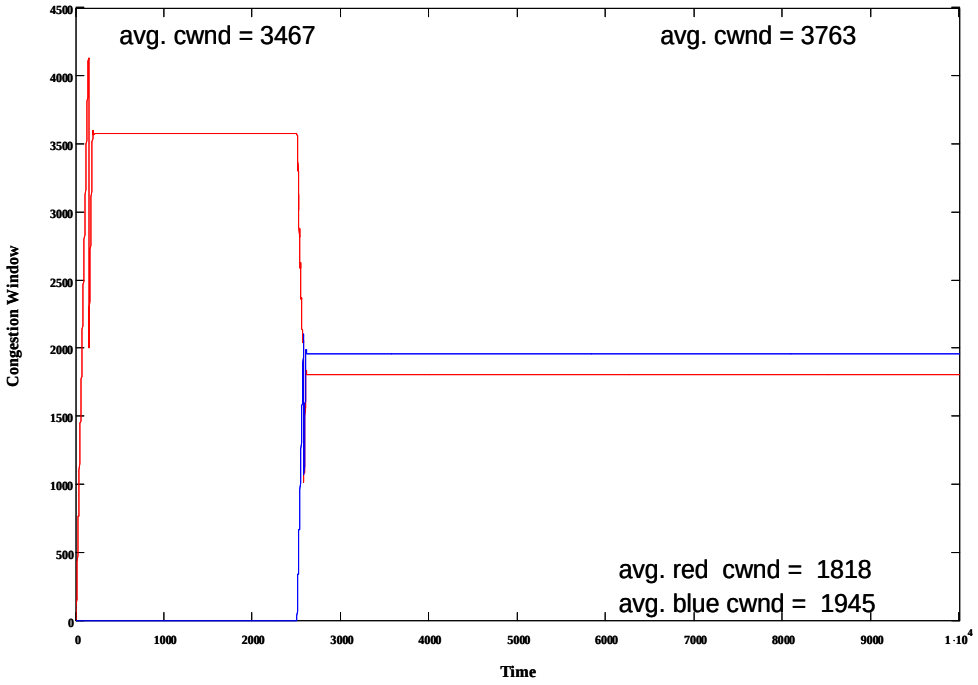


Figure 5-23. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 200, *rtt* = 162 ms)

MesoNet simulations achieved closest convergence among *cwnd* for competing flows with α_F = 80, which was the value we determined as best for the simulated network conditions.

We simulated with α_F = 200 to match values used by the designers of FAST in some empirical studies. Where buffers were sufficient, MesoNet simulations also achieved close convergence with this larger α_F.

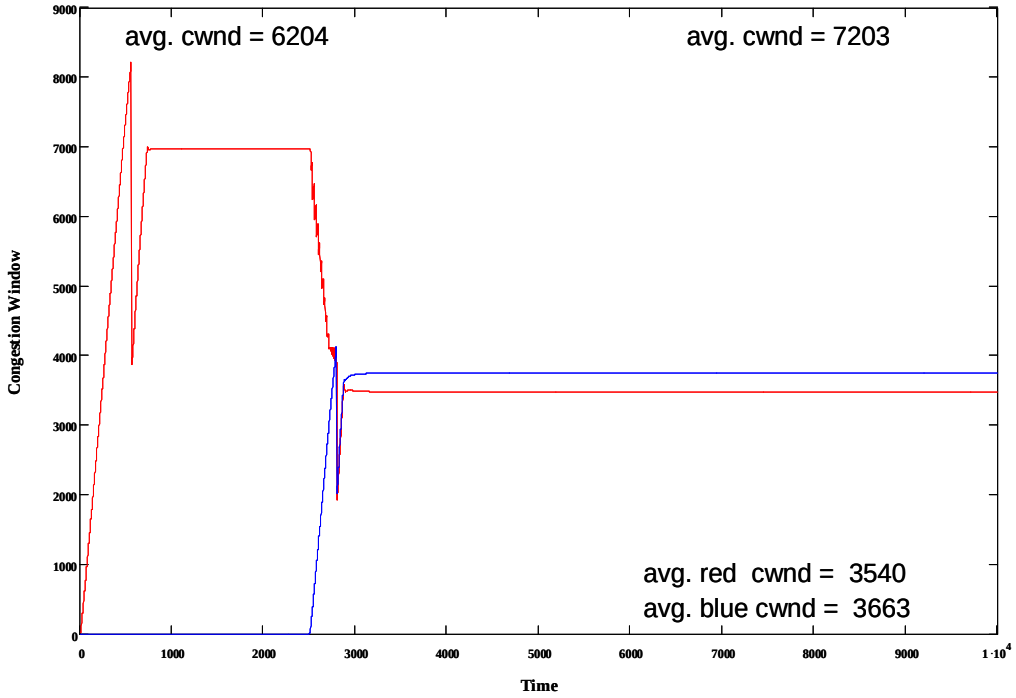


Figure 5-24. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two FAST Flows (α_F = 200, *rtt* = 324 ms)

5.4.5 Behavior of HSTCP Congestion Control Model

The MesoNet simulation results for HSTCP, shown in Figs. 5-25 to 5-27, agree with results from the study by Li, Leith and Shorten. HSTCP flows converge to fairness, but this requires significant time, which increases with increasing *rtt*.

5.4.6 Behavior of H-TCP Congestion Control Model

Figs. 5-28 to 5-30 display the *cwnd* evolutions produced by the MesoNet simulation of H-TCP, which appear quite similar in shape to those reported in the empirical study. H-TCP flows in the simulation appear to converge slightly slower than those reported in the empirical study. Convergence times for simulated H-TCP are second fastest among the congestion control mechanisms simulated. This agrees with results from the study by Li, Leith and Shorten.

5.4.7 Behavior of Scalable TCP Congestion Control Model

MesoNet simulation results for Scalable TCP are shown in Figs. 5-31 to 5-33. For the three *rtt* values simulated, Scalable TCP did not converge to a fair allocation of bandwidth. Scalable TCP implements what amounts to a multiplicative-increase, multiplicative-decrease (MIMD) algorithm, which previous theoretical analysis [1] shows cannot guarantee convergence. Li, Leith and Shorten [67] also found that Scalable TCP either does not converge or converges very slowly. Scalable TCP flows did not converge to fair bandwidth allocation over the 10-minute

duration of the tests used by Li, Leith and Shorten in their empirical study. This agrees with the MesoNet simulation results.

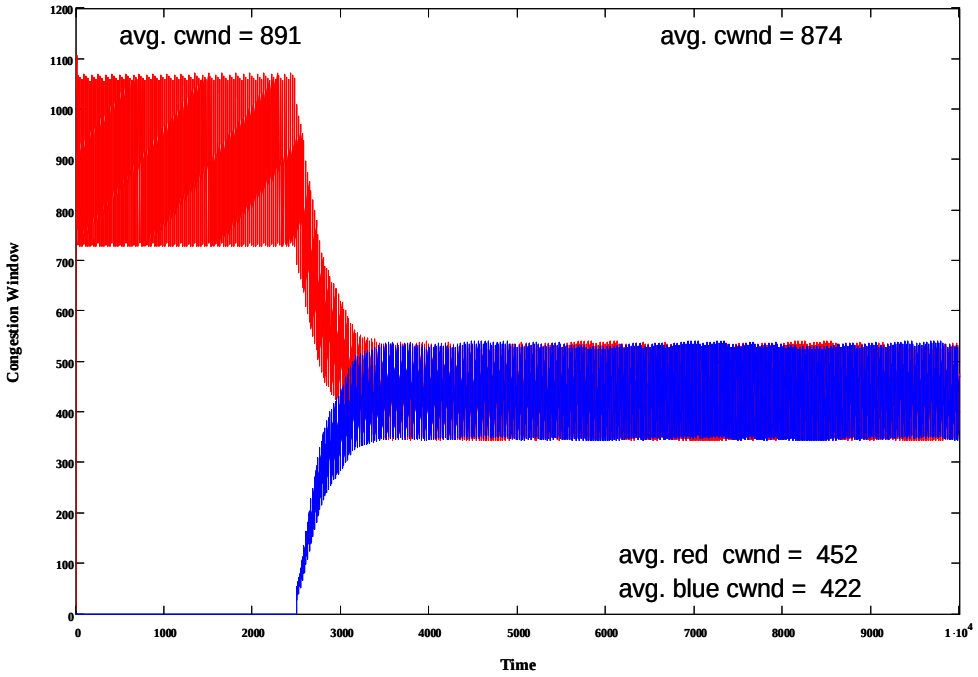


Figure 5-25. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two HSTCP Flows (*rtt* = 42 ms)

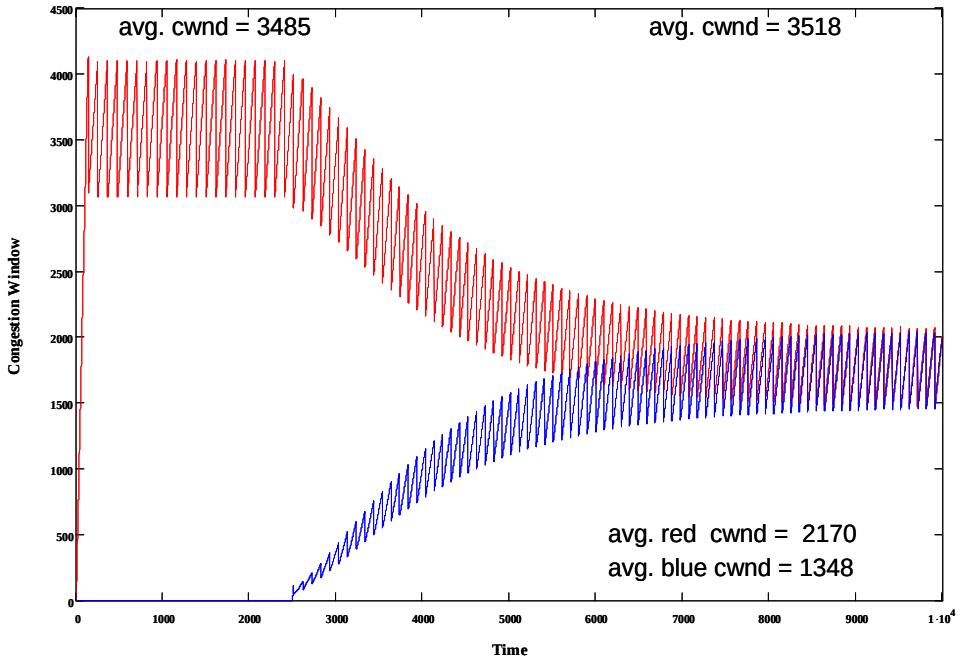


Figure 5-26. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two HSTCP Flows (*rtt* = 162 ms)

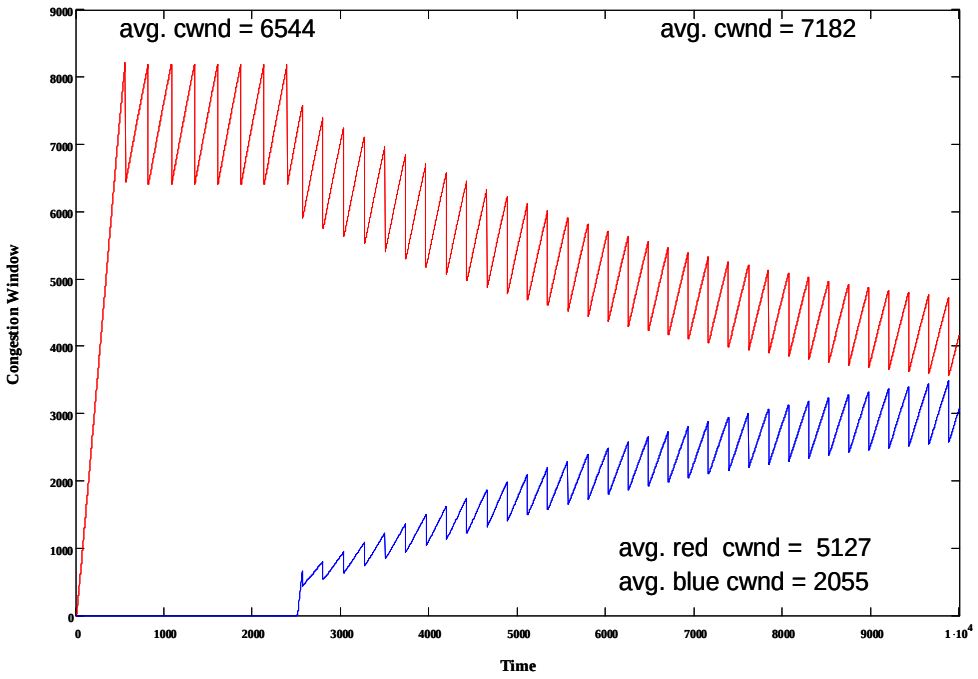


Figure 5-27. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two HSTCP Flows (*rtt* = 324 ms)

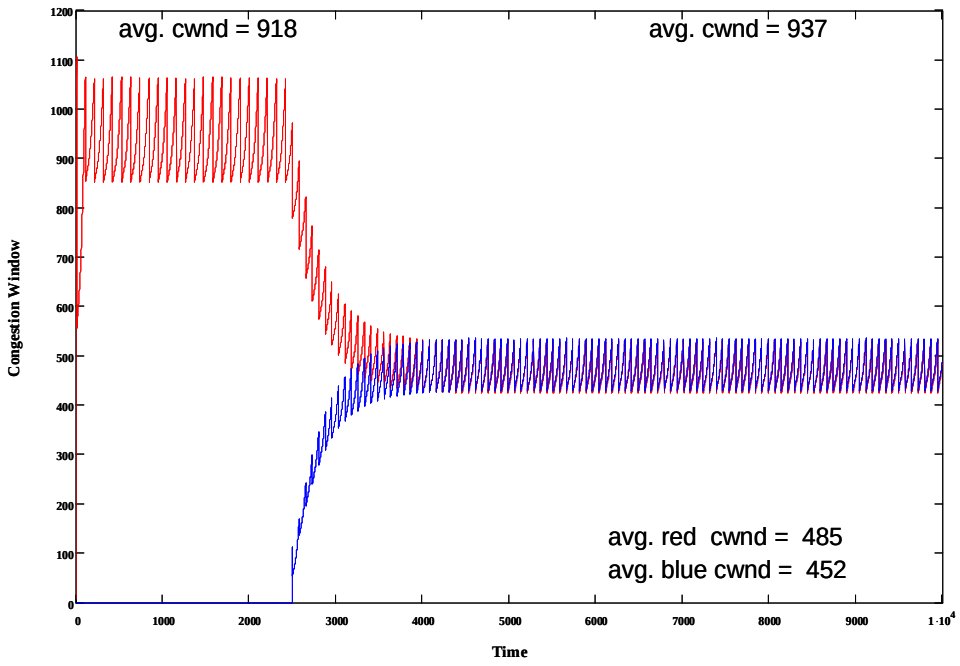


Figure 5-28. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two H-TCP Flows (*rtt* = 42 ms)

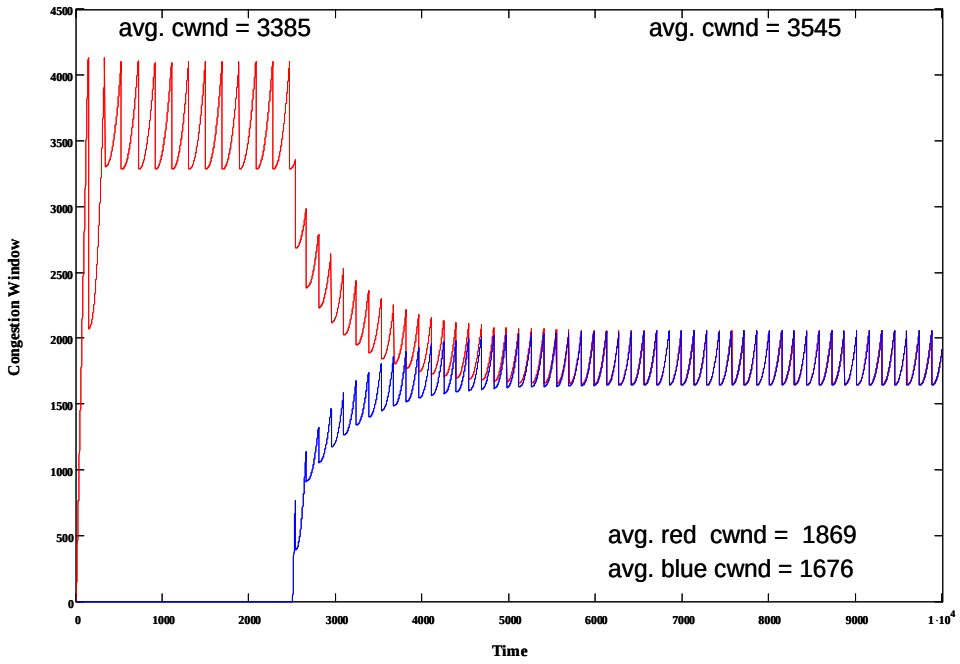


Figure 5-29. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two H-TCP Flows (*rtt* = 162 ms)

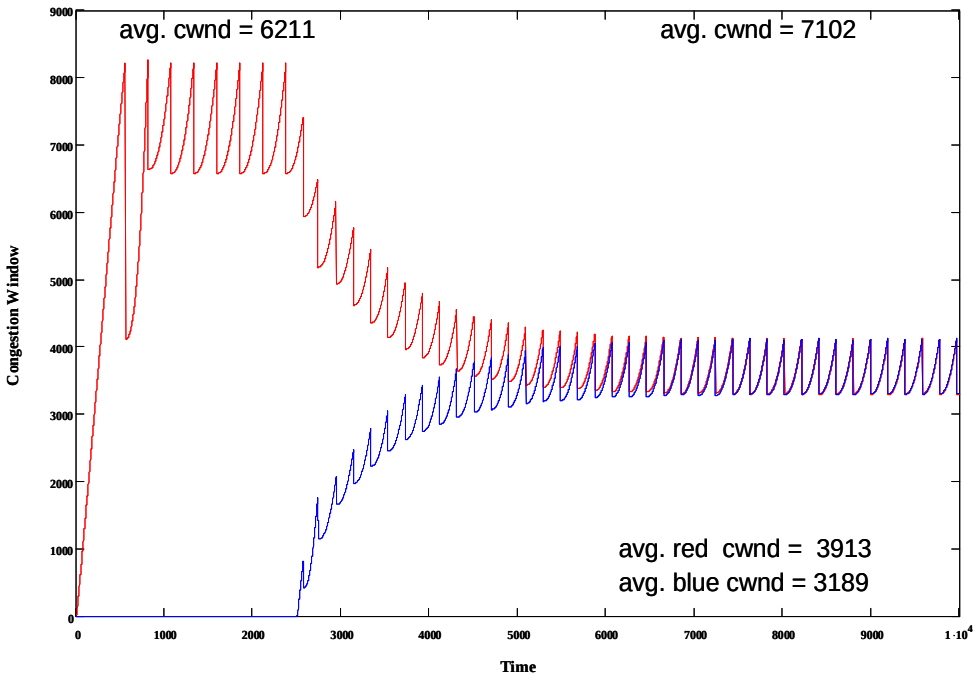


Figure 5-30. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two H-TCP Flows (*rtt* = 324 ms)

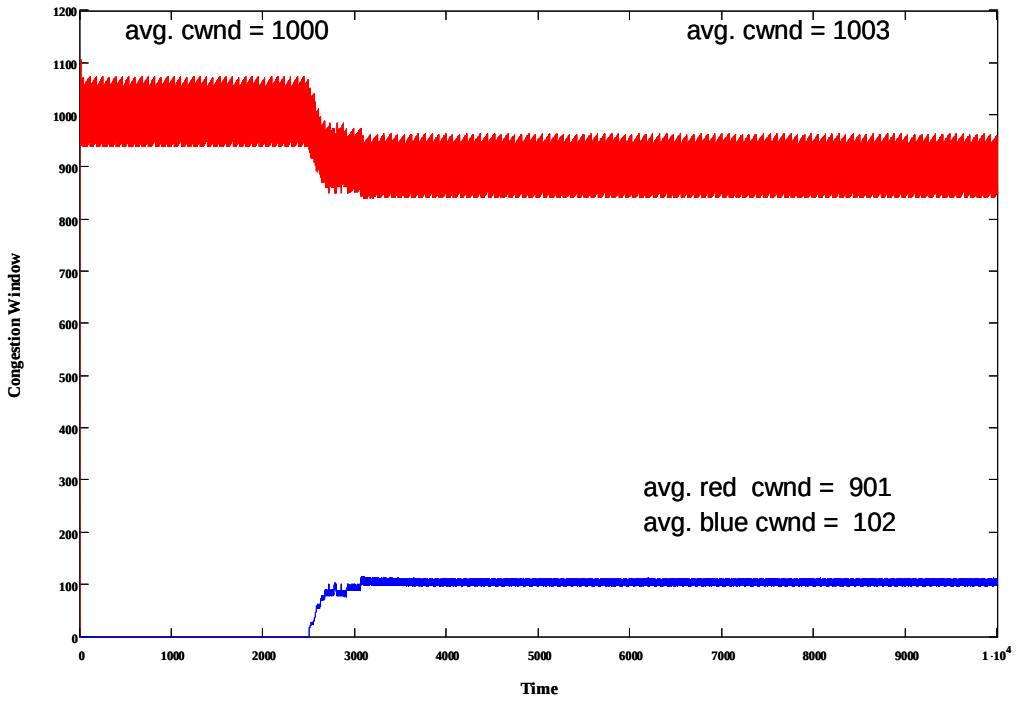


Figure 5-31. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two Scalable TCP Flows (*rtt* = 42 ms)

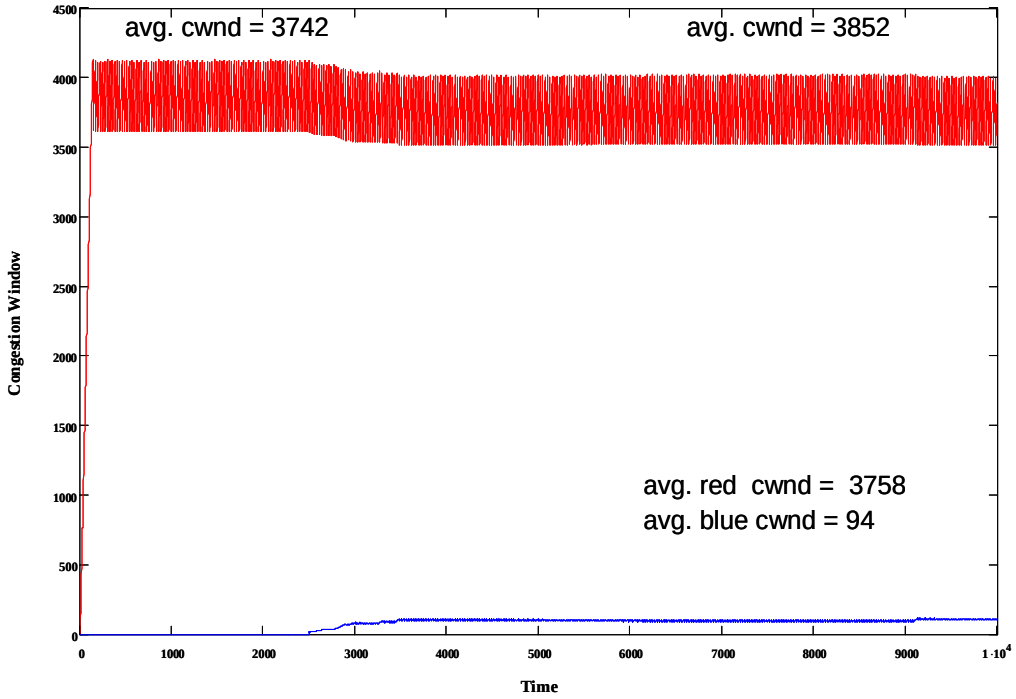


Figure 5-32. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two Scalable TCP Flows (*rtt* = 162 ms)

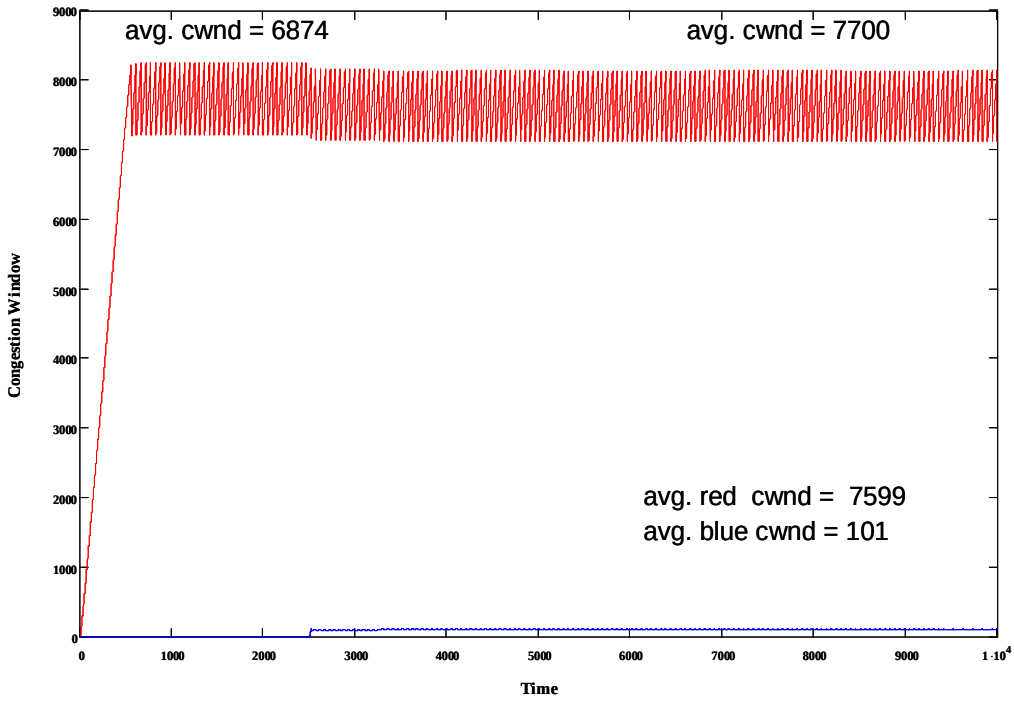


Figure 5-33. Change in *cwnd* (packets) vs. Time (0.1 s units) for Two Scalable TCP Flows (*rtt* = 324 ms)

5.4.8 Summary of Behavior of MesoNet Congestion Control Models

In this section, we provide a summary of the comparative behavior of MesoNet simulations across all seven congestion control mechanisms, including the two extra FAST configurations. We consider three aspects of performance: link utilization, buffer utilization and fairness. In making our comparisons, we use average *cwnd* as a surrogate for average throughput. We limit our numerical analyses to two (rounded) decimal places, so we do not discuss smaller differences in performance among the congestion control mechanisms.

Given a single path with a set of long-lived flows, an ideal congestion control mechanism would yield a situation where each flow has the same average *cwnd* and the sum of the average *cwnd* over all flows equals the bandwidth-delay product (BDP). In such a situation the link is fully utilized, buffers are empty and each flow receives fair (i.e., the same) bandwidth. While congestion control mechanisms are unlikely to be ideal, we can compare congestion control mechanisms by examining relative link and buffer utilizations and fairness.

Table 5-9 (first row) displays the capacity (in packets) of the network path modeled by the dumbbell topology (Fig. 5-6) as a function of *rtt*. These figures define the throughput limits on a path, which caps the maximum link utilization. Once a path contains a sufficient number of packets, then some source will always be able to transmit. As an example, given *rtt* = 42, the path will hold 882 packets in aggregate. Average link utilization can be determined by summing the average *cwnd* over all flows on the path and dividing by the BDP. For example, from Fig. 5-7 we see two TCP flows with average *cwnd* of 409 and 395 packets, respectively. The average link utilization can then be computed as (409 + 395)/882 = 0.91.

In cases where the aggregate average *cwnd* exceeds the BDP, then the excess packets must be sitting in buffers on the path. Table 5-9 (second row) shows the buffer sizes (20% of BDP) as a function of *rtt*. We can estimate the buffer utilization on a path by subtracting the BDP from the aggregate average *cwnd* and then dividing the residual by the number of buffers

on the path. When the residual is ≤ 0, buffers are empty. For example, Fig. 5-31 (*rtt* = 42 ms) shows that Scalable TCP leads to an aggregate average *cwnd* of (901 + 102 =) 1003 packets, giving a residual of (1003 – 882 =) 121 packets in buffers on the path. Thus, buffer utilization is (121/176 =) 0.69. In general, given several congestion control mechanisms that yield 100 % link utilization, we might prefer the one that leads to lowest buffer utilization.

Table 5-9. Capacity (in Packets) of the Dumbbell Topology with Various Round-Trip Times

	rtt = 42 ms	*rtt* = 162 ms	*rtt* = 324 ms
Bandwidth-Delay Product (packets)	882	3402	6804
Buffers (packets)	176	680	1360
Buffers + Bandwidth-Delay Product	1058	4082	8164

Among several congestion control mechanisms with high link utilization, we might also prefer the one allocating bandwidth most fairly. To measure fairness, we use Jain's fairness index [64] but applied to *cwnd* rather than throughput. We use the following formulation.

$$\frac{\left(\sum_{i=1}^{n} cwnd_i\right)^2}{n \times \sum_{i=1}^{n} (cwnd_i)^2} \tag{41}$$

Jain's fairness index ranges between 0 and 1, with a higher value denoting better fairness.

Table 5-10 gives link and buffer utilizations for each simulated congestion control mechanism as a function of *rtt*. Even at the shortest *rtt* (= 42 ms), several of the congestion control mechanisms fail to achieve full link utilization. For TCP and CTCP this results from slow recovery from packet losses. For FAST with α–tuning low utilization arises from two factors: prior to reaching equilibrium α_F is too high, which leads to substantial packet losses, and after reaching equilibrium α_F is too low to fully utilize the link. α_F is too high for FAST with α_F = 200, which leads to packet losses and an oscillating *cwnd*.

As *rtt* increases, all congestion control mechanisms except CTCP and standard TCP achieve full link utilization. (CTCP does achieve 100% at *rtt* = 162 ms, while maintaining an average of four buffered packets.) Among the congestion control mechanisms achieving full utilization, H-TCP, HSTCP and FAST (α_F = 80) lead to relatively low buffer utilizations. BIC and Scalable TCP exhibit relatively high buffer utilizations.

Table 5-11 shows Jain's fairness index for the simulated congestion control mechanisms as a function of *rtt*. As expected, Scalable TCP shows substantial unfairness. The unfairness of BIC and HSTCP increases with *rtt*. Also as expected, FAST with α–tuning leads to unfairness. Several congestion control mechanisms (CTCP, FAST with fixed α_F, and H-TCP) yield fairness across all values of *rtt*.

Table 5-10. Link and Buffer Utilizations for Simulated Congestion control Mechanisms

	rtt = 42 ms		*rtt* = 162 ms		*rtt* = 324 ms	
	Link Util.	Buffer Util.	Link Util.	Buffer Util.	Link Util.	Buffer Util.
TCP	0.91	0.00	0.89	0.00	0.89	0.00
BIC	1.00	0.45	1.00	0.51	1.00	0.63
CTCP	0.95	0.00	1.00	0.01	0.92	0.00
FAST α Tuning	0.82	0.00	1.00	0.44	1.00	0.21
FAST α_F = 80	1.00	0.90	1.00	0.21	1.00	0.08
FAST α_F = 200	0.92	0.00	1.00	0.53	1.00	0.29
HSTCP	0.99	0.00	1.00	0.17	1.00	0.28
H-TCP	1.00	0.31	1.00	0.21	1.00	0.22
Scalable TCP	1.00	0.69	1.00	0.66	1.00	0.66

Table 5-11. Bandwidth Fairness (Jain's Index) for Simulated Congestion Control Mechanisms

	rtt = 42 ms	*rtt* = 162 ms	*rtt* = 324 ms
TCP	1.00	0.96	1.00
BIC	1.00	0.96	0.77
CTCP	1.00	1.00	1.00
FAST α Tuning	0.97	0.88	0.89
FAST α_F = 80	1.00	1.00	1.00
FAST α_F = 200	1.00	1.00	1.00
HSTCP	1.00	0.95	0.85
H-TCP	1.00	1.00	0.99
Scalable TCP	0.61	0.52	0.51

As evident from our simulations, several of the proposed congestion control mechanisms approach ideal performance under the limited cases reported here. H-TCP, FAST and HSTCP give full link utilizations. H-TCP and HSTCP also tend to limit buffer utilization at full link

utilization. FAST limits buffer utilization under some circumstances. H-TCP and FAST with fixed α_F also show good fairness across values of *rtt*. How will the various congestion control mechanisms compare in a larger topology with varying network conditions? We explore this question in the next four chapters (6-9).

Chapter 6 – Comparing Congestion Control Regimes in a Large, Fast Network

6 Comparing Congestion Control Regimes in a Large, Fast Network

We continue our investigation of congestion control mechanisms by comparing relative behaviors given a large (up to 278 x 10^3 sources), fast (backbone routers operating up to 192 Gbps), simulated network with Web traffic, a few long-lived flows and periods of heavy file transfers among selected sites. We adopt an unrealistic assumption that all sources within our simulated network use the same congestion control regime.[1] We simulate our network under a range of conditions, then change the congestion control regime and repeat the simulation for the same conditions. In this way, we can determine how each congestion control regime responds to various conditions and then identify any differences. Various data analyses given later in this chapter refer to congestion control regimes by the identifiers shown in Table 6-1. The details of each regime were explained previously in Chapter 5.

Table 6-1. Congestion Control Mechanisms Compared

Identifier	Label	Name of Congestion Avoidance Algorithm
1	BIC	Binary Increase Congestion Control
2	CTCP	Compound Transmission Control Protocol
3	FAST	Fast Active-Queue Management Scalable Transmission Control Protocol
4	HSTCP	High-Speed Transmission Control Protocol
5	HTCP	Hamilton Transmission Control Protocol
6	Scalable	Scalable Transmission Control Protocol
7	TCP	Transmission Control Protocol (Reno)

We begin by describing (in Sec. 6.1) our experiment design, including the topology simulated, the input factors varied (and fixed), the conditions adopted, the temporal scenario and measured responses. Subsequently (in Sec. 6.2), we describe how we executed our experiments and collected data. In Sec. 6.3, we discuss our approach to data analysis. We display our most salient results in Sec. 6.4 and then (in Sec. 6.5) report our main findings before concluding in Sec. 6.6.

6.1 Experiment Design

The experiment was conducted within a single topology, illustrated in Fig. 6-1. This four-tier topology was explained and justified in Chapter 3. The top tier is formed by 11 backbone routers and 14 pairs of long-distance links. The second tier consists of 22 POP routers, while the third tier comprises 139 access routers. Access routers come in three varieties: normal (gray), fast (green) and directly connected (red). Fast and directly connected access routers connect sites to the topology at higher speeds than normal access routers. Directly connected access routers bypass POP routers and connect directly to the backbone. The fourth tier, not shown in Fig. 6-1, consists of various sources and receivers distributed throughout the topology and located under access routers.

[1] We change this assumption in Chapters 8 and 9.

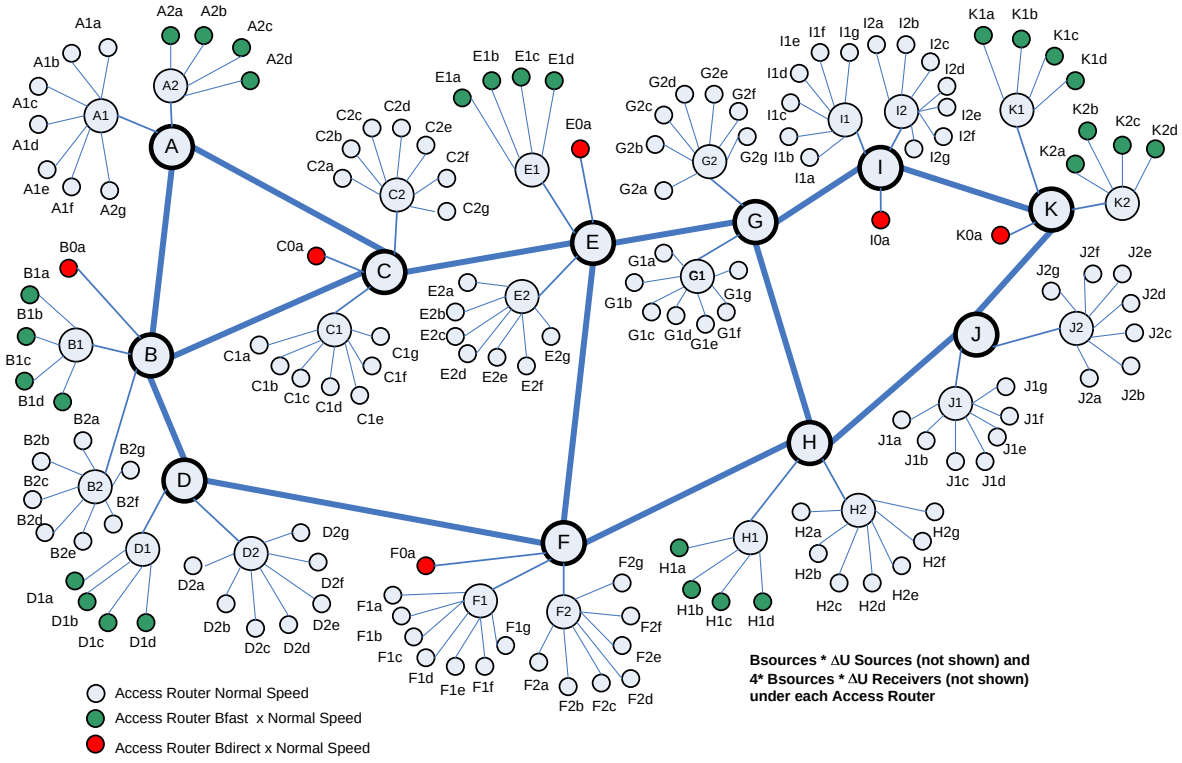


Figure 6-1. Topology Adopted for Experiments

One of the reasons for adopting such a topology is to permit flows to transit paths with a variety of characteristics, so congestion control mechanisms can be compared with respect to path class as defined in Table 6-2, where each path class consists of one or more flow type. A flow type is defined by the type of the access routers under which the source and receiver are located. The flow types in Table 6-2 are color coded to match the access routers depicted in Fig. 6-1. Since a flow cannot expect better performance than access routers provide, a flow is placed into the class dictated by its slowest access router. Thus, the "Very Fast" path class includes only DD flows, while DF flows are allocated to the "Fast" path class and DN flows are allocated to the "Typical" path class and so on.

Table 6-2. Definition of Three Path Classes (note that the correspondent of a source is a receiver and the correspondent of a receiver is a source)

Path Class	Flow Type	Definition
Very Fast	DD	Source & receiver under directly connected access router
Fast	DF	Source or receiver under directly connected access router and correspondent under fast access router
	FF	Source & receiver under fast access router
Typical	DN	Source or receiver under directly connected access router and correspondent under normal access router
	FN	Source or receiver under fast access router and correspondent under normal access router
	NN	Source & receiver under normal access router

6.1.1 Simulation Parameters

Within the framework provided by the topology given in Fig. 6-1, a wide range of network conditions may be simulated by specifying values for various parameters, or input factors, as discussed previously in Chapters 3 and 4. Guided by our sensitivity analysis, we selected six parameters, shown in Table 6-3, which we vary to establish the conditions under which we compare the congestion control mechanisms listed in Table 6-1. These six parameters are called *robustness factors* because any conclusions we draw hold (i.e., are robust) only over our simulated combinations of these parameters. Other simulation parameters are fixed across all experiments, as we document below.

Table 6-3. Robustness Factors Selected for Comparing Congestion Control Mechanisms

Identifier	Definition	PLUS (+1) Value	Minus (-1) Value
x1	Network Speed	8000 packets/ms	4000 packets/ms
x2	Think Time	5000 ms	2500 ms
x3	Source Distribution	Uniform (.33/.33/.33)	Skewed (.1/.6/.3)
x4	Propagation Delay	2	1
x5	File Size	100 packets	50 packets
x6	Buffer Sizing Algorithm	RTTxCapacity	RTTxCapacity/SQRT(*N*)

In Chapter 4, seven of the 11 parameters considered exhibited most significant influence. We adopted six of those seven as robustness factors for our current experiment. We omitted the multiplier on number of sources and receivers because we will consider a smaller network separately in Chapter 7. For each factor, we selected two settings, so we use a two-level experiment design. Network speed (x1) defines the fundamental capacity of backbone routers in packets per ms (p/ms). Recall, however, that this fundamental capacity is multiplied by BBspeedup to determine the full capacity of each backbone router. The speeds of other routers within the topology are derived from the value of x1 using various transformations, as shown in Table 6-4, which lists fixed parameters associated with the network model.

Table 6-4. Fixed Network Parameters

Parameter	Definition	Value
BBspeedup	Backbone router speed = x1xBBspeedup	2
R2	POP routers speed = x1/R2	4
R3	Access routers speed = x1/R2/R3	10
Bdirect	Directly connected access router speed = x1/R2/R3xBdirect	10
Bfast	Fast access router speed = x1/R2/R3xBfast	2
Hbase	Speed of basic sources (96 Mbps)	8
Hfast	Speed of fast sources (960 Mbps)	80
P(FastHost)	Probability that a source is fast	0.4
Qfactor	Factor by which buffer size will be multiplied	1

Sources and receivers may operate at one of two speeds: Hbase (8 p/ms) or Hfast (80 p/ms). This simulates the situation in real networks, where some computers connect at 100 Mbps, while others connect at 1 Gbps. For this experiment, we permit 40 % of

sources (and receivers) to connect at the fast speed, while remaining sources (and receivers) connect at the slower speed.

Factors x4 (propagation delay) and x6 (buffer-sizing algorithm) also alter characteristics of the network. When x4 = 2, the fundamental propagation delays encoded in the topology are doubled. Factor x6 selects the algorithm used to size router buffers. Setting the Qfactor = 1 ensures that the results of the chosen algorithm are used without further scaling of buffer sizes.

The factors controlling network characteristics may be translated into domain-specific values to give a sense of the nature of the network being simulated. For example, the speed of a backbone router when x1 = 8000 p/ms may be translated as 8000 p/ms x 2 x 1000 sec/ms x 12000 bits/packet = 192 Gbps. Table 6-5 shows the simulated speeds for all types of routers given the two values for factor x1. Similar reasoning indicates that fast sources operate at 960 Mbps and basic sources operate at 96 Mbps.

Table 6-5. Domain View of Router Speeds

Router	PLUS (+1)	Minus (-1)
Backbone	192 Gbps	96 Gbps
POP	24 Gbps	12 Gbps
Normal Access	2.4 Gbps	1.2 Gbps
Fast Access	4.8 Gbps	2.4 Gbps
Directly Connected Access	24 Gbps	12 Gbps

Table 6-6 illustrates the range of propagation delays being used within the experiment. Setting x6 = 1 (Minus) simulates a topology with an average one-way path propagation delay comparable to a network in the continental United States that has some links to Europe. Setting x6 = 2 (PLUS) simulates a topology that could span from East Asia to Western Europe, while transiting across North America. Since buffer sizes are computed based on router speed and propagation delay, Table 6-7 gives the range of buffer sizes that are simulated in our experiments.

Table 6-6. Path Propagation Delays Simulated

	Min	Avg	Max
PLUS (+1)	12	81	200
Minus (-1)	6	41	100

Table 6-7. Buffer Sizes Simulated

	PLUS (+1)			Minus (-1)		
Router	Min	Avg	Max	Min	Avg	Max
Backbone	325.528×10^3	732.437×10^3	130.211×10^4	1.153×10^3	2.606×10^3	4.654×10^3
POP	40.691×10^3	91.555×10^3	162.764×10^3	221	505	908
Access	6.47×10^3	14.557×10^3	25.879×10^3	91	207	369

Factor x2, think time, represents the average (exponentially distributed) interval (in ms) before a source initiates a new flow after completing a previous flow. A longer think time leads to lower demand on the network. Factor x3 controls the distribution of sources throughout the topology. The uniform distribution tends to spread congestion more evenly across the topology, while the skewed distribution tends to concentrate congestion more toward fast access routers. The number of sources in the topology is determined by a combination of factor x3 and two fixed factors, Bsources and ΔU, shown in Table 6-8. The net effect on the maximum number of simulated sources is given in Table 6-9. Table 6-8 also gives the fixed distribution of receivers, which creates a bias toward placing receivers under typical access routers. Further, Table 6-8 records the initial slow-start threshold – fixed to an arbitrarily large number of packets for the current simulation experiment.

Table 6-8. Fixed Parameters Related to Sources and Receivers

Parameter	Definition	Value
Bsources	Basic number of sources per access router	1000
ΔU	Avg. sources per access router = Bsources x ΔU	2
P(Nr)	Probability receiver under normal access router	0.6
P(Nrf)	Probability receiver under fast access router	0.2
P(Nrd)	Probability receiver under directly connected access router	0.2
sst_{INT}	Initial slow-start threshold in packets	$2^{31}/2$

Table 6-9. Number of Simulated Sources

PLUS (+1)	Minus (-1)
278×10^3	174.6×10^3

Several fixed parameters, shown in Table 6-10, control the operation of the simulation. The basic simulation time step is set to 1 ms and measurements are taken 5 times/sec, i.e., measurement interval (mi) duration is 200 ms. Total simulated time is (7500 mi/5 mi/s) = 1500 s, which amounts to (1500 s/60 s/m =) 25 minutes simulated for each condition. In order to reduce memory consumption, measures are buffered for only (1500 mi/5 mi/s/60 s/m =) 5 minutes before being written to disk. Table 6-10 also shows the fixed random number seed used for each run.

Table 6-10. Fixed Simulation Control Parameters

Parameter	Definition	Value
M	Number of Time Steps per Measurement Interval	200
MI	Number of Measurement Intervals Simulated	7500
MB	Number of Measurement Intervals Buffered	1500
Rnseed	Random Number Seed	200000
TSD	Duration of Each Time Step in seconds	0.001

For each condition, the 25 simulated minutes are orchestrated into the same scenario, shown in Fig. 6-2. Each time period consists of simulated traffic with specific properties, as defined below. The first 10 minutes, used primarily as a warm-up period,

consists of simulated Web traffic. The subsequent 15 minutes are divided into three, five-minute periods. At the beginning of the first time period (TP1), ongoing Web traffic is augmented by three long-lived flows, which continue for the duration of the simulation. All flows initiated on very fast paths (i.e., DD flows) during TP2 carry jumbo file transfers. At the onset of TP3, all newly initiated flows return to a pattern of simulated Web traffic; any residual backlog of ongoing, jumbo file transfers started during TP2 will continue into TP3 until they complete or the simulation ends. As explained below in Sec. 6.1.3, separate measurements are made in each time period, and selected measurements are totaled over the entire 25 minutes of the simulated scenario.

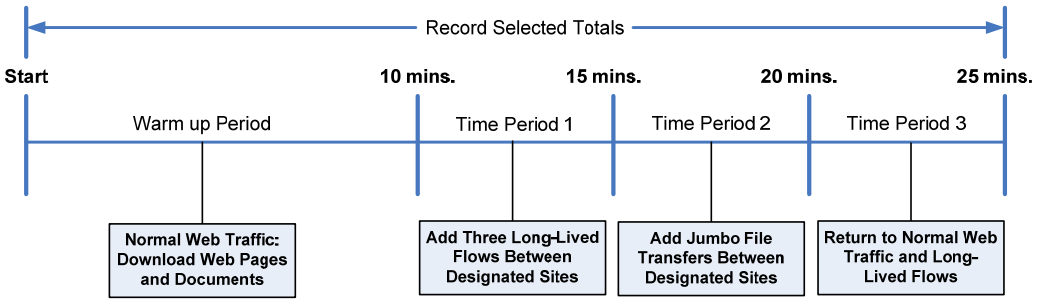


Figure 6-2. Scenario Adopted for Each Simulated Condition

Table 6-11. Fixed Parameters Specifying Simulated User Traffic

Parameter	Definition	Value
α	Shape parameter for Pareto distribution of file sizes	1.5
Fx	Document size = x5 x Fx	10
P(F)	Probability a file is a document	0.01
Jon	Jumbo file transfers begin after Jon x 25 minutes	0.6
Joff	Jumbo file transfers cease after Joff x 25 minutes	0.8
Jx	Jumbo file size = file (or document) size x Jx	100

Table 6-11 specifies the primary fixed parameters controlling generation of user traffic over the 25-minute scenario. Table 6-12 gives fixed parameters for the three long-lived flows. Fundamental file sizes within the simulation are chosen from a Pareto distribution with a mean given by factor x5, which equals either 50 or 100 packets depending on the level of the factor. The shape parameter for the Pareto distribution is fixed at 1.5. Factor x5 represents Web pages with an average size of (50 packets x 1500 bytes/packet =) 75 Kbytes or (100 packets x 1500 bytes/packet =) 150 Kbytes. Recall, however, that MesoNet packets have no size, so file sizes are specified in packets. With a fixed probability of 0.01, i.e., P(F), a document will be downloaded from a Web site. Document sizes are determined by multiplying a file size selected for a Web page by a fixed factor of 10, i.e., Fx, so downloaded documents average either 500 packets (750 Kbytes) or 1000 packets (1.5 Mbytes), depending on the value of x5. This combination of Web pages and documents makes up the pattern of user traffic labeled as normal Web traffic.

Jumbo file transfers, initiated on all DD flows started during TP2, are controlled by three parameters. Jon determines the proportion of elapsed simulation time before jumbo transfers begin and Joff defines the proportion after which initiation of jumbo transfers cease. The size of a jumbo transfer is determined by multiplying the file size chosen for normal Web traffic by a factor of 100 (Jx). This means that jumbo file transfers will average ((50 x .99 + 500 x .01) x 100 =) 251 packets (376.5 Kbytes) or ((100 x .99 + 1000 x .01) x 100 =) 1089 packets (1.63 Mbytes), depending upon the setting of factor x5. Note that all transfers – whether Web pages, documents or jumbo files – are subject to the heavy-tailed property of the Pareto distribution, so transfers may be much larger than the average size.

Table 6-12. Fixed Parameters Specifying Long-Lived Flows

Identifier	Definition	Source Router	Receiver Router	Start Time
L1	Long-distance flow	B0a	K0a	0.4 x 25 mins.
L2	Medium-distance flow	C0a	I0a	0.4 x 25 mins.
L3	Short-distance flow	E0a	F0a	0.4 x 25 mins.

Table 6-12 gives the details for the three long-lived flows that commence in TP1 and continue throughout the remainder of the simulated scenario. Each long-lived flow transmits continuously at whatever rate can be achieved over a very fast (DD) path. The maximum transmission rate for long-lived flows is 80 x 10^3 pps (i.e., long-lived sources and receivers operate at the rate defined by Hfast). Flow L1 traverses the length of the topology. Flow L3 traverses the width of the topology. Flow L2 traverses the middle of the topology. These flows serve several purposes. First, the flows can be individually tracked and measured in detail. This reveals the temporal evolution of the flows, as well as how the flows are influenced by other flows. Second, since the flows transit different distances across the network, measurements can be taken to determine the lag time before each flow reaches its maximum transmission rate. Third, the flows transit directly-connected access routers, so in TP2 the influence of jumbo file transfers may be observed.

6.1.2 Conditions Simulated

For the six factors enumerated in Table 6-2, a two-level experiment design would require simulating (2^6 =) 64 conditions. Given the size and speed of the network we wished to simulate, we decided we could afford examining only 32 conditions. For this reason, we adopted a 2^{6-1} orthogonal fractional factorial (OFF) design. To generate the subset of conditions required by the design, we selected values from Table 6-2 as specified in Table 6-13, a template where each row defines a condition as a combination of the six input factors. The resulting experiment design (in Table 6-14) provides a good balance of individual factors as well as orthogonal combinations of factors. The 2^{6-1} design is a resolution VI design, which means that main effects will be confounded (explained in Sec. 2.5.1) only with five-factor interactions. In addition, two-factor interactions will be confounded only with four-factor interactions. Our previous sensitivity analysis revealed that our model is driven primarily by main effects; even two-factor interactions were not

very evident. For these reasons, we can obtain all necessary information by simulating only 32 of the 64 conditions defined by our input factors.

Table 6-13. Template Specifying a 2^{6-1} Orthogonal Fractional Factorial Design

Factor->	X1	X2	X3	X4	X5	X6
Condition	--	--	--	--	--	--
1	-1	-1	-1	-1	-1	-1
2	+1	-1	-1	-1	-1	+1
3	-1	+1	-1	-1	-1	+1
4	+1	+1	-1	-1	-1	-1
5	-1	-1	+1	-1	-1	+1
6	+1	-1	+1	-1	-1	-1
7	-1	+1	+1	-1	-1	-1
8	+1	+1	+1	-1	-1	+1
9	-1	-1	-1	+1	-1	+1
10	+1	-1	-1	+1	-1	-1
11	-1	+1	-1	+1	-1	-1
12	+1	+1	-1	+1	-1	+1
13	-1	-1	+1	+1	-1	-1
14	+1	-1	+1	+1	-1	+1
15	-1	+1	+1	+1	-1	+1
16	+1	+1	+1	+1	-1	-1
17	-1	-1	-1	-1	+1	+1
18	+1	-1	-1	-1	+1	-1
19	-1	+1	-1	-1	+1	-1
20	+1	+1	-1	-1	+1	+1
21	-1	-1	+1	-1	+1	-1
22	+1	-1	+1	-1	+1	+1
23	-1	+1	+1	-1	+1	+1
24	+1	+1	+1	-1	+1	-1
25	-1	-1	-1	+1	+1	-1
26	+1	-1	-1	+1	+1	+1
27	-1	+1	-1	+1	+1	+1
28	+1	+1	-1	+1	+1	-1
29	-1	-1	+1	+1	+1	+1
30	+1	-1	+1	+1	+1	-1
31	-1	+1	+1	+1	+1	-1
32	+1	+1	+1	+1	+1	+1

6.1.3 Responses Measured

The remainder of the experiment design addresses the system responses measured for each simulated condition. At the top level, we measured a collection of 45 instantaneous responses averaged over each time period and we aggregated 28 measures across all 25 minutes of the simulated scenario. We designate the instantaneous responses as y1 through y45 and we designate the aggregate responses as T.y1 through T.y28. We begin by defining the instantaneous average measures, which may be divided into three categories: (1) measures of macroscopic network behavior, (2) measures of user experience and (3) measures of buffer usage in designated access routers.

6.1.3.1 Measures of Macroscopic Behavior. We selected 12 responses (see Table 6-15) to represent macroscopic behavior in the simulated network. Each response is measured during each measurement interval, which forms a time series. The measured values are then averaged over the relevant time period. Five responses (highlighted in yellow) characterize the status of non-idle flows. Idle flows are those flows waiting within a think period. Non-idle flows are either connecting (y42) or active (y1). Active flows may be operating within initial slow start (y43) or within the normal TCP congestion control

regime (y44) or an alternate regime (y45). The precise nature of the alternate congestion control regime depends upon which congestion avoidance algorithm (recall Table 6-1) is adopted for a particular set of runs. As one would expect, y45 will always be zero when normal TCP Reno congestion avoidance is in use and y44 will be zero for FAST.

Table 6-14. Instantiated Robustness Conditions for 2^{6-1} Experiment Design

Factor->	X1	X2	X3	X4	X5	X6
Condition	--	--	--	--	--	--
1	4000	2500	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
2	8000	2500	.1/.6/.3	1	50	RTTxCapacity
3	4000	5000	.1/.6/.3	1	50	RTTxCapacity
4	8000	5000	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
5	4000	2500	.3/.3/.3	1	50	RTTxCapacity
6	8000	2500	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
7	4000	5000	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
8	8000	5000	.3/.3/.3	1	50	RTTxCapacity
9	4000	2500	.1/.6/.3	2	50	RTTxCapacity
10	8000	2500	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
11	4000	5000	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
12	8000	5000	.1/.6/.3	2	50	RTTxCapacity
13	4000	2500	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
14	8000	2500	.3/.3/.3	2	50	RTTxCapacity
15	4000	5000	.3/.3/.3	2	50	RTTxCapacity
16	8000	5000	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
17	4000	2500	.1/.6/.3	1	100	RTTxCapacity
18	8000	2500	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
19	4000	5000	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
20	8000	5000	.1/.6/.3	1	100	RTTxCapacity
21	4000	2500	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
22	8000	2500	.3/.3/.3	1	100	RTTxCapacity
23	4000	5000	.3/.3/.3	1	100	RTTxCapacity
24	8000	5000	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
25	4000	2500	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
26	8000	2500	.1/.6/.3	2	100	RTTxCapacity
27	4000	5000	.1/.6/.3	2	100	RTTxCapacity
28	8000	5000	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
29	4000	2500	.3/.3/.3	2	100	RTTxCapacity
30	8000	2500	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
31	4000	5000	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
32	8000	5000	.3/.3/.3	2	100	RTTxCapacity

Table 6-15. Responses Characterizing Macroscopic Behavior

Response	Definition
y42	Average number of connecting flows
y1	Average number of active (i.e., connected) flows
y43	Average number of active flows in initial slow start
y44	Average number of active flows in normal congestion-control mode
y45	Average number of active flows in alternate congestion-control mode
y3	Average packets output per measurement interval
y5	Average flows completed per measurement interval
y6	Average retransmission rate
y7	Average smoothed round-trip time (SRTT)
y8	Average round-trip queuing delay
y2	Average congestion-window increases per active flow
y4	Average congestion window per active flow

Two responses (highlighted in blue) represent network-wide throughput, either as packets output (y3) per measurement interval or as flows completed (y5) per measurement interval. Three responses (highlighted in orange) summarize network-wide congestion. One reflection of congestion is the average retransmission rate (y6). Two

other responses reflect congestion-induced delay: average SRTT (y7), from which the average round-trip propagation delay may be subtracted to reveal the average round-trip queuing delay (y8).

The two remaining responses (in green) relate to the congestion window for an average active flow. One (y2) measures the average number of window increases per flow in a measurement interval, while the other (y4) measures the average congestion window size (in packets) per flow. These measures reflect congestion but can also reflect details associated with the operation of specific congestion control algorithms.

6.1.3.2 Measures of User Experience. We use *goodput* as a fundamental measure of user experience. We define goodput as the number of packets per second (pps) received at the user level on a given flow. Thus, goodput excludes retransmissions. Since various flows transit the topology on paths that possess different characteristics, we measure user experience for flows on each path class (recall Table 6-2). Recognizing that goodput can be influenced by the number of flows sharing the same path, we measure the relevant characteristics. For example, Table 6-16 shows how we characterize user experience for flows on very fast (DD) paths. We measure not only average goodput (y9) but also the average number of active flows (y10) and the average number of completed flows (y11). We assume that completed flows finish at uniformly distributed times in a given measurement interval. We then compute (y12) the average aggregate number of pps delivered on all DD flows. This allows us to investigate average goodput in a nuanced fashion. We make similar measurements for (DF and FF) flows transiting fast paths (Table 6-17) and for those (DN, FN, NN) flows transiting typical paths (Table 6-18).

Table 6-16. Responses Characterizing User Experience on Very Fast Paths

Response	Definition
y9	Average goodput (pps) for DD flows
y10	Average number of active DD flows
y11	Average number of DD flows completed per measurement interval
y12	Average aggregate number of DD packets delivered per second = y9x(y10+(y11/2))

Table 6-17. Responses Characterizing User Experience on Fast Paths

Response	Definition
y13	Average goodput (pps) for DF flows
y14	Average number of active DF flows
y15	Average number of DF flows completed per measurement interval
y16	Average aggregate number of DF packets delivered per second = y13x(y14+(y15/2))
y21	Average goodput (pps) for FF flows
y22	Average number of active FF flows
y23	Average number of FF flows completed per measurement interval
y24	Average aggregate number of FF packets delivered per second = y21x(y22+(y23/2))

Table 6-18. Responses Characterizing User Experience on Typical Paths

Response	Definition
y17	Average goodput (pps) for DN flows
y18	Average number of active DN flows
y19	Average number of DN flows completed per measurement interval
y20	Average aggregate number of DN packets delivered per second = y17x(y18+(y19/2))
y25	Average goodput (pps) for FN flows
y26	Average number of active FN flows
y27	Average number of FN flows completed per measurement interval
y28	Average aggregate number of FN packets delivered per second = y25x(y26+(y27/2))
y29	Average goodput (pps) for NN flows
y30	Average number of active NN flows
y31	Average number of NN flows completed per measurement interval
y32	Average aggregate number of NN packets delivered per second = y21x(y22+(y23/2))

We also measure user experience individually for the three long-lived flows defined in the scenario. For these flows, we measure only average goodput, as shown in Table 6-19.

Table 6-19. Responses Characterizing User Experience on Long-Lived Flows

Response	Definition
y33	Average goodput (pps) for the long-distance flow (L1)
y34	Average goodput (pps) for the medium-distance flow (L2)
y35	Average goodput (pps) for the short-distance flow (L3)

Table 6-20. Responses Charactering Buffer Usage in Directly Connected Access Routers

Response	Definition
y36	Average buffer saturation for router B0a
y37	Average buffer saturation for router C0a
y38	Average buffer saturation for router E0a
y39	Average buffer saturation for router F0a
y40	Average buffer saturation for router I0a
y41	Average buffer saturation for router K0a

6.1.3.3 Measures of Buffer Usage. The construction of the simulated topology ensures that most (if not all) significant buffer usage occurs at the access routers, most of which have much lower speeds than the POP and backbone routers. The topology used in the simulation consists of 139 access routers. We chose to analyze buffer usage only for the six directly connected access routers, as shown in Table 6-20. For each router, we measure average buffer saturation, defined as the ratio of buffers in use to buffers available.

6.1.3.4 Aggregate Measures. We measure 28 responses over the course of the entire 25-minute scenario, including the warm-up period. These responses fall into three broad categories: (1) measures of macroscopic behavior, (2) measures of user experience and (3) measures of flow distribution among backbone routers. We discuss each of these in turn.

As shown in Table 6-21, we aggregate the number of data packets injected (T.y1) into the network as well as the number of packets delivered (T.y2) over the entire 25 minutes simulated. We provide similar measures for flows connected (T.y3) and completed (T.y4). For connected flows, we also measure (T.y5) the average number of SYN packets sent per flow. This provides some measure of the degree to which congestion impedes the ability of flows to connect.

Table 6-21. Aggregate Responses Characterizing Macroscopic Behavior

Response	Definition
T.y1	Aggregate packets input
T.y2	Aggregate packets output
T.y3	Aggregate flows connected
T.y4	Aggregate flows completed
T.y5	Average SYNs sent per flow

We characterize user experience for completed flows in each path class using two measures: (1) aggregate number of flows completed and (2) average per-flow goodput on the completed flows. We consider completed flows in aggregate for two reasons. First, we can include flows across the entire 25 simulated minutes. Second, some flows may have trouble completing, so we can view goodput for completed flows as a best case measure of user experience. Below, we identify the measures for each path class: very fast paths (Table 6-22), fast paths (Table 6-23) and typical paths (Table 6-24).

Table 6-22. Responses Characterizing User Experience for Completed Flows on Very Fast Paths

Response	Definition
T.y6	Aggregate number of DD flows completed
T.y7	Average goodput (pps) for completed DD flows

Table 6-23. Responses Characterizing User Experience for Completed Flows on Fast Paths

Response	Definition
T.y8	Aggregate number of DF flows completed
T.y9	Average goodput (pps) for completed DF flows
T.y12	Aggregate number of FF flows completed
T.y13	Average goodput (pps) for completed FF flows

Table 6-24. Responses Characterizing User Experience for Completed Flows on Typical Paths

Response	Definition
T.y10	Aggregate number of DN flows completed
T.y11	Average goodput (pps) for completed DN flows
T.y14	Aggregate number of FN flows completed
T.y15	Average goodput (pps) for completed FN flows
T.y16	Aggregate number of NN flows completed
T.y17	Average goodput (pps) for completed NN flows

The final set of responses measure the distribution of flows transiting the 11 backbone routers. As shown in Table 6-25, we simply total the number of completed flows that transit each backbone router during the 25 simulated minutes. Measuring these responses enables us to detect whether any of the congestion control regimes shift the workload experienced by backbone routers.

Table 6-25. Responses Characterizing Distribution of Flows among Backbone Routers

Response	Definition
T.y18	Aggregate completed flows transiting backbone router A
T.y19	Aggregate completed flows transiting backbone router B
T.y20	Aggregate completed flows transiting backbone router C
T.y21	Aggregate completed flows transiting backbone router D
T.y22	Aggregate completed flows transiting backbone router E
T.y23	Aggregate completed flows transiting backbone router F
T.y24	Aggregate completed flows transiting backbone router G
T.y25	Aggregate completed flows transiting backbone router H
T.y26	Aggregate completed flows transiting backbone router I
T.y27	Aggregate completed flows transiting backbone router J
T.y28	Aggregate completed flows transiting backbone router K

6.2 Experiment Execution and Data Collection

In this section, we shift gears to discuss the mechanics of executing the experiments and collecting the data. We describe the resources available for conducting the simulations and also the resource requirements. In addition, we define the format in which we collected data to capture our measured responses.

6.2.1 Experiment Execution

We simulated seven congestion control mechanisms (recall Table 6-1) under the same 32 conditions (recall Table 6-14), requiring (7 x 32 =) 224 separate simulation runs. We had six available compute servers with the characteristics defined in Table 6-26. Each compute server provided 8 processors, so we had a total of (6 x 8 =) 48 processors on which we could execute simulations in parallel. Each of the compute servers was

provisioned with 32 Gbytes of memory. Two of the servers (ws9 and ws10) had four dual-core AMD Opertron™ 8218 processors operating at 2.6 GHz, while the remaining servers (ws11-ws4) had four dual-core AMD Opertron™ 8222 SE processors operating at 3 GHz. All of the compute servers executed under the control of the 64-bit version of Microsoft Windows[2] Server 2003™.

Table 6-26. Characteristics of Compute Servers Used to Execute the Simulations

Compute Server	Physical Processors	Speed (GHz)	Memory (GB)	Operating System
ws9	8	2.6	32	Windows Server 2003 R2 x64 Edition SP2
ws10	8	2.6	32	Windows Server 2003 R2 x64 Edition SP2
ws11	8	3.0	32	Windows Server 2003 R2 x64 Edition SP2
ws12	8	3.0	32	Windows Server 2003 R2 x64 Edition SP2
ws13	8	3.0	32	Windows Server 2003 R2 x64 Edition SP2
ws14	8	3.0	32	Windows Server 2003 R2 x64 Edition SP2

Given 48 processors (also referred to as central-processing units, or CPUs), we were able to run one congestion control mechanism simultaneously against all 32 conditions and we could run another congestion control mechanism against half (16) of the conditions. As shown in Table 6-27, we ran simulations for five congestion control mechanisms (BIC, CTCP, FAST, HTCP and TCP) on the four faster compute servers (ws11-ws14) and we ran simulations for the other two (HSTCP and Scalable TCP) on the slower compute servers (ws9-ws10).

Table 6-27. Processing Requirements for Simulations Mapped to Specific Compute Servers (Units are Processor Days)

	Compute Servers ws11-ws14						Compute Servers ws9-ws10		
	BIC	CTCP	FAST	HTCP	TCP	Totals	HSTCP	Scalable	Totals
CPU time (32 runs)	91.5	97.2	93.4	96.4	94.2	472.5	108.6	110.5	219.1
Avg. CPU time (per run)	2.86	3.04	2.92	3.01	2.94	14.77	3.39	3.46	13.70 (6.85x2)
Min. CPU time (one run)	1.16	1.33	1.44	1.40	1.28		1.61	1.51	
Max. CPU time (one run)	5.94	5.85	5.17	5.84	5.63	28.42	6.57	6.61	26.37 (13.18x2)

Each simulated condition required about 1.25 Gbytes of memory, so running 8 simulations in parallel on one compute server required about 10 Gbytes, or about 1/3 of

[2] Our simulation model, MesoNet, is written in the SLX simulation language. The SLX compiler and run-time require the Microsoft Windows™ operating system.

the available memory. On the other hand, as indicated in Table 6-27, running all 224 simulations required substantial processing resources: (472.5 + 219.1 =) 691.6 processor (CPU) days. Running 48 simulations in parallel, we could potentially have finished the experiment in (691.6 processor days/48 processors =) 14.4 days. Achieving this goal required some astute management of the runs. For example, launching 32 runs for a given congestion control mechanism and then waiting for all runs to complete prior to starting the next set would advance progress at a pace congruent with the maximum processor time required among the 32 simulations run for each congestion control mechanism. As shown in the last row of Table 6-27, this naïve approach would have completed the simulations in 28.42 days, which is the time required to run the five congestion control mechanisms on ws11-ws14. Using the same naïve approach, the two mechanisms simulated on ws9 and ws10 could complete (28.42 - 26.37 =) two days sooner. Note that since only 16 of the 32 conditions could be run in parallel on ws9 and ws10 the processor time required must be doubled, e.g., (6.57 + 6.61) x 2 = 26.37 days.

To complete the simulations in about two weeks one needs to achieve a rate of progress close to the average processor time per run, shown in the second row of Table 6-27. This can be done by first estimating the relative run time required by each simulated condition, and then sorting the conditions by estimated run time into two lists: (1) shortest-to-longest and (2) longest-to-shortest. The two lists define a mapping function for scheduling simulation runs. Whenever a simulation finishes for a specific condition on the first list, select the next condition to start based on its mate from the second list. In this way, as short conditions finish they are replaced by long conditions and vice versa. This enables completing the simulations in just over two weeks, the maximum of 14.77 days and 13.7 days, as shown in the second row of Table 6-27.

Why does the simulation require so much processor time? Each experiment simulates the operation of up to hundreds of thousands of simultaneously active flows over a period of 25 simulated minutes. Each flow that starts during the simulation must be modeled, as well as every packet sent on each flow. Each packet transits several routers as it propagates through the simulated topology. As shown in Table 6-28, the average condition requires simulating just over 74 million flows during the 25 simulated minutes. This amounts to simulating around 7 billion data packets, each of which has a matching acknowledgment. Thus, in a given simulation run 14 billion packets are sent on average. For all conditions across all congestion control algorithms, more than 16.5 billion flows and 3 trillion packets (1.5 trillion data packets and 1.5 trillion acknowledgments) must be simulated. In Chapter 7 we investigate whether a scaled down network simulation can provide sufficient information while requiring less processor time.

Table 6-28. Characterization of the Number of Flows and Data Packets Simulated

Statistic	Flows Completed	Data Packets Sent
Avg. Per Condition	74.033×10^6	6.912×10^9
Min. Per Condition	40.966×10^6	3.147×10^9
Max. Per Condition	154.914×10^6	11.917×10^9
Total All Runs	16.583×10^9	1.548×10^{12}

6.2.2 Data Collection

We collected summary response measurements into four files: one file for each of three five-minute time periods (recall Fig. 6-2) and one file containing data aggregated across the entire 25-minute scenario. Table 6-29 shows the format used for each time-period file. Each file consists of (7 x 32 =) 224 rows of 47 columns. The header row, shown for clarity in Table 6-29, was not included in the data file. The first column identifies the congestion control algorithm and the second column identifies the condition. Each of the remaining columns contains the value for one of the 45 responses measured for the relevant time period (recall Sec. 6.1.3). A response represents the average value across all measurement intervals within the time period.

Table 6-29. Format Adopted for Each Time-Period Data File

Algorithm	Run	y1	y2	…	y44	y45
1	1	33473.81	9.111708	…	21090.14	1834.653
1	2	15370.5	41.35026	…	0.758	0.555333
…	…	…	…	…	…	…
1	31	107357	24.93412	…	602.5607	110.8673
1	32	28287.67	9.126458	…	23397.35	83.21733
…	…	…	…	…	…	…
7	1	34108.22	9.115773	…	23487.56	0
7	2	15333.99	41.4579	…	0.116	0
…	…	…	…	…	…	…
7	31	108421.6	1.899736	…	99872.35	0
7	32	27644.4	25.66166	…	363.9507	0

As shown in Table 6-30, we adopted a similar format for the file containing aggregate responses. In this case, the file included only 30 columns because the number of responses was limited to 28. As discussed in Sec. 6.1.3.4, most values represent an aggregation across the entire 25-minute scenario, while some values represent an average goodput or SYN rate across the scenario.

Table 6-30. Format Adopted for Reporting Aggregate Measures

Algorithm	Run	T.y1	T.y2	…	T.y27	T.y28
1	1	6141323986	5849131237	…	13251058	21098541
1	2	7569164137	7565416432	…	16627869	28921523
…	…	…	…	…	…	…
1	31	6267074928	5653265836	…	8205819	6654958
1	32	9246017312	9244473772	…	13946759	10825599
…	…	…	…	…	…	…
7	1	6132681434	5854382727	…	13255301	21163097
7	2	7564415186	7562570309	…	16634422	28934311
…	…	…	…	…	…	…
7	31	6158229285	5557616933	…	8215894	6648529
7	32	9245995194	9245121085	…	13993455	10880051

To support some detailed analyses (as discussed below in Sec. 6.3) we also used selected time-series data files as output directly by MesoNet. A time series for a particular response simply provides the raw measurement data that was used to create the summarization reported in Table 6-29.

6.3 Data Analysis Approach

In this section, we introduce and explain our approach to data analysis. For illustrative purposes, we also provide a few insights into the behavior of our simulated network. We defer a complete presentation of key results until Sec. 6.4.

We employed three main techniques for analyzing data: (1) cluster analysis, (2) detailed analysis of individual responses and (3) summary analysis of all responses across all conditions. Where advantageous, we also adopted some useful strategies to explore the data. We address each of these topics in turn, beginning with cluster analysis.

6.3.1 Cluster Analysis

We use cluster analysis to provide a comprehensive comparison of differences among all congestion control algorithms for all responses and conditions. Results from the cluster analysis establish whether any of the algorithms generate a distinctive response to the various conditions. To perform the analysis we used hierarchical clustering tools from the MATLAB™ Statistics Toolbox™ [87]. Hierarchical clustering requires selection of a function to compute distances between points in the vector space composed by the response data. We used the standardized Euclidean distance function.

$$Dist(Y_i, Y_j) \equiv \sqrt{\sum_{m=1}^{45} \frac{(Y_{im} - Y_{jm})^2}{(\sigma_m)^2}} \tag{1}$$

Equation (1) computes the inter-algorithm distance in 45-dimension space, where each dimension *m* represents one response. Here, Y_i and Y_j represent the response vectors for the *i*th and *j*th congestion control algorithms. (Note that we use a 28-dimension space when clustering aggregate results.) Distances for each response are normalized with respect to response variance. This enables distances to be placed on a similar scale. (Any response with zero standard deviation is excluded from the distance computation.) A pair of algorithms with close proximity may be linked together within a cluster.

We measure the linkage between clusters of algorithms as the average distance between responses associated with each algorithm in each cluster. The linkage function, shown in (2), uses the Euclidean-distance function from (1).

$$D(r,s) \equiv \frac{1}{n_r n_s} \sum_{k=1}^{n_r} \sum_{l=1}^{n_s} Dist(Y_{k,r}, Y_{l,s}) \tag{2}$$

Equation (2) computes the linkage between any two clusters *r* and *s*, containing n_r and n_s congestion control algorithms, respectively. $Y_{k,r}$ represents the response vector for the *k*th congestion control algorithm in cluster *r*; similarly, $Y_{l,s}$ represents the response vector for

the *l*th congestion control algorithm in cluster *s*. The linkage function is used to place binary clusters into larger clusters, forming a hierarchical tree.

The final step in hierarchical clustering is to suggest which congestion control algorithms should be included within the same cluster. For this purpose, we use the MATLAB™ dendrogram ( ) function to color the lines on the hierarchical tree whenever the linkage value between two clusters falls below 70 % of the maximum linkage value. The net result from clustering is a diagram, such as Fig. 6-3, suggesting relationships among the congestion control algorithms. Identifiers for the seven congestion control algorithms (Table 6-1) are plotted on the x axis and the y axis displays standardized distances between algorithms in the subordinate cluster(s). Here, the clustering suggests algorithms 4 and 6 give similar results and algorithms 1 and 2 give similar results. The remaining algorithms are dissimilar, with algorithm 3 being most dissimilar from the others.

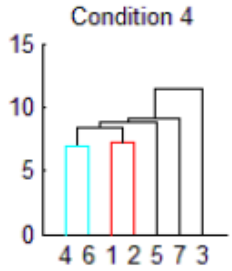


Figure 6-3. Dendrogram Illustrating Clustering Based on Responses for Condition 4 During Time Period One (TP1) – x axis gives the algorithm identifier from Table 6-1 and y axis gives the standardized Euclidean distance between algorithms or clusters of algorithms

Clustering must be performed individually on the various conditions because the conditions can yield results that are quite dissimilar. One may obtain an overall picture of clustering across conditions by plotting together 32 dendrograms, one per condition. Fig. 6-4 shows such a plot for seven congestion control algorithms and related responses covering TP1. Review of the plot reveals that algorithm 3 appears distinctive under about 23 of the 32 conditions. Further, the responses generated by the different algorithms are indistinguishable in six conditions – in fact, are identical for condition 12, where the corresponding dendrogram shows zero distance between the algorithms. The remaining three conditions (2, 27 and 32) find small distinctions among the algorithms. As Fig. 6-4 illustrates, clustering analysis can reveal some significant overall patterns in the data.

A natural next step is to consider why algorithm 3 (FAST) is distinctive in many of the conditions but not in all. In other words, can we determine properties that distinguish among the conditions and then map those properties into hypotheses regarding the operation of algorithm 3? Given the input factors (x1…x6) defining the conditions, we suspect that distinct conditions represent differing levels of congestion within the simulated network. To confirm our suspicion, we can sort the conditions using some property, such as loss rate or retransmission rate, which reflects congestion. Fig. 6-5 displays a bar chart where conditions on the x axis are sorted in order of increasing retransmission rate (response y6) on the y axis. The bar chart shows that 16 conditions have much higher retransmission rates (reflecting higher congestion) than the others. Thus, half the conditions lead to significant congestion and half do not. To quantify the

difference, we include an inset bar chart in Fig. 6-5. The inset shows that the highest retransmission rate (for condition 11) among the uncongested conditions is an order of magnitude or more lower than the lowest retransmission rate (for condition 18) among the congested conditions. Examining the uncongested conditions in detail, one can declare somewhat arbitrary distinctions between conditions with no congestion (N), little congestion (L) and moderate congestion (M). We label Fig. 6-5 accordingly.

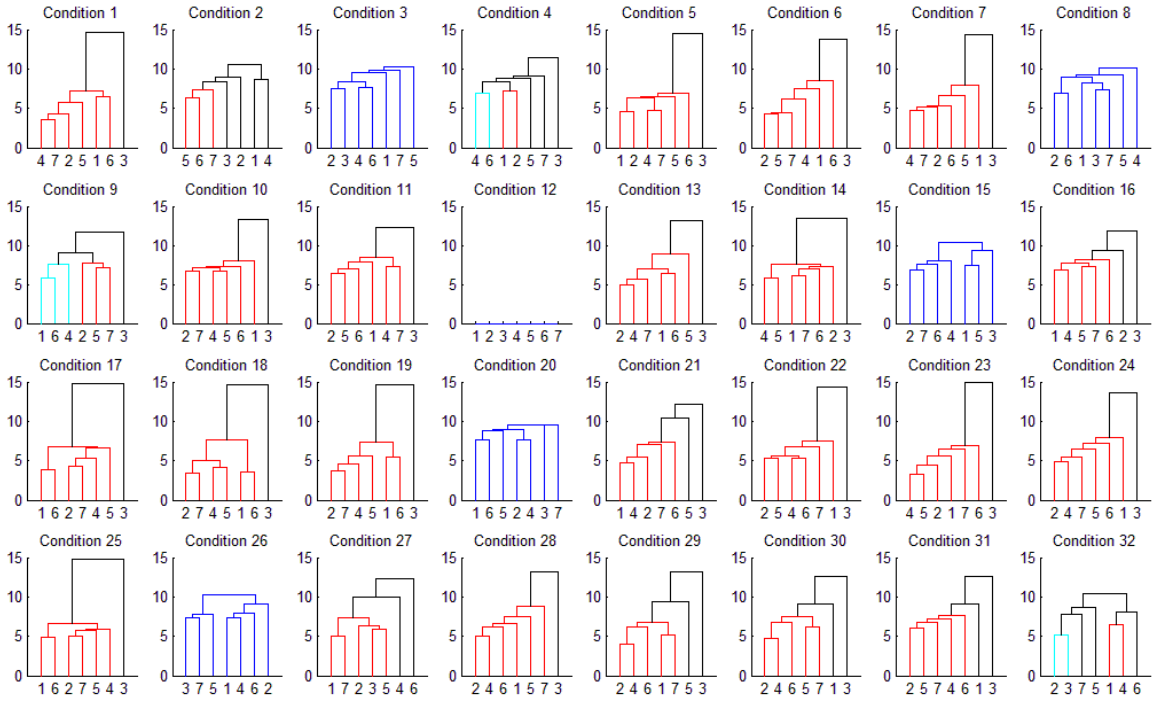


Figure 6-4. Cluster Analysis for 32 Conditions Using Data from Time Period One

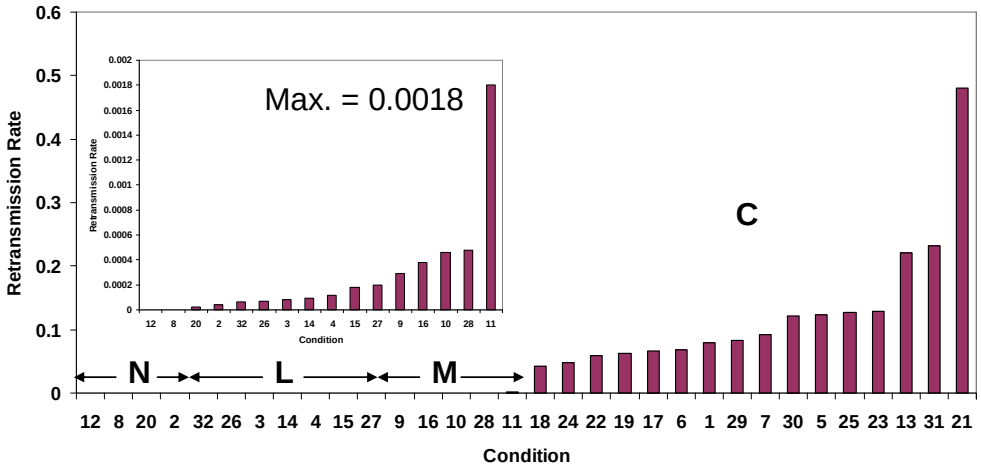


Figure 6-5. Conditions (x axis) Ordered from Least to Most Congested vs. Retransmission Rate (y axis), which is the proportion of all sent packets that were retransmissions

We can select one uncongested and one congested condition to examine more closely. Fig. 6-6 plots several time series that, taken together, show the distribution of flow states for (uncongested) condition 4 under standard TCP congestion control. The x

axis displays time over all three time periods measured for the simulated scenario. The y axis indicates the number of active (red curve) and connecting (yellow curve) flows. Additional curves decompose the active flows by congestion control state. For TP1 (3000-4500) the plot shows that most of the active flows operate in initial slow-start (green curve). This means that the network is sufficiently uncongested that most file transfers complete without a lost packet. Things change during TP2 (4500-6000) as jumbo file transfers induce congestion in the directly connected access routers. Congestion leads to losses, which increases the number of flows operating under normal congestion control procedures (brown curve). As jumbo file transfers diminish during TP3 (6000-7500), congestion ebbs so that, by time 6500, most active flows again complete file transfers without a lost packet. The same curves plotted for the other 15 uncongested conditions show similar patterns.

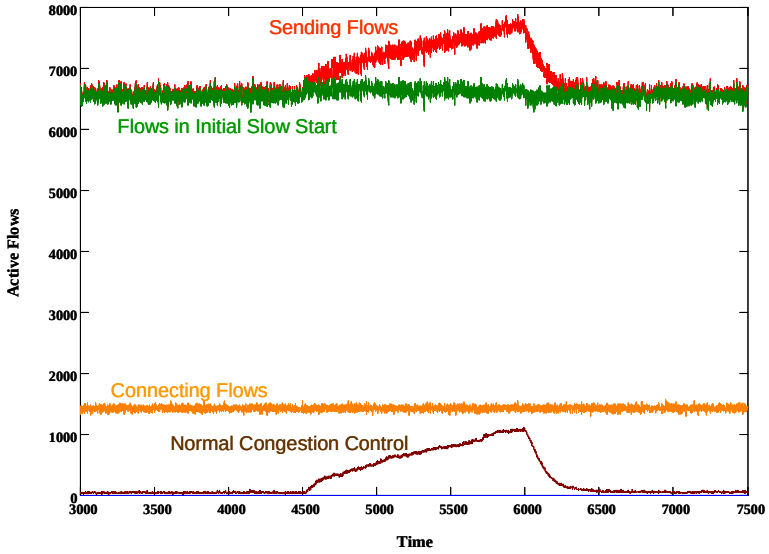


Figure 6-6. Distribution of Flow States over Three Time Periods under Condition 4 for Standard TCP – x axis shows time in 200 ms increments and y axis shows number of active flows in each state

The situation is much different for congested conditions. Fig. 6-7 plots the distribution of flow states for condition five under standard TCP congestion control. Notice that the number of active flows (red) averages about 125 x 10^3. Here, the vast majority (about 105 x 10^3) of those flows are operating under normal congestion control procedures (brown), which means these flows have suffered lost packets. Notice also that network congestion is sufficiently high so that introducing jumbo file transfers during TP 2 (4500-6000) makes very little difference in the overall distribution of flow states. The same curves plotted for the other 15 congested conditions show similar patterns.

Combining this new information with the previous cluster analysis provides substantial insight about conditions that lead to the distinctive behavior of algorithm 3. Fig. 6-8 reproduces an augmented version of Fig. 6-4. Here, we annotate the cluster plot for each condition with a character indicating the relative level of associated congestion. Reviewing the plot reveals that algorithm 3 is distinctive under conditions showing moderate to heavy congestion. The distinctiveness of algorithm 3 fades under conditions

with little or no congestion. Further, under the least congested condition (12), all seven congestion control mechanisms produced identical responses.

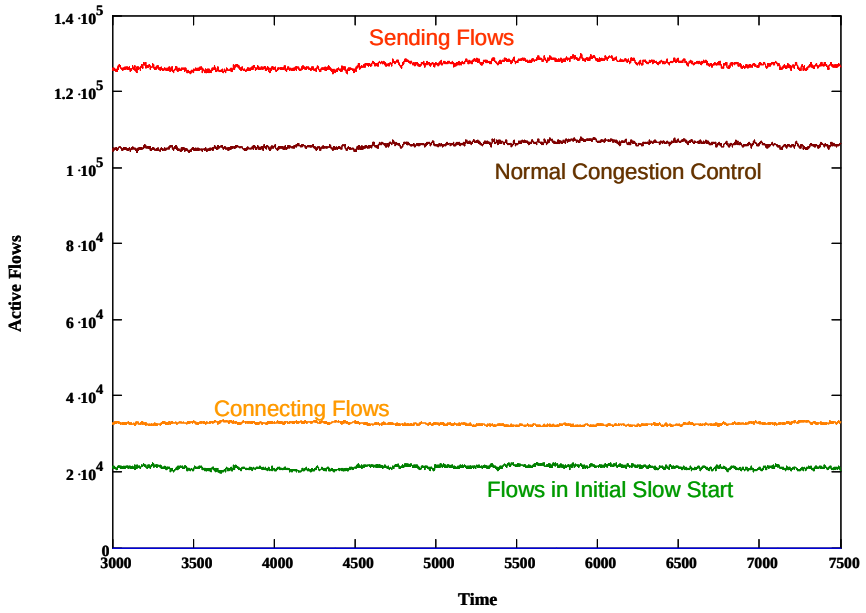


Figure 6-7. Distribution of Flow States over Three Time Periods under Condition 5 for Standard TCP– x axis shows time in 200 ms increments and y axis shows number of active flows in each state

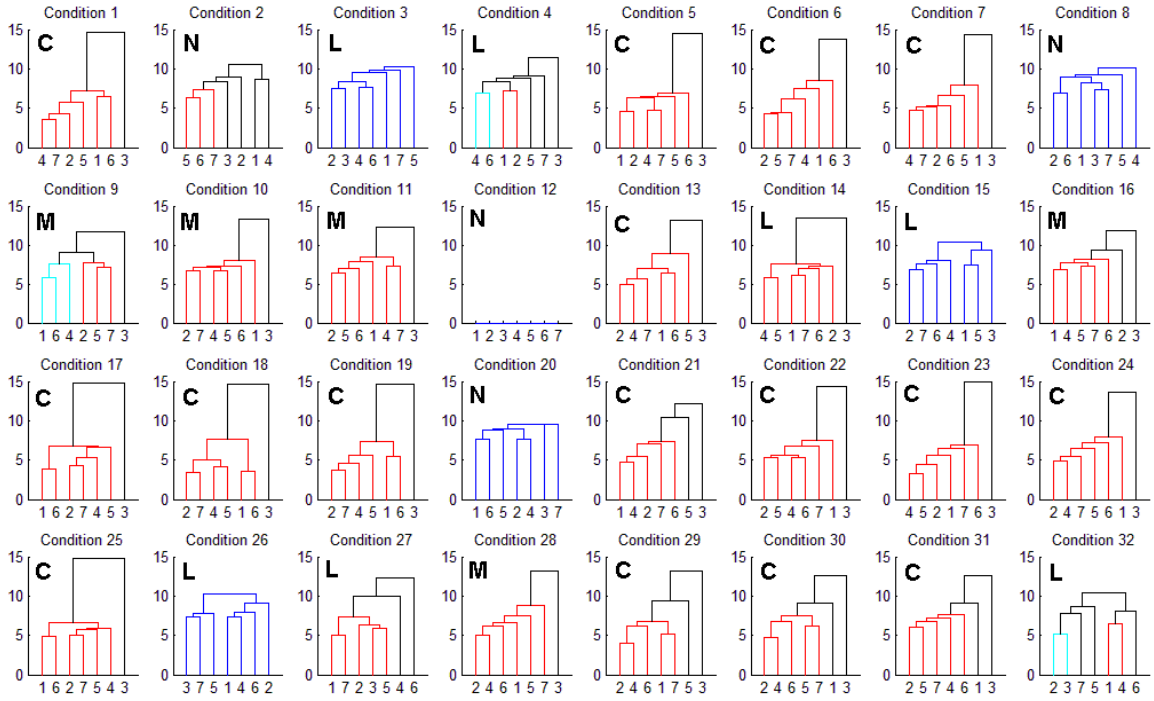


Figure 6-8. Cluster Analysis for Time Period One – Conditions Labeled with Congestion Level

Clustering, combined with some supplementary data analyses, can provide us with a useful overall view of differences among the congestion control algorithms. In our example, during TP1, algorithm 3 shows a distinctive behavior that appears tied to the level of congestion in the network. Further, under little or no congestion, the congestion control algorithms are largely indistinguishable. Unfortunately, cluster analysis does not identify the precise nature of the distinctions among the various alternative congestion control algorithms. For more insight, we need to apply a technique for the detailed analysis of individual responses. We next explain the technique we used to investigate each response.

6.3.2 Detail Analysis of Individual Responses

For each time period, we subjected each response to a statistical analysis for each of the 32 conditions simulated, and we then generated a corresponding plot displaying the relevant information. Such a plot shows, for each condition, which algorithm produced the largest difference (compared to the average for all algorithms) in the response variable. The plot also reports the results of a numerical test to determine whether the largest difference was statistically significant. In addition, the plot reports the absolute and relative magnitudes of the largest effect. We produced (45 x 3 =) 135 plots; each plot represents a single response for a single time period. The best approach to explaining the analysis is to discuss a sample plot, such as Fig. 6-9.

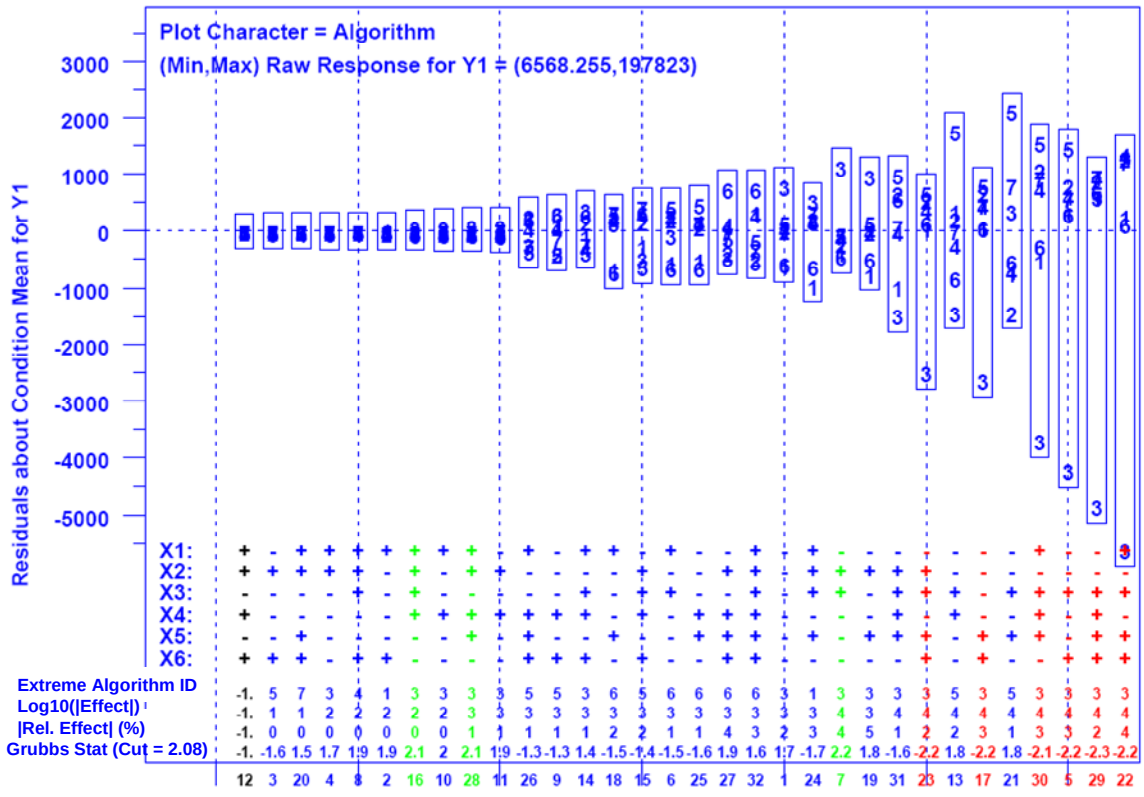


Figure 6-9. Sample Plot Analyzing the Influence of Condition and Congestion Control Algorithm on the Average Number of Active Flows (y1) – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

The x axis in Fig. 6-9 shows the 32 conditions. Here, conditions are sorted by increasing magnitude of the largest difference in the response variable produced by a congestion control algorithm. The upper left corner of the plot gives the minimum and maximum values for the raw responses when considering the data across all algorithms and conditions. The y axis gives the spreads of residuals about the mean. Here, each residual is computed by subtracting the mean response for all algorithms for a given condition from the response for a given algorithm and the same condition. For each condition, we plot a box within which we place algorithm identifiers (1-7). The location of each identifier indicates the distance of the response generated by that algorithm (i.e., the residual) from the mean response over all algorithms for the same condition. Here, the residuals range from zero (all algorithms in condition 12) to about negative 5500 (for algorithm 3 in condition 22).

Below each box we display vertically the settings (+/-) for each input factor (x1…x6) that generated the relevant condition. The remainder of the plot consists of four 32-column rows of quantitative information, where each column gives four statistics applicable to the algorithms and responses for the related condition. The first statistic identifies the extreme algorithm – that is the algorithm with the largest residual. The identifier is listed as -1 when the algorithms cannot be distinguished numerically. This arises for condition 12 in Fig. 6-9. Explicitly listing the extreme algorithm is helpful when the residuals are too close together to be visible in the box – for example in conditions 12 to 26.

The second statistic reports the absolute magnitude (log 10) associated with the maximum residual. The exponent of the absolute magnitude can be reported concisely on the plot at the cost of some numerical precision. The third statistic reports the relative effect as a percentage of the mean response. A domain analyst can consider both the absolute and relative differences when judging whether an effect is significant from an engineering view.

The fourth statistic reports *G*, which results from a Grubbs' test for outlying observations [91] associated with the extreme residual for each condition. The Grubbs' test computes *G* by dividing the largest residual by the sample standard deviation.

$$G \equiv \frac{\max\left(\left|Y_i - \mathrm{mean}(Y)\right|\right)}{s} \tag{3}$$

Assuming no significant differences among congestion control algorithms, we would expect measured residuals to be normally distributed. For this reason, residuals that deviate too far from the mean could be characterized as statistically significant outliers. For our plots we declare an outlier significant (5 %) when $G > 2.08$. The entire column (factors and statistics) is highlighted for conditions where the Grubbs' test identifies an outlier. Green identifies positive outliers (e.g., conditions 16, 28 and 7 in Fig. 6-9) and red identifies negative outliers (e.g., conditions 23, 17, 30, 5, 29 and 22 in Fig. 6-9). Columns are printed in black when no numerical difference could be detected among the responses (e.g., condition 12 in Fig. 6-9). The remaining columns are printed in blue.

What can we conclude from Fig. 6-9 alone? Not much. Algorithm 3 appears as a significant negative outlier under six conditions (all congested). This result could occur

by chance with a probability of about 0.17. Algorithm 3 also appears as a statistically significant positive outlier under three conditions (one congested). This result has about a 0.70 probability of occurring by chance. We selected Fig. 6-9 as an example because it illustrates most traits of such plots. We defer more interesting findings until we discuss pattern seeking in Sec. 6.3.4.2 and overall results in Sec. 6.4. Next, we discuss a technique to summarize our detailed analyses of individual responses. The summary considers all responses and all conditions for each time period.

6.3.3 Condition-Response Summary Analysis

The plots associated with detailed analysis of individual responses can be quite revealing, but they do not give a broad view across responses and conditions in a form similar to that provided by the dendrogram plots from cluster analysis. We can, however, extract information from analyzing individual responses and then present a condensed overview across responses and conditions. Fig. 6-10 shows such a condition-response summary plot for TP1.

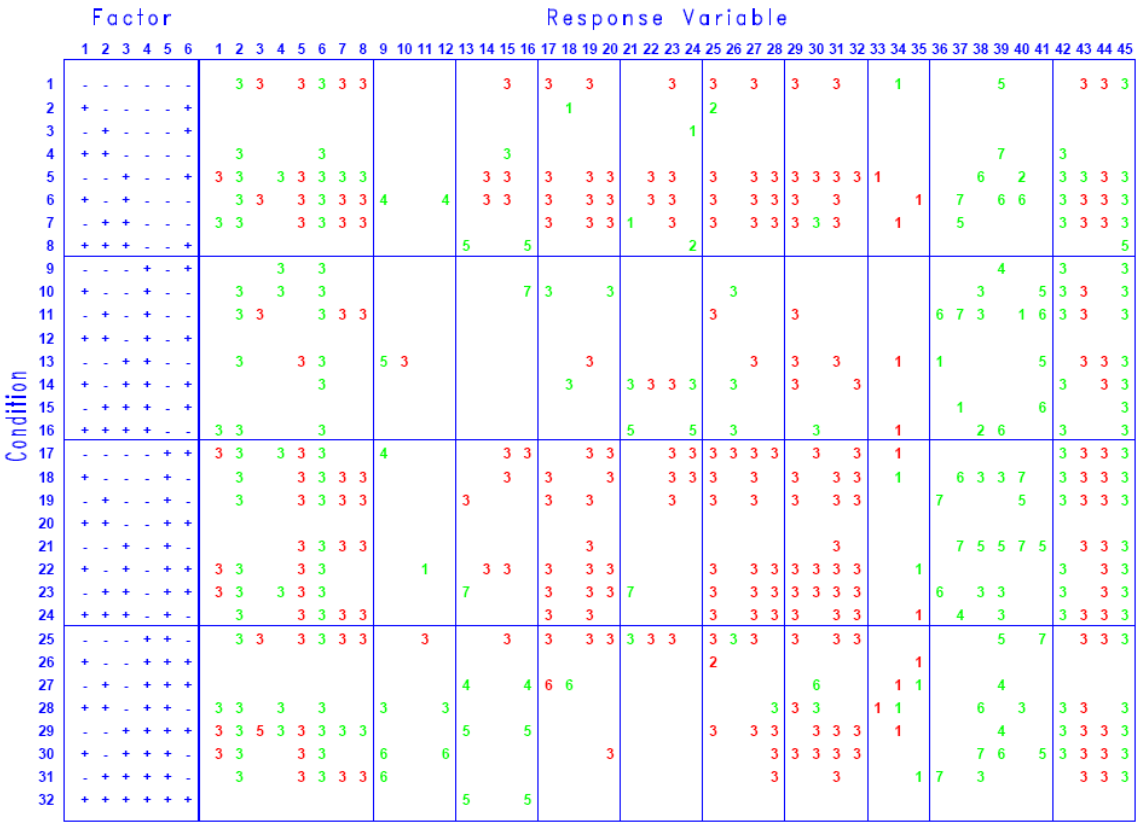


Figure 6-10. Summary of Statistically Significant Outliers in Time Period One – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm manifesting a Statistically Significant Outlier, where blank cells mean there was no significant outlier

Each row in Fig. 6-10 corresponds to a specific condition (identified on the left). The first six columns report settings (+/-) for the six input factors defining the condition. The remaining columns represent individual responses. Vertical blue lines group related responses. For example, responses 1 through 8 relate to macroscopic behavior, responses

9 through 12 relate to goodput on DD flows, responses 42 through 45 relate to distribution of flow states and so on. Cells, formed by condition-response intersections, contain an algorithm identifier when there is a statistically significant outlier – red denotes low outliers and green denotes high outliers.

A scan of Fig. 6-10 shows that algorithm 3 arises as a statistically significant outlier in many cells. The highest concentration of outliers appears for congested conditions; fewer outliers appear for less congested conditions. No algorithm appears as an outlier for condition (i.e., row) 12. These results agree with the cluster analysis (recall Fig. 6-4) for the same time period. Both analyses identify algorithm 3 as distinctive under congested conditions. Fig. 6-10 has the advantage of identifying precisely the particular responses for which algorithm 3 exhibits different behavior.

To focus analysis on the most significant behavioral differences, we can apply various filters when generating a condition-response summary plot. For example, Fig. 6-11 shows a summary plot reporting statistically significant outliers that also achieve a relative difference greater than 10%. The pattern of outliers is now sparser, so we can focus our analysis on responses y2 (congestion window increase rate), y6 (retransmission rate), y42 (average number of connecting flows), y44 (average number of active flows in normal congestion control mode) and y45 (average number of active flows in alternate congestion control mode). The responses measuring buffer usage (y36 – y41) exhibit outliers but there is no evident pattern.

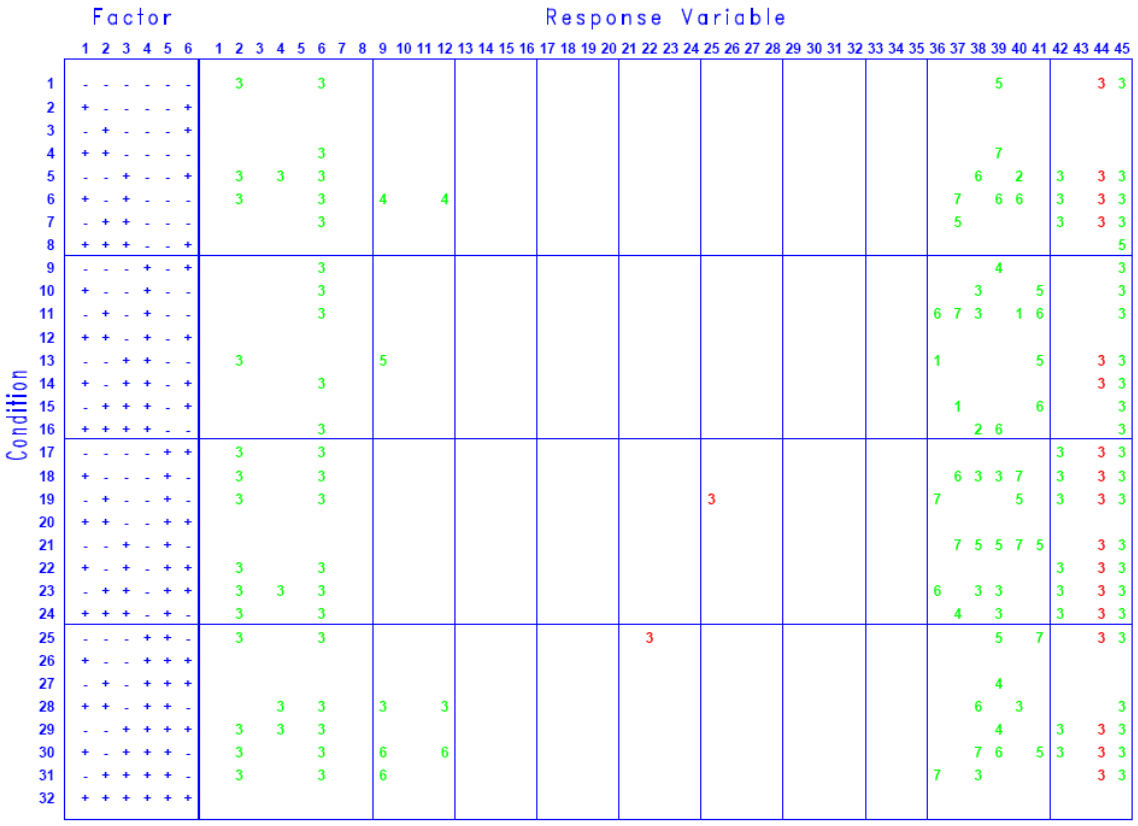


Figure 6-11. Filtered Summary Plot for Time Period One Identifying Statistically Significant Outliers with Associated Relative Effect > 10%

We can exclude y44 and y45 from further consideration because algorithm 3 (FAST) never operates in normal congestion control mode. This means that we should expect algorithm 3 to be an outlier exhibiting a large effect for responses y44 and y45. This is certainly the case in all the analyses we conducted. With this knowledge, we completed a revised cluster analysis with responses y44 and y45 excluded. The revised clustering results (reported in Sec. 6.4) continue to identify algorithm 3 as distinctive.

Fig. 6-11 suggests that algorithm 3 is most different with respect to response y6 – retransmission rate. Here, algorithm 3 produces retransmission rates more than 10 % higher than the other algorithms in 21 of the 32 conditions. In 12 conditions the retransmission rate for algorithm 3 is more than 30 % higher – more than 50 % higher in five conditions. Clearly, this is a significant finding, which we discuss more fully in later sections.

Fig. 6-11 also shows that for 14 conditions algorithm 3 (FAST) produces more than a 10 % higher rate of window increase than the other algorithms. All 14 conditions are among the most congested. Recall from Chapter 5 that FAST aims to provide a stable congestion window that reaches equilibrium, changing very little over time. The simulations in Chapter 5 also showed that when FAST had insufficient buffers a rapid oscillating behavior ensued where the congestion window was cut in half on a loss and then quickly increased up to another loss and so on. Under these rapid oscillations, FAST would tend to increase congestion windows very frequently. Thus, under FAST, the larger the retransmission rate, the higher the rate of window increases.

What about y42 (average number of connecting flows)? A high retransmission rate arises from a high loss rate. To establish flows, a source and receiver must exchange SYN and SYN+ACK packets. Since these packets are also subject to being lost, we expect that a high loss rate can impede connection establishment. This means that on average more SYNs must be sent to connect a flow. Thus, given a higher retransmission rate for algorithm 3, we should expect more flows to be pending in the connecting state.

This discussion illustrates that condition-response summary plots can be quite powerful – allowing an analyst to identify key differences separating algorithms. In Sec. 6.4 we report summary plots for all three time periods, as well as for the aggregate responses. As we will demonstrate, the summary plots impart substantial insight regarding system behavior.

6.3.4 Data Exploration

In previous sections we introduced the main techniques we used to analyze system behavior. We augmented these analysis techniques with some exploratory approaches that allowed us to investigate specific questions. In this section we briefly describe and illustrate selected augmentations.

6.3.4.1 Extrapolating from Time Series. MesoNet samples responses at each measurement interval and produces related time series. We generate our summary responses by averaging time series of interest over particular intervals. As discussed in Sec. 6.3.1, an analyst may examine raw time series as necessary to gain additional insight. Here, we give an example that illustrates pitfalls that may arise when focusing on time series for only a few selected conditions.

Fig. 6-12 plots seven time series (one for each congestion control algorithm) for condition 4, a lightly congested condition. Each time series reports the average goodput for DD flows (y9) over the final three time periods (15 minutes total) of the experiment scenario. The plot shows the general effect of the scenario on DD flows. During the first time period (3000-4500) average per-flow goodputs fluctuate in the neighborhood of 500 pps. Jumbo flows commence at time 4500, which leads to a rapid increase in average goodput up to around 10^4 pps. As additional jumbo flows arrive, average goodput falls as bandwidth must be shared among more flows. New jumbo flows cease to arrive starting at time 6000, which enables average goodput to increase as residual jumbo flows are cleared. As the mix of flows moves away from jumbo flows and back to normal Web traffic, average goodput trails off. Had the scenario continued, all residual jumbo flows would eventually clear the system and average goodput would return to levels seen in the first time period. This general behavior is representative of the time varying scenario across all conditions. Fig. 6-13 plots seven time series for the number of active DD flows (y10) over the same time periods and under the same condition.

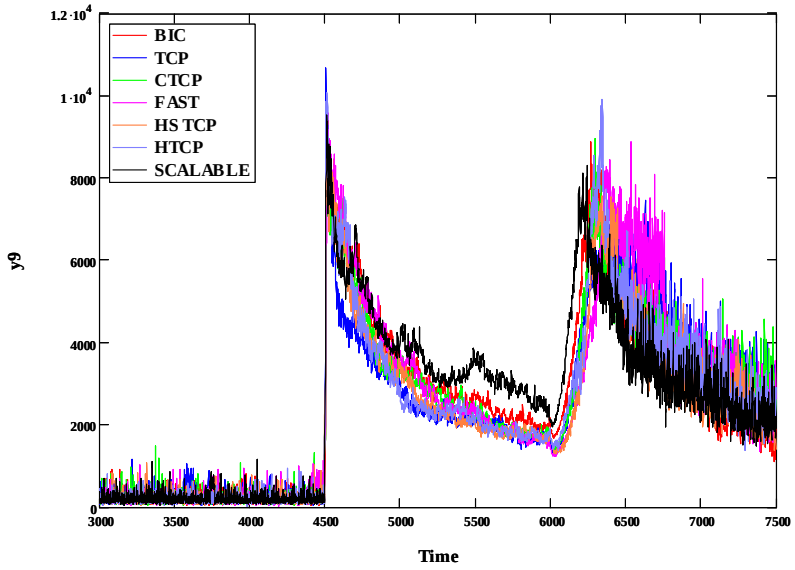


Figure 6-12. Average Per-Flow Goodput on DD Flows (y9) for Seven Congestion Control Algorithms under Condition 4 over Three Time Periods – x axis gives time in 200 ms increments and y axis gives goodput in packets per second

Fig. 6-12 indicates that Scalable TCP (black curve) provided higher average goodput during the period of jumbo file transfers. Recall that in Chapter 5 we found that under a restricted topology with few flows, Scalable TCP tended to provide unfair allocation of bandwidth. Does relative unfairness relate to the behavior shown in Fig. 6-12? The current simulation scenario was set up to ensure that a concentration of jumbo files would be transferred on DD flows between times 4500 and 6000. Yet, Fig. 6-13 reveals that Scalable TCP (black curve) has the fewest number of active DD flows in that time period; BIC (red curve) has second fewest. Given a finite (bottleneck) capacity to deliver packets, each flow will naturally receive higher average goodput when the bottleneck is shared by fewer flows. Fig. 6-14 shows that a bottleneck capacity exists, as

the total rate of packets delivered on DD flows (y12) during the second time period reaches a level of just under 2 million pps for each of the congestion control algorithms.

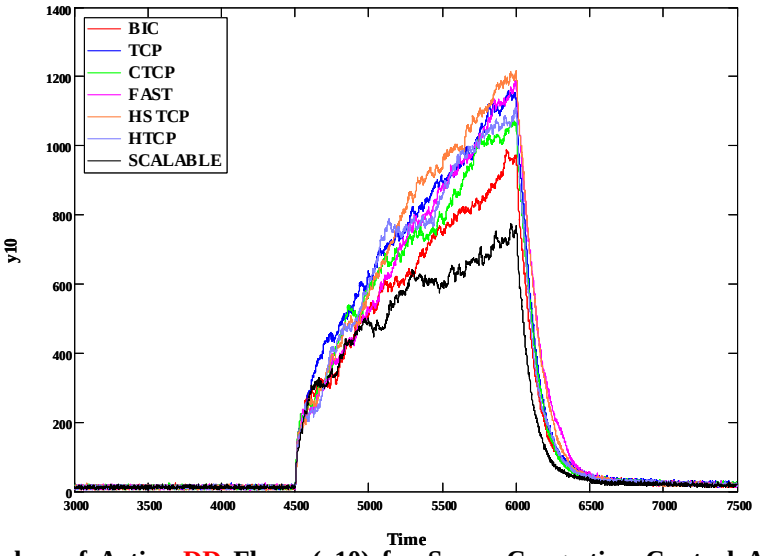


Figure 6-13. Number of Active DD Flows (y10) for Seven Congestion Control Algorithms under Condition 4 over Three Time Periods – x axis gives time in 200 ms increments and y axis number of active flows

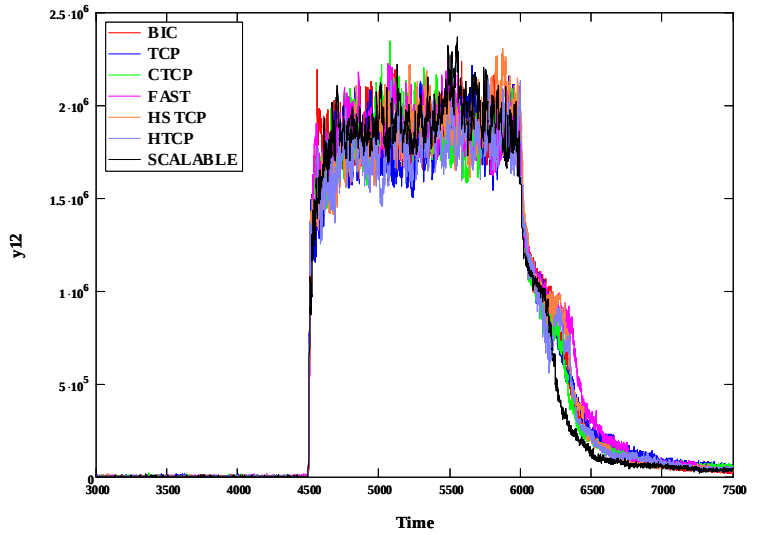


Figure 6-14. Aggregate Packet Delivery Rate DD Flows (y12) for Seven Congestion Control Algorithms under Condition 4 over Three Time Periods – x axis gives time in 200 ms increments and y axis packet delivery rate in packets per second

How can we explain the fact that fewer jumbo flows are active simultaneously under condition 4 in TP2 for Scalable TCP? The answer can be found by examining the completion rate for DD flows (y11) during TP2, as shown in Table 6-31. Scalable TCP completes slightly more (.3 to .4) DD flows per measurement interval than other congestion control algorithms. Remember that the measurement interval is only 200 ms in duration. Considered over the entire 5 minutes (1500 measurement intervals) comprising TP2, Table 6-31 shows that Scalable TCP completes 500 to 600 more DD

flows. More flows completed per unit time leads to fewer active flows, which yields higher goodput per flow.

Table 6-31. Flows Completed per 200 ms interval and Total Completions for DD Flows in Time Period Two under Condition 4

Algorithm	DD Flow Completion Rate	DD Flows Completed in Time Period 2
BIC	7.74	11.610×10^3
CTCP	7.60	11.401×10^3
FAST	7.57	11.353×10^3
HSTCP	7.42	11.133×10^3
HTCP	7.54	11.307×10^3
SCALABLE	7.88	11.814×10^3
TCP	7.53	11.296×10^3

Does this behavior repeat across a wide range of conditions? In selected uncongested conditions (such as 8 and 12) Scalable TCP provides the worst goodput for DD flows during TP2. An overall examination of y9 across all conditions (see Fig. 6-15) reveals no particular pattern, which illustrates why we must rely on comprehensive results and not focus in detail on particular conditions to the exclusion of others.

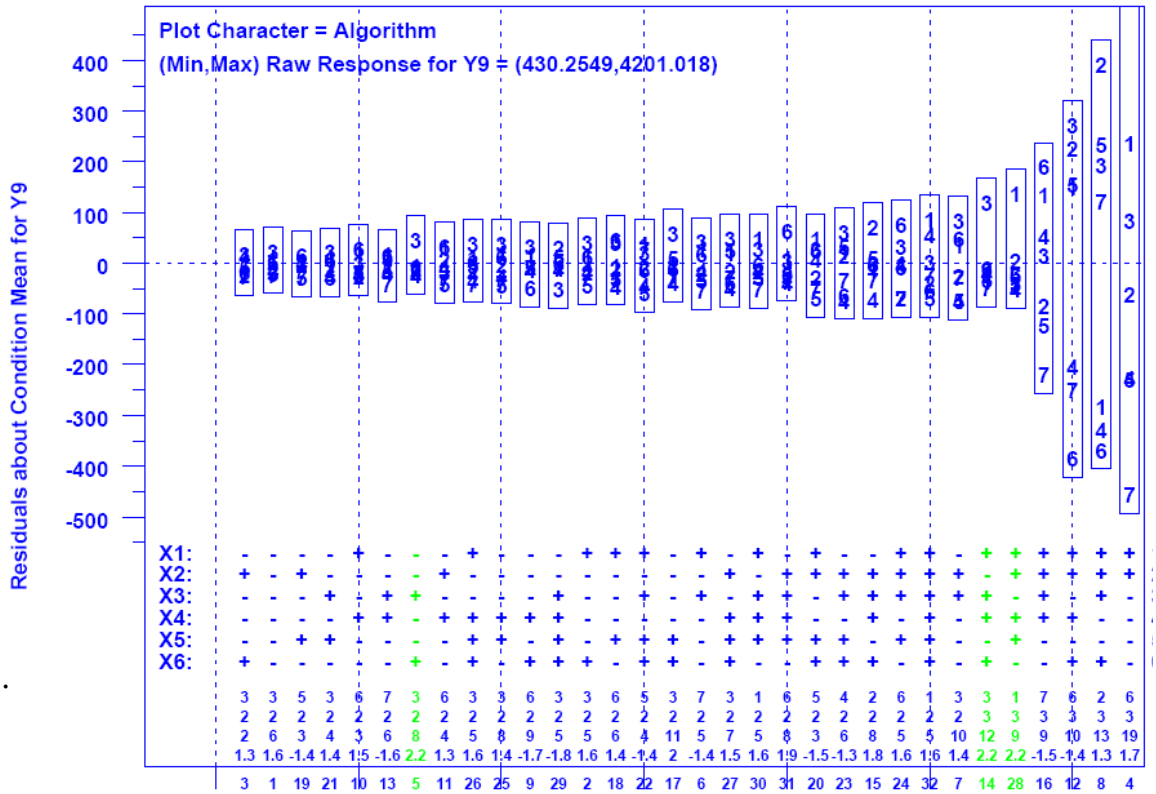


Figure 6-15. Analyzing the Influence of Condition and Congestion Control Algorithm on the Average Goodput (pps) for DD Flows (y9) during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

In summary, examining time series for individual conditions can provide interesting and useful information regarding the movement of system responses over time. Unfortunately, one must examine a response across all conditions in order to reveal general trends and patterns. Comprehensive examination across responses and conditions can best be achieved through cluster analysis and condition-response summary plots. Examination of individual time series must be used for focused purposes, such as causality analysis, and not to draw general inferences.

6.3.4.2 Seeking Patterns. Above, we characterized Fig. 6-15 as revealing no particular pattern. This implies that we will be seeking patterns not only in the dendrogram and condition-response summary plots but also in plots giving detailed analyses of individual responses. Such patterns correspond to columns in a condition-response summary plot where identical algorithms are reported as statistically significant outliers across a substantial number of conditions. For example, Fig. 6-16 illustrates a pattern in a detailed analysis plot for congestion window size (y4) during TP3. Comparing Fig. 6-15 and Fig. 6-16 illustrates the difference between a non-pattern and a pattern. The pattern in Fig. 6-16 reports that during TP3 algorithm 2 (CTCP) yields a congestion window significantly larger than the other algorithms. This result occurs in 30 of the 32 conditions (in conditions 3 and 32 algorithm 3 is the outlier) and is statistically significant (5 %) in 28 of the 30 conditions, and nearly significant in the other two (i.e., in conditions 19 and 29 the Grubbs' test statistic is 2, which is just below the 2.08 cutoff for 5 % significance).

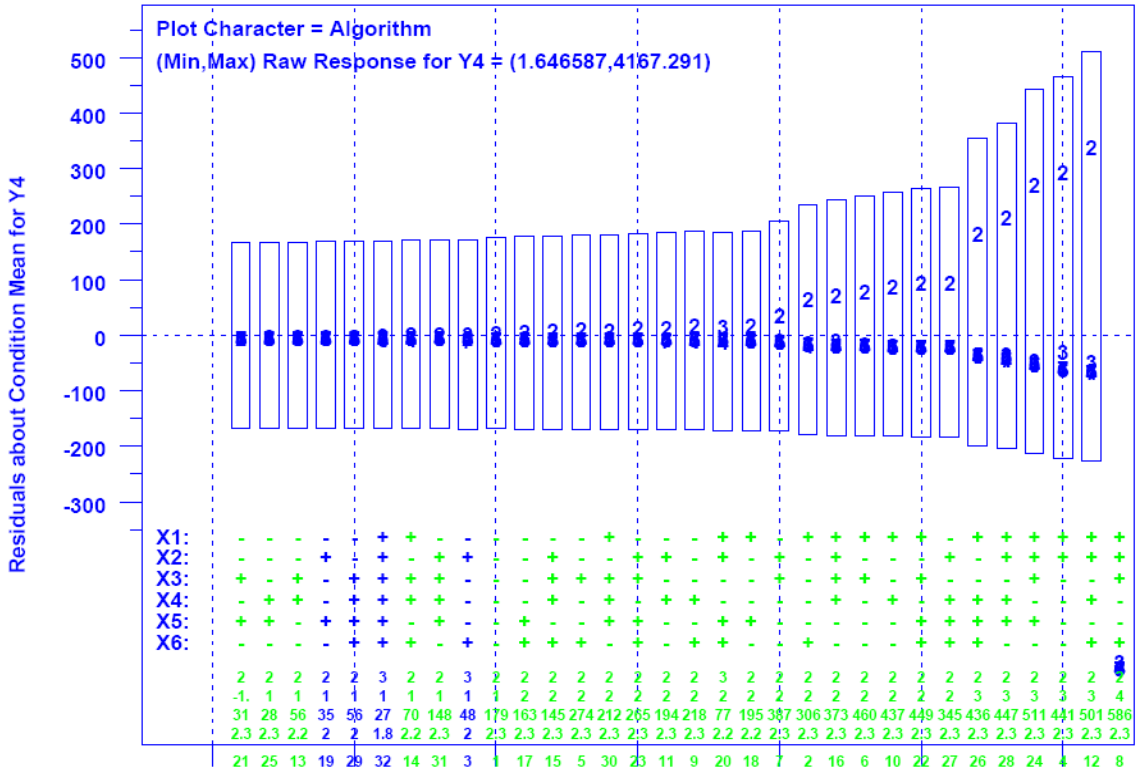


Figure 6-16. Analyzing the Influence of Condition and Congestion Control Algorithm on Congestion Window Size (y4) in packets during Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.3.4.3 Investigating Data Subsets. In cases where a summary plot reveals one particular algorithm as distinctive, an analyst may naturally wonder whether the distinction might be sufficient to mask more subtle distinctions among the remaining algorithms. To investigate such questions, one can exclude response data for the distinctive algorithm and then reconsider the analysis on the remaining subset of response data. For example, Fig. 6-17 gives dendrograms resulting from a cluster analysis for TP1 when response data for algorithm 3 is omitted. The resulting plot reveals that the responses are very similar across the remaining algorithms in about half the conditions. For some conditions there appears to be a slight tendency for algorithms 1 (BIC) and 6 (Scalable TCP) to cluster together, while algorithm 5 (HTCP) is somewhat distinctive under four conditions. Overall, the cluster analysis for TP1 with algorithm 3 excluded shows the behavior among the remaining algorithms to be largely indistinguishable. There appears some tendency for algorithms 1 and 6 to exhibit slightly similar behaviors somewhat different from other algorithms. A condition-response summary plot for the same subset of data identifies few statistically significant outliers.

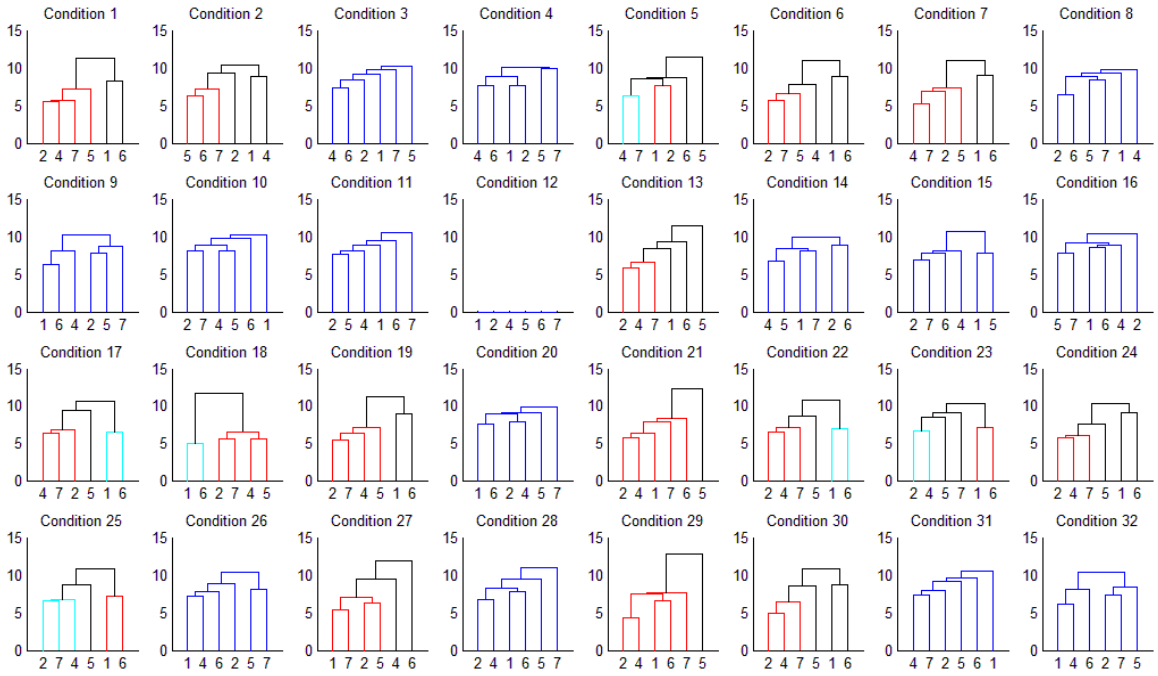


Figure 6-17. Cluster Analysis Using Data from Time Period One – Algorithm 3 Excluded

6.3.4.4 Interactive Animation. MesoNet produces a large amount of data – simulation of a congestion control algorithm under one condition can produce about 165 Mbytes of measurements. Much of the data relates to temporal behavior in individual routers in the network topology. Such data naturally lends itself to animation within a layout of the network topology. To accommodate such animation, as well as to support abstract analysis of multidimensional data, colleagues produced DiVisa [86], an interactive system for multidimensional data visualization. DiVisa, freely available for public use, requires only access to a Java™ run-time environment, so DiVisa is portable to a range of operating systems.

Fig. 6-18 depicts a sample screenshot where we used DiVisa to monitor packet losses throughout the network topology for algorithm 1 (BIC) under condition 10. The screenshot shows three main panels: a (leftmost) visualization-control panel and two plots. The control panel permits a user to define plot characteristics. In this case, we assigned the leftmost plot panel to hold the network topology (routers and links only), while the rightmost plot panel graphs packet losses over time. Further, in the topology panel we assign color to represent the rate of packet losses – from orange for minimal losses to red for high losses. The particular screenshot shows one frame from an animation of the evolution of packet losses – the animation has reached time 5510, which is within TP2. At that time, only two routers in the topology show any appreciable losses: access router I0a (yellow) and access router K0a (blue). We can select specific routers in the topology and the related curve in the time plot will be emphasized. We can also interactively explore other router characteristics, such as utilization and buffer saturation. DiVisa animations helped us discover that backbone routers could be overrun under some conditions in TP2. Using this information, we increased the simulated speed of our backbone routers. DiVisa animations also helped us to determine that access router K0a was the most heavily utilized of the access routers during TP2. In summary, availability of a data exploration tool and animator, such as DiVisa, can help an analyst gain global views of spatiotemporal patterns in a simulated system. Of course, one must remember that looking at animations of individual time series does not provide sufficient information to discern significant overall patterns across conditions and responses

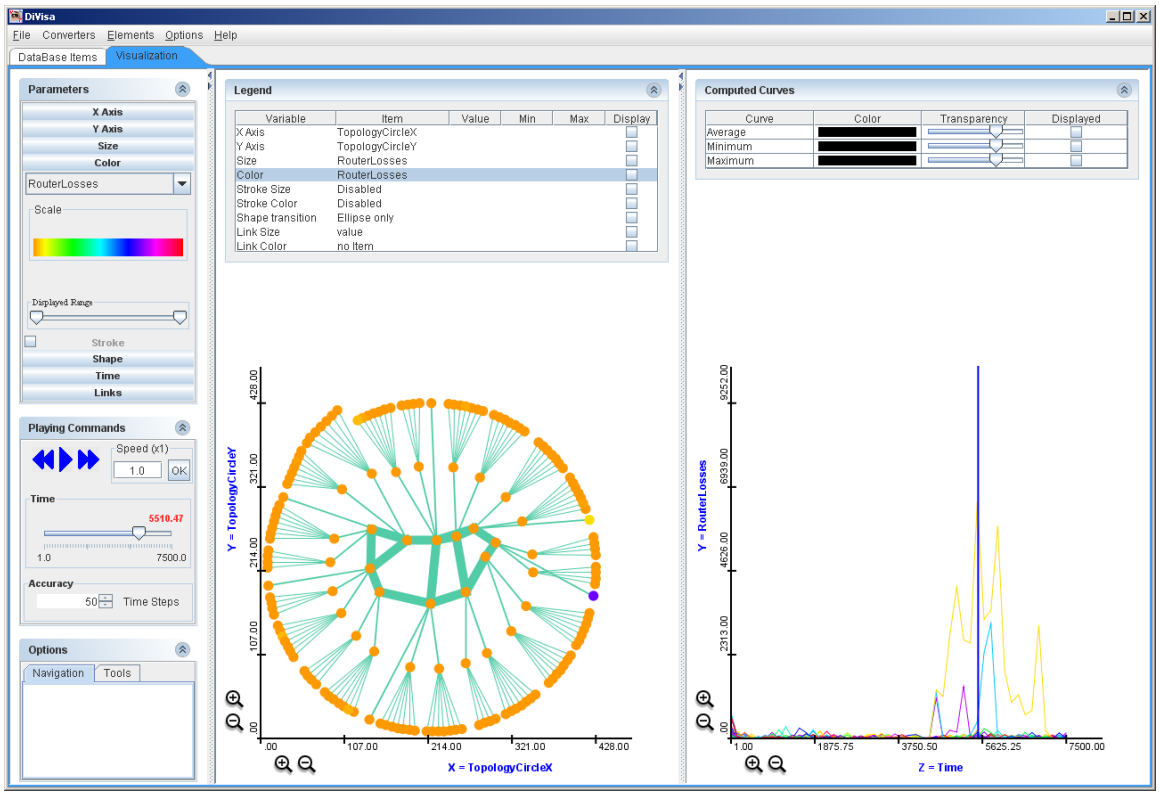


Figure 6-18. Screenshot from DiVisa Animation of the Temporal Evolution of a MesoNet Simulation

6.4 Results

In this section, we report salient results from our analysis of summarized response data (described in Sec. 6.2.2). As necessary, we provide brief commentaries to explain the results presented. We give results in four segments: one for each of the three 5-minute time periods and one for response data aggregated over the entire 25-minute scenario. We follow a similar plan for each segment: (1) present results from cluster analysis, (2) present results from condition-response summaries, (3) present detailed analysis of significant responses and (4) give a summary of the results for the segment. We defer drawing inferences from the results until Sec. 6.5, where we report our findings.

6.4.1 Time Period One (TP1)

Recall that TP1 comprises a five-minute period where three long-lived flows commence within an overall background of normal Web traffic, which includes downloading Web pages, and occasionally documents. As for any time period, we consider seven congestion control algorithms under a range of 32 conditions, where half the conditions can be considered uncongested and half congested.

6.4.1.1 Cluster Analysis for TP1. We present two dendogram plots for TP1. Fig. 6-19 gives the cluster analysis for all seven congestion control algorithms. We annotate the individual dendograms with a 3 when algorithm 3 appears distinctive. Fig. 6-20 gives the cluster analysis after omitting response data for algorithm 3.

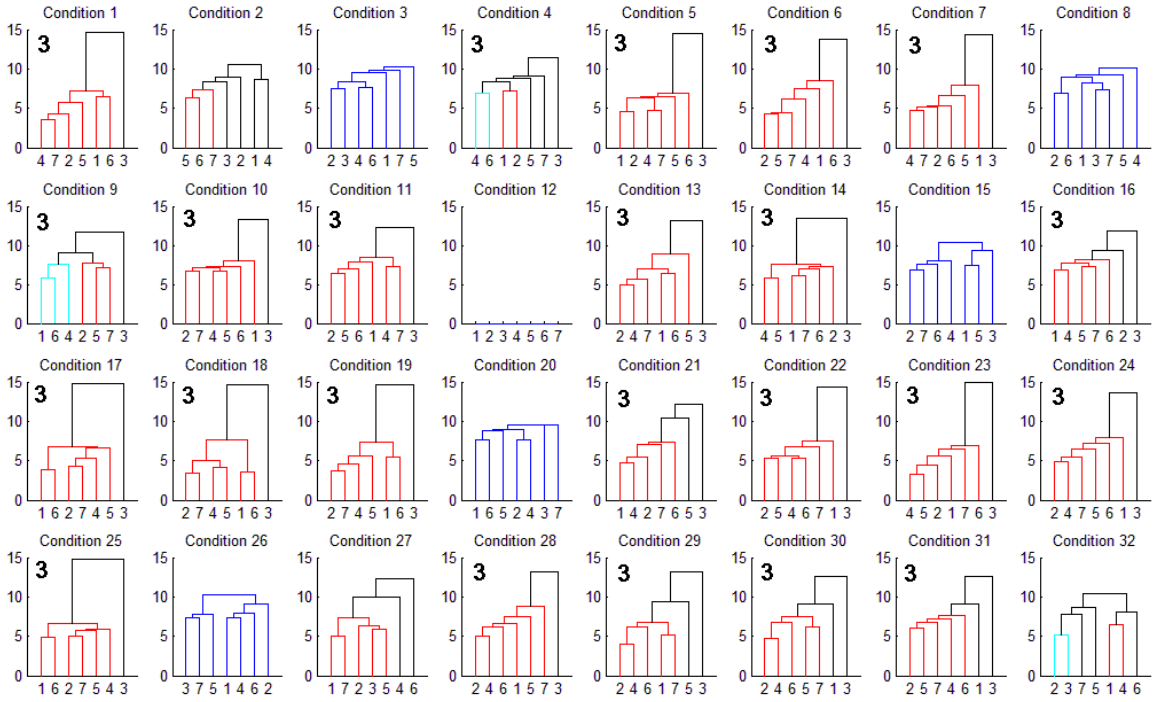


Figure 6-19. Clustering for Time Period One – Annotated to Identify Distinctive Algorithm 3

6.4.1.2 Condition-Response Summary for TP1. Fig. 6-21 gives the condition-response summary for TP1. Fig. 6-22 shows the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 10 %.

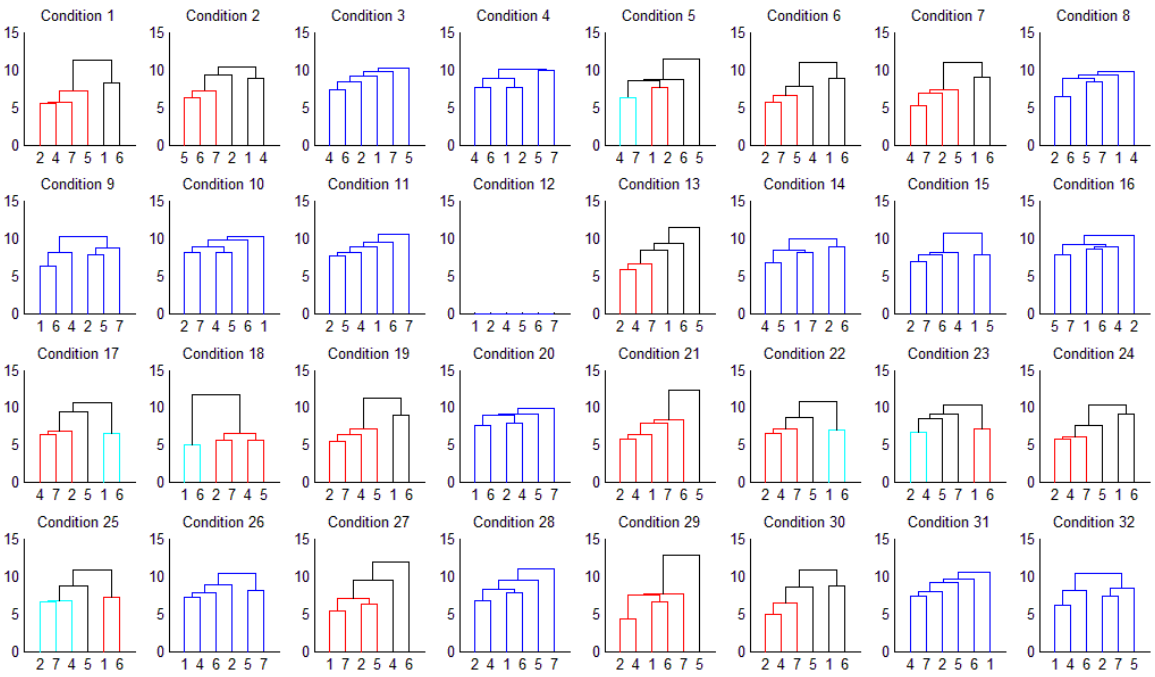


Figure 6-20. Clustering for Time Period One – Algorithm 3 Omitted

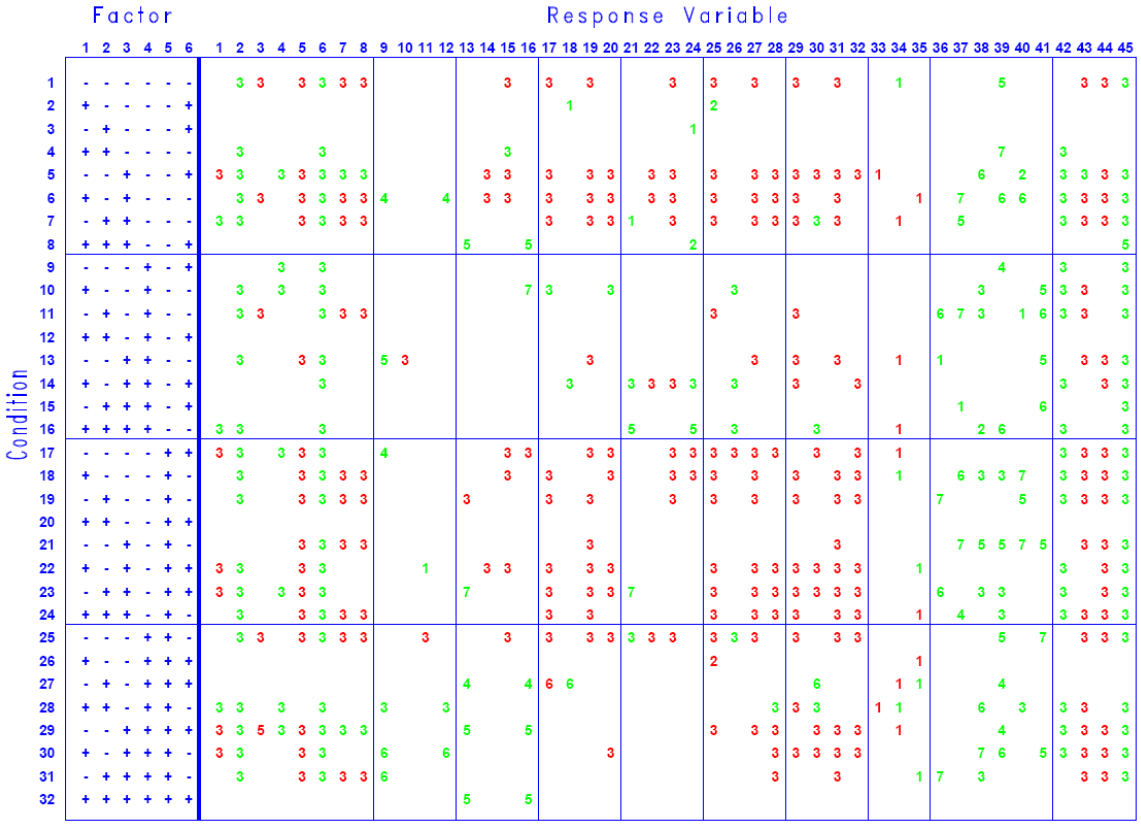


Figure 6-21. Condition-Response Summary for Time Period One – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm manifesting a Statistically Significant Outlier

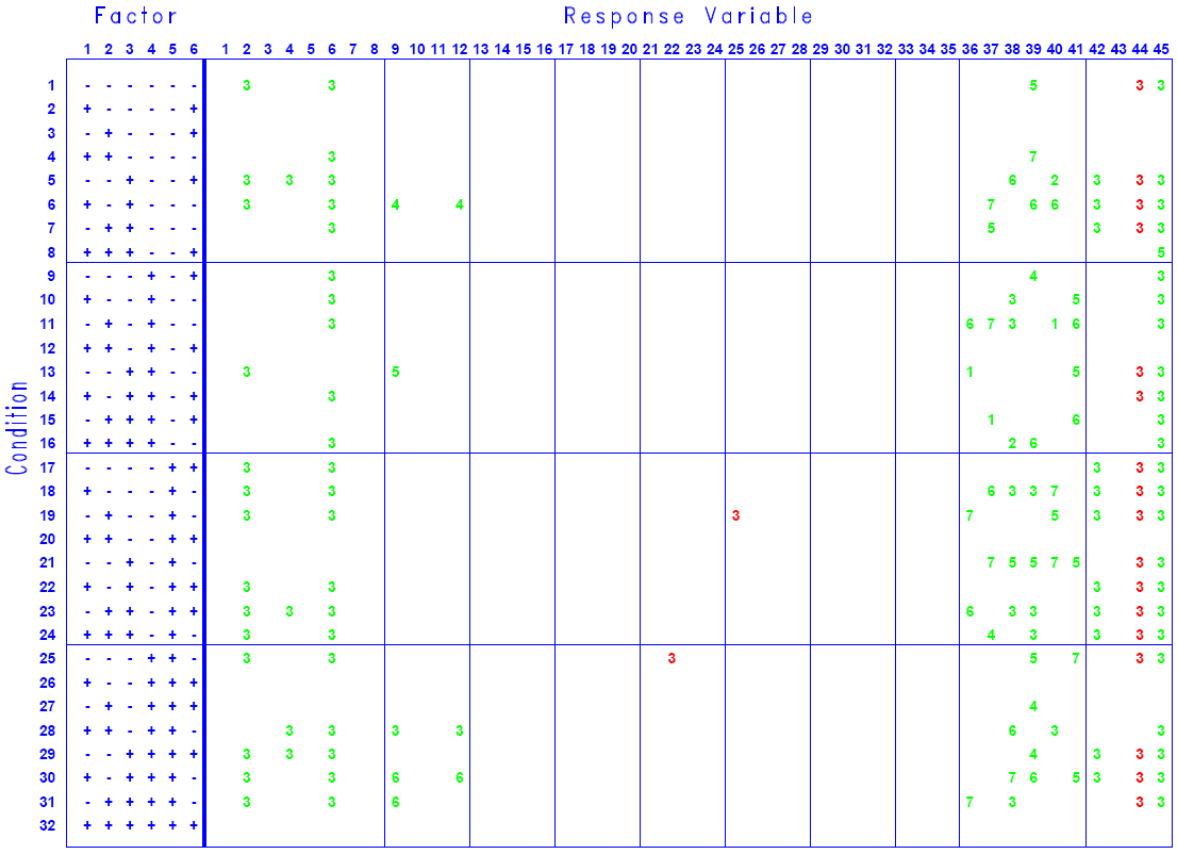


Figure 6-22. Filtered Summary Plot for Time Period One Identifying Statistically Significant Outliers with Associated Relative Effect > 10%

6.4.1.3 Analysis of Significant Responses for TP1. Based on Figs. 6-21 and 6-22 we selected several responses for more detailed analysis. Specifically, in Figs. 6-23 to 6-27, we report analyses for congestion window increase rate (y2), flow completion rate (y5), retransmission rate (y6), completion rate for NN flows (y31), and average number of connecting flows (y42). We omitted y44 and y45 because (as we explained earlier) they provide little insight into differences in behavior among the congestion control algorithms. We selected y5 and y31 based on Fig. 6-21 even though they did not pass the 10 % filter required for reporting in Fig. 6-22. We made these additional selections because the absolute magnitude of each effect within an individual measurement interval appears large enough to influence system behavior when accumulated over time. While we could have chosen completion rates for other flow classes (e.g., y19 or y25), NN flows make up the largest proportion of all flows active at any given time, so the significance of this flow class appears to be highest.

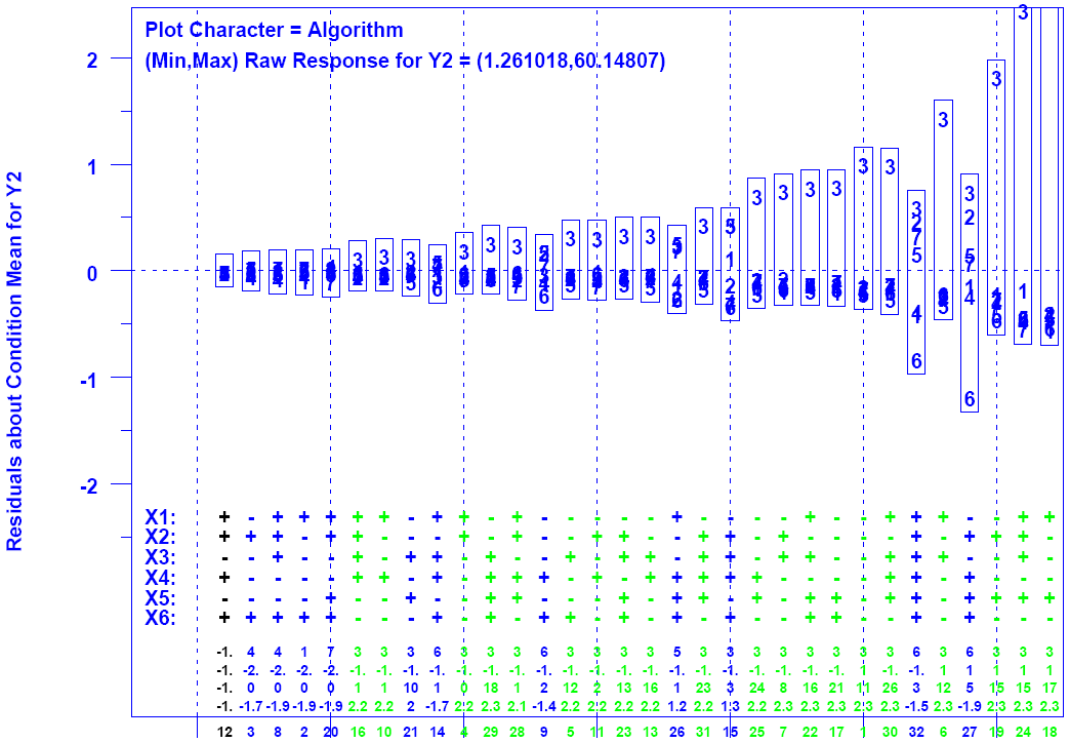


Figure 6-23. Detailed Analysis for Congestion Window Increase Rate Per Flow (increases per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

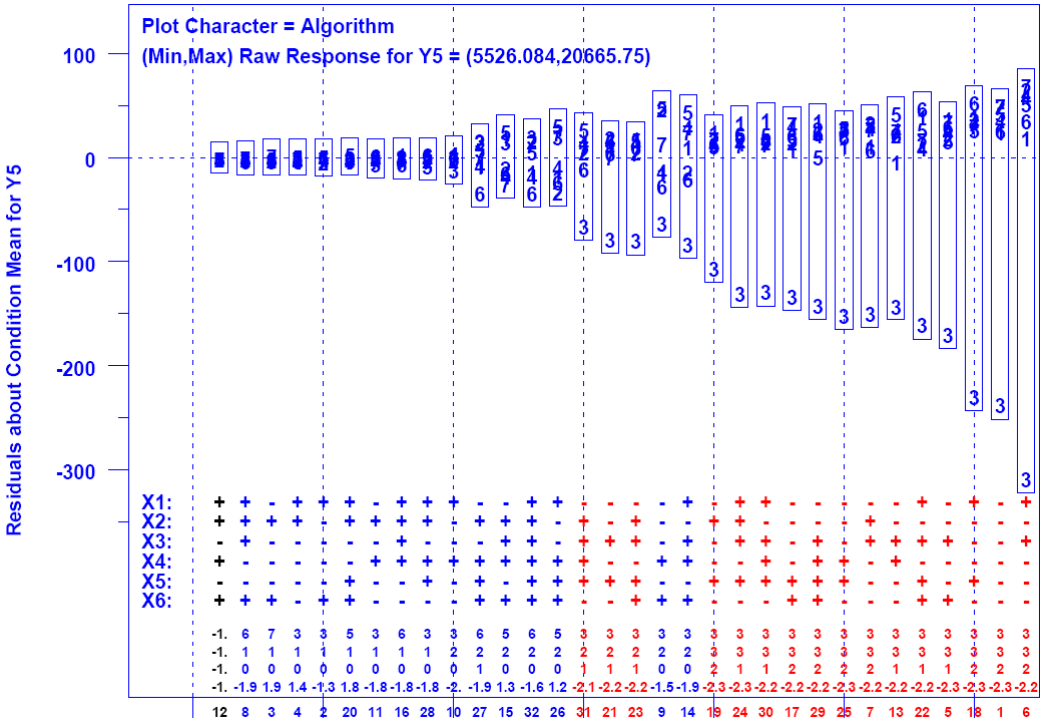


Figure 6-24. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

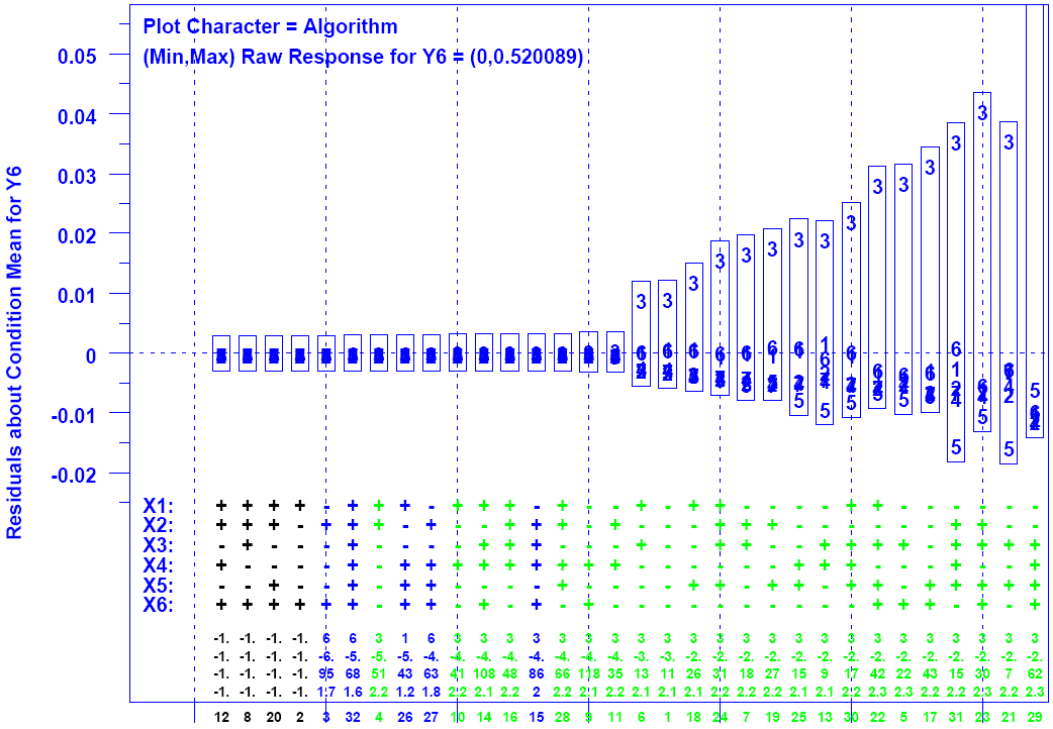


Figure 6-25. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

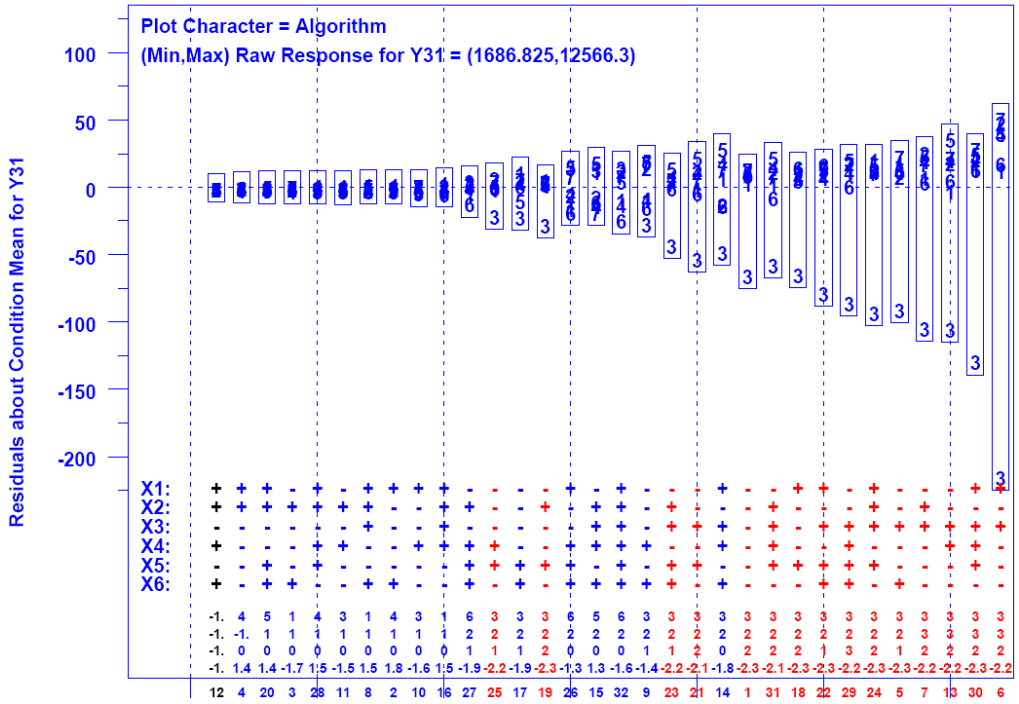


Figure 6-26. Detailed Analysis for NN Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

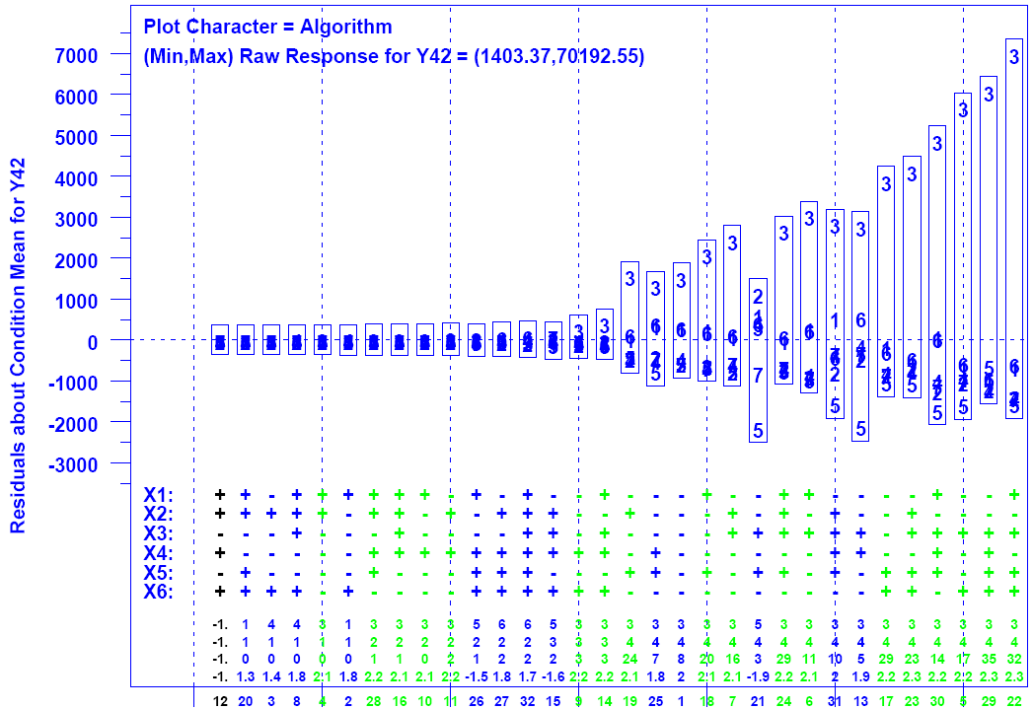


Figure 6-27. Detailed Analysis for Number of Connecting Flows in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.4.1.4 Summary of Results for TP1. Given normal Web traffic, FAST (algorithm 3) exhibits distinctive behavior, which appears to grow more distinctive with increasing congestion. The other algorithms behave quite similarly under most conditions, though BIC (algorithm 1) and Scalable TCP (algorithm 6) appear to cluster together under some conditions. When faced with congestion, FAST exacerbates the situation, as shown by the higher rate of increase in congestion windows, which leads to more packet losses and then to a higher rate of retransmissions. Increased losses under FAST also appear to increase the difficulty for establishing flows because more SYN and SYN+ACK packets are lost – as a result, on average more flows are pending in the connecting state. Increased retransmissions also cause flows to send more packets in order to ensure all data is successfully received. This means that flows take longer to finish, as shown by the lower completion rate for flows in general and for NN flows in particular.

6.4.2 Time Period Two (TP2)

During TP2 DD flows become jumbo file transfers, which lead to increased congestion within directly connected routers and also increases packet load on the network backbone. The remaining flow classes continue to generate normal Web traffic during TP2, but the net effect of adding the jumbo flows is to increase network-wide congestion.

6.4.2.1 Cluster Analysis for TP2. Fig. 6-28 shows an annotated set of 32 dendrograms for TP2. Since the level of congestion has increased throughout the network and algorithm 3 appears sensitive to congestion, one might expect the behavior of algorithm 3 to become more distinctive. Note that algorithm 3 now appears as distinctive in 28 of the conditions

– versus only 23 conditions in TP1. Fig. 6-29 gives dendrograms for TP2 but with the data for algorithm 3 omitted – none of the other algorithms stand out.

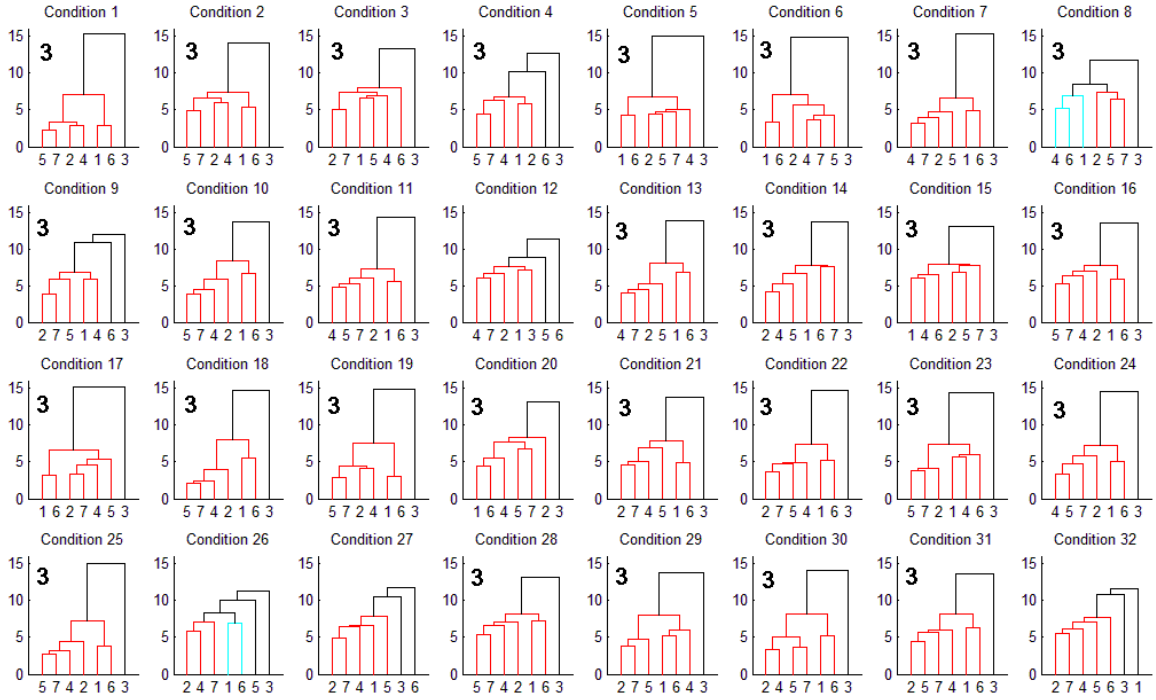


Figure 6-28. Clustering for Time Period Two – Annotated to Identify Distinctive Algorithm 3

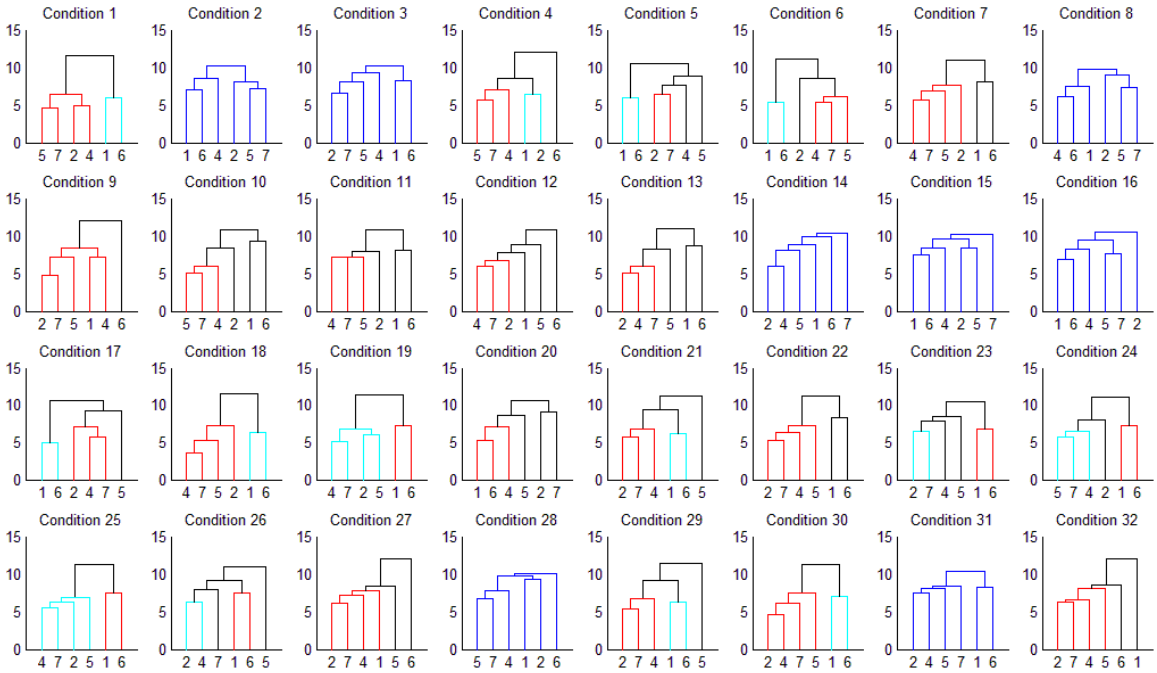


Figure 6-29. Clustering for Time Period Two – Algorithm 3 Omitted

6.4.2.2 Condition-Response Summary for TP2. Fig. 6-30 gives the condition-response summary for TP2. Fig. 6-31 shows the same summary after applying a filter showing

only statistically significant outliers for which the relative effect exceeds 30 %. Algorithm 3 stands out in both figures.

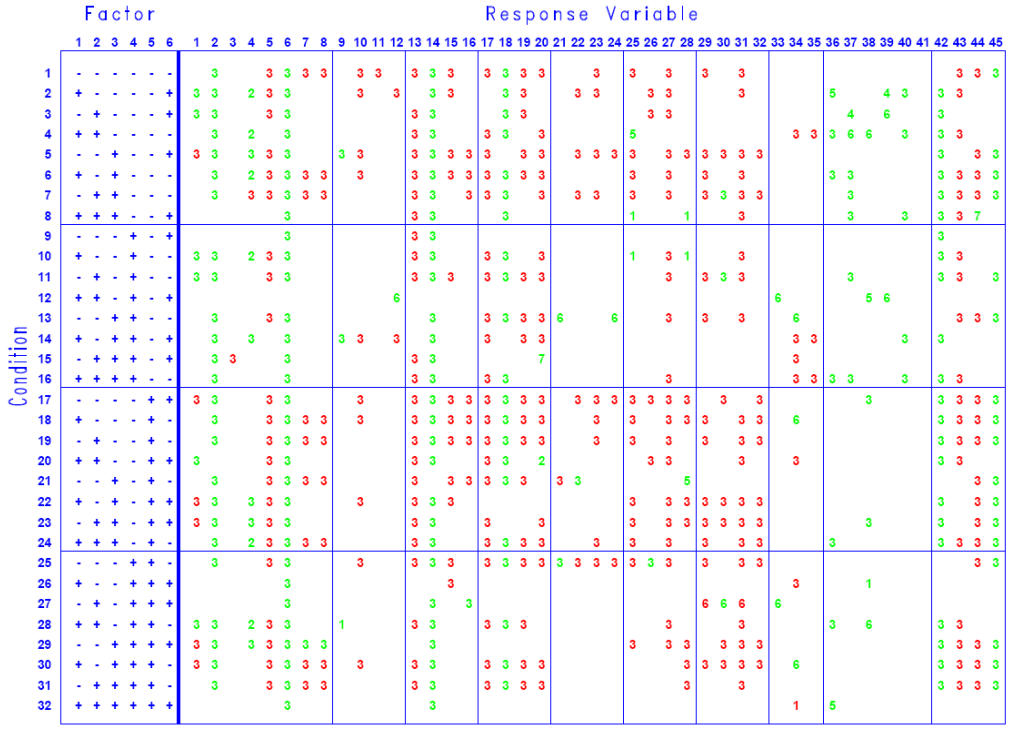


Figure 6-30. Condition-Response Summary for Time Period Two – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm manifesting a Statistically Significant Outlier

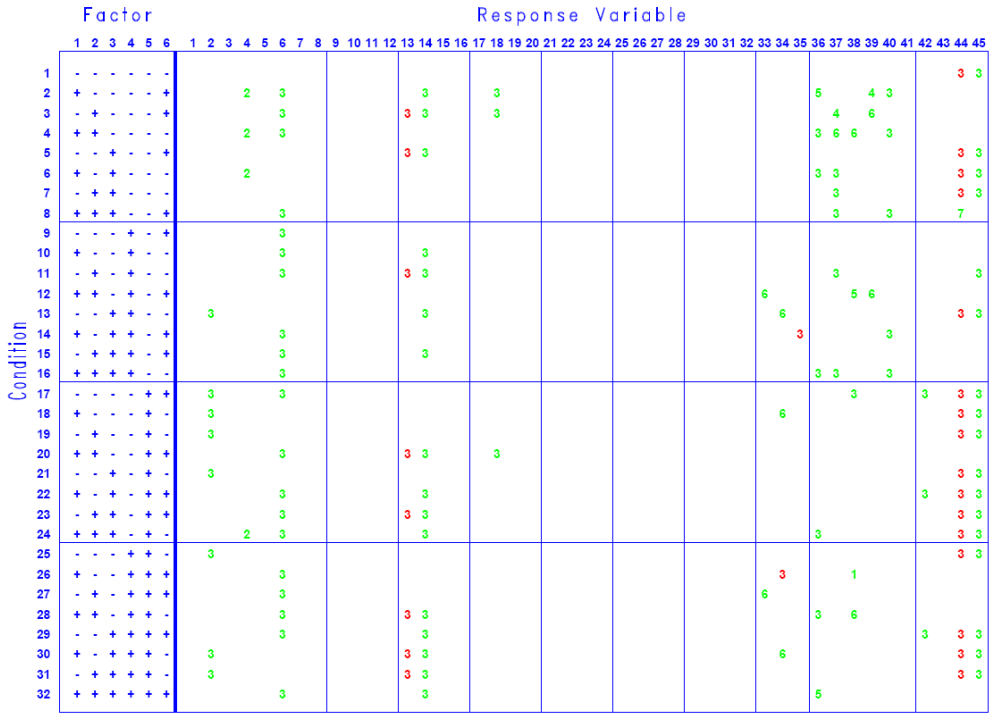


Figure 6-31. Filtered Summary Plot for Time Period Two Identifying Statistically Significant Outliers with Associated Relative Effect > 30%

6.4.2.3 Analysis of Significant Responses for TP2. Based on Figs. 6-30 and 6-31 we selected several responses for more detailed analysis. Specifically, in Figs. 6-32 to 6-37, we report analyses for congestion window increase rate (y2), flow completion rate (y5), retransmission rate (y6), average goodput for DF flows (y13), average number of active DF flows (y14), and average number of connecting flows (y42). In Figs. 6-38 and 6-39 we show the analyses for average goodput on the long (L1) and medium (L2) distance long-lived flows. We selected y5 based on Fig. 6-30 even though it did not pass the 30 % filter required for reporting in Fig. 6-31. We made this additional selection because the absolute magnitude of the effect within an individual measurement interval appears large enough to influence system behavior when accumulated over time.

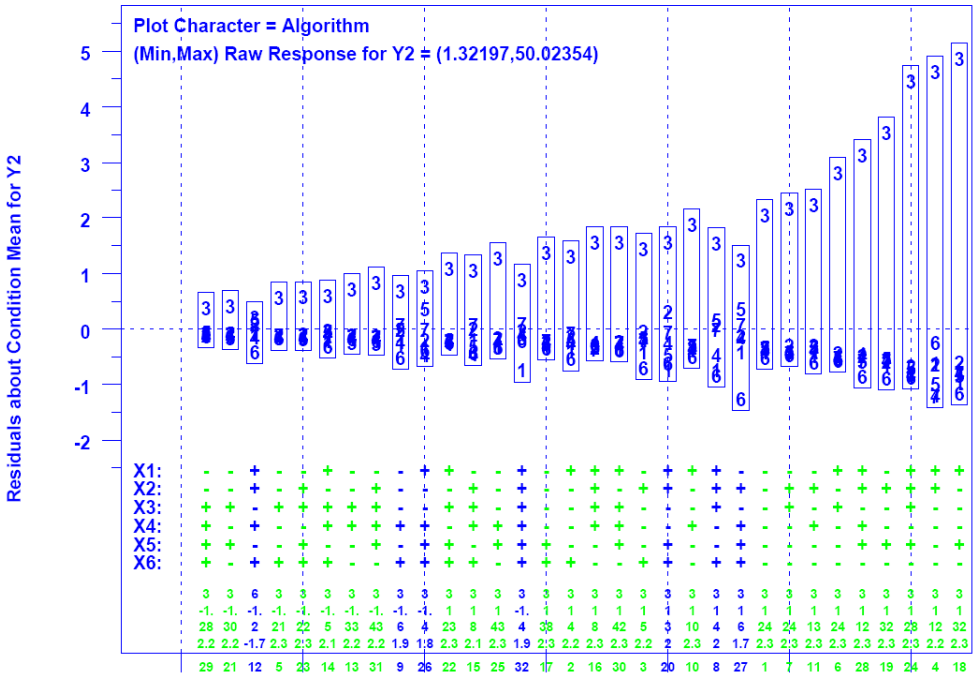


Figure 6-32. Detailed Analysis for Congestion Window Increase Rate (increase per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.4.2.4 Summary of Results for TP2. FAST (algorithm 3) exhibits most of the same distinctive behaviors seen during TP1. The increased congestion in TP2 seems to enhance these effects, most of which now show up as relative differences of 30 % or more. A new pattern of behavior arises with respect to DF flows. The number of active DF flows accumulates for algorithm 3 during TP2, which leads to lower average goodput on those flows. We can again attribute this to the congestion sensitivity demonstrated by FAST. Under normal Web traffic, network parameter settings for the experiment tend to generate congestion at fast access routers. During TP2, DD flows experience jumbo file transfers, so DF flows are affected by the normal congestion pattern as well as increased congestion due to jumbo files. Given this increased congestion, algorithm 3 has more trouble completing DF flows than the other algorithms – increased retransmissions on DF flows lead to longer holding times to complete the flows.

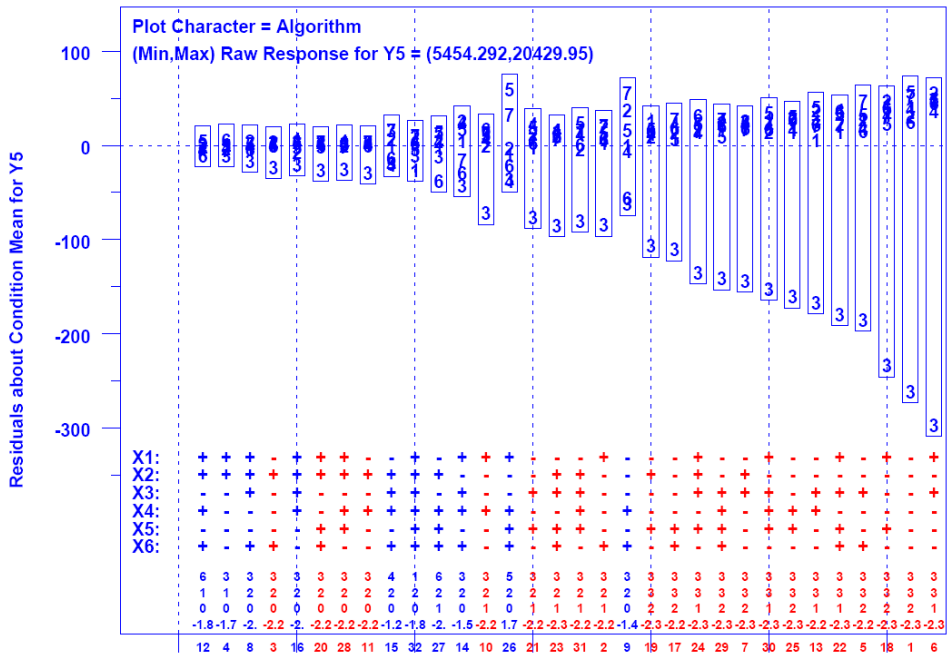


Figure 6-33. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

A new behavior appears in TP2 with respect to the long and moderate distance long-lived flows. Algorithm 6 (Scalable TCP) tends to achieve higher average goodput. We consider this a tendency because there is no widespread pattern of statistical significance. We attribute this tendency to unfairness inherent in Scalable TCP. Under Scalable TCP (during TP1) long-lived flows establish a high congestion window. DD flows arising during TP2 have difficulty claiming a fair share of bandwidth from the entrenched long-lived flows.

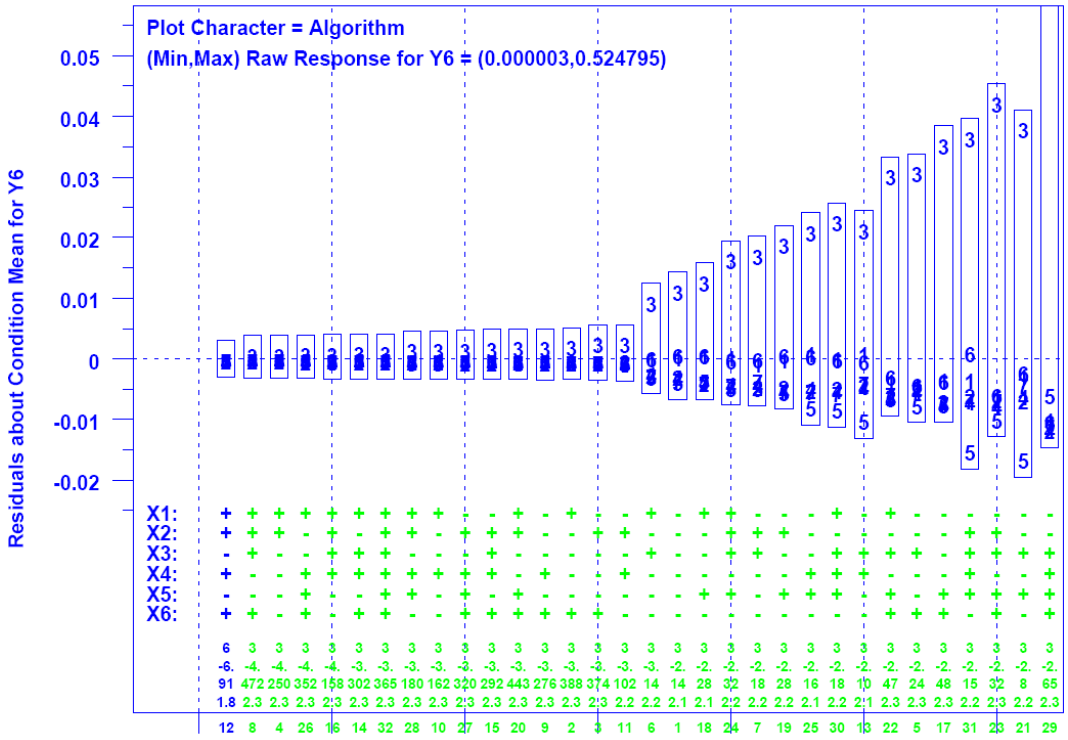


Figure 6-34. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

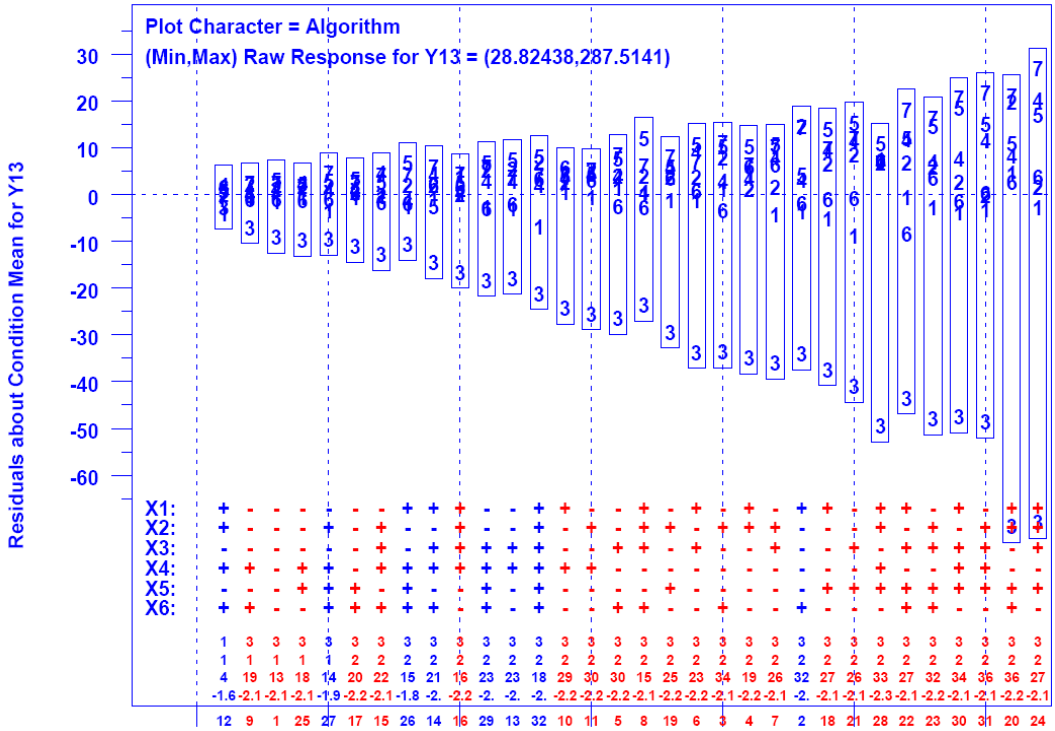


Figure 6-35. Detailed Analysis for Average Goodput (pps) on DF Flows in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

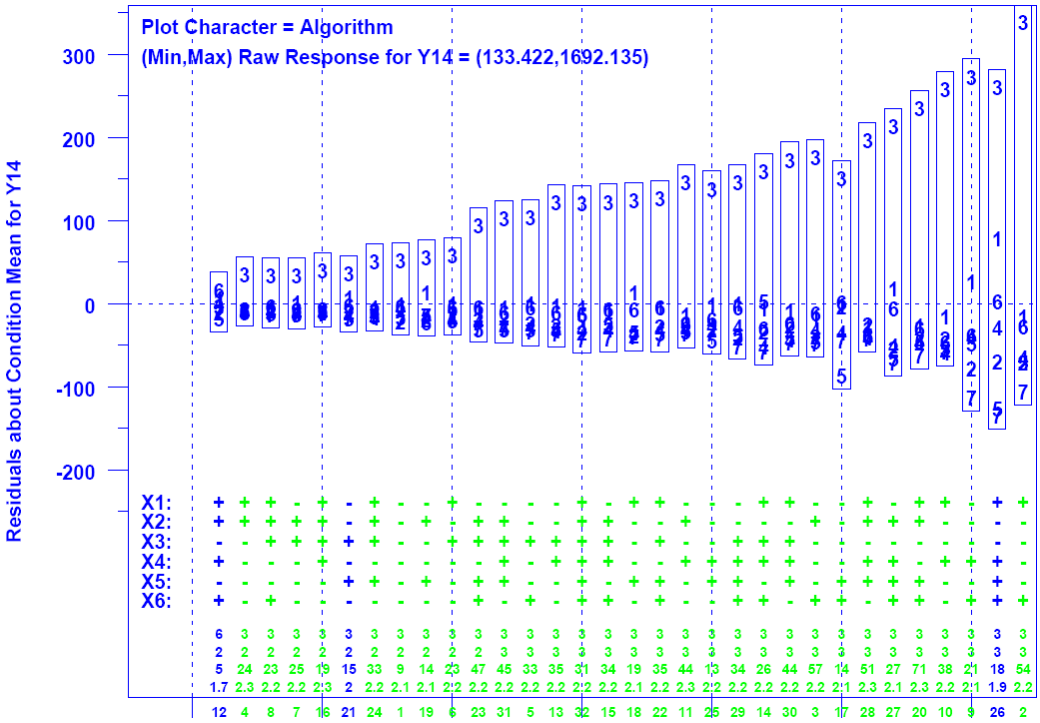


Figure 6-36. Detailed Analysis for Average Number of Active DF Flows in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

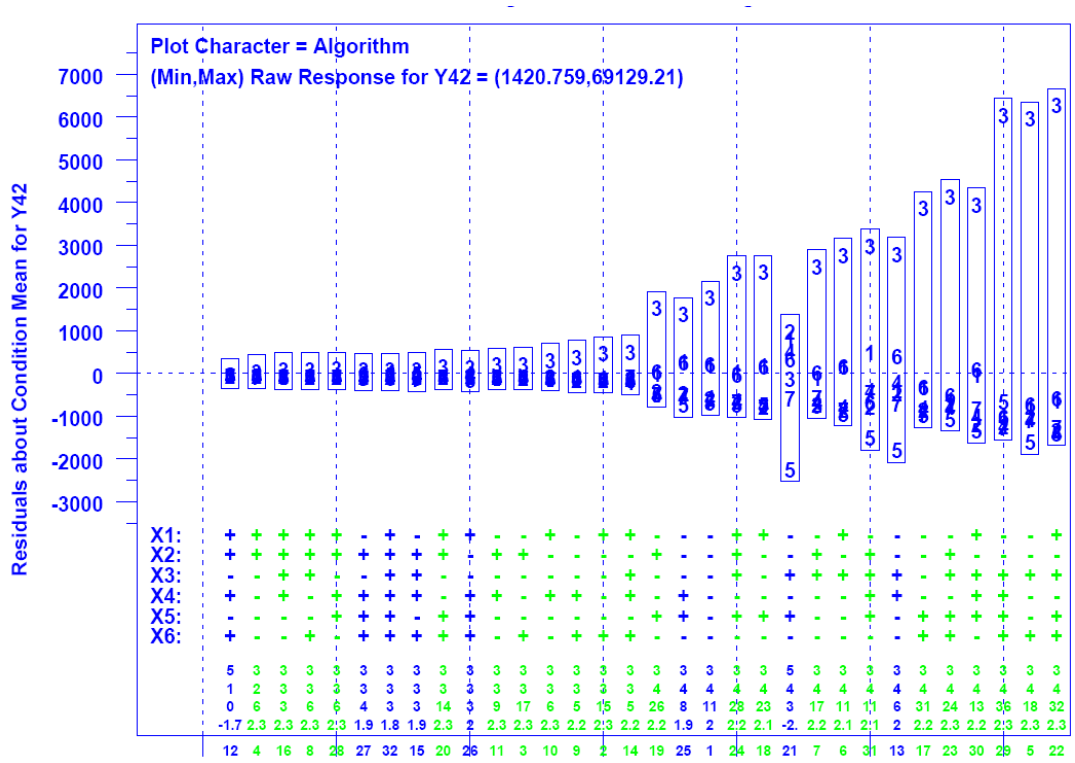


Figure 6-37. Detailed Analysis for Average Number of Connecting Flows in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

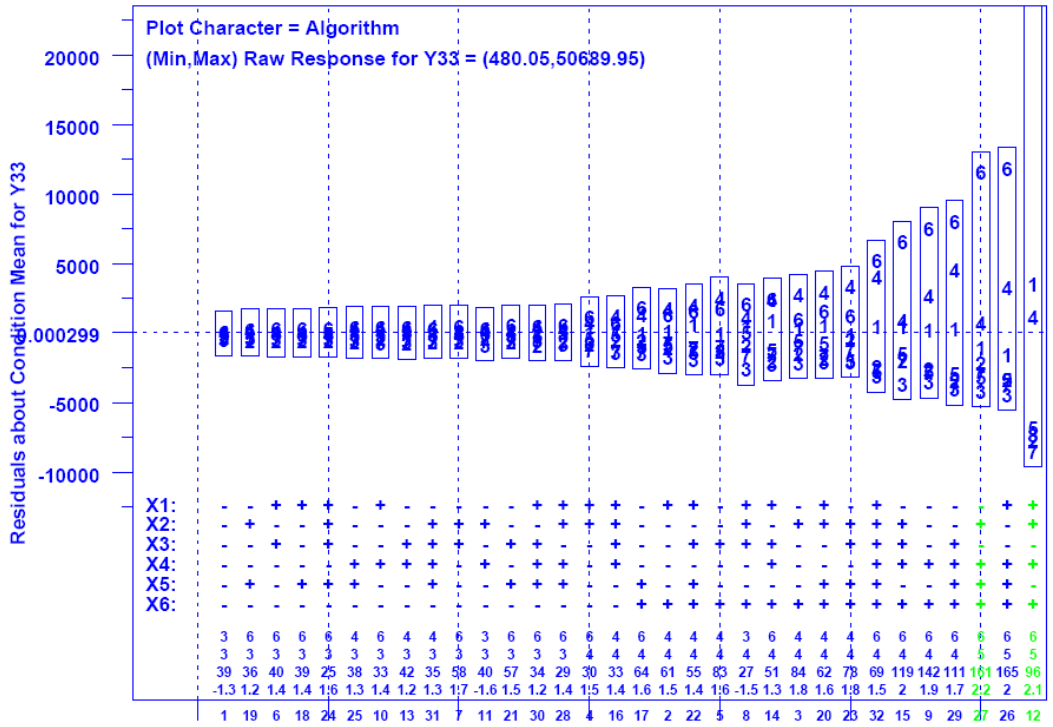


Figure 6-38. Detailed Analysis for Average Goodput (pps) on Long-lived Flow L1 in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

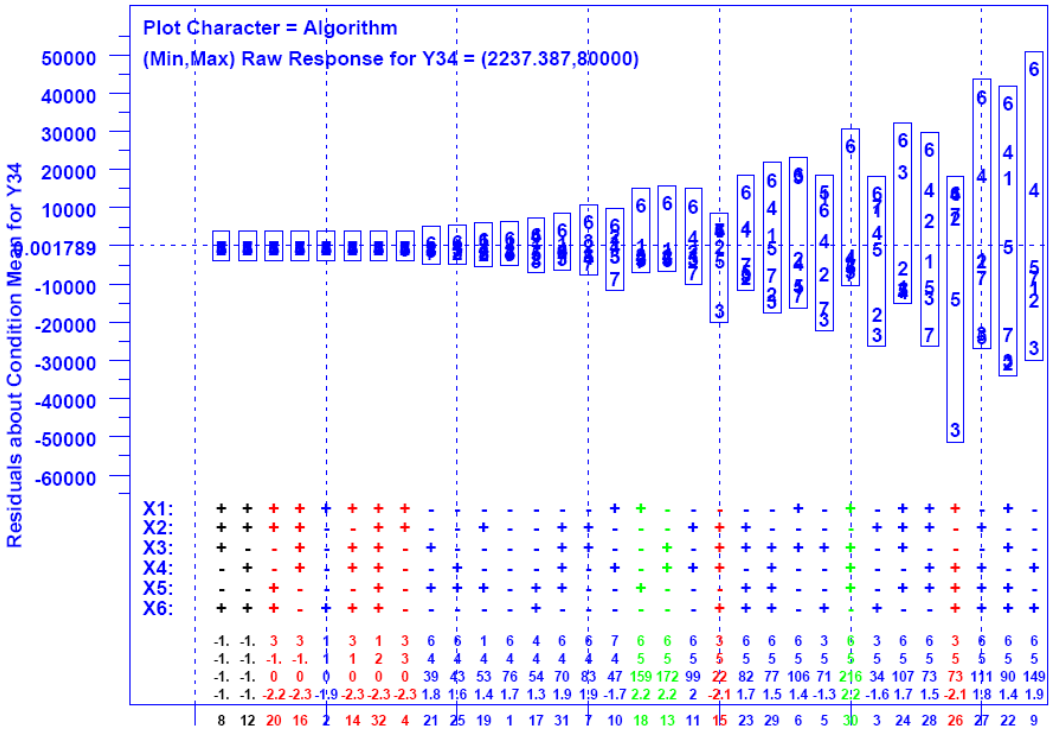


Figure 6-39. Detailed Analysis for Average Goodput (pps) on Long-lived Flow L2 in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.4.3 Time Period Three (TP3)

During TP3 no new jumbo file transfers are initiated on DD flows; what remains is for residual jumbo transfers to complete as the network transitions back toward normal Web traffic. The degree to which normal conditions can be restored depends upon the number and size of jumbo transfers created during TP2.

6.4.3.1 Cluster Analysis for TP3. Fig. 6-40 shows an annotated set of 32 dendrograms for TP3. Since the level of congestion stays relatively high, as residual jumbo file transfers drain from the system, algorithm 3 remains distinctive. When omitting responses for algorithm 3, cluster analysis (Fig. 6-41) identifies no distinctive algorithm.

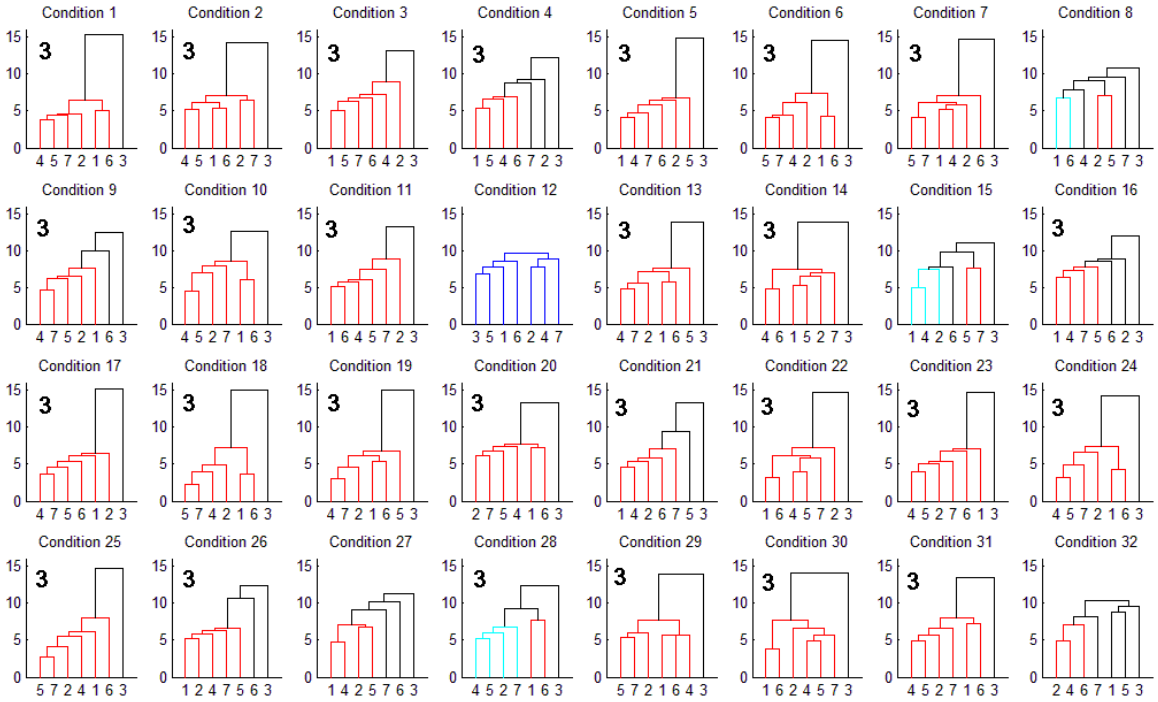


Figure 6-40. Clustering for Time Period Three – Annotated to Identify Distinctive Algorithm 3

6.4.3.2 Condition-Response Summary for TP3. Fig. 6-42 gives the condition-response summary for TP3. Fig. 6-43 shows the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 30 %. Algorithm 3 stands out in both figures – the distinctiveness is quite similar to that seen for TP2. Fig. 6-43 also reveals two new patterns. First, algorithm 2 (CTCP) shows a large increase in the average congestion window, which is pervasive over many conditions during TP3. Second, average goodput lags on the higher propagation, long-lived TCP flows (L1 and L2) as the DD paths recover from the period of jumbo file transfers.

6.4.3.3 Analysis of Significant Responses for TP3. Based on Figs. 6-42 and 6-43 we selected several responses for more detailed analysis. Specifically, in Figs. 6-44 to 6-49, we report analyses for congestion window increase rate (y2), flow completion rate (y5), retransmission rate (y6), average goodput on DF flows (y13), number of active DF flows (y14) and number of connecting flows (y42). Fig. 6-50 illustrates the substantial increase

in congestion window size (y4) for algorithm 2. Fig. 6-51 illustrates (with flow L2) how goodput on long-lived flows tends to lag under standard TCP (algorithm 7).

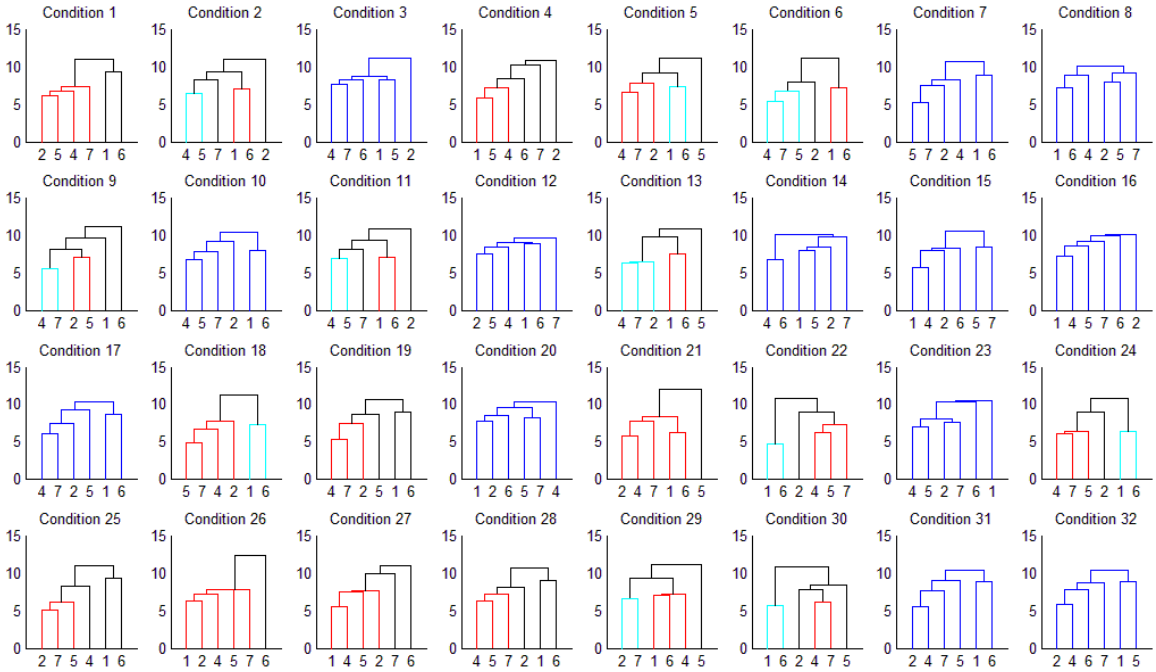


Figure 6-41. Clustering for Time Period Three – Algorithm 3 Omitted

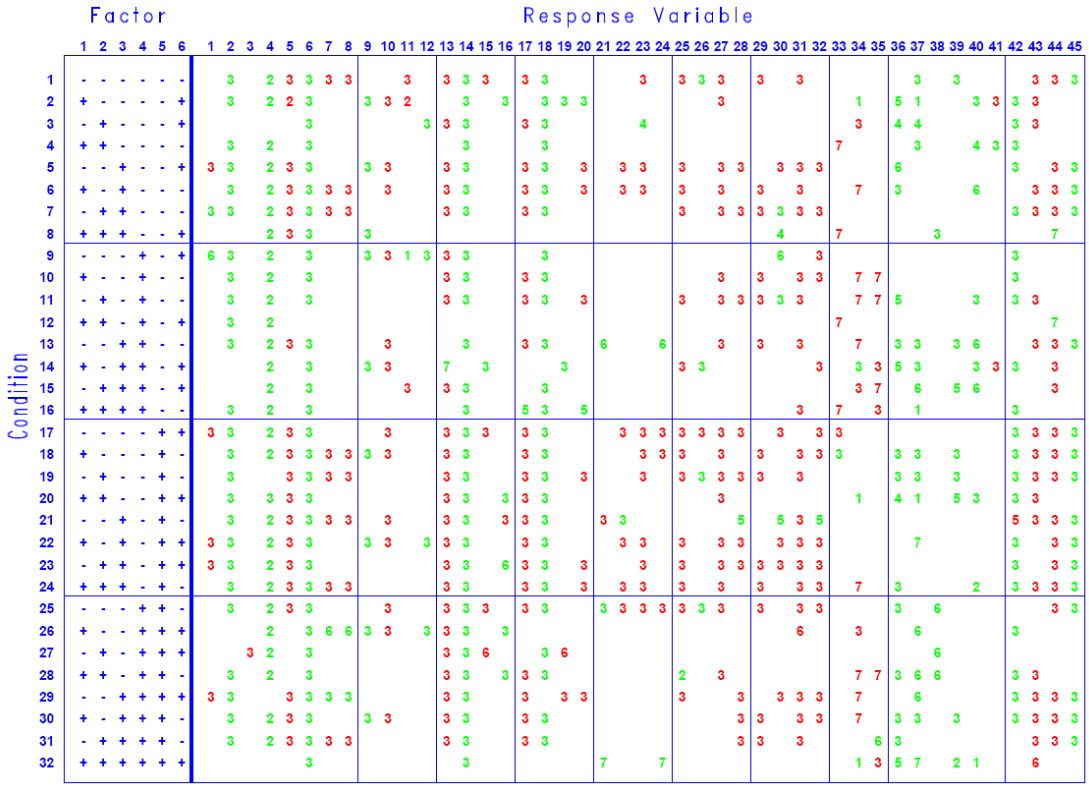


Figure 6-42. Condition-Response Summary for Time Period Three – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm manifesting a Statistically Significant Outlier

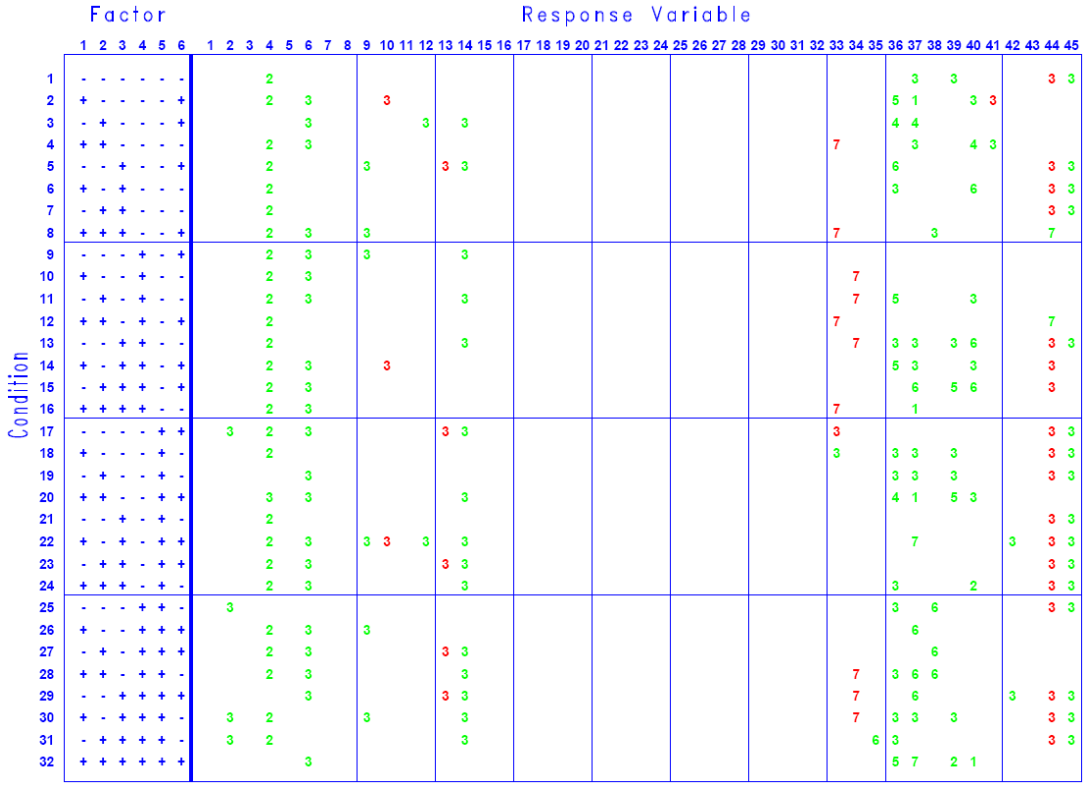


Figure 6-43. Filtered Summary Plot for Time Period Three Identifying Statistically Significant Outliers with Associated Relative Effect > 30%

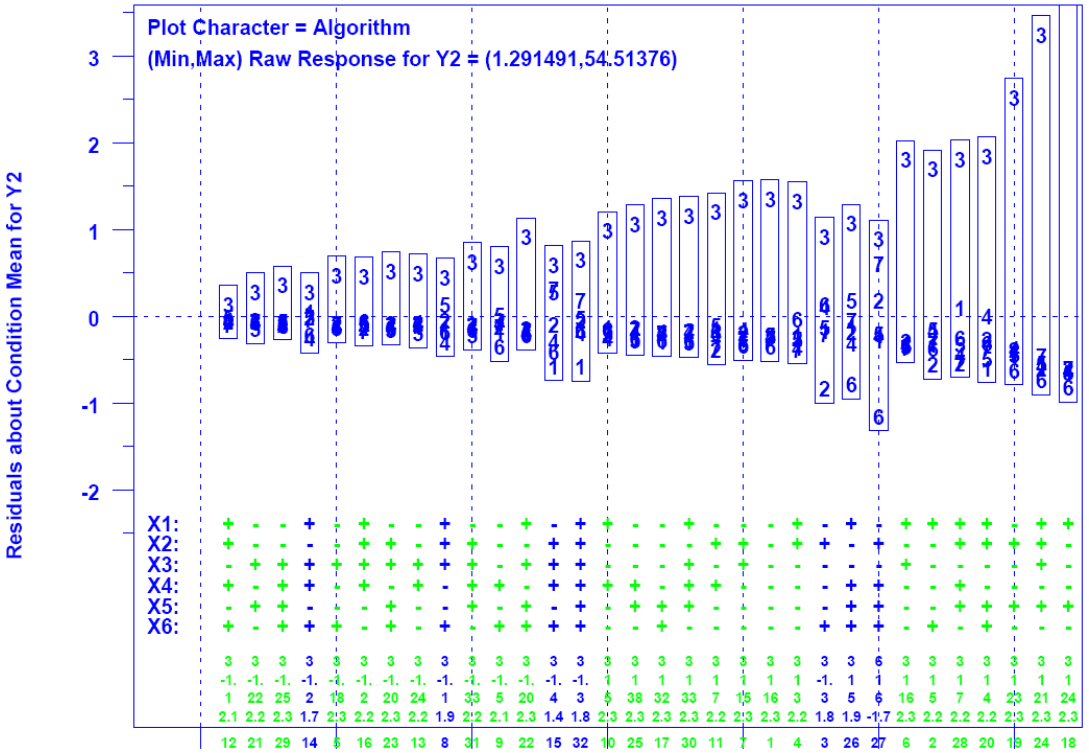


Figure 6-44. Detailed Analysis for Congestion Window Increase Rate (increases per 200 ms) in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

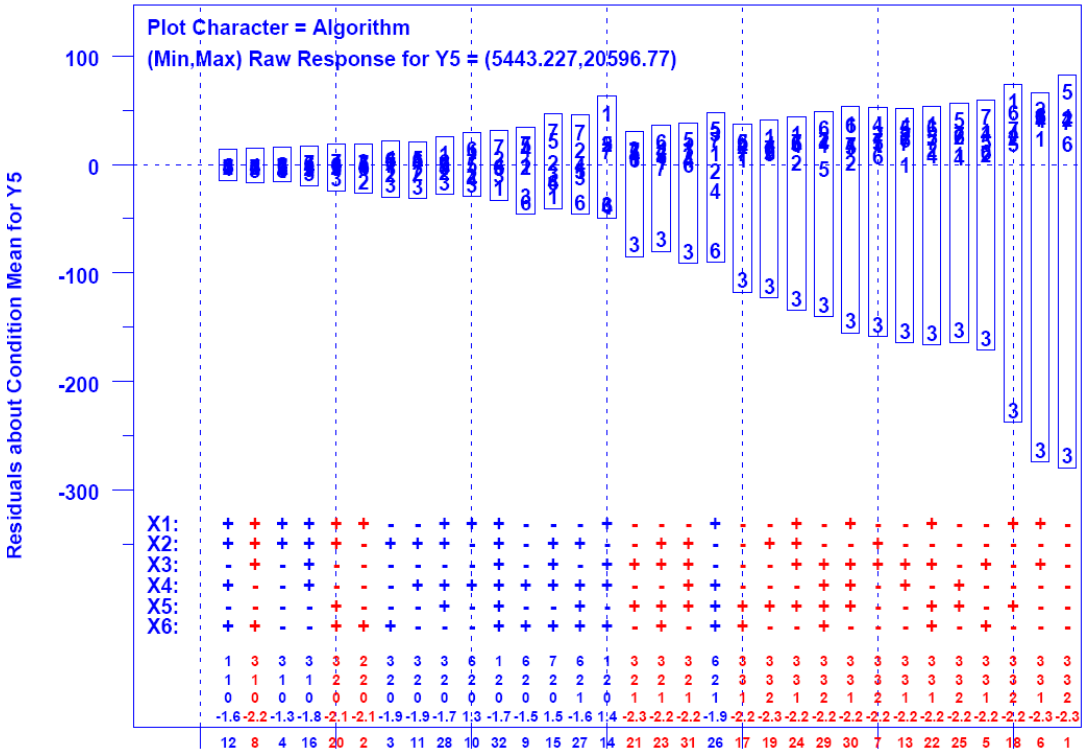


Figure 6-45. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

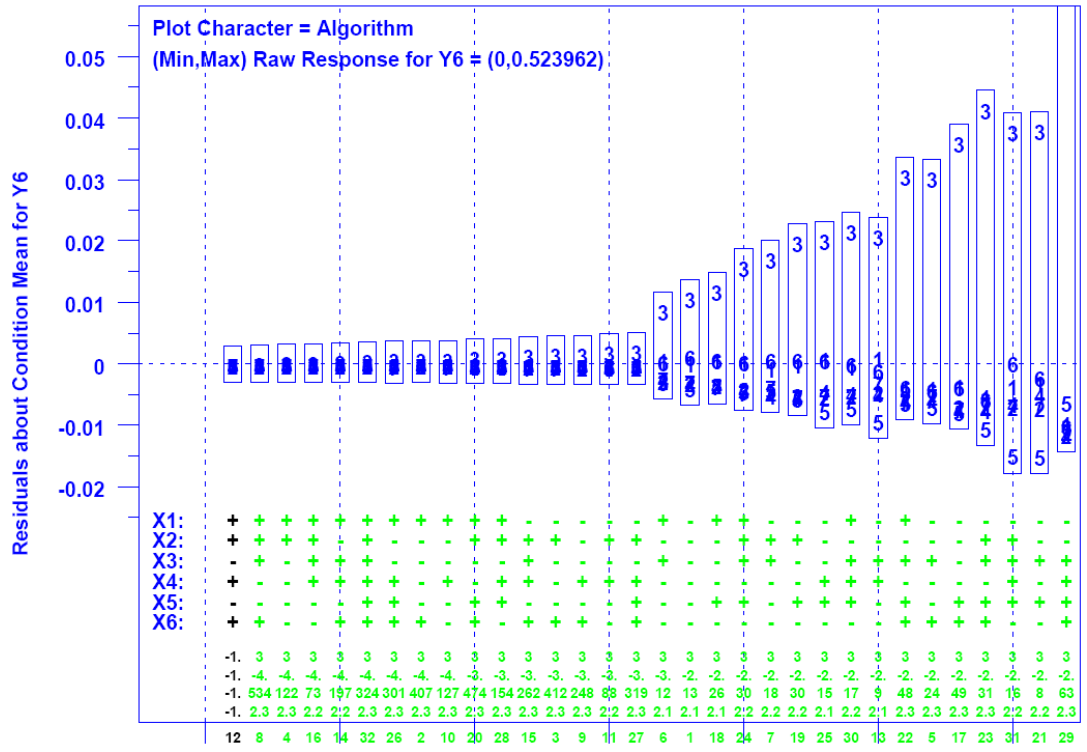


Figure 6-46. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

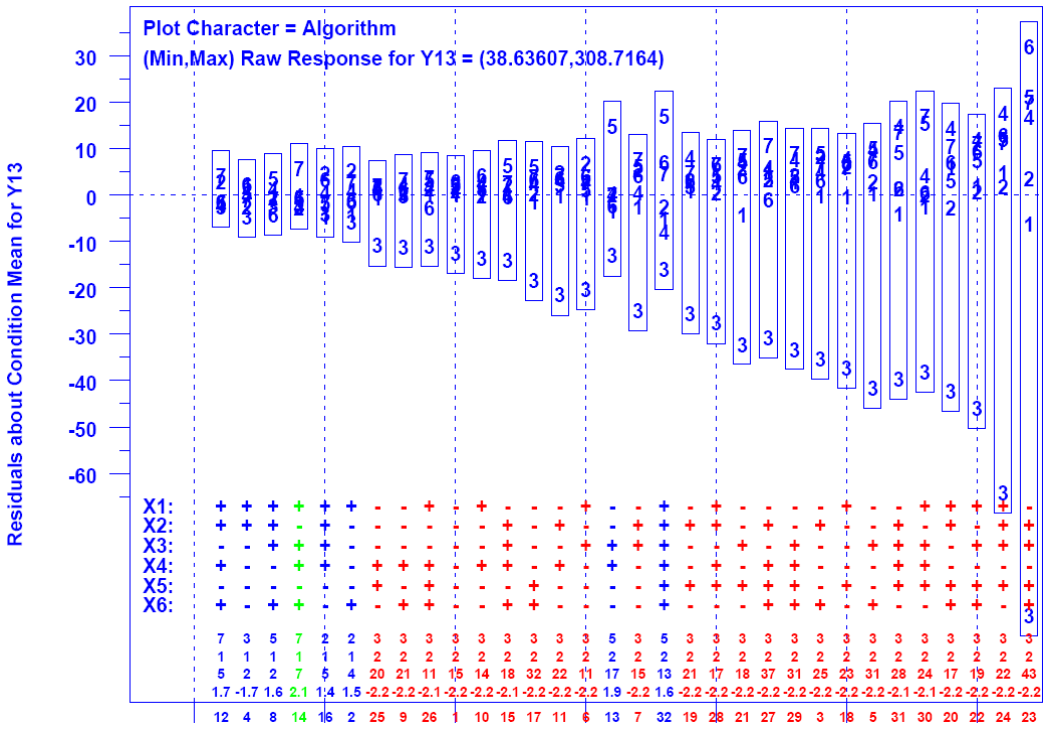


Figure 6-47. Detailed Analysis for Average Goodput (pps) on DF Flows in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

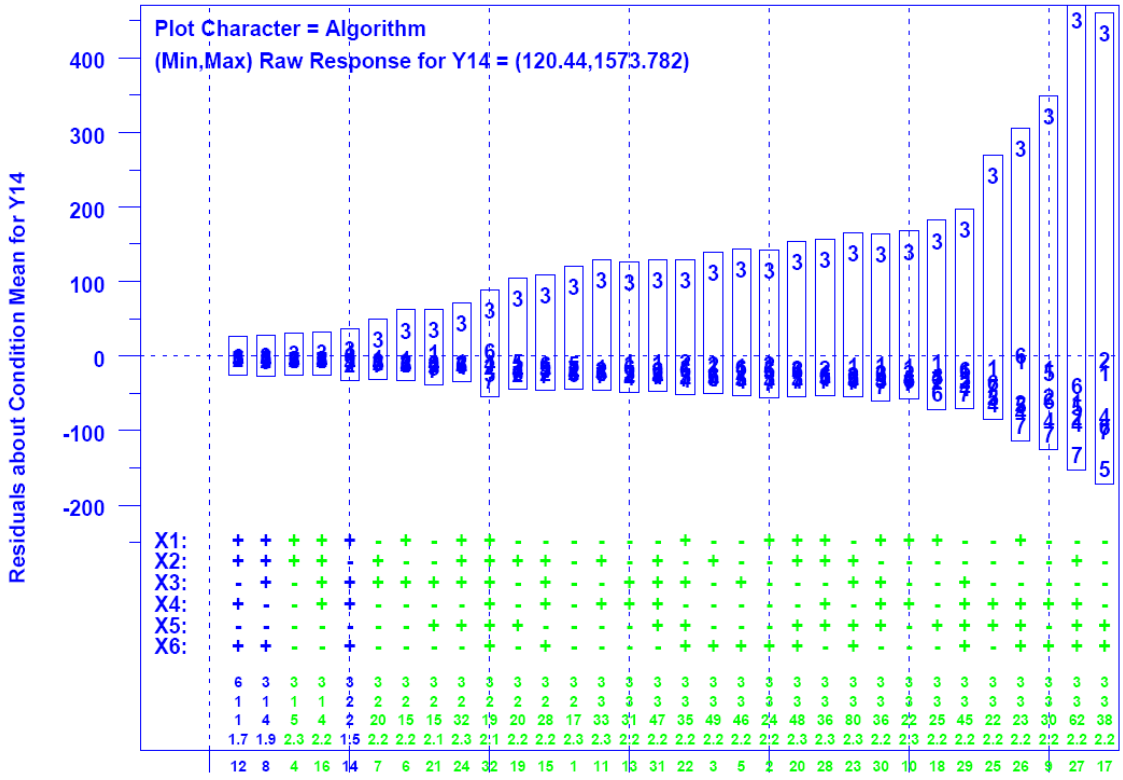


Figure 6-48. Detailed Analysis for Average Number of Active DF Flows in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

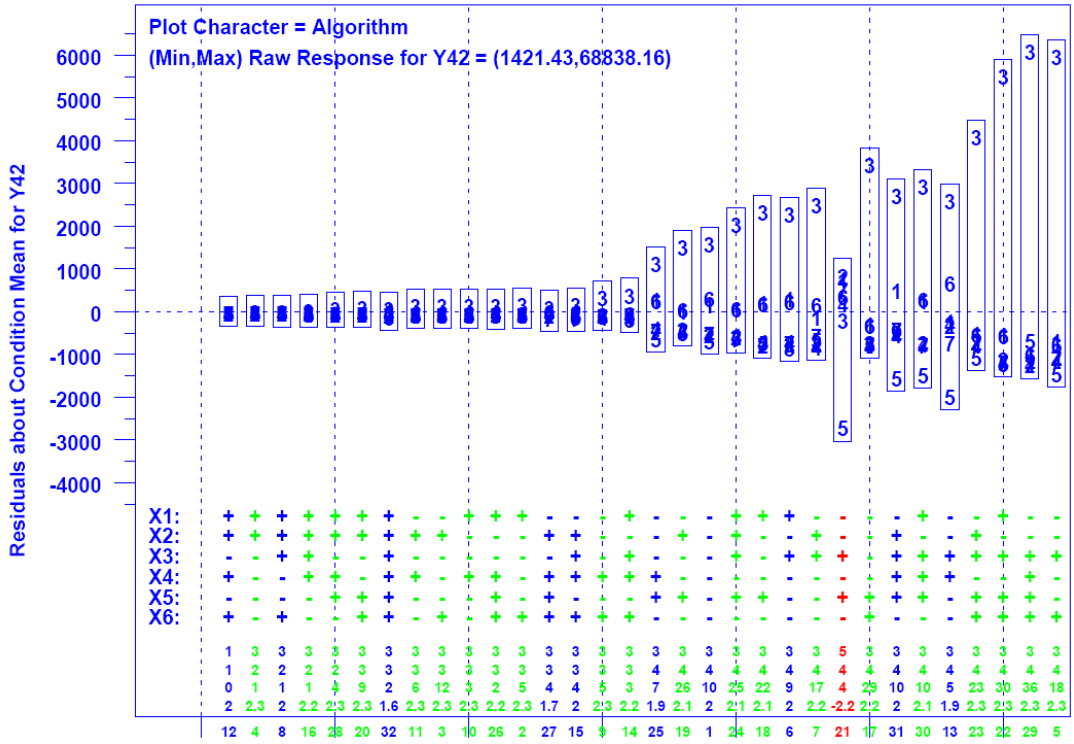


Figure 6-49. Detailed Analysis for Average Number of Connecting Flows in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

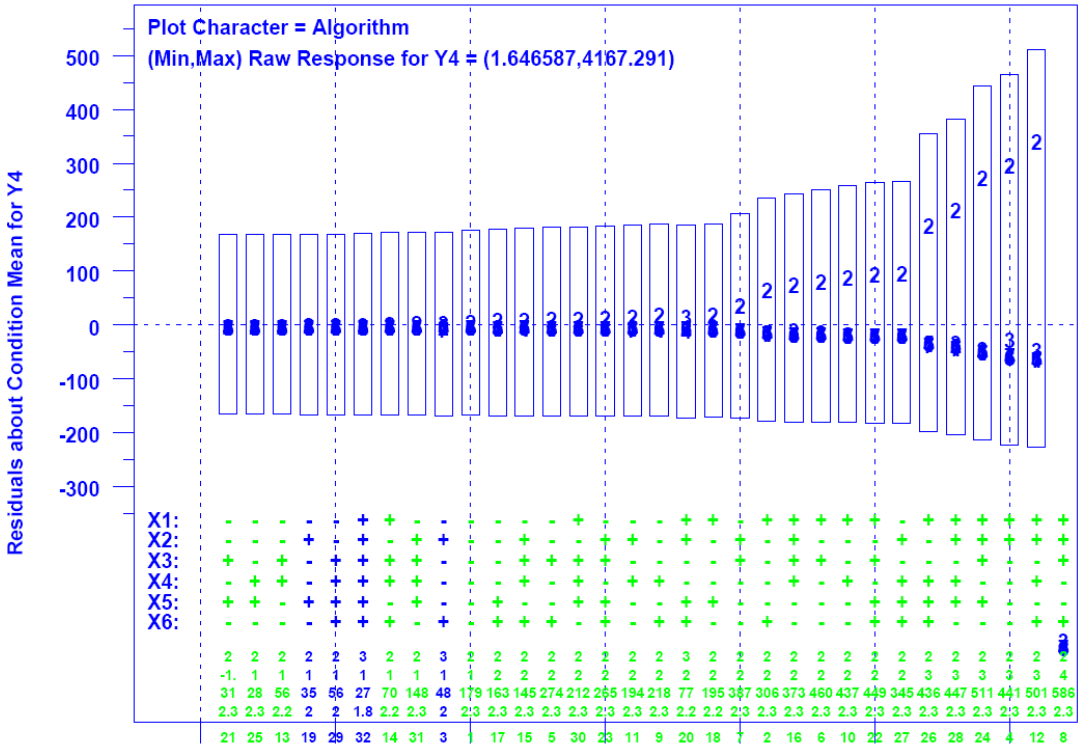


Figure 6-50. Detailed Analysis for Average Congestion Window Size (packets) in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

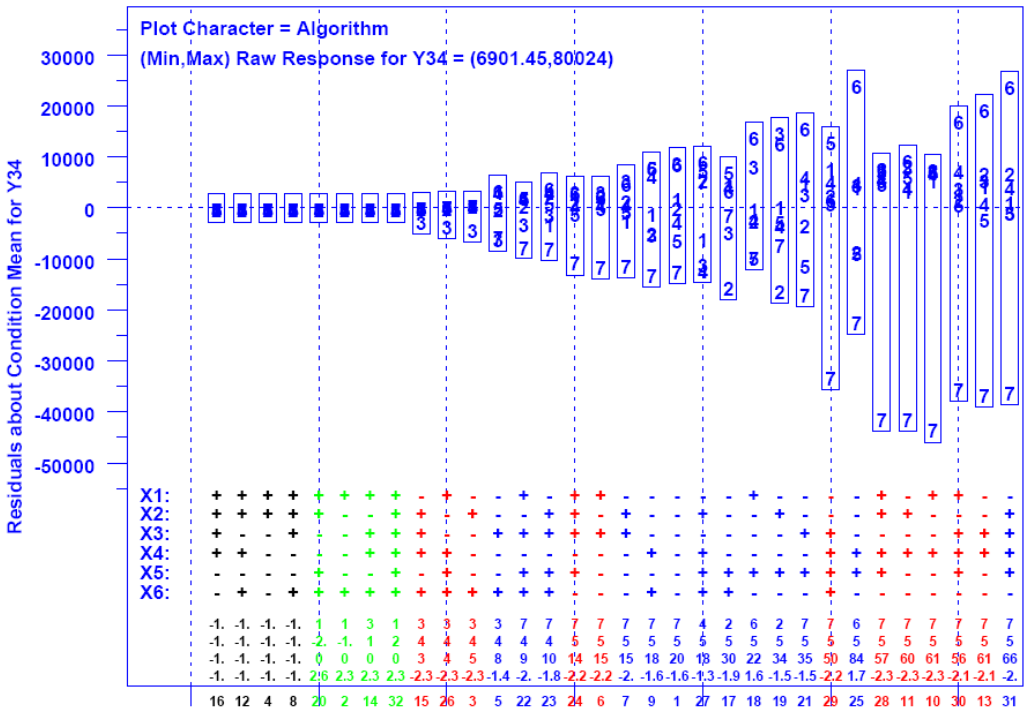


Figure 6-51. Detailed Analysis for Average Goodput (pps) on Long-lived Flow L2 in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.4.3.4 Summary of Results for TP3. FAST (algorithm 3) exhibits the same distinctive behaviors seen during TP1 and TP2. Residual congestion from TP2 maintains these effects at an enhanced level; most effects continue to show relative differences of 30 % or more. A new pattern of behavior arises with respect to average congestion window size, where algorithm 2 (CTCP) shows a substantial increase over other algorithms – this difference is limited to TP3. Results also show that average goodput for long-lived flows using algorithm 7 (TCP) tends to lag when recovering from the congested period (TP2). This trait of standard TCP congestion control was an initial stimulus for many of the proposals for alternate congestion control mechanisms.

6.4.4 Aggregated Responses (Totals)

Here we present analyses for the 28 responses collected over the entire 25-minute scenario. Recall that most of these responses are aggregated counts, but SYN rate on connected flows and goodput on completed flows are averages. Whereas the previous analyses focused on differences in instantaneous behavior averaged over 5-minute intervals, the current analysis examines the effects of behavioral differences viewed over a longer period.

6.4.4.1 Cluster Analysis for Totals. Fig. 6-52 shows the usual annotated set of 32 dendrograms, but this time clustering based on the 28 aggregate responses. Similar to the cluster analyses for the three time periods, algorithm 3 appears distinctive in many (24) of the conditions. Fig. 6-53 shows the results from clustering with algorithm 3 responses excluded. No significant difference appears among the remaining algorithms, though algorithms 1 and 6 exhibit some tendency to be paired.

6.4.4.2 Condition-Response Summary for Totals. Fig. 6-54 gives the condition-response summary for the aggregate responses. One finding from the figure is that algorithm 3 (FAST) tends to input more packets but not to output more packets – this is congruent with a higher loss rate, and consequent increased retransmission rate. For path classes prone to congestion, algorithm 3 (FAST) provides lower average goodput, which means these flows require more retransmissions and take longer to complete. In addition, algorithm 3 (FAST) connects and completes fewer flows – among a wide range of flow classes and across the entire set of backbone routers. As expected, based on analysis of the time periods, algorithm 3 (FAST) shows a higher average SYN rate over most conditions. This is congruent with a larger number of flows pending in the connecting state, and with a higher retransmission rate due to lost packets.

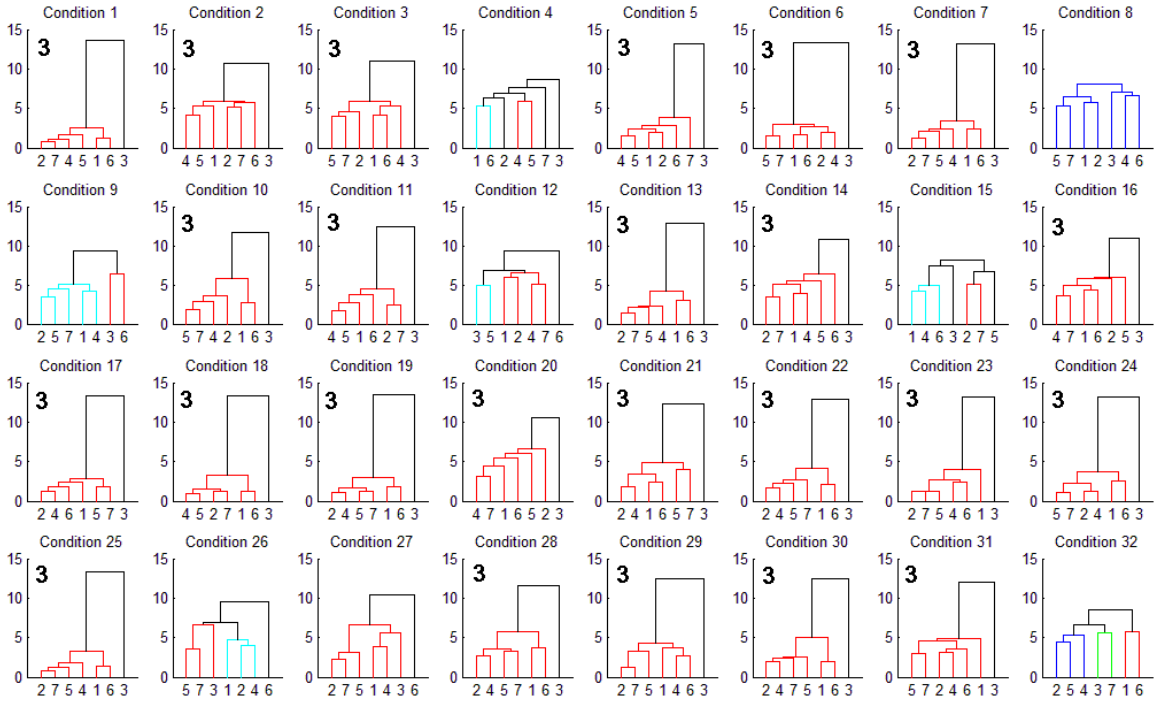


Figure 6-52. Clustering for Totals – Annotated to Identify Distinctive Algorithm 3

6.4.4.3 Analysis of Significant Responses for Totals. Based on Fig. 6-54 we selected two responses for detailed analysis. Algorithm 3 completed fewer flows under most conditions for most flow classes – including DF flows (T.y8), DN flows (T.y10), FF flows (T.y12), FN flows (T.y14) and NN flows (T.y16). For these flow classes, algorithm 3 also usually exhibited lower average goodput. Algorithm 3 completed fewer flows across all backbone routers in the network. Rather than show detailed analyses for all of these categories, we present, in Fig. 6-55, an analysis of the aggregate number of flows completed (T.y4), where algorithm 3 underperforms under most conditions. We also show, in Fig. 6-56, a detailed analysis of the average SYN rate. In all but two conditions (the least and most congested), algorithm 3 leads to more SYNs being sent on average to establish flows. This supports earlier observations that algorithm 3 tends to have substantially more flows pending in the connecting state at any instant in time.

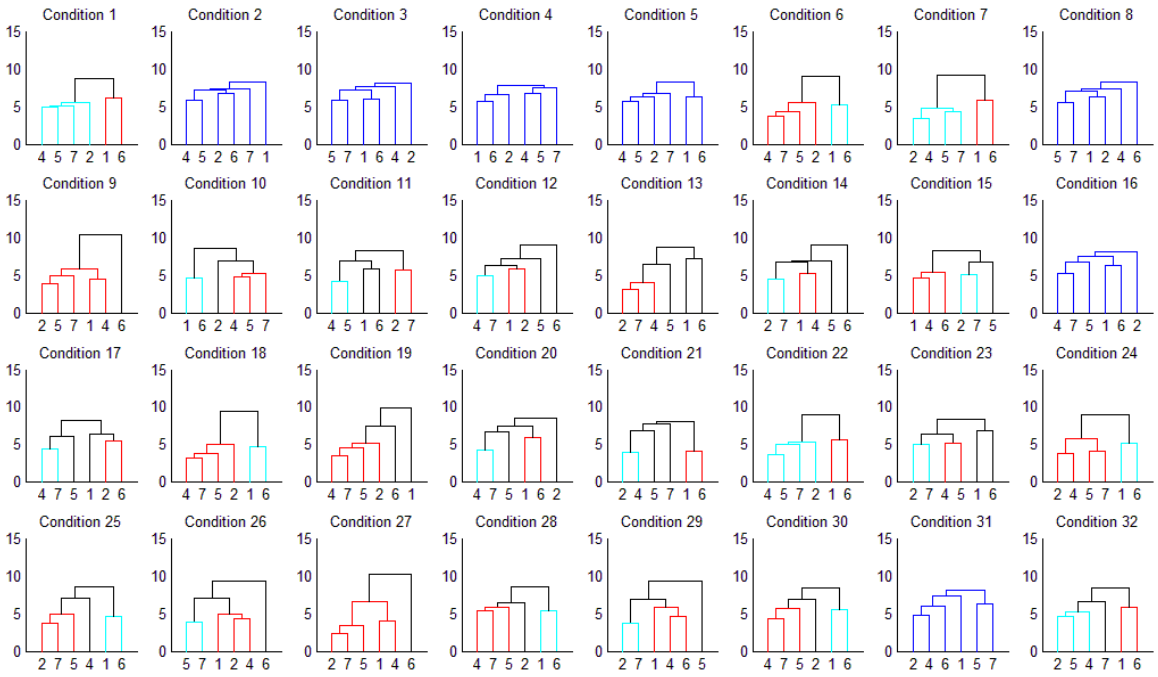


Figure 6-53. Clustering for Totals – Algorithm 3 Omitted

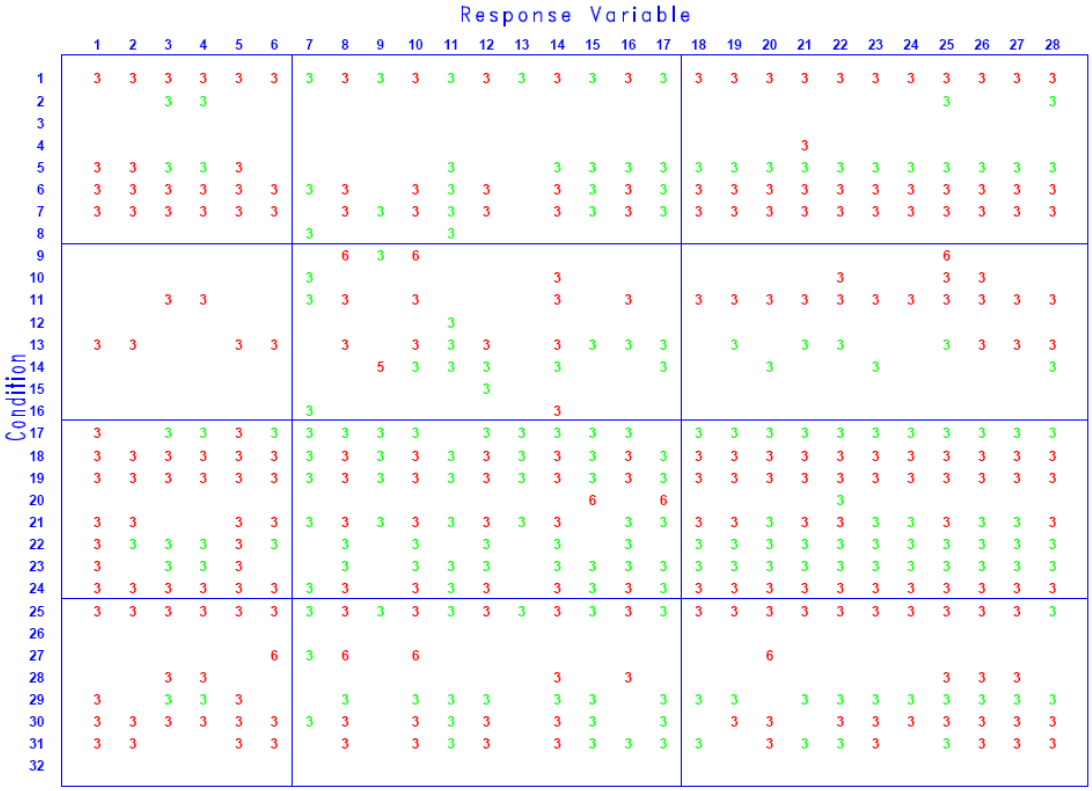


Figure 6-54. Condition-Response Summary for Totals – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm manifesting a Statistically Significant Outlier

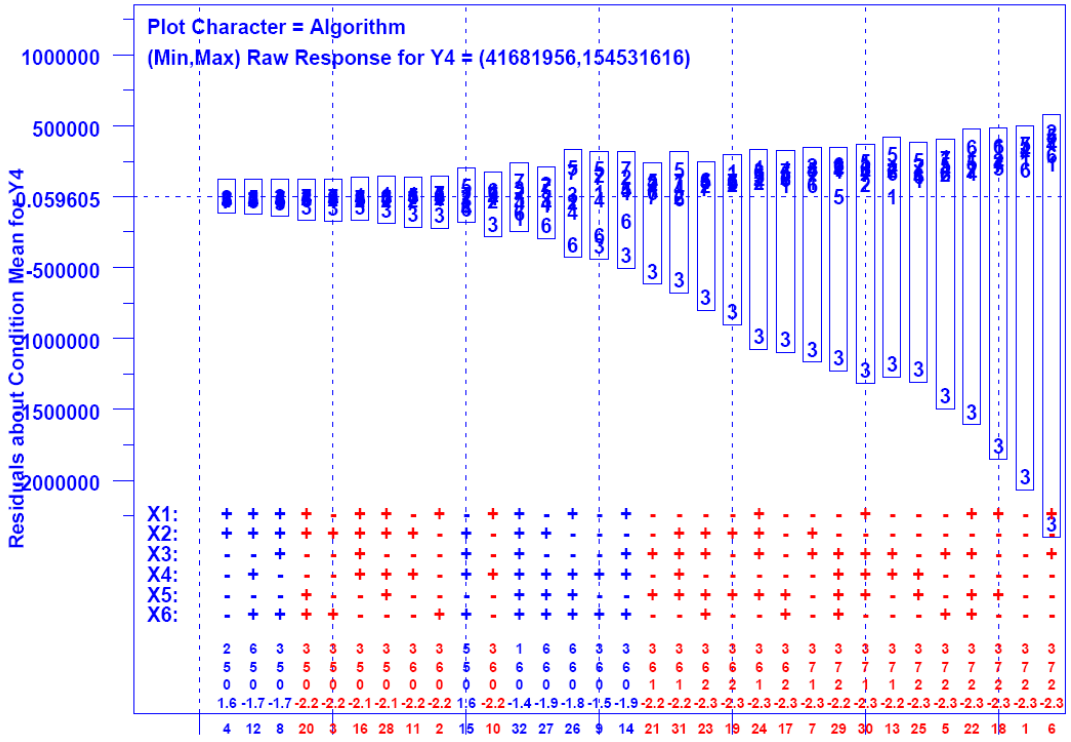


Figure 6-55. Detailed Analysis for Aggregate Number of Flows Completed over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

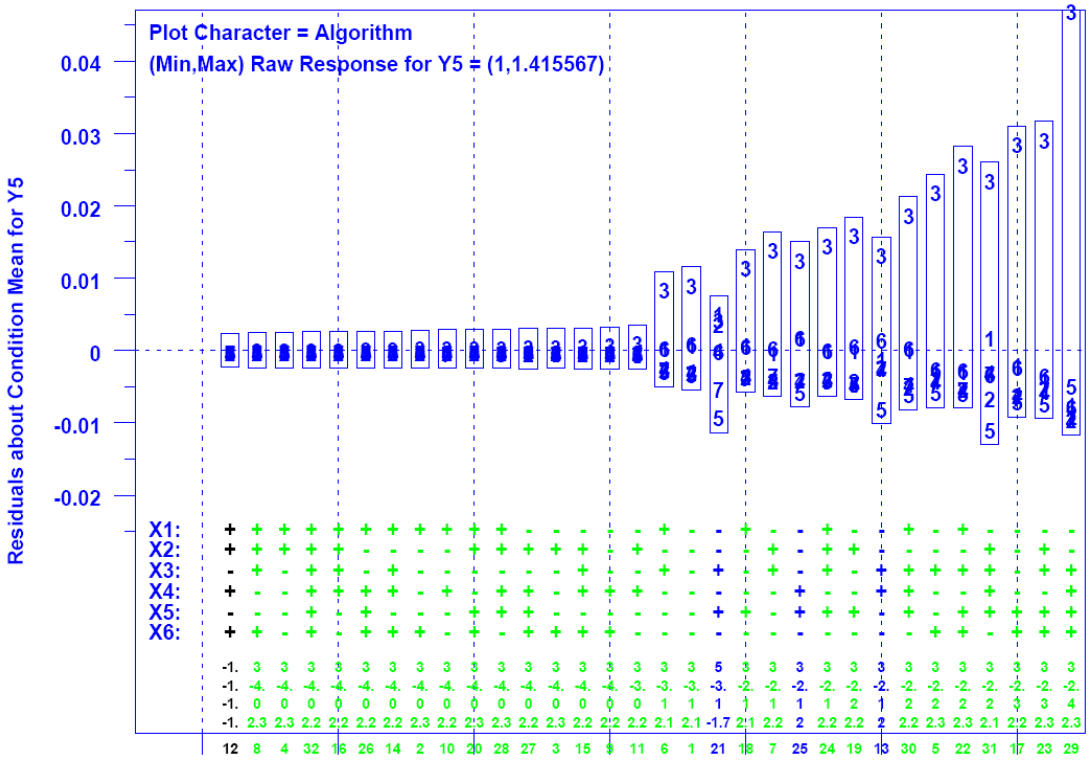


Figure 6-56. Detailed Analysis for Average SYN Rate (count of SYNs sent per connection attempt) for Connecting Flows over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

6.4.4.4 Summary of Results for Totals. Under the conditions investigated in this experiment, FAST (algorithm 3) completes fewer flows during the 25-minute scenario. While FAST completes only up to 2 % fewer flows, this amounts to between a million and 10 million flows over 25 minutes – 40×10^3 to 400×10^3 flows a minute. In addition, FAST impedes the ability of flows to establish connections.

6.5 Findings

From the results reported in Sec. 6.4, we identified four main findings, as discussed below. In addition, detailed analysis of individual responses, when excluding algorithm 3, identified some tendencies, which we outline in Sec. 6.5.5.

6.5.1 Finding #1

Setting aside algorithm 3 (FAST), for the experiment scenario and conditions examined in this section, the alternate congestion control algorithms exhibited indistinguishable macroscopic behavior and modest differences in user experience. In other words there was no overall advantage to be gained in switching the entire network to a particular alternate congestion avoidance scheme, nor was there any overall disadvantage in switching. (Remember we are excluding FAST from this finding.) Selected users could experience somewhat higher throughputs when using alternate congestion control algorithms during periods of competing large file transfers, but no widespread improvement in user experience should be expected.

To understand this finding, recall that slow-start procedures are unaffected by alternate congestion control mechanisms, which define replacements only for the TCP congestion avoidance phase. No matter what congestion control mechanism is used, a flow commences operating in initial slow-start and switches to congestion avoidance only after a packet loss (because we used a high initial slow-start threshold). Aside from FAST and TCP Reno, the alternate congestion avoidance procedures specify an activation threshold (either a certain congestion window size or duration since the most recent loss). Below that threshold, a flow adopts standard TCP congestion avoidance procedures; above that threshold the flow adopts alternate congestion avoidance procedures.

Recall that in our experiment we simulated 32 conditions covering a range of congestion patterns, which could be classified roughly into 16 uncongested and 16 congested conditions. Condition 12 created the least congestion, while condition 21 created the most congestion. Of course, even uncongested conditions include localized congestion arising from the onset of jumbo file transfers during TP2, as well as from hot spots appearing from time-to-time at particular access routers. For example, in Fig. 6-57, we plot data under condition 12 for algorithm 1 (we chose to plot BIC because it has the lowest activation threshold: congestion-window > 14 packets). Note that most of the 1.2×10^4 or so active flows (red) in TP1 (3000 – 4500) and TP3 (6000 – 7500) operate in initial slow start (green). This means that these active flows complete their file transfers without packet loss. For flows of this nature, congestion avoidance is never activated, so one would expect alternate congestion avoidance procedures to make no difference. During TP2 (4500 – 6000), jumbo file transfers on DD flows cause concentrated congestion at directly connected access routers. As Fig. 6-57 shows, even during TP2 the number of flows operating in congestion avoidance reached a level of around 10^3 (under 10 %) out of 1.3×10^4 active flows. Half of the flows operated in normal (brown)

congestion control mode (i.e., congestion window $\leq$ 14) and half operated using BIC (blue) congestion avoidance procedures. One would expect DD flows operating in alternate congestion control mode to achieve higher throughput than the DD flows operating in normal congestion control mode. So, selected users could experience improved throughput over others during TP2.

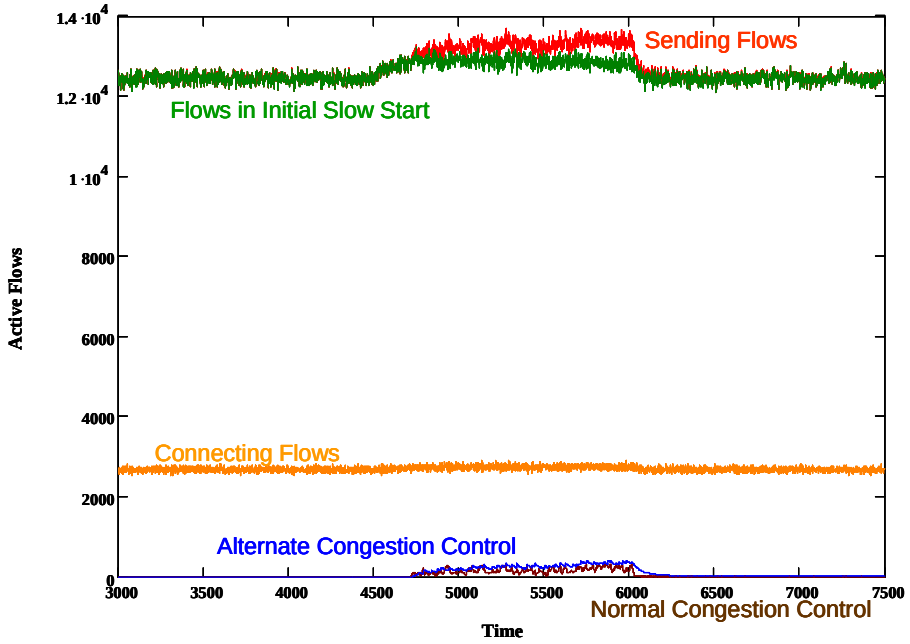


Figure 6-57. Five Time Series Showing the Distribution of Flow States over Three Time Periods for Algorithm 1 (BIC) under Condition 12 – x axis shows time in 200 ms increments and y axis shows number of active flows in each state

In Fig. 6-58, we plot the equivalent distribution of flow states for BIC under the most congested condition: 21. Of the 1.45 x 10^5 active flows (red) in TP1 and TP3 about 1.4 x 10^5 flows (brown) operate in normal congestion control mode and the rest (green) operate in initial slow start. Under these conditions, alternate congestion avoidance procedures are not activated. During TP2, the onset of jumbo file transfers leads to about 1.5 x 10^3 flows (blue) (around 1 %) using alternate congestion avoidance procedures. This small proportion of flows adopting alternate procedures cannot be expected to make a large difference in macroscopic network behavior.

What about user experience? Most flows in a heavily congested network, or in heavily congested portions of a network, will be sharing paths with many other flows. For this reason, one should expect most flows to be operating within normal congestion control mode; these flows cannot achieve a large enough congestion window size (or avoid losses for long enough) to activate alternate congestion avoidance procedures. On the other hand, flows transiting very fast (DD) paths may be able to benefit from alternate congestion control procedures. Overall pattern analysis found that average goodput on DD flows in TP2 showed statistically significant improvement for the extreme algorithm in only three (4, 15 and 28) of 32 conditions; the three conditions were all uncongested. On the other hand, Table 6-32 gives, for each congestion control algorithm, the average goodput on DD flows when averaged across all conditions during TP2, as well as the

minimum and maximum average goodputs. The figures in Table 6-32 suggest that the alternate congestion control mechanisms do, on average, provide better user experience on DD flows during TP2. In fact, during TP2 TCP yields lowest average goodput, but this is only 1 % to 7 % lower than for the other algorithms. The detailed analysis of average goodput on DD flows (y9) during TP2 also shows that a particular alternate congestion control algorithm can improve goodput by 2 % to 19 % over the average for specific conditions. However, there is no particular pattern as to which alternate congestion control algorithm provides best goodput. From this, we conclude that under some conditions users can experience higher goodput when using alternate congestion control algorithms on DD flows that compete to complete large file transfers. The overall improvement when averaged across a wide range of conditions would, however, likely be below 10 %.

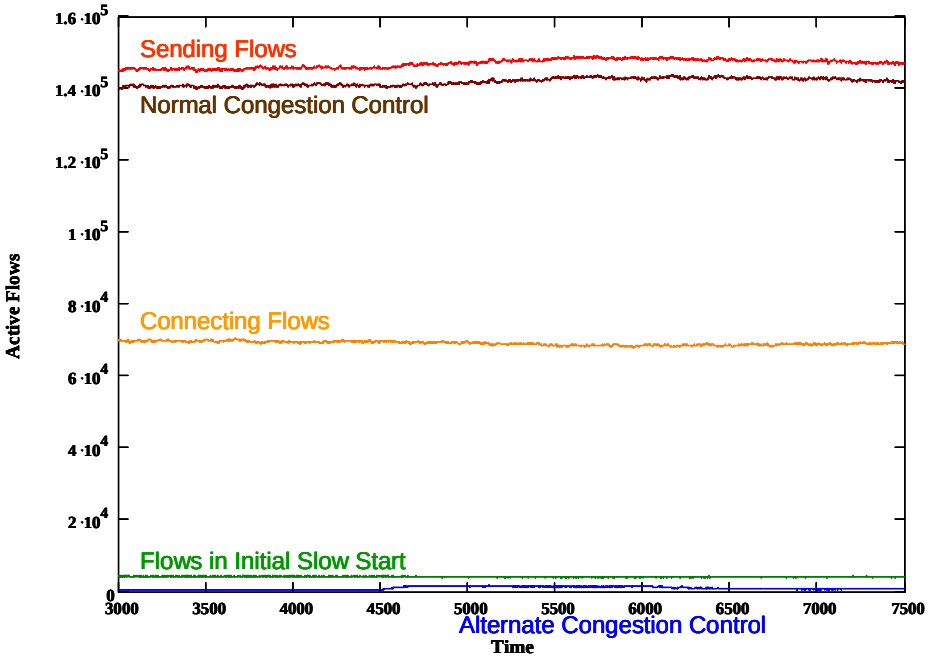


Figure 6-58. Five Time Series Showing the Distribution of Flow States over Three Time Periods for Algorithm 1 (BIC) under Condition 21 – x axis shows time in 200 ms increments and y axis shows number of active flows in each state

Table 6-32. Average, Minimum and Maximum Goodput (pps) on DD Flows for Each Congestion Control Algorithm during TP2 when Averaged over All 32 Conditions

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP
Average Goodput	Average	1260.62	1242.59	1277.23	1193.47	1216.96	1241.56	1184.58
	Minimum	461.27	436.93	474.47	438.84	431.58	437.13	430.25
	Maximum	4079.63	4155.30	4201.02	3724.64	4085.80	3907.04	3680.50

In summary, switching the entire network from standard TCP congestion control to BIC, CTCP, HSTCP, HTCP or Scalable TCP should not cause large shifts in macroscopic network behavior. Further, Web-browsing users would see little difference in their experience. Under uncongested conditions typical file transfers complete in initial slow start. Under heavily congested conditions typical file transfers enter normal congestion avoidance mode. On the other hand, switching to an alternate congestion control mechanism could modestly benefit selected users with high capacity access paths during periods where large file transfers compete for bandwidth on shared, high-capacity paths. These findings are limited to cases where all users on the network: (a) have a high initial slow-start threshold and (b) adopt the same congestion control mechanism. In Chapter 7 we investigate the case of a lower initial slow-start threshold. We address the case of heterogeneity among congestion control mechanisms in Chapters 8 and 9.

6.5.2 Finding #2

When deployed network wide, alternate congestion control algorithm 3 (FAST) can produce macroscopic changes in network behavior at congested places in the topology and during congested periods. Further, these changes can present Web-browsing users with lower average goodputs and longer connection times. The influence of these effects increases with increasing congestion. These findings suggest that deploying FAST on a wide scale could incur significant risk.

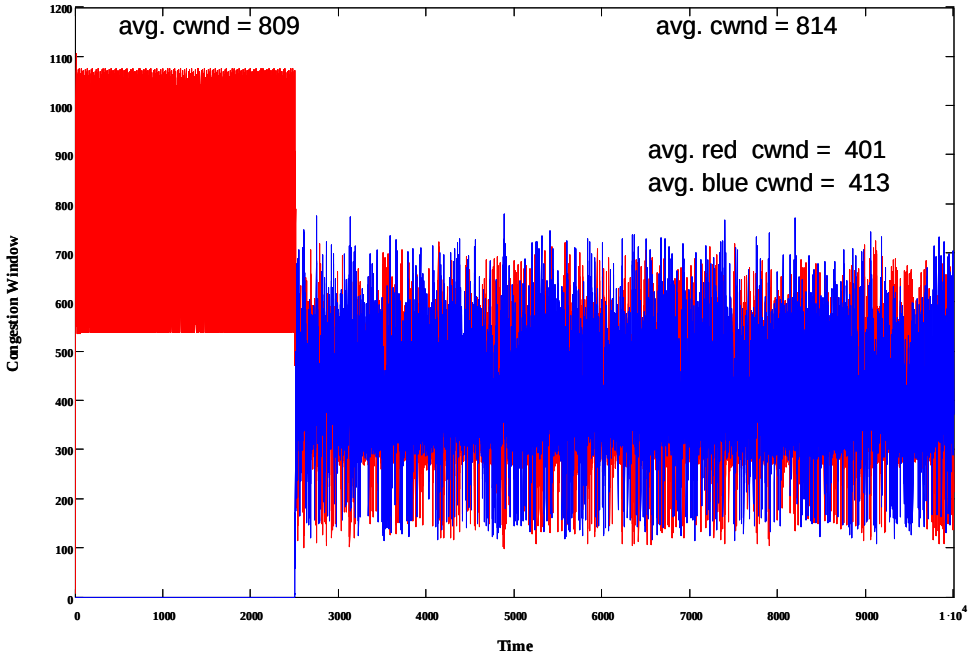


Figure 6-59. Reproduction of Fig. 5-22, Showing Change in *cwnd* for Two FAST Flows (α_F = 200, *rtt* = 42 ms) – x axis gives time in 200 ms increments and y axis gives congestion window in packets ranging from 0 to 1200

To understand this finding, recall from Sec. 5.4.4 that FAST exhibits rapid oscillations in congestion window size when a path has insufficient buffers to contain the packets that the FAST algorithm attempts to maintain queued at a bottleneck. The resulting behavior is illustrated by Fig. 5-22, which, for convenience, we reproduce here

as Fig. 6-59. In this figure, two FAST flows are attempting to maintain 100 packets each through a bottleneck router that has buffers for only 176 packets. Insufficient buffer space results in packet losses, followed by (50 %) window reduction, followed by rapid increase in congestion window. This cycle repeats quite rapidly because FAST flows update their target congestion window frequently (every 20 ms here). This rapid oscillation in congestion window appears to be the source for the deleterious behavior exhibited by FAST in congested locations and at times of significant network-wide congestion, as we elaborate below.

When a large number of flows simultaneously transit a network router, the overall effect can be to flood the router with many packets. When the number of flows is sufficient to overrun the available buffers in the router, FAST flows exhibit an oscillatory behavior that can create additional congestion that causes the flows to remain in oscillation for an extended time. For example, Fig. 6-60 shows the evolution of the congestion window for long-lived FAST flow L2 during 500 measurement intervals within TP2 under (the most congested) condition 21. For comparison, Fig. 6-61 gives the behavior of standard TCP Reno under the same circumstances. Faced with congestion, the other alternate congestion control algorithms we simulated oscillate with a frequency closer to TCP than to FAST. Figs. 6-62 through 6-66 show the behavior for the remaining congestion control algorithms under condition 21 for the same measurement intervals in TP2.

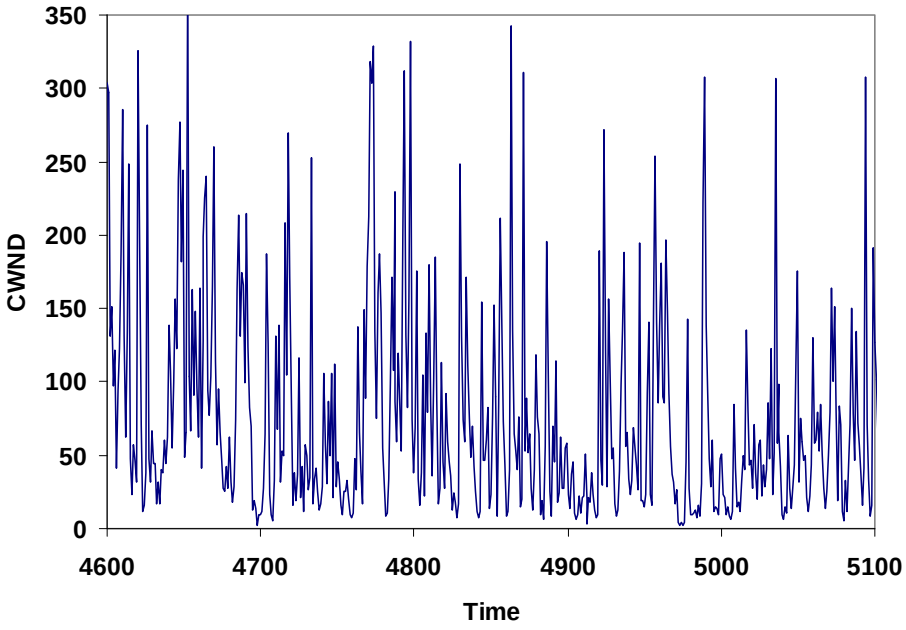


Figure 6-60. Change in Congestion Window (packets) under FAST for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

The rapid oscillatory behavior of FAST results in numerous packet losses, which leads to a larger rate of congestion window increase (as shown in Figs. 6-23, 6-32 and 6-44) and to a higher retransmission rate (as shown in Figs. 6-25, 6-34 and 6-46). The higher loss rate also causes a higher SYN rate (as shown in Fig. 6-56), which leads to a larger number of flows pending in a connecting state (as shown in Figs. 6-27, 6-37 and 6-49) because flows take longer to connect. Flows also take longer to complete because a larger number of packets must be retransmitted. This effect can be seen in Figs. 6-24, 6-

26, 6-33 and 6-45, which show that FAST flows have a significantly lower completion rate. The net effect of a lower completion rate appears in Fig. 6-55, which shows that FAST completes many fewer flows (than other algorithms) over a 25-minute period of network operation. A lower rate of flow completions also means that more flows can be active simultaneously in congested locations in the topology. See, for example, Figs. 6-36 and 6-48. As a result, the average goodput will be lower for flows transiting congested areas, as shown in Figs. 6-35 and 6-47.

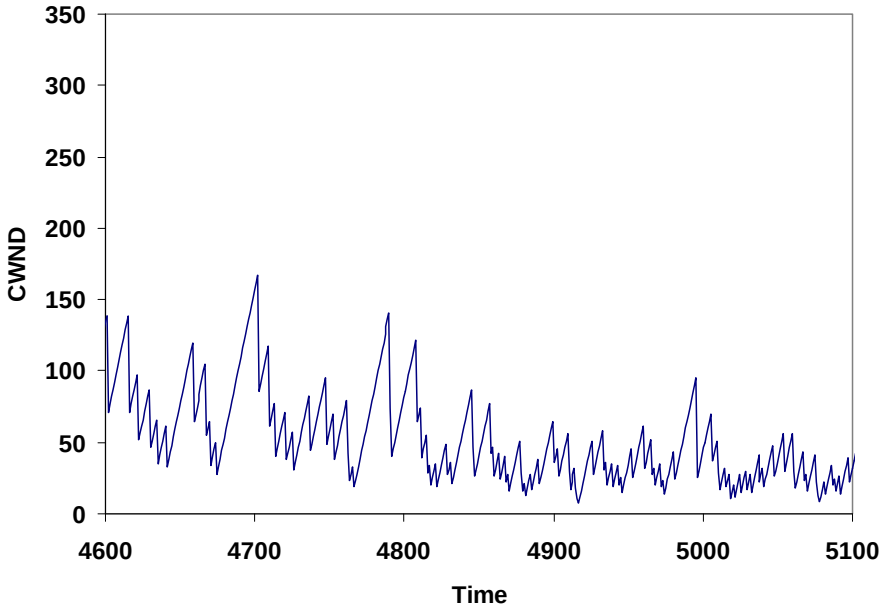


Figure 6-61. Change in Congestion Window (packets) under TCP Reno for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

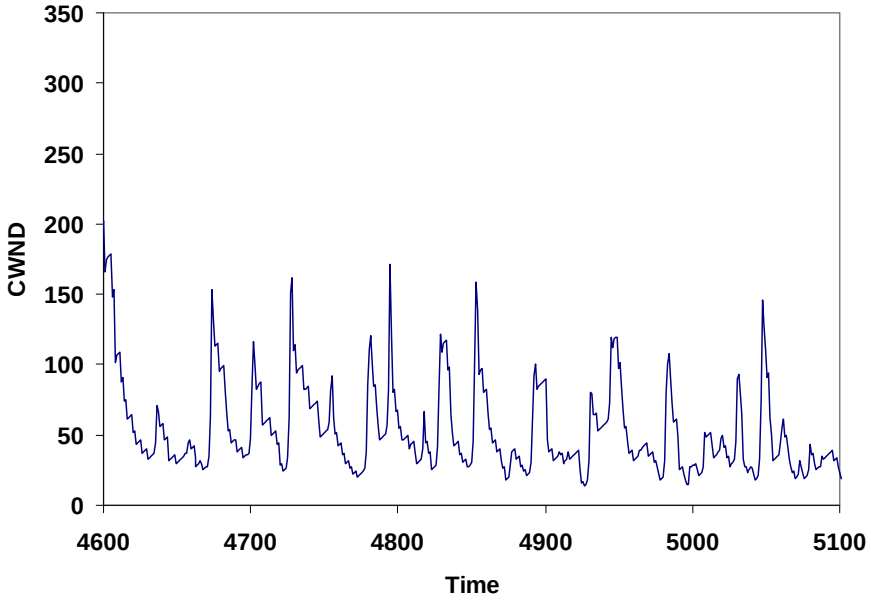


Figure 6-62. Change in Congestion Window (packets) under BIC for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

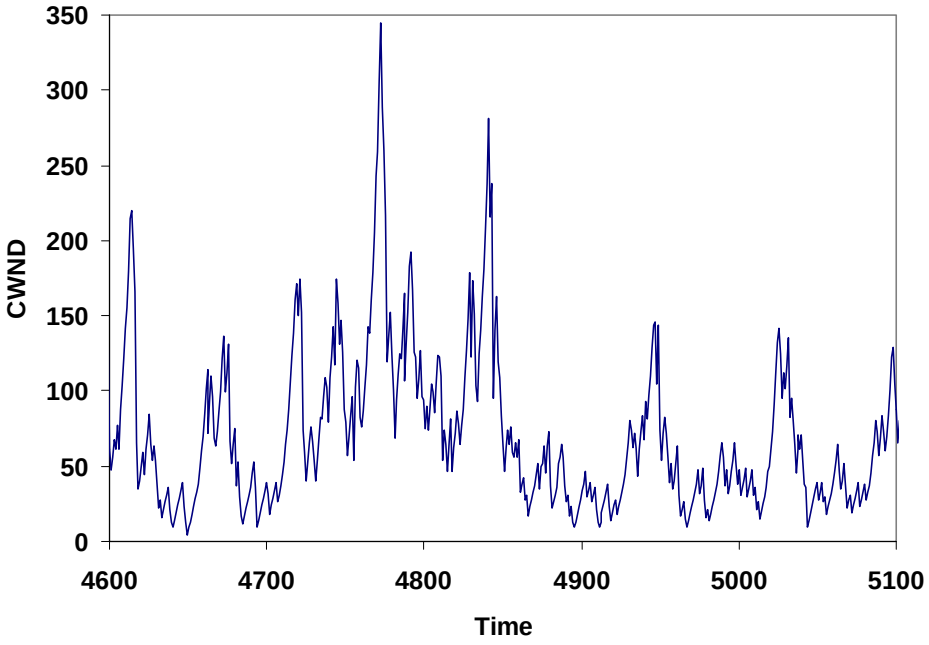


Figure 6-63. Change in Congestion Window (packets) under CTCP for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

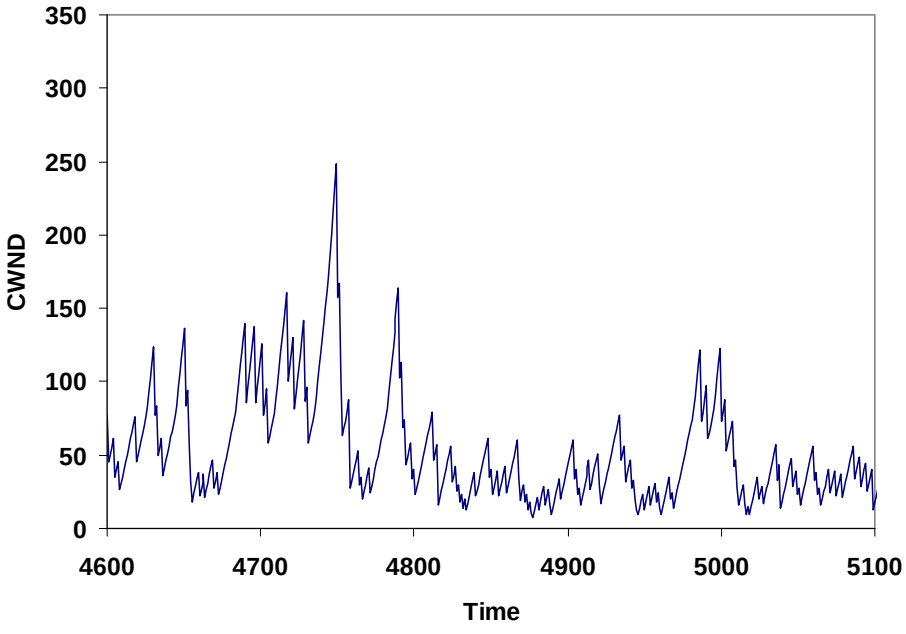


Figure 6-64. Change in Congestion Window (packets) under HSTCP for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

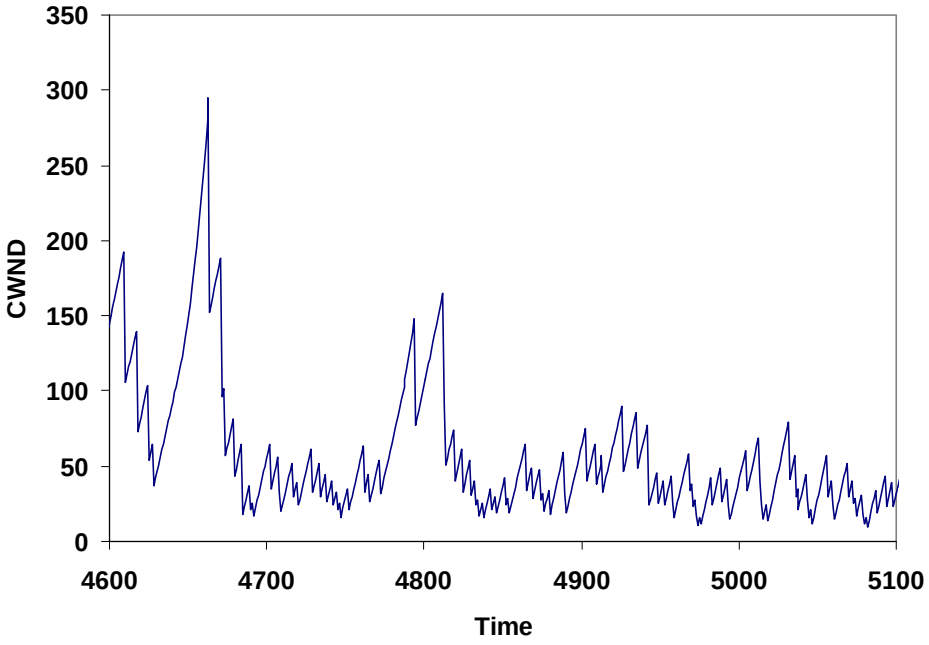


Figure 6-65. Change in Congestion Window (packets) under HTCP for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

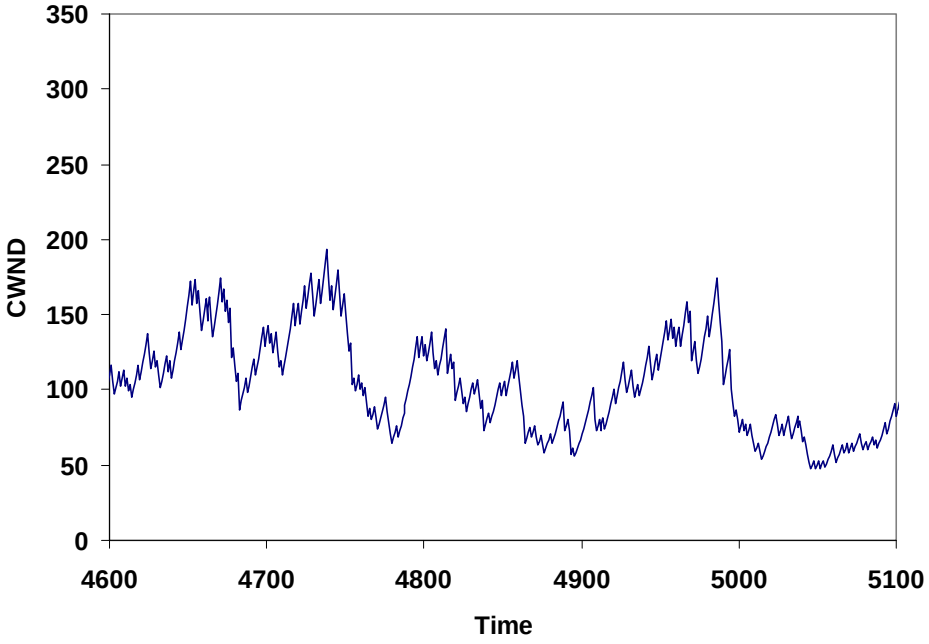


Figure 6-66. Change in Congestion Window (packets) under Scalable TCP for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

In summary, a large network with many simultaneously active flows can induce congestion at various times and locations within the topology. When congestion is sufficient to induce losses, flows using the FAST algorithm can enter a rapid oscillatory behavior that exacerbates congestion. As a result, the network can exhibit a higher overall loss rate with consequent increase in retransmissions. Flows can take longer to connect and complete. The number of flows completed in such a network can be significantly reduced over long time spans. Should FAST be deployed throughout a network, typical Web-browsing users could experience lower average goodput on flows transiting through congested areas. These findings are limited to cases where all users on the network: (a) adopt FAST and (b) FAST is configured as discussed in Sec. 5.2.3 with fixed αF = 200. In Chapter 7 we also investigate the case of FAST configured with α-tuning enabled.

6.5.3 Finding #3

Under certain conditions, CTCP (algorithm 2) can drive congestion window size to substantially higher values than the other congestion control algorithms we simulated. In our experiment, this behavior arose during TP3, as shown in Fig. 6-50, which analyzes average congestion window size. Detailed examination of the relevant time series revealed that this increase in congestion window size can be attributed solely to DD flows.

Recall that during TP2 jumbo file transfers were initiated on DD flows, which introduced substantial congestion within directly connected access routers. At the onset of TP3 no further jumbo transfers are initiated and congestion eases as residual jumbo transfers complete. During this easing period, the congestion window on DD flows can increase – the rate of increase depends upon the level of congestion created during TP2. For example, Fig. 6-67 plots, for six congestion control algorithms, the increase in average congestion window for DD flows during TP3 under condition 12.

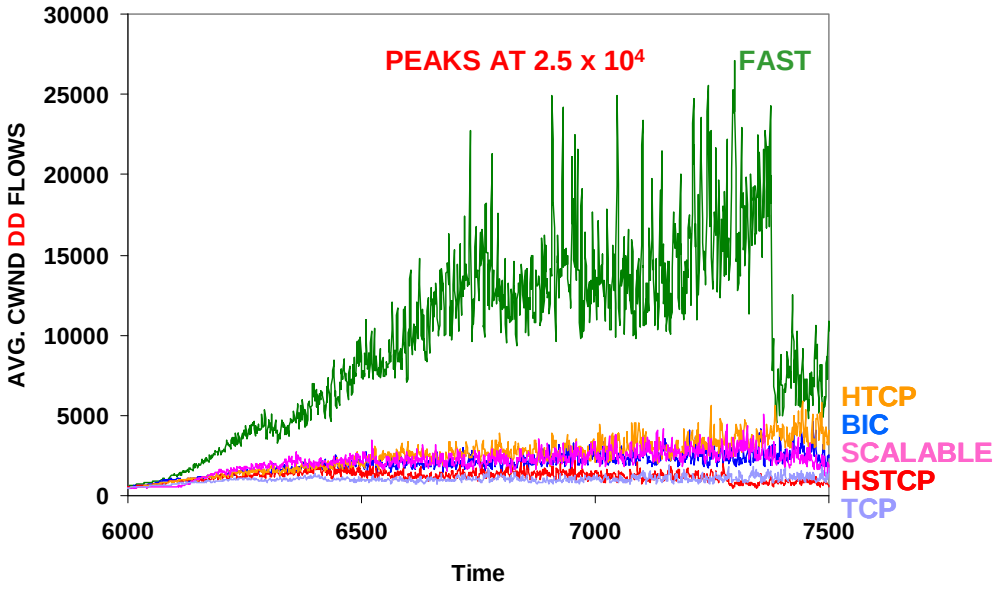


Figure 6-67. Average Congestion Window Size (packets) of DD Flows during TP3 (spanning 1500 200 ms measurement intervals) under Condition 12 for BIC, FAST, HSTCP, HTCP, Scalable TCP and TCP Reno

Fig. 6-67 shows that five of the congestion control algorithms provide a linear increase (with a small slope) in average congestion window size, up to a maximum of about 4 x 10^3 packets. The increase for FAST, which also appears approximately linear but with larger slope, peaks at around 25 x 10^3 packets. The situation for CTCP is much different, as shown in Fig. 6-68, where under the same conditions the average congestion window size increases exponentially, reaching a peak of about 1 million packets.

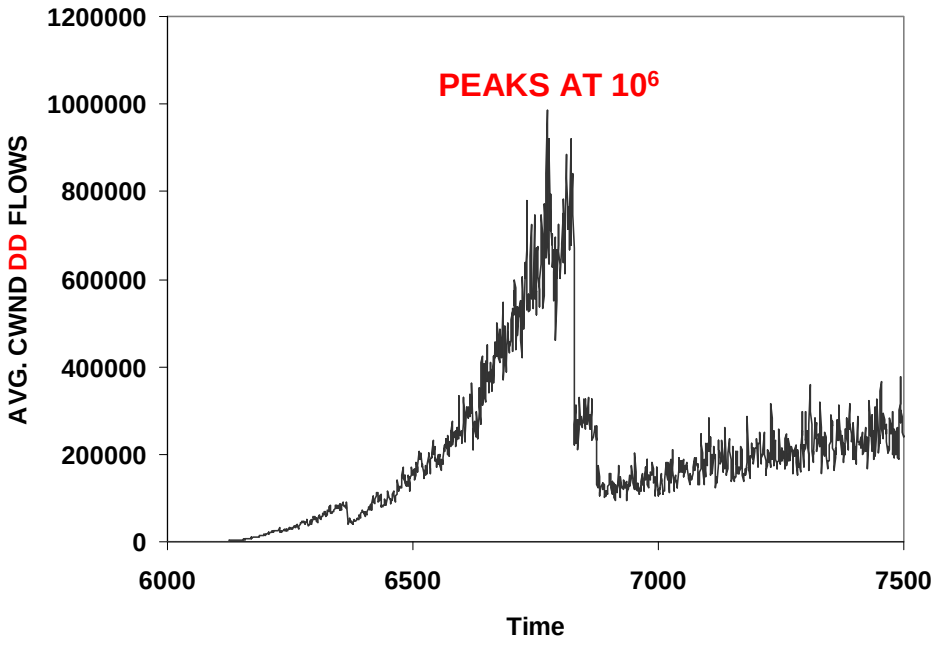


Figure 6-68. Average Congestion Window Size (packets) of DD Flows during TP3 (spanning 1500 200 ms measurement intervals) under Condition 12 for CTCP

To understand this distinctive behavior we must revisit the window management algorithm used by CTCP. Specifically, we are interested in equation (17) from Sec. 5.2.2. We repeat the equation here for convenience.

$$
\begin{aligned}
\textbf{every}(SRTT_C) \equiv\ & E_C \leftarrow \frac{cwnd}{minRTT_C} \\
& A_C \leftarrow \frac{cwnd}{SRTT_C} \\
& D_C \leftarrow (E_C - A_C) \times minRTT_C \\
& \textit{if}\ \ CD_C = \textbf{true} \\
& \quad dwnd \leftarrow \min\left[0, cwnd \times (1 - \beta_C) - \frac{cwnd}{2}\right] \\
& \quad CD_C \leftarrow \textbf{false} \\
& dwnd \leftarrow dwnd + \min\left(0, \alpha_C \times cwnd^{K_C} - 1\right) \ \ \textit{if}\ \ CD_C = \textbf{false} \wedge D_C < \gamma_C \\
& dwnd \leftarrow \min\left[0, dwnd - (\zeta_C \times D_C)\right] \ \ \textit{otherwise} \\
& cwnd \leftarrow \max(\textbf{int\_max}, cwnd + dwnd)
\end{aligned}
\qquad (17)
$$

Equation (17) specifies a periodic algorithm used by CTCP to adjust the delay window as needed every round-trip time (RTT). The CTCP delay window augments the congestion window. The highlighted line in (17) shows that CTCP will increase the delay window exponentially when no congestion has been detected and the actual congestion window is within (γ_C =) 30 packets of the expected congestion window. In other words, if there is no congestion and the actual window is close to what is expected from previous measurements, then perhaps the window can be increased because congestion is easing.

In our scenario, DD flows that start during TP2 are likely to face stiff congestion, which implies that the initial minimum RTT for these flows will be somewhat high. At the onset of TP3, congestion eases as residual jumbo transfers complete. Easing congestion causes measured SRTT (smoothed RTT) to fall, thus minimum RTT recorded on these flows will be driven down. As a result, the minimum RTT and the measured SRTT will be identical, or nearly so. Thus, the difference in expected and actual congestion window, as computed by the CTCP algorithm, will be around zero. As SRTT continues to fall, and minimum RTT falls with it, the highlighted line in (17) will be executed during each RTT. Naturally, this leads to an exponential increase in the congestion window.

Under our scenario, this exponential congestion window increase has little practical implication because a source cannot transmit faster than its maximum interface speed (or the maximum interface speed of a slower receiver). Note, however, that under easing congestion and no packet losses the CTCP congestion window continues to increase exponentially until a transfer completes even though the source is unable to increase its transmission rate. This situation is analogous to initial slow start, which also increases the congestion window exponentially. Given an arbitrarily high initial slow-start threshold, a large file transfer that proceeds without packet loss will likely remain in initial slow start until the transfer completes. Under these circumstances the congestion window grows exponentially even though the source is unable to increase its transmission speed beyond a physical maximum. In theory, a CTCP flow (or any flow operating within initial slow start) could achieve a very high window (e.g., millions of packets). A subsequent loss on a flow that has achieved such a high window could require many losses to reduce the window (by 50 % per loss) to a point where the transmission rate is throttled sufficiently to respond to the congestion signal. The possibility for such an outcome suggests that some practical upper limit should be placed on delay window size, though most TCP implementations place an upper limit on the size of the congestion window.

6.5.4 Finding #4

Focusing on longed-lived flows reveals several points of interest. First, during TP1 all congestion control algorithms showed nearly identical goodput on the three long-lived flows – the less the congestion, the closer the goodput. This occurs because the initial slow-start threshold was set to an arbitrarily high value. During TP1, when the long-lived flows commenced amid a background of Web traffic, initial slow-start was typically able to carry the long-lived flows to the maximum achievable transmission rate (960 Mbps). Since all congestion control algorithms adopted identical initial slow-start procedures, this finding should not be surprising. (In Chapter 7 we investigate effects from a lower initial slow-start threshold.)

When heavy congestion strikes, as jumbo file transfers commence in TP2, algorithm 6 (Scalable TCP) exhibited a tendency to provide higher goodput on the long-lived flows than did the other congestion control algorithms. This comparative advantage of Scalable TCP tended to increase with increasing propagation delay and decrease with increasing congestion. Detailed analyses of long-lived flows (e.g., 6-38 and 6-39) did not find the goodput advantage of Scalable TCP to be statistically significant (5 %) under many conditions, but this appears influenced by the wide range of goodputs exhibited. The reason that Scalable TCP tended to provide higher goodputs on long-lived flows during TP2 is that newly arriving flows have more difficulty claiming their share of bandwidth when the competing flows are all using the Scalable congestion avoidance algorithm. This difficulty was illustrated in Chapter 5 (see Figs. 5-31 to 5-33). Further evidence of this effect is shown in Fig. 6-69, which compares algorithms 3 (FAST) and 6 (Scalable TCP) with respect to decrease in congestion window size for all three long-lived flows at the onset of TP2 under condition 27 (light-to-moderate congestion).

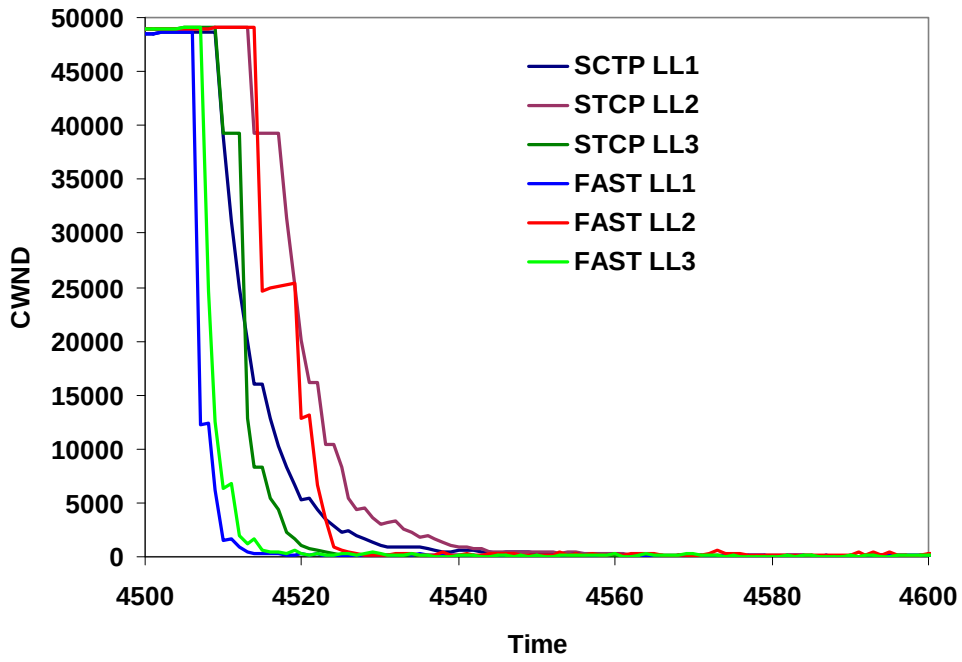


Figure 6-69. Comparing Congestion Window Size (packets) of Scalable TCP (STCP) and FAST with respect to Falling Congestion Window for Three Long-Lived Flows during the First 100 measurement intervals (200 ms each) of TP2 under Condition 27

The comparative advantage of Scalable TCP vanished in TP3 as congestion abated and most of the alternate congestion control algorithms recovered well. The most notable effect for long-lived flows during TP3 is that standard TCP lags in recovering peak goodput. This finding is as expected. In fact, the sluggishness shown by standard TCP when recovering from congestion provides motivation for researchers to propose alternate congestion control algorithms.

6.5.5 Tendencies

Given that algorithm 3 showed several distinctive behaviors, we discarded its response data and then conducted our detailed analyses a second time on the remaining congestion control algorithms. As already demonstrated, the remaining algorithms could not be distinguished using tests for statistically significant differences. On the other hand, we noted earlier that algorithms 1 (BIC) and 6 (Scalable TCP) showed some tendency to behave similarly to each other and distinctly from other algorithms. Based on our supplementary analyses, we identified some tendencies that, though they cannot be considered findings, might illuminate differences among alternate congestion control algorithms (excluding FAST). The tendencies we identify are from rather small differences in relative and absolute effect. If nothing else, these tendencies help to explain why BIC and Scalable TCP clustered together under many conditions.

The first observation to note is that BIC and Scalable TCP behaved more similarly under congested conditions. In part this is due to the fact that all the algorithms tended to behave similarly under uncongested conditions, so behavioral distinctions appeared only with increasing congestion. We note that BIC and Scalable TCP tended to push more packets through the network, while completing fewer flows. Algorithms 5 (HTCP) and 7 (TCP) exhibited the opposite tendencies (i.e., fewer packets pushed through and more flows completed). One factor affecting these trends is that BIC and Scalable TCP tended to complete fewer NN flows, which were most numerous and also had the lowest potential for goodput, while CTCP (algorithm 2), HTCP (algorithm 5) and TCP tended to complete more of such flows. From this, we conclude that BIC and Scalable TCP showed a tendency to push more packets through the network for flows that could achieve higher goodputs (e.g., long-lived flows and other flows over fast and very fast paths). Another way to look at this is that (in this experiment) CTCP, HTCP and TCP provided fairer bandwidth sharing under heavy congestion than either BIC or Scalable TCP. This confirms differences demonstrated earlier in Sec. 5.4. These differences led to some distinctions in network-wide behavior.

The average congestion window size tended to be higher under BIC and Scalable TCP; this higher average was due largely to bigger windows on advantaged flows. Pushing more packets into the network also led BIC and Scalable TCP to have higher retransmission rates, larger queuing delays and higher SYN rates (along with more flows pending in the connecting state). While not statistically significant in this experiment, the differences we highlight provide some tendencies that might separate BIC and Scalable TCP qualitatively from the other congestion control algorithms.

6.6 Conclusions

In this section we described an experiment comparing alternate congestion control algorithms deployed in a large, fast network with typical Web traffic, a few long-lived flows and a period of large file transfers between selected locations, followed by easing congestion. The specific experiment design we described follows a general approach that we will apply repeatedly in subsequent chapters to compare congestion control algorithms under various circumstances. In addition, we defined a data analysis approach that allowed us to find key differences, where they existed, among various congestion control algorithms. We applied the experiment design and data analysis approaches to compare seven alternate congestion control algorithms within a simulated network. In a

given simulation, all sources in the network used the same congestion control algorithm. This unrealistic assumption of homogeneity aided our analysis and allowed us to identify differences among the algorithms we compared. We subjected each congestion control algorithm to the same 32 conditions, which provided a range of congestion levels.

We demonstrated that (aside from FAST) under our scenario and conditions the alternate congestion control algorithms exhibited indistinguishable macroscopic behavior and modest differences in user experience. We showed that the behaviors were more similar in uncongested conditions. We also explained why this was the case. We showed that FAST can exhibit distinctive, undesirable network-wide behavior, which grows more distinctive under increasing congestion. We described the root cause of this distinctive behavior, and we argued that deploying FAST throughout the Internet might entail significant risk. We identified an element of the CTCP delay-window adjustment algorithm that can lead to an exponential increase in congestion window under particular circumstances associated with easing congestion. We showed that Scalable TCP tends to retain a higher congestion window for a longer time on long-lived flows under periods of increasing congestion. We identified some tendencies for BIC and Scalable TCP to provide higher goodputs on large flows with high available bandwidth, while providing lower quality of service on more numerous, typical flows with lower available bandwidth.

In the next section, we repeat the current experiment while changing only a few parameters. We scale down the network by one order of magnitude in size (number of sources and receivers) and speed. We intend to show that a scaled-down simulation, which requires much less computing resources, can reveal findings similar to a larger simulation. We also lower the initial slow-start threshold to a relatively small number of packets. Decreasing the initial slow-start threshold will allow flows to enter congestion avoidance earlier. More frequent activation of congestion avoidance under low congestion might reveal additional information about differences among congestion control algorithms. Finally, we add the α-tuning variant of FAST as an eighth congestion control algorithm to consider. Here, we seek to understand whether activating α-tuning might lead to improved behavior for FAST.

Chapter 7 – Comparing Congestion Control Regimes in a Scaled-Down Network

7 Comparing Congestion Control Regimes in a Scaled-Down Network

In this section, we repeat the previous experiment (from Chapter 6) making a few parameter changes and including an additional congestion control regime: FAST with α-tuning enabled. Our parameter changes include reducing network size (number of sources and receivers) and speed by about an order of magnitude and reducing the initial slow-start threshold to a relatively low value. As shown in Table 7-1, we retain the algorithm identifiers from Chapter 6, adding FAST with α-tuning enabled (FAST-AT) as algorithm number eight. We also retain the topology (Fig. 6-1), scenario (Fig. 6-2), path classes (Table 6-2), and fixed parameters for the network (Table 6-4), for simulation control (Table 6-10), for user traffic (Table 6-11) and for long-lived flows (Table 6-12). In addition, we measure the same responses defined in Chapter 6 (recall Tables 6-15 through 6-25). We collect data in the same fashion as described in Sec. 6.2.2. We also adopt the same fundamental approach to data analysis (as described in Sec. 6.3).

Table 7-1. Congestion Control Mechanisms Compared

Identifier	Label	Name of Congestion Avoidance Algorithm
1	BIC	Binary Increase Congestion Control
2	CTCP	Compound Transmission Control Protocol
3	FAST	Fast Active-Queue Management Scalable Transmission Control Protocol
4	HSTCP	High-Speed Transmission Control Protocol
5	HTCP	Hamilton Transmission Control Protocol
6	Scalable	Scalable Transmission Control Protocol
7	TCP	Transmission Control Protocol (Reno)
8	FAST-AT	FAST with α-tuning Enabled

We expect the scaled-down network simulation to require an order of magnitude fewer resources, while confirming the main findings from simulating a large, fast network. Scaling down network size and speed by a similar factor should generate conditions with relative congestion aligned to those described in Chapter 6 (recall Figs. 6-5 through 6-8). We also expect FAST-AT to exhibit similar response to congested conditions as FAST. This expectation arises from the fact that FAST and FAST-AT showed similar oscillatory behavior when simulated in a single-path topology with insufficient buffers. We also expect that lowering the initial slow-start threshold from $2^{31}/2$ to 100 packets will have only a limited effect on our results. This expectation arises from the fact that Web objects retain an average size of no more than 100 packets, which should enable the transfer of Web objects to complete under initial slow start during uncongested conditions. In uncongested conditions, document transfers, with average size of up to 10^3 packets, might be affected by the lower initial slow-start threshold, but such flows make up only about 1 % of all transfers. During congested conditions initial slow

start tends to end before the congestion window reaches the initial slow-start threshold. We do expect a lower initial slow-start threshold to have a negative influence on long-lived flows that use standard TCP congestion avoidance. Long-lived flows have an infinite size, so congestion avoidance procedures are activated before such flows reach a maximum achievable transfer rate. Since standard TCP congestion avoidance procedures lead to a linear increase in the congestion window, long-lived TCP flows transiting high-delay paths should take quite some time to achieve maximum rate.

We organize what follows into six sections. Sec. 7.1 describes the experiment design, concentrating on changes from the previous experiment where we simulated a large, fast network. Sec. 7.1 also explains how the revised experiment design influences the domain view of the simulated network. Sec. 7.2 compares resource requirements for simulating a large, fast network against resource requirements for simulating a scaled-down network. Sec. 7.3 explains the nature of (and rationale for) a tactical change in the data analysis approach we used to investigate the scaled-down network. Sec. 7.4 presents selected results from simulating a scaled-down network, while Sec. 7.5 discusses key findings from the results. We conclude in Sec. 7.6.

7.1 Experiment Design

We adopt the same 2^{6-1} orthogonal fractional factorial design template (see Table 6-13) used in Chapter 6. As discussed below, we change only one robustness factor and two fixed parameters and then instantiate the design template to create 32 simulated conditions.

7.1.1 Changes in Robustness Factors and Fixed Factors

Table 7-2 specifies the robustness factors and values we used for this experiment. Recall that robustness factors define the range of parameter combinations over which experiment conclusions will hold. We highlight (in red) our changes (from Table 6-3) to robustness factor x1 (network speed). These changes result in a network that operates at only 15 % of the speed we simulated in Chapter 6.

Table 7-2. Robustness Factors Adopted for Comparing Congestion Control Mechanisms

Identifier	Definition	PLUS (+1) Value	Minus (-1) Value
x1	Network Speed	1200 packets/ms	600 packets/ms
x2	Think Time	5000 ms	2500 ms
x3	Source Distribution	Uniform (.33/.33/.33)	Skewed (.1/.6/.3)
x4	Propagation Delay	2	1
x5	File Size	100 packets	50 packets
x6	Buffer Sizing Algorithm	RTTxCapacity	RTTxCapacity/SQRT(*N*)

We make only two other parameter changes from our previous experiment. The changes, shown below in red in Table 7-3, affect fixed parameters related to sources and receivers. We reduce the base number of sources under an access router from 1000 to 100 and we reduce the initial slow-start threshold from $2^{31}/2$ to 100 packets.

7.1.2 Orthogonal Fractional Factorial Design of Robustness Conditions

We inject the robustness factors from Table 7-2 into the design template from Table 6-13 to yield 32 instantiated robustness conditions, as shown in Table 7-4. Changes in the robustness conditions (from Table 6-14) are emphasized in red.

Table 7-3. Fixed Parameters Related to Sources and Receivers

Parameter	Definition	Value
Bsources	Basic unit for sources per access router	100
ΔU	Avg. sources per access router = Bsources x ΔU	2
P(Nr)	Probability receiver under normal access router	0.6
P(Nrf)	Probability receiver under fast access router	0.2
P(Nrd)	Probability receiver under directly connected access router	0.2
sst_{INT}	Initial slow-start threshold (packets)	100

Table 7-4. Instantiated Robustness Conditions

Factor->	X1	X2	X3	X4	X5	X6
Condition	--	--	--	--	--	--
1	600	2500	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
2	1200	2500	.1/.6/.3	1	50	RTTxCapacity
3	600	5000	.1/.6/.3	1	50	RTTxCapacity
4	1200	5000	.1/.6/.3	1	50	RTTxCapacity/SQRT(N)
5	600	2500	.3/.3/.3	1	50	RTTxCapacity
6	1200	2500	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
7	600	5000	.3/.3/.3	1	50	RTTxCapacity/SQRT(N)
8	1200	5000	.3/.3/.3	1	50	RTTxCapacity
9	600	2500	.1/.6/.3	2	50	RTTxCapacity
10	1200	2500	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
11	600	5000	.1/.6/.3	2	50	RTTxCapacity/SQRT(N)
12	1200	5000	.1/.6/.3	2	50	RTTxCapacity
13	600	2500	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
14	1200	2500	.3/.3/.3	2	50	RTTxCapacity
15	600	5000	.3/.3/.3	2	50	RTTxCapacity
16	1200	5000	.3/.3/.3	2	50	RTTxCapacity/SQRT(N)
17	600	2500	.1/.6/.3	1	100	RTTxCapacity
18	1200	2500	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
19	600	5000	.1/.6/.3	1	100	RTTxCapacity/SQRT(N)
20	1200	5000	.1/.6/.3	1	100	RTTxCapacity
21	600	2500	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
22	1200	2500	.3/.3/.3	1	100	RTTxCapacity
23	600	5000	.3/.3/.3	1	100	RTTxCapacity
24	1200	5000	.3/.3/.3	1	100	RTTxCapacity/SQRT(N)
25	600	2500	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
26	1200	2500	.1/.6/.3	2	100	RTTxCapacity
27	600	5000	.1/.6/.3	2	100	RTTxCapacity
28	1200	5000	.1/.6/.3	2	100	RTTxCapacity/SQRT(N)
29	600	2500	.3/.3/.3	2	100	RTTxCapacity
30	1200	2500	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
31	600	5000	.3/.3/.3	2	100	RTTxCapacity/SQRT(N)
32	1200	5000	.3/.3/.3	2	100	RTTxCapacity

7.1.3 Domain View of Robustness Conditions

Changes in simulated speed and size influence the domain view of our simulated network. Table 7-5 shows the simulated router speeds for this experiment, which are reduced by about an order of magnitude over the values given in Table 6-5. Changes in router speeds are highlighted in red in Table 7-5. Reduction in **Bsources** (base number of sources) leads to an order of magnitude fewer sources, as shown in Table 7-6, which highlights in red changes from Table 6-9.

Table 7-5. Domain View of Router Speeds

Router	PLUS (+1)	Minus (-1)
Backbone	28.8 Gbps	14.4 Gbps
POP	3.6 Gbps	1.8 Gbps
Normal Access	360 Mbps	180 Mbps
Fast Access	720 Mbps	360 Mbps
Directly Connected Access	3.6 Gbps	1.8 Gbps

Table 7-6. Number of Simulated Sources

PLUS (+1)	Minus (-1)
27.8×10^3	17.46×10^3

We use the same topology as the previous experiment and we simulate the same propagation delays (shown in Table 6-6). Recall, though, that buffer sizing is influenced by up to three factors: network speed, propagation delay and number of sources. Since we have changed two of these factors, simulated buffer sizes also change, as shown in Table 7-7, which highlights in red changes in buffer sizes from the previous experiment (see Table 6-7).

Table 7-7. Buffer Sizes Simulated

	PLUS (+1)			Minus (-1)		
Router	Min	Avg	Max	Min	Avg	Max
Backbone	48.830×10^3	109.866×10^3	195.317×10^3	547	1.236×10^3	2.208×10^3
POP	6.104×10^3	13.734×10^3	24.415×10^3	105	240	431
Access	971	2.184×10^3	6.104×10^3	44	99	105

Overall, the order of magnitude reduction in network speed and number of sources should scale the network model so that simulated conditions exhibit about the same relative congestion levels as the previous experiment. Fig. 7-1 plots the retransmission rates (y axis) for each of the 32 simulated conditions. The x axis is plotted in order of increasing retransmission rate. Comparing this plot with Fig. 6-5, we see that

the order of the conditions shifts around slightly. Congestion levels in the scaled-down network are reduced somewhat, as can be seen by the fact that the most congested condition (21) has a retransmission rate of around 30 % as compared with 50 % for the larger network. Uncongested conditions in the scaled-down network have retransmission rates two orders of magnitude lower than exhibited by the larger network. Selected conditions with moderate congestion (4, 10, 16, 28 and 11) exhibit increased retransmission rates in the scaled-down network, as compared with the larger network. In fact, condition 11 can now be considered congested – implying that the scaled-down network has 17 congested conditions, labeled **C** in Fig. 7-1, as compared with 16 congested conditions in the previous experiment (see Fig. 6-5). As one can see, condition 11 exhibits a substantially higher retransmission rate than condition 28 but a substantially lower rate than condition 22. We arbitrarily divide the remaining conditions into three categories reflecting no (**N**), limited (**L**) and moderate (**M**) congestion.

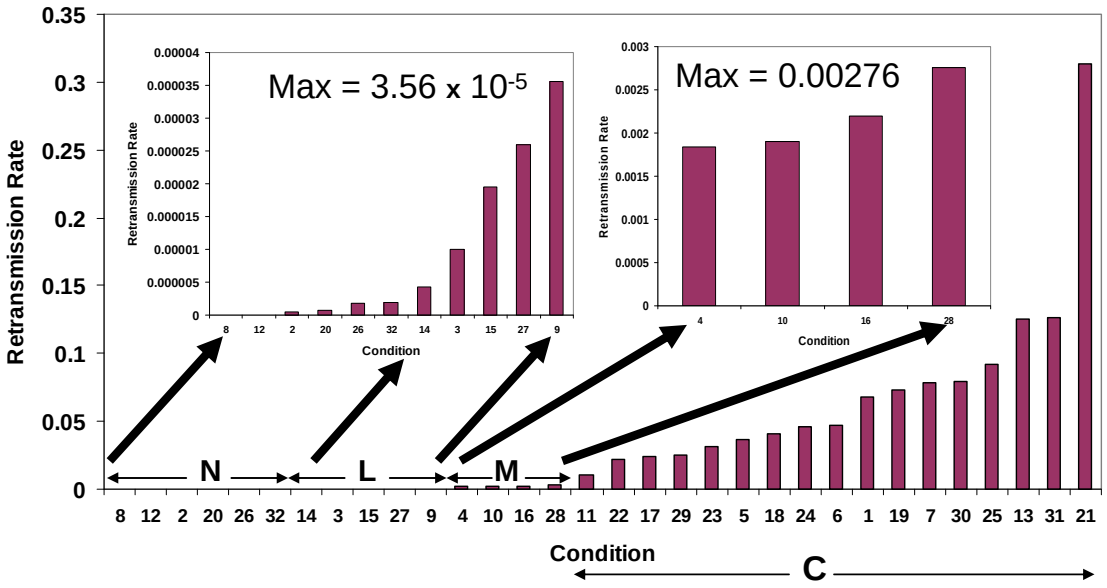


Figure 7-1. Retransmission Rate (proportion of retransmitted packets) vs. Simulated Conditions Ordered from Least to Most Congested

The range of retransmission rates in Fig. 7-1 spans about six orders of magnitude, as was the case in Fig. 6-5. On the other hand, the two least congested conditions (8 and 12) in Fig. 7-1 show no retransmissions. In the large, fast network (Fig. 6-5) condition 12 had 6 retransmissions in 10^9 packets and condition 8 had 6 retransmissions in 10^7 packets. This difference can be attributed to the fact that the larger network typically had an order of magnitude more active flows and a higher slow-start threshold, both of which increase the likelihood of lost packets.

As with the previous experiment, we select one uncongested and one congested condition to examine more closely. For the large, fast network we examined conditions 4 (Fig. 6-6) and 5 (Fig. 6-7) under standard TCP congestion control. For the scaled-down network we also examine congested condition 5, but we select uncongested condition 3 because condition 4 has moved from the category of little congestion (in Fig. 6-5) to moderate congestion (in Fig. 7-1).

Fig. 7-2 plots several time series that show the change in flow states for uncongested condition 3 under standard TCP congestion control. The x axis displays time (in 200 ms intervals since the beginning of the simulation) over the three time periods (final 15 simulated minutes) measured for the scenario. The y axis indicates the number of active (red curve) and connecting (yellow curve) flows. Additional curves decompose active flows by congestion control states: initial slow start (green curve) and normal congestion control (brown curve). Fig. 7-2 resembles Fig. 6-6, which plots the change in flow states for uncongested condition 4 in the large, fast network. The network is sufficiently uncongested that most transfers during TP1 (3000-4500) complete in initial slow start. Things change during TP2 (4500-6000) as jumbo transfers induce congestion in directly connected access routers. Congestion leads to losses, which increases the number of flows operating under normal congestion control procedures. As jumbo transfers diminish during TP3 (6000-7500), congestion decreases so that, by time t = 6600, most active flows again complete transfers without packet loss. Plots for other uncongested conditions show similar patterns.

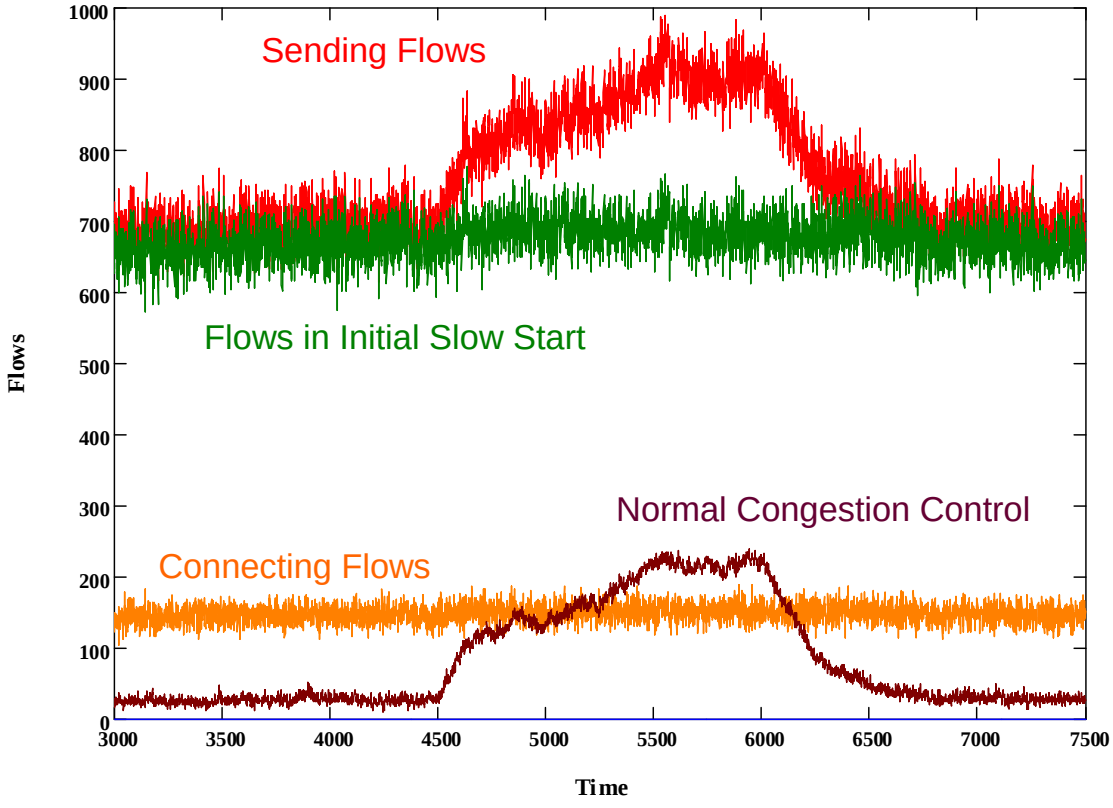


Figure 7-2. Change in Flow States over Three Time Periods under Condition 3 for Standard TCP – x axis gives time in 200 ms increments from the beginning of the simulation, covering the final 15 minutes of the simulated scenario, and y axis gives the number of flows

Fig. 7-3 illustrates the change in flow states for condition 5, a representative congested condition. The number of active flows (red curve) shows an order of magnitude increase over uncongested condition 3. Comparing Fig. 7-3 with Fig. 6-7 reveals some similarities and some differences. Both plots show that network congestion is sufficiently high that introducing jumbo transfers in TP2 (4500-6000) makes little

difference in the overall distribution of flow states. In the scaled-down network, about 60 % of active flows operate under normal congestion control (brown curve) and 40 % operate in initial slow start (green curve). On the other hand, Fig. 6-7 shows that under the large, fast network about 85 % of active flows operate in normal congestion control. This difference occurs over the range of all congested conditions until the three most congested conditions (13, 31 and 21). Under these highest levels of congestion, the relative proportion of active flows using normal congestion control appears similar for both the large and scaled-down network.

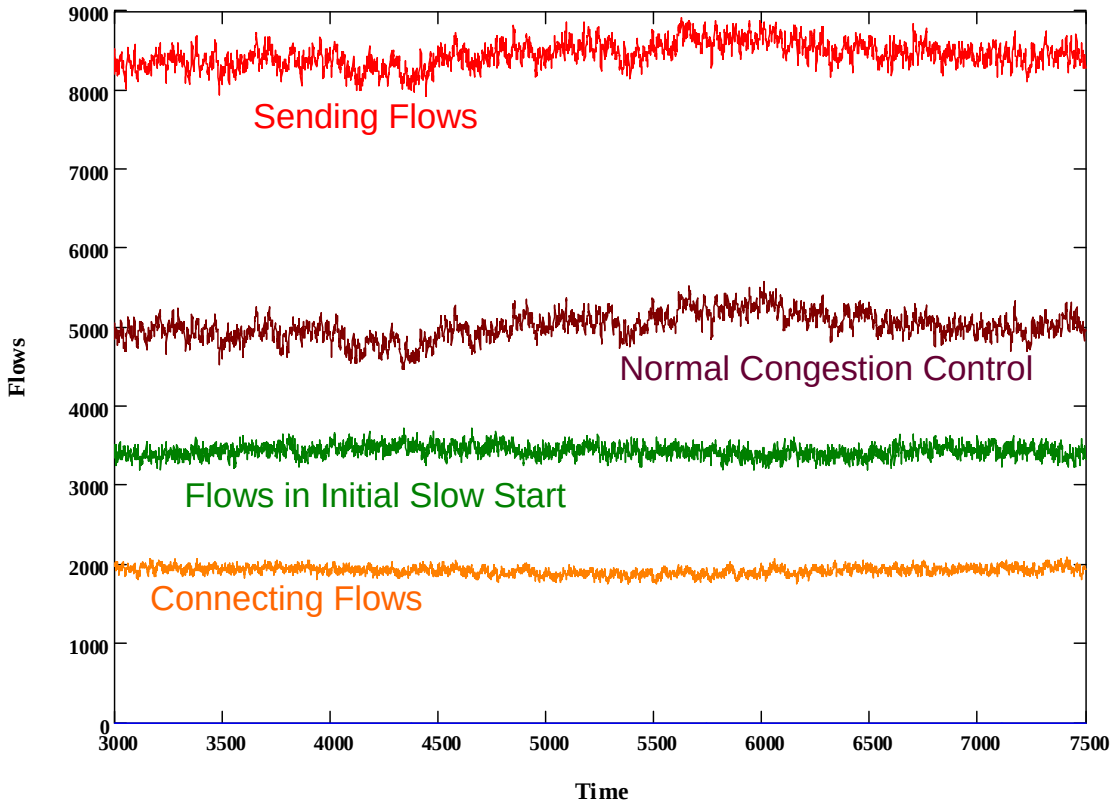


Figure 7-3. Change in Flow States over Three Time Periods under Condition 5 for Standard TCP – x axis gives time in 200 ms increments from the beginning of the simulation, covering the final 15 minutes of the simulated scenario, and y axis gives the number of flows

To further understand similarities and differences in conditions created by the scaled-down network versus the large, fast network, we can plot a cluster analysis (similar to Fig. 6-8) for each of the 32 conditions and eight congestion control algorithms across all response variables and then label each condition with the congestion class identified in Fig. 7-1. We show the annotated cluster analysis as Fig. 7-4, which encompasses the first time period (TP1). Comparing Fig. 7-4 and Fig. 6-8 we find that 27 of the 32 conditions retain the same classification in both figures. Three conditions (9, 26 and 32) remain classified as uncongested but fall into the next lower congestion category as we scale down the network simulation. One condition (4) remains classified as uncongested but with a movement to the next higher congestion category (from **L** to **M**). Only condition 11 becomes congested from uncongested (**M**).

The foregoing analysis demonstrates that scaling down the network model leads to congestion conditions that show reasonable alignment with the large, fast network. Overall, conditions, especially uncongested conditions, tend to be less congested under the scaled-down network. This lessening of congestion can be attributed to a lower initial slow-start threshold and a decrease in the number of active flows throughout the network.

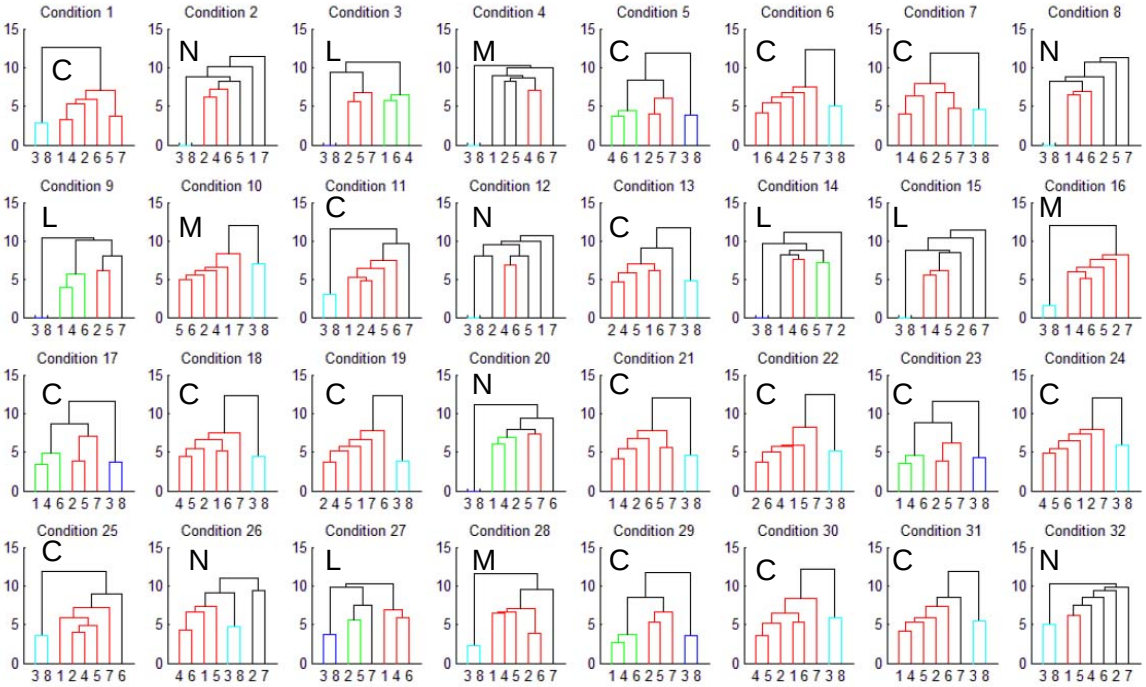


Figure 7-4. Cluster Analysis for Time Period One – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in the response space between pairs of algorithms and clusters of algorithms as indicated - each sub-plot also labeled with Congestion Level

7.2 Experiment Execution and Data Collection

Table 7-8 compares processing and memory requirements for simulating the FAST congestion control algorithm in the scaled-down network against resource requirements for simulating FAST in a large, fast network (Chapter 6). All simulations compared in Table 7-8 were executed on identical compute servers (ws11 through ws14, described in Table 6-26). As expected, scaling down the network simulation by an order of magnitude leads to a similar reduction in the processing time and memory usage. Table 7-9 shows that this reduction in resource usage arises because the scaled-down network simulates 10 times fewer flows and packets per run. We collect data as described in Sec. 6.2.2, so reducing the scale of the simulation does not reduce the amount of data collected.

7.3 Data Analysis Approach

While we use the same fundamental data analysis approach described in Sec. 6.3, we adopt a tactical change in order to enhance clarity. Careful review of Fig. 7-4 reveals that algorithms 3 (FAST) and 8 (FAST with α-tuning enabled) exhibit similar performance under all conditions. In fact, the cluster analysis groups the two algorithms together into the same exclusive cluster for each of the 32 conditions. This indicates that the two

algorithms respond similarly to similar conditions. This should come as no surprise because algorithms 3 and 8 share the same underlying FAST procedures, differing only with respect to treatment of the α parameter.

Table 7-8. Comparing Resource Requirements for Simulating the FAST Congestion Control Algorithm in a Large, Fast Network and a Scaled-Down Network

	Large, Fast Network	Scaled-Down Network
CPU hours (32 Runs)	2241.6	197.3
Avg. CPU hours (per run)	70.1	6.2
Min. CPU hours (one run)	34.6	2.52
Max. CPU hours (one run)	124.1	12.3
Avg. Memory Usage (Mbytes)	1250	139

Table 7-9. Comparing Number of Simulated Flows and Packets for a Large, Fast Network and a Scaled-Down Network under All Congestion Control Algorithms

	Large, Fast Network Chapter 6		Scaled-Down Network Chapter 7	
Statistic	Flows Completed	Data Packets Sent	Flows Completed	Data Packets Sent
Avg. Per Condition	74.033×10^6	6.912×10^9	8.329×10^6	897.379×10^6
Min. Per Condition	40.966×10^6	3.146×10^9	4.329×10^6	380.349×10^6
Max. Per Condition	154.915×10^6	11.917×10^9	16.730×10^6	1.749×10^9
Total All Runs	16.583×10^9	1.548×10^{12}	2.132×10^9	229.729×10^9

Similarity in responses for algorithms 3 and 8 lead to paired, extreme response values under many conditions. As a result, responses for the two algorithms are not necessarily distinguished as significant outliers by a Grubbs' test. For example, examine Fig. 7-5, which gives a detailed analysis of the retransmission rate among all algorithms and conditions during the first time period (TP1). The figure shows that algorithms 3 and 8 have extreme, high retransmission rates under congested conditions. Note, however, that in none of these conditions does a response satisfy our selected cutoff (> 2.08 deviations from the residual mean) for the Grubbs' test. The existence of two, similar, extreme values for two related congestion control algorithms inflates the mean, which causes the Grubbs' value to reach only 1.6. This prevents automated highlighting (green for high and red for low) of either algorithm as an outlier. Without highlighting, the detailed analyses become more difficult to interpret, especially because similar response values for algorithms 3 and 8 can lead to an overstrike of the two numbers on the plots, as shown in Fig. 7-5.

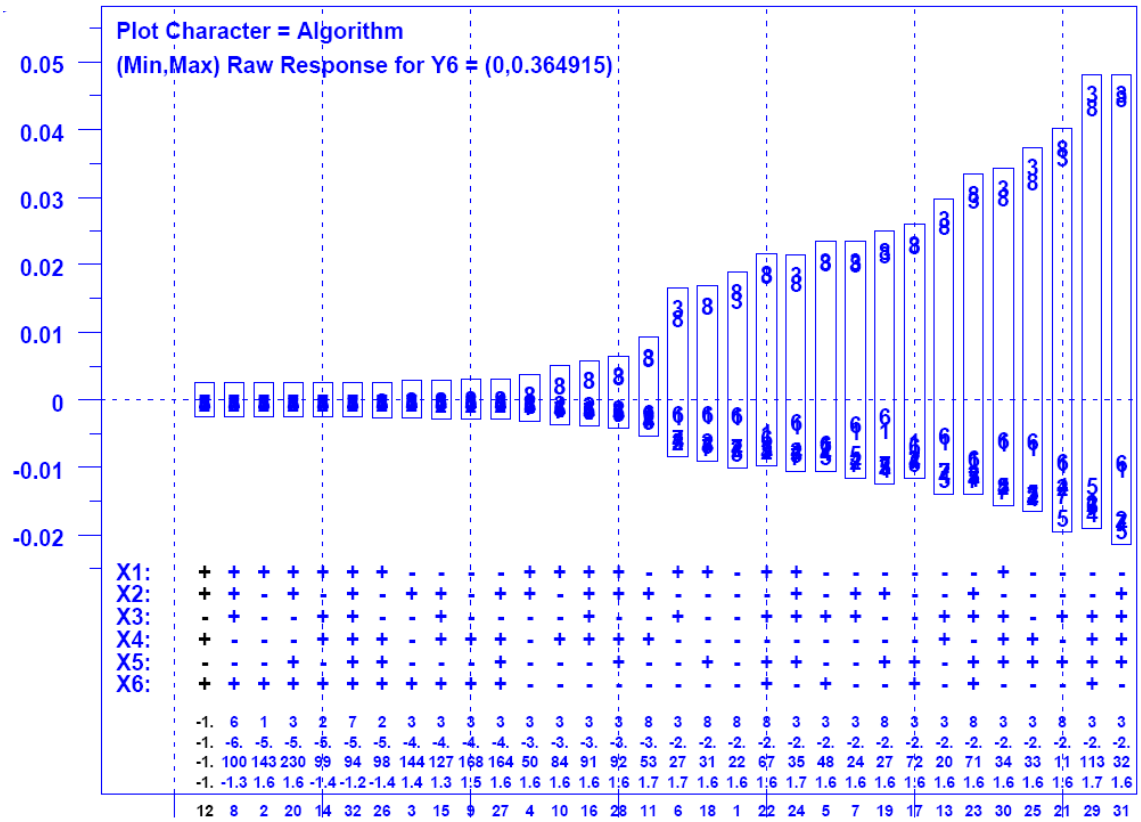


Figure 7-5. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) in Time Period One for All Algorithms – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

To enhance clarity, we adopt an approach that enables algorithm 8 to be highlighted as an outlier wherever both algorithms exhibit similar extreme responses. We omit responses for algorithm 3 from the detailed analyses. Omitting responses for algorithm 3 reduces the values of the mean residuals and thus enables responses for algorithm 8 to exceed our chosen 2.08 cutoff under the Grubbs' test for outliers. For example, Fig. 7-6 shows the result of using our tactic to reconsider retransmission rate under all conditions for TP1. Note that numerous responses for algorithm 8 are now highlighted in green, which signifies a statistically significant increase in retransmission rate over the other algorithms (excluding algorithm 3).

To justify this, we point out that the cluster analyses grouped together algorithm 3 and algorithm 8 as a pair for each of the 32 conditions. We chose to omit algorithm 3 rather than algorithm 8 because results for algorithm 3 were analyzed and discussed previously in Sec. 6.4. By focusing in this section on algorithm 8, we will be better positioned to identify significant similarities in the behavior of the two algorithms. Should we wish to seek significant differences, we could conduct the detailed analyses excluding responses for algorithm 8 instead of algorithm 3 and compare the two analyses.

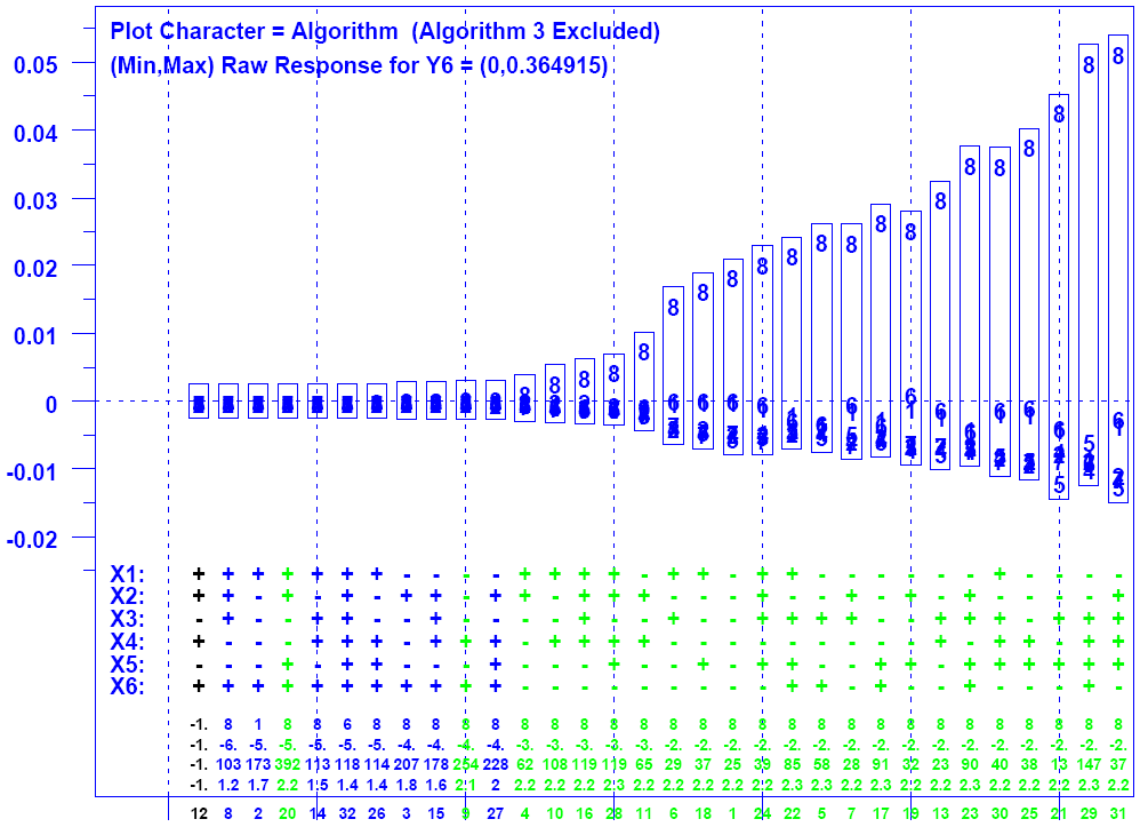


Figure 7-6. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) in Time Period One when Excluding Responses for Algorithm 3 – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4 Results

In this section, we report the key results from our analysis of summarized response data (described in Sec. 6.2.2). We provide brief commentaries to explain the results presented. We give results in four segments: one for each of the three time periods and one for response data aggregated over the entire 25-minute scenario. We follow a similar plan for each segment: (1) present results from cluster analysis, (2) present results from condition-response summaries, (3) present detailed analysis of significant responses and (4) give a summary of the results for the segment. We defer drawing inferences from the results until Sec. 7.5, where we give our findings.

7.4.1 Time Period One (TP1)

TP1 comprises a five-minute period (200 ms measurement intervals 3000-4500) where three long-lived flows commence within an overall background of normal Web traffic, which includes downloading Web pages and documents. In this experiment, each long-lived flow exits initial slow-start once the congestion window passes 100 packets, so maximum transfer rate can only be achieved after entering congestion avoidance. For this reason, TP1 should reveal any differences (among the eight congestion control algorithms) in the time needed to achieve maximum transfer rate. In the previous experiment (Chapter 6), no such differences arose because the initial slow-start threshold

was set arbitrarily high (at $2^{32}/2$ packets) and all long-lived flows achieved maximum transfer rate within initial slow-start.

7.4.1.1 Cluster Analysis for TP1. Fig. 7-7 presents a cluster analysis comparing all eight algorithms (recall Table 7-1) under each of the 32 simulated conditions. We annotate the individual dendrograms with the algorithm identifier of a congestion control algorithm that stands out. Where more than one algorithm (usually 3 and 8) stands out we include all relevant algorithms separated by slashes, e.g., 3/8.

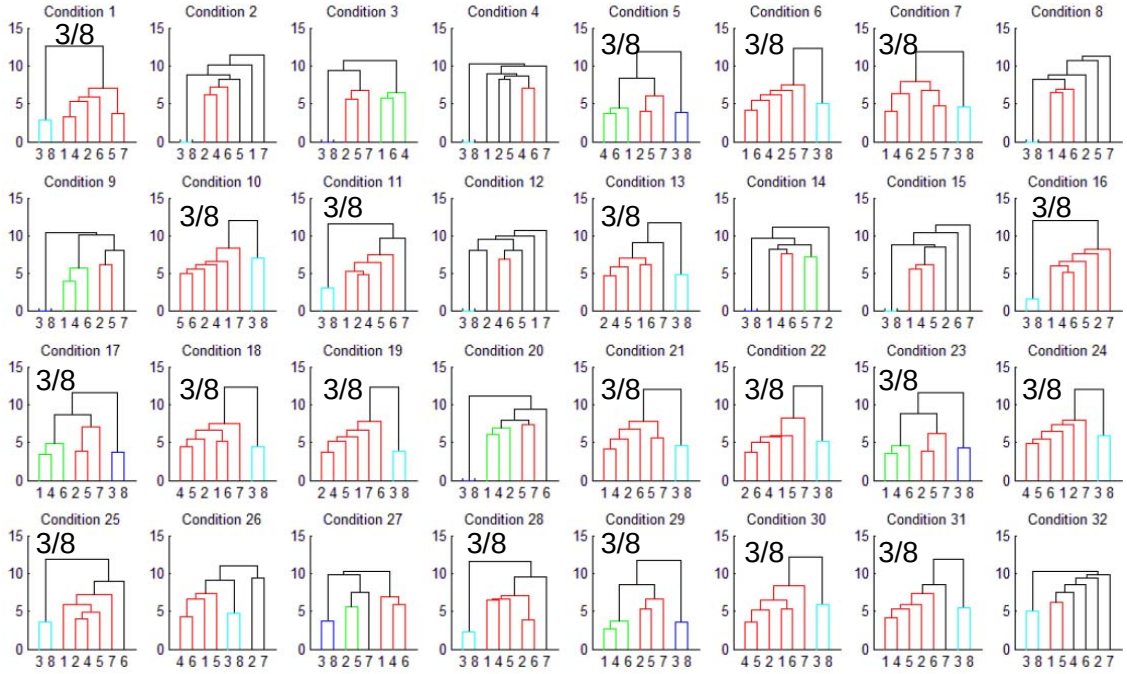


Figure 7-7. Clustering for Time Period One – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in the response space between pairs of algorithms and clusters of algorithms as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

7.4.1.2 Condition-Response Summary for TP1. Fig. 7-8 gives the condition-response summary for TP1 – but with the data for algorithm 3 excluded in order to highlight specific responses for which algorithm 8 may be distinguished. Fig. 7-9 gives the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 10 %.

7.4.1.3 Analysis of Significant Responses for TP1. Based on Figs. 7-8 and 7-9, we selected responses for more detailed analysis. In Figs. 7-10 to 7-15, we report analyses for congestion window increase rate (y2), flow completion rate (y5), retransmission rate (y6), goodput (y29) and flows completed (y31) on typical (NN) paths and average number of connecting flows (y42). We selected y5 and y31 based on Fig. 7-8 even though they did not pass the 10 % filter applied to generate Fig. 7-9. When accumulated over time, the absolute magnitude of each effect within a measurement interval appears large enough to affect overall system performance.

In Figs. 7-16 to 7-18, we provide additional, detailed analyses selected using the condition-response summaries shown in Figs. 7-8 and 7-9. Fig. 7-16 reveals that algorithm 7 (TCP Reno) lags significantly in the average rate at which packets exit the network. Fig. 7-17 shows a similar lag by algorithm 7 in average goodput provided on the moderate-distance long-lived flow (L2). We omit similar analyses for the short- and long-distance long-lived flows. Finally, Fig. 7-18 explores average buffer utilization in directly connected access router K0a. We selected this analysis to show a tendency for differences in buffer utilization among the various congestion control algorithms.

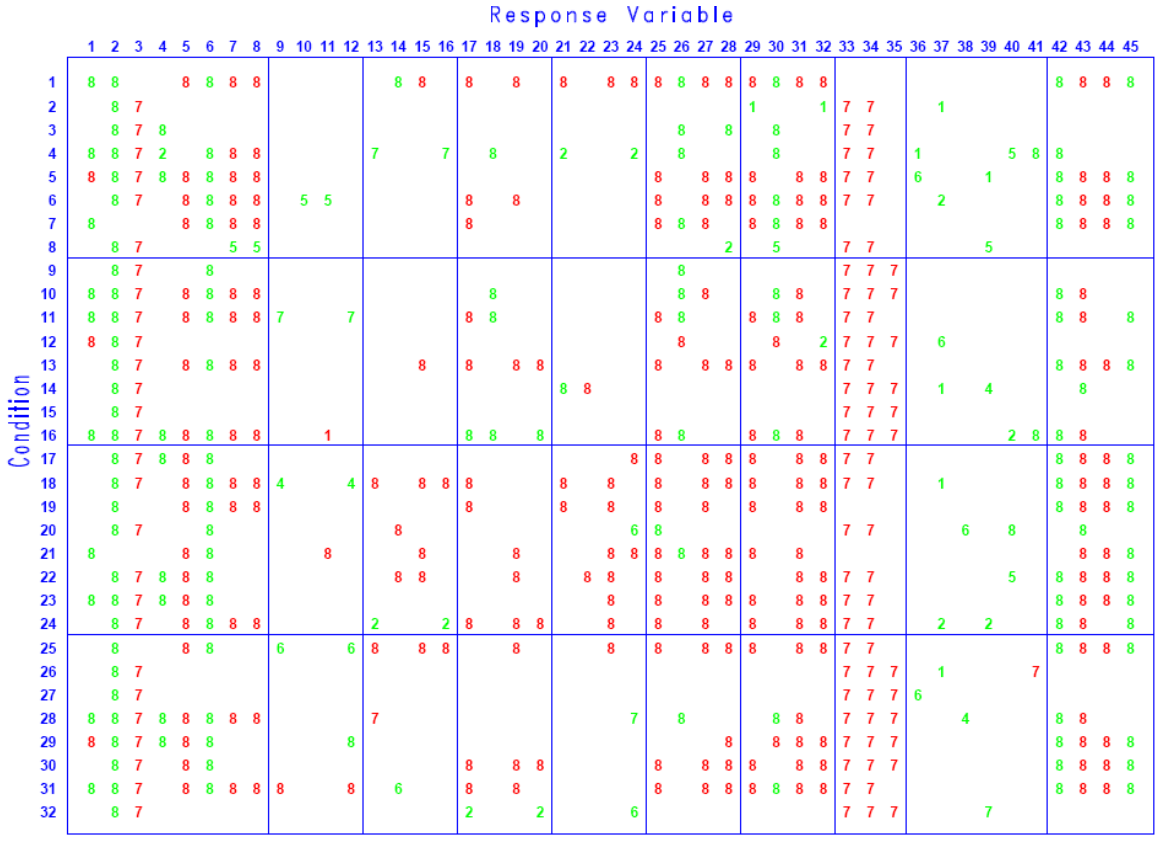


Figure 7-8. Condition-Response Summary for Time Period One – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

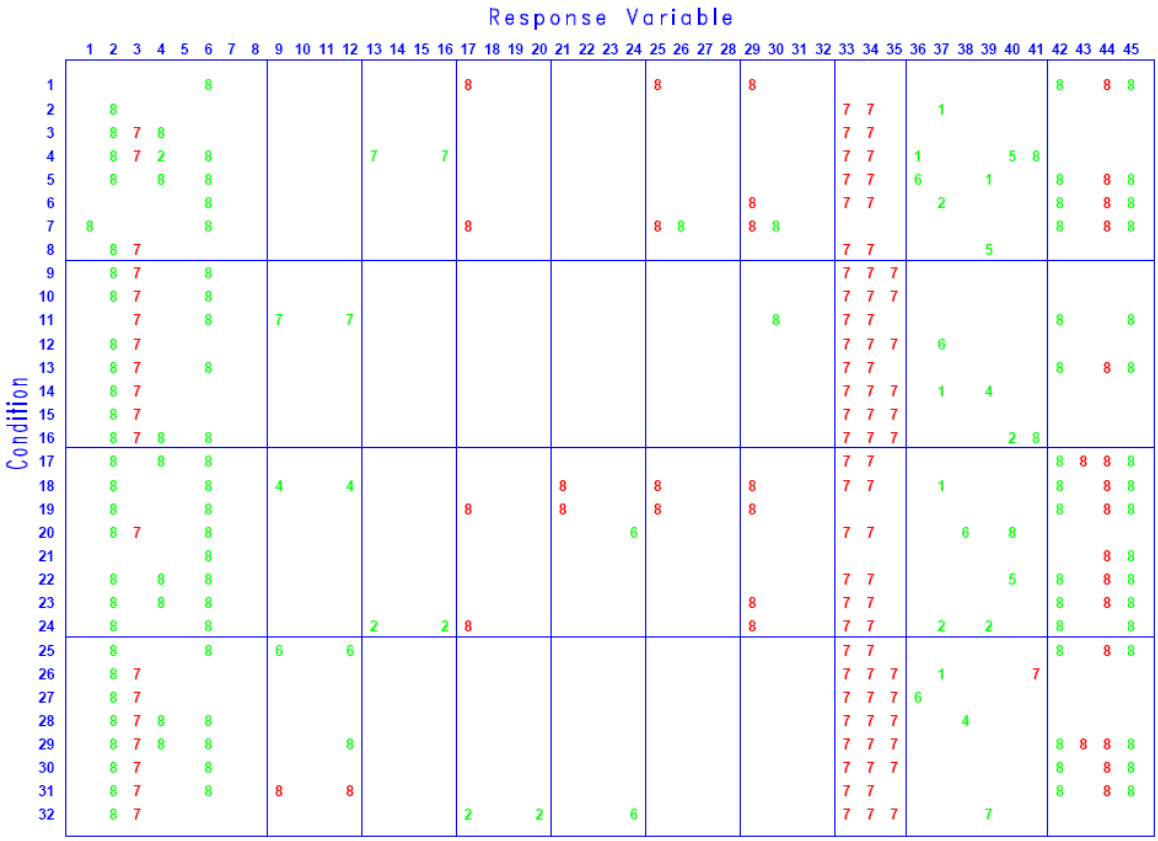


Figure 7-9. Condition-Response Summary for Time Period One – 10% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

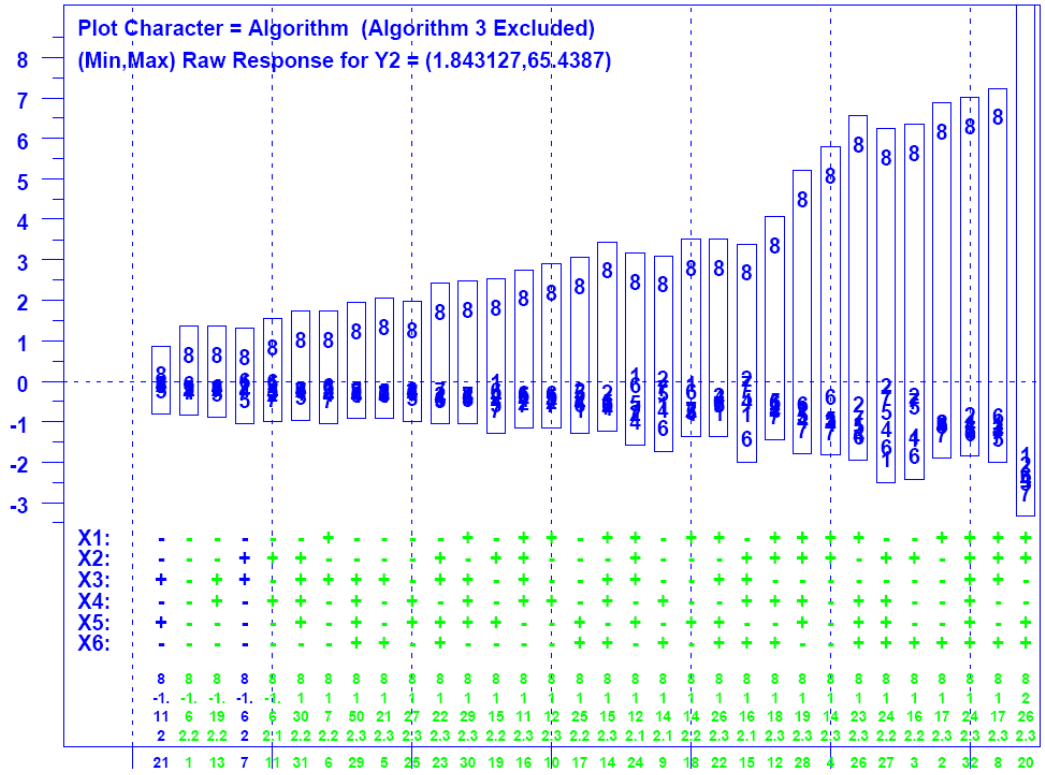


Figure 7-10. Detailed Analysis for Congestion Window Increase Rate Per Flow (increases per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

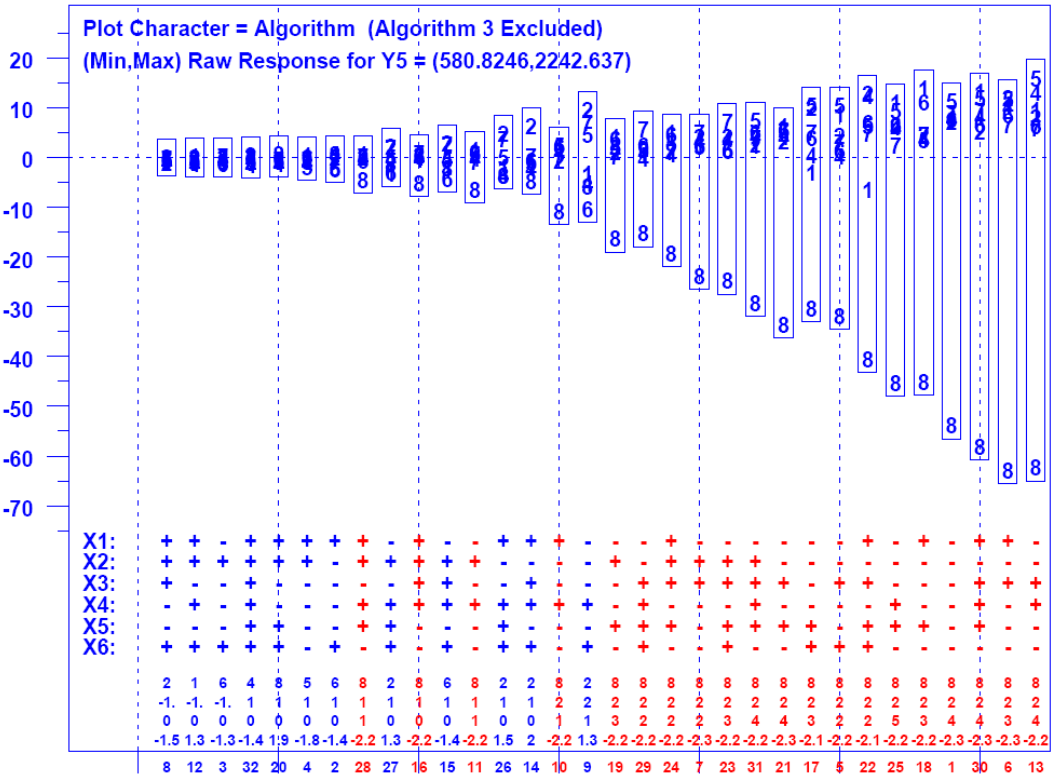


Figure 7-11. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

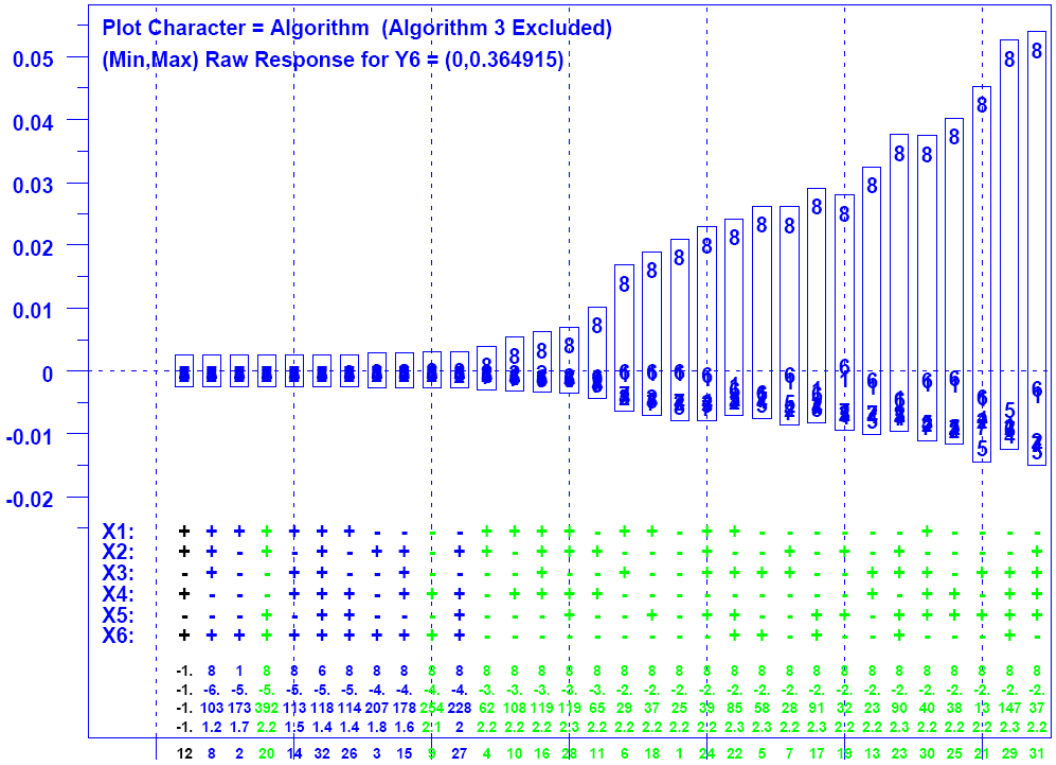


Figure 7-12. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

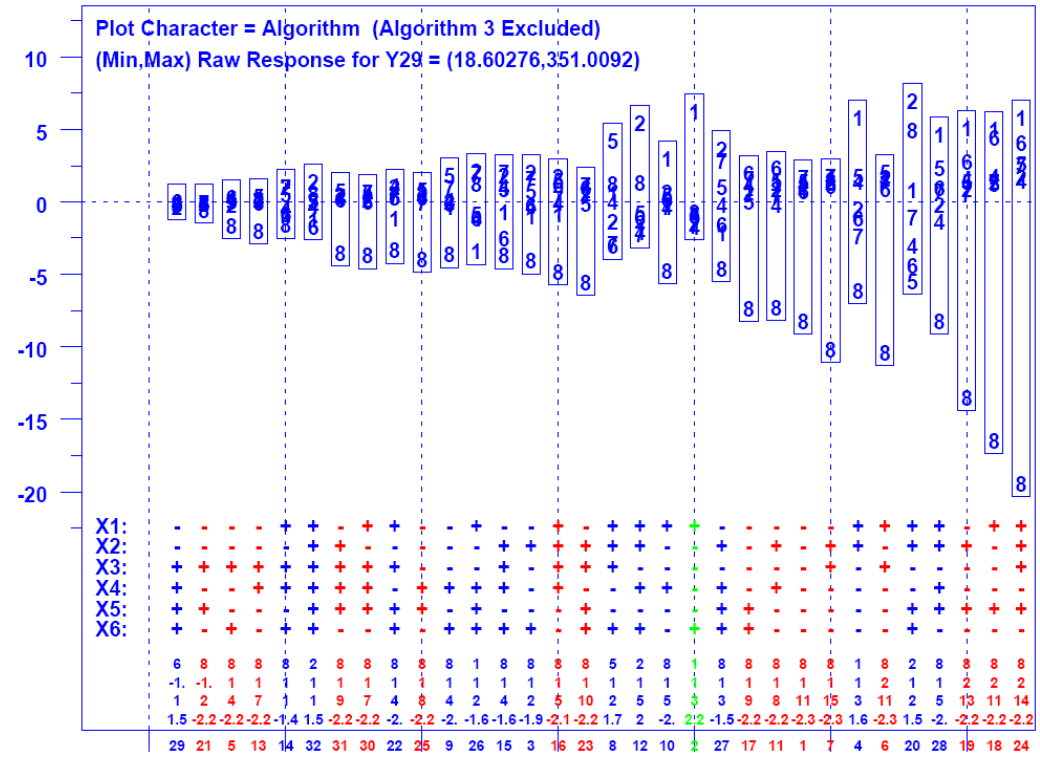


Figure 7-13. Detailed Analysis for Average Goodput (packets per second) on NN Flows in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

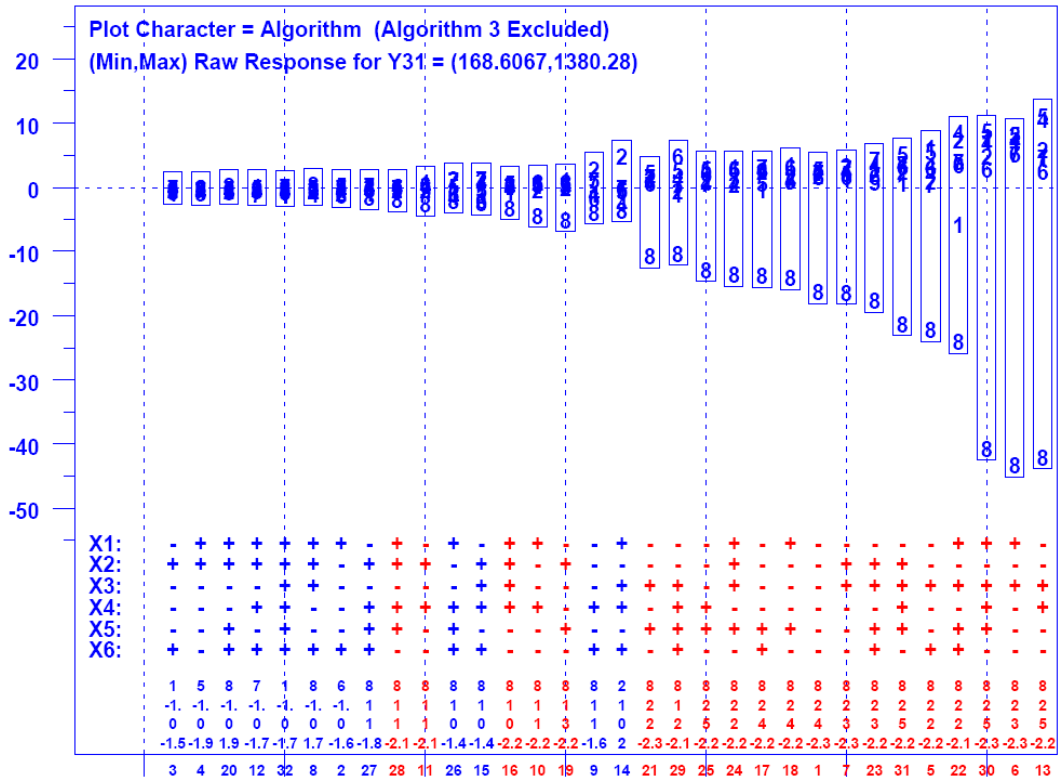


Figure 7-14. Detailed Analysis for NN Flow Completion Rate (flows per 200 ms) in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

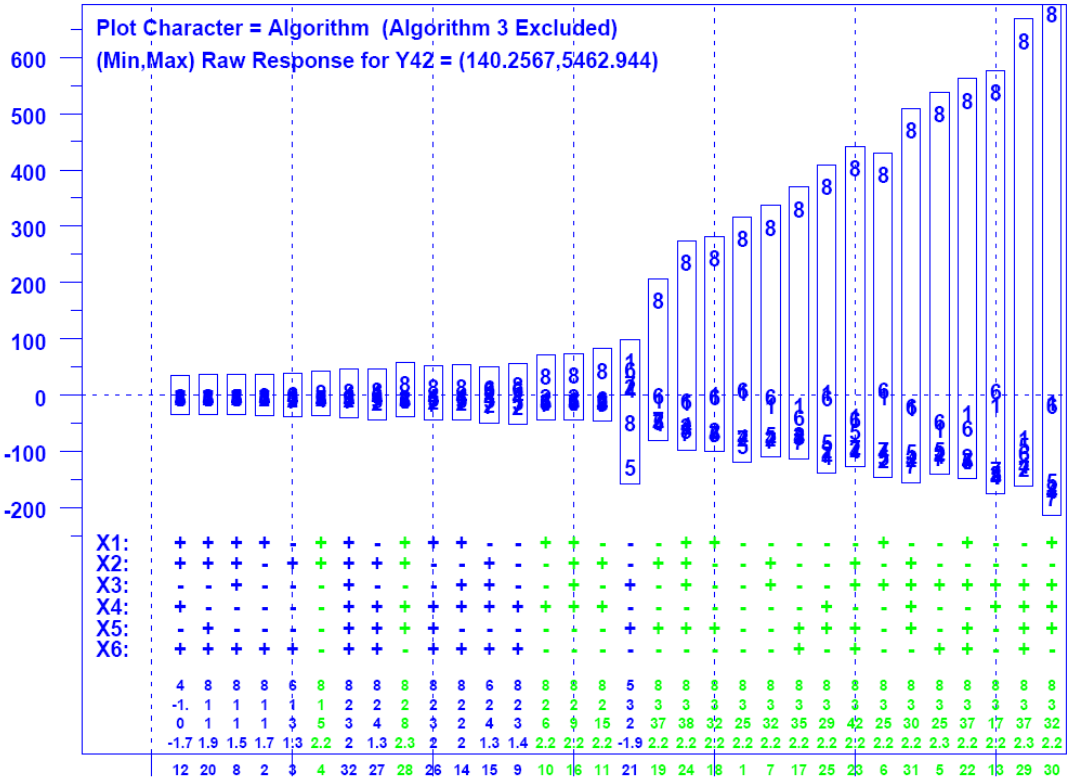


Figure 7-15. Detailed Analysis for Number of Connecting Flows in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

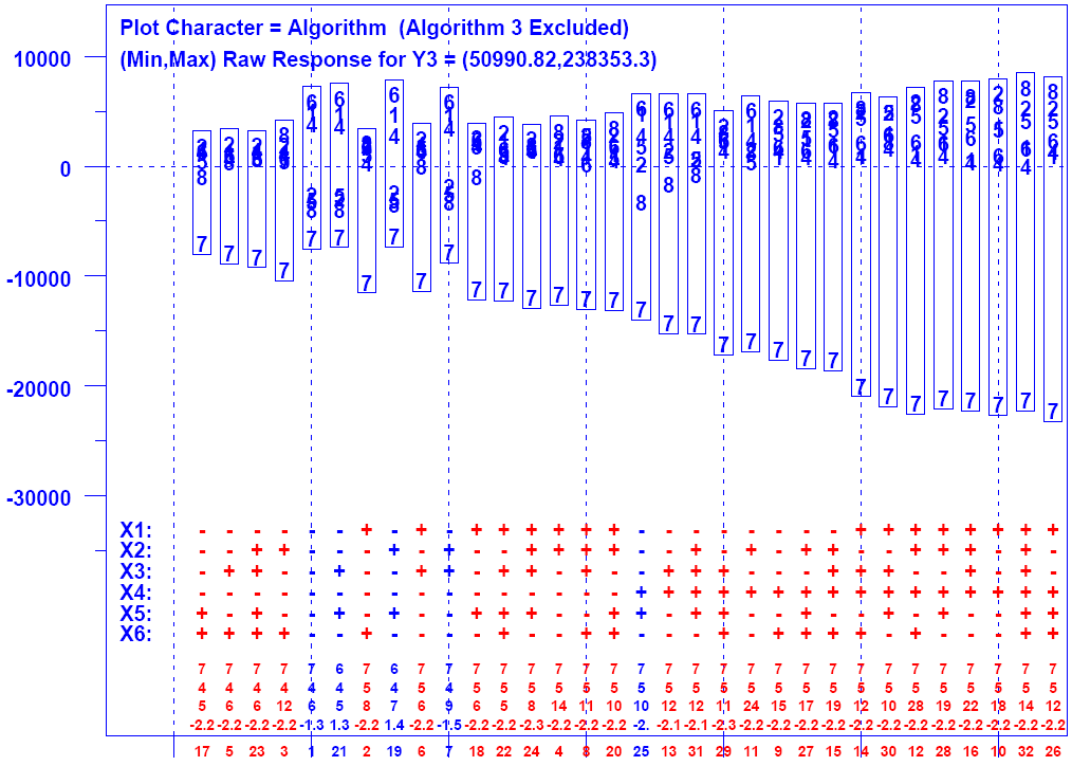


Figure 7-16. Detailed Analysis for Average Packets Output Per 200 ms Interval in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

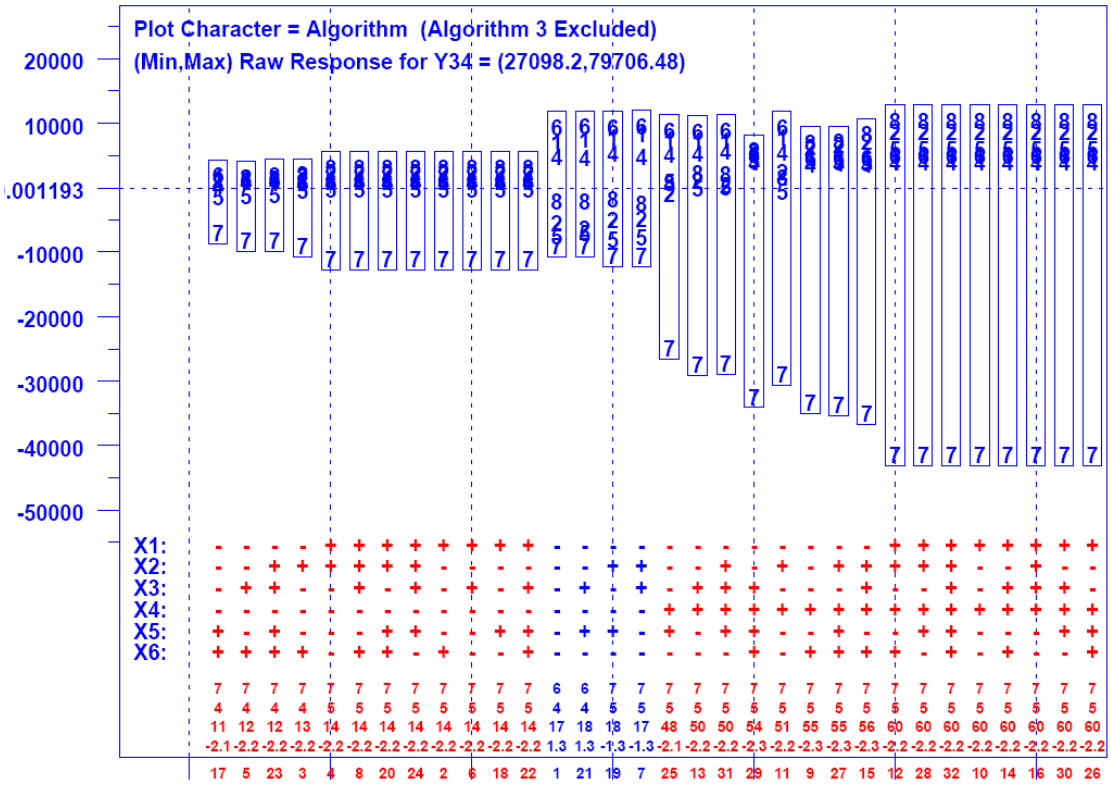


Figure 7-17. Detailed Analysis for Average Goodput (packets per second) on Long-lived Flow L2 in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

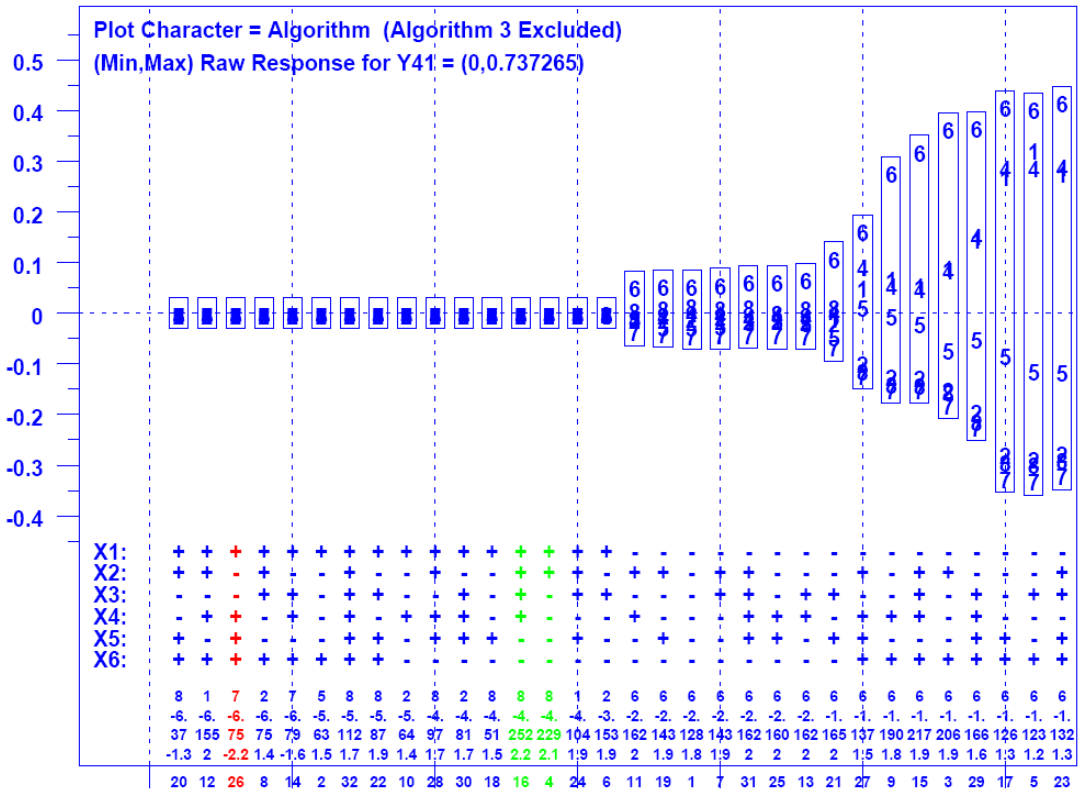


Figure 7-18. Detailed Analysis for Average Buffer Utilization (percentage of buffers occupied) at Router K0a in Time Period One – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.1.4 Summary of Results for TP1. Given normal Web traffic, FAST with α-tuning enabled (FAST-AT, i.e., algorithm 8) – exhibits distinctive behavior, which appears to grow more distinctive with increasing congestion. When faced with congestion, FAST-AT exacerbates the situation (as shown by the higher rate of increase in congestion window), which leads to more packet losses and then to a higher rate of retransmissions. Increased losses under FAST-AT also appear to increase the difficulty for establishing flows because more SYN and SYN+ACK packets are lost. As a result, on average more flows are pending in the connecting state. Increased retransmissions also cause flows to send more packets (y4) in order to ensure all data is successfully received. This means that flows take longer to finish, as shown by the lower completion rate for flows in general and for NN flows in particular. This mirrors the findings for FAST with α-tuning disabled, as reported previously (Sec. 6.4.1).

Two additional results for TP1 should be mentioned. First, as expected, when the initial slow-start threshold is set small, standard TCP Reno (algorithm 7) provides substantially lower (compared with the other algorithms) average goodput on long-lived flows. This result occurs during the period when each long-lived flow attempts to achieve maximum transmission rate using congestion avoidance increase procedures. Since most packets exiting the network during TP1 are associated with long-lived flows, the lower goodput provided by TCP Reno also leads to a lower aggregate rate of packet output. Finally, our results show that buffer utilization in routers supporting long-lived flows appears highest for Scalable TCP (algorithm 6), followed by BIC (algorithm 1) and HSTCP (algorithm 4). HTCP (algorithm 5) tends to follow next, while the other

algorithms (CTCP, FAST-AT and TCP Reno) appear to utilize the smallest proportion of buffers.

7.4.2 Time Period Two (TP2)

During the five minutes (200 ms intervals 4500 to 6000) of TP2, DD flows (explained in Sec. 6.1) become jumbo file transfers, which lead to increased congestion within directly connected access routers, and also to increased packet load on the network backbone. Jumbo file transfers compete with the three long-lived flows transiting the same directly connected access routers. The remaining flow classes continue to generate normal Web traffic and occasional document downloads during TP2.

7.4.2.1 Cluster Analysis for TP2. Fig. 7-19 shows an annotated set of 32 dendrograms for TP2. Since the level of congestion increases throughout the network during this period and algorithms 3 and 8 appear sensitive to congestion, one might expect the behavior of those algorithms to become more distinctive when compared with TP1. Note that algorithms 3 and 8 now standout under 28 of the conditions – compared with only 20 conditions in TP1. Fig. 7-19 also shows algorithm 6 (Scalable TCP) standing out in a couple of conditions.

7.4.2.2 Condition-Response Summary for TP2. Fig. 7-20 gives the condition-response summary for TP2. Fig. 7-21 gives the same summary after applying a filter showing only statistically significant outliers for which the relative effect exceeds 30 %. Remember that these figures omit data for algorithm 3. Algorithm 8 stands out in both figures and algorithms 4 (HSTCP) and 6 (Scalable TCP) tend to stand out with respect to buffer utilization (y36 through y41).

7.4.2.3 Analysis of Significant Responses for TP2. Guided by Figs. 7-20 and 7-21, we selected several responses for detailed analyses, which are reported in Figs. 7-22 through 7-30. We provide analyses for congestion window increase rate (y2), average packets output per measurement interval (y3), flow completion rate (y5), retransmission rate (y6), average goodput (y25) and completions (y27) for FN flows, average goodput for the moderate-distance (L2), long-lived flow (y34) and average number of connecting flows (y42). Taken together, these analyses highlight the key similarities and differences in behavior between TP1 and TP2. Fig. 7-30 reports the average buffer utilization for directly connected router I0a.

7.4.2.4 Summary of Results for TP2. FAST-AT exhibits the same distinctive behaviors seen during TP1. Increased congestion in TP2 enhances these effects, most of which now show up as relative differences of 30 % or more. Under congestion associated with jumbo file transfers, FAST-AT outputs the fewest packets per measurement interval under most conditions. This represents a change from TP1, when TCP Reno output the fewest packets. During TP1, the packet output rate is dominated by long-lived flows, which are attempting to achieve maximum transfer rate. Here, TCP Reno lags. During TP2, packet output rate is dominated by jumbo flows and long-lived flows, which share bandwidth at directly connected routers. Here, FAST-AT lags. Congestion also causes FAST-AT to provide lower goodput on other flows, such as FN and NN flows. FAST-AT also

completes fewer FN and NN flows per interval. Note that Scalable TCP provides highest goodput on long-lived flows and continues its tendency to use the most buffers in directly connected routers.

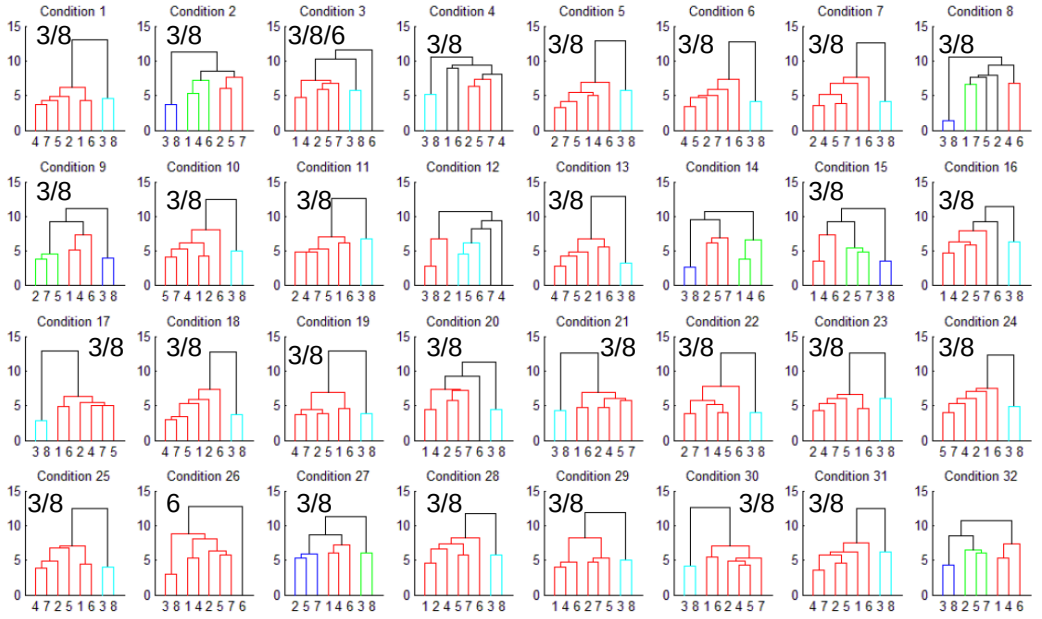


Figure 7-19. Cluster Analysis for Time Period Two – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

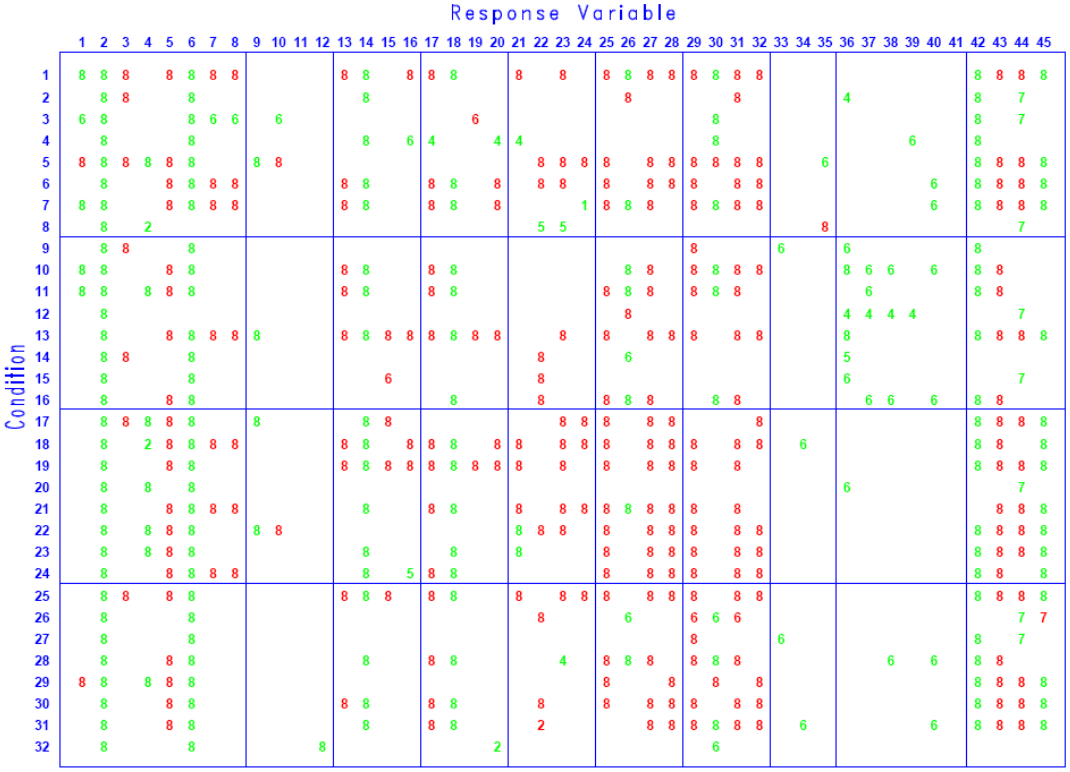


Figure 7-20. Condition-Response Summary for Time Period Two – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

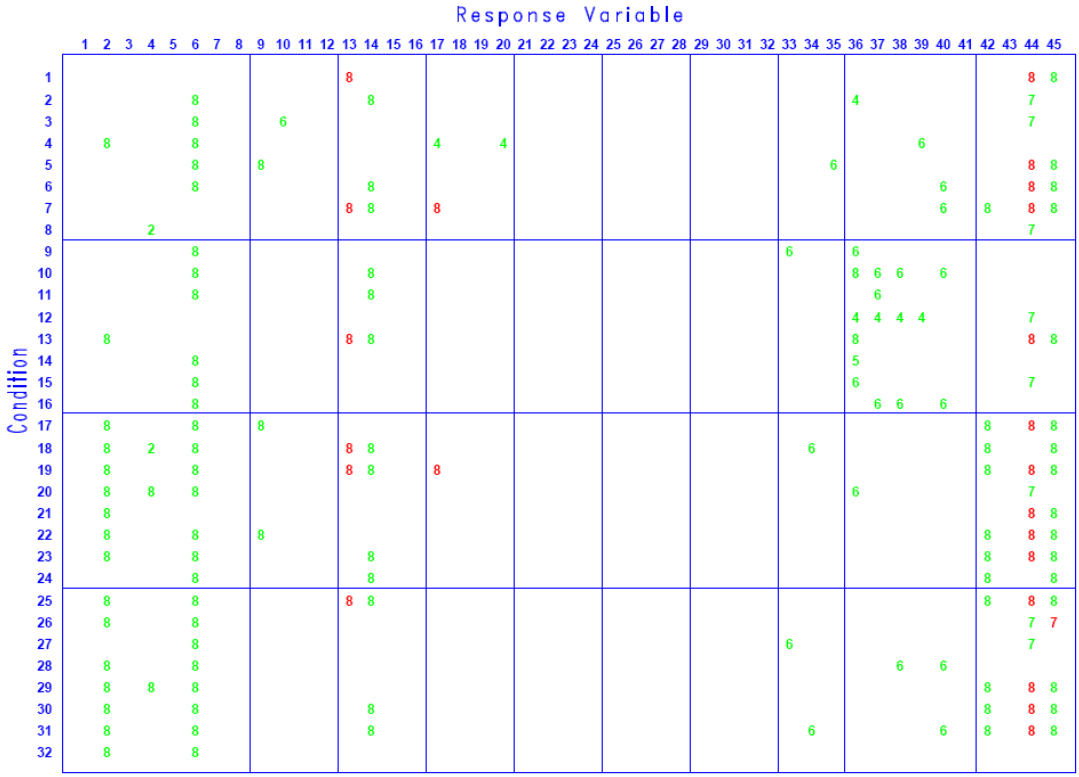


Figure 7-21. Condition-Response Summary for Time Period Two – 30% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

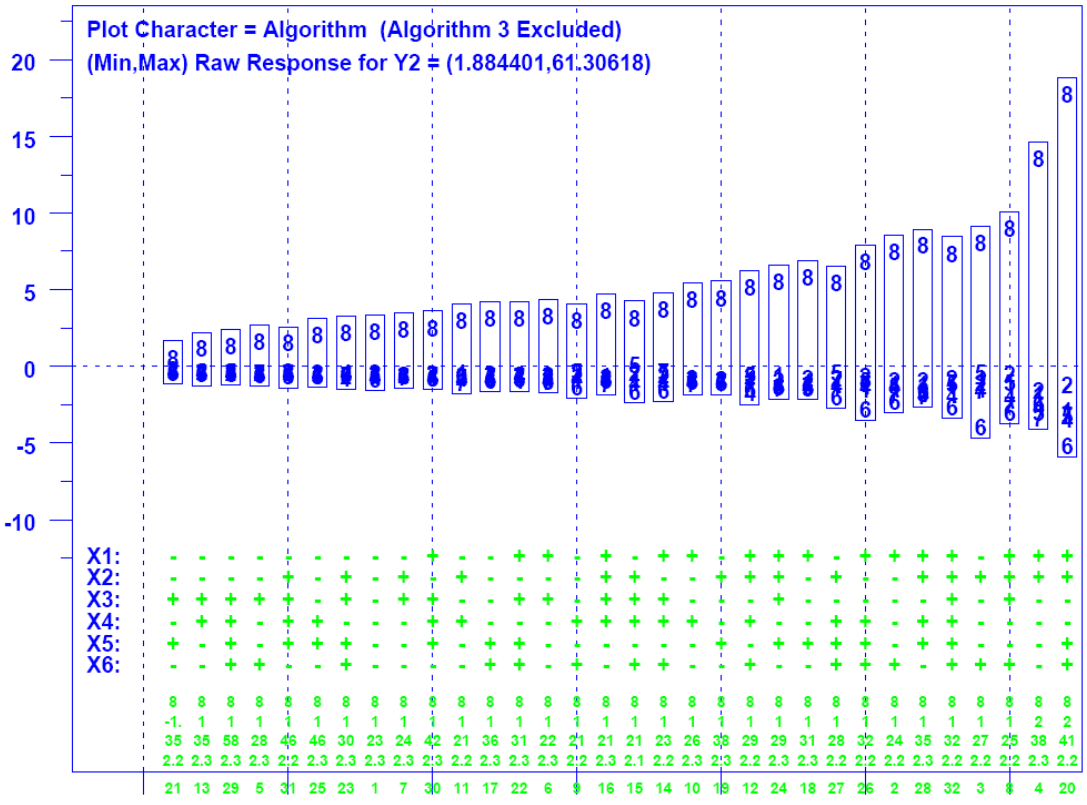


Figure 7-22. Detailed Analysis for Congestion Window Increase Rate (increases per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

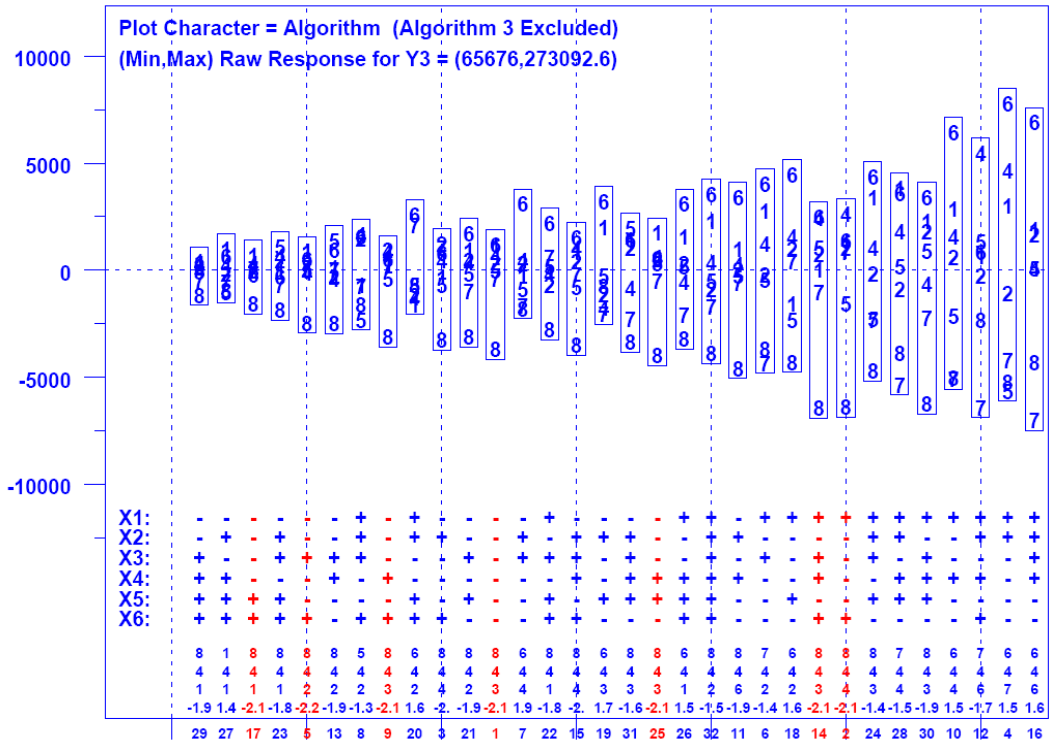


Figure 7-23. Detailed Analysis for Packet Output Rate (packets per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

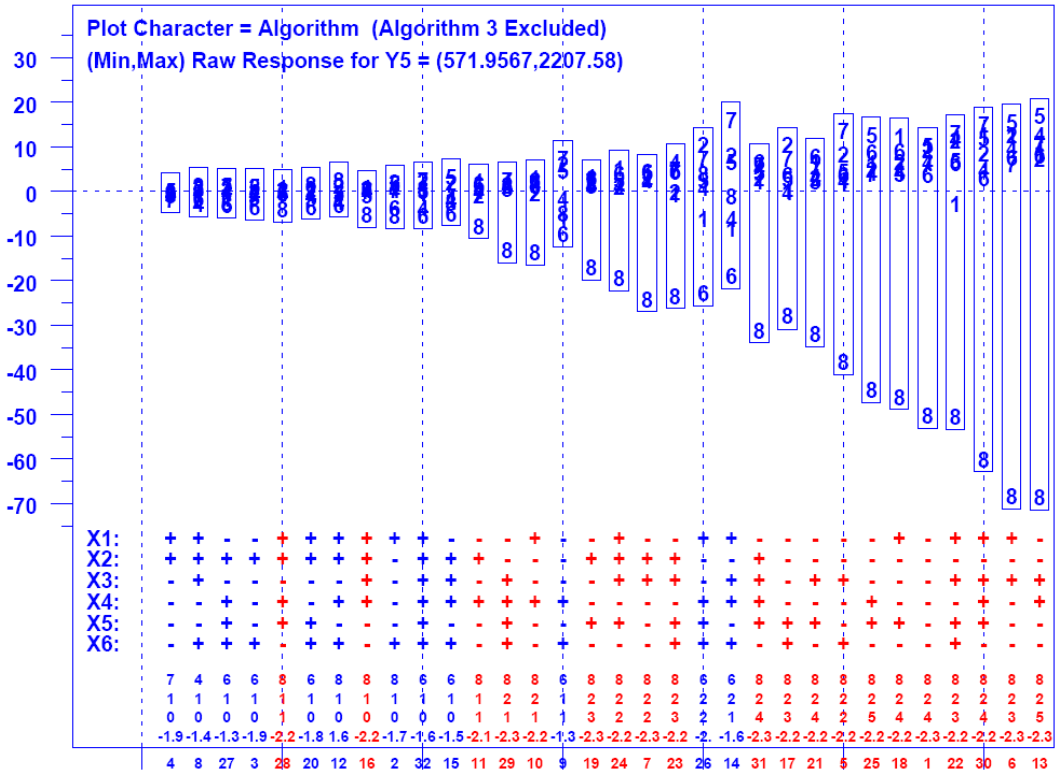


Figure 7-24. Detailed Analysis for Flow Completion Rate (flows per 200 ms) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

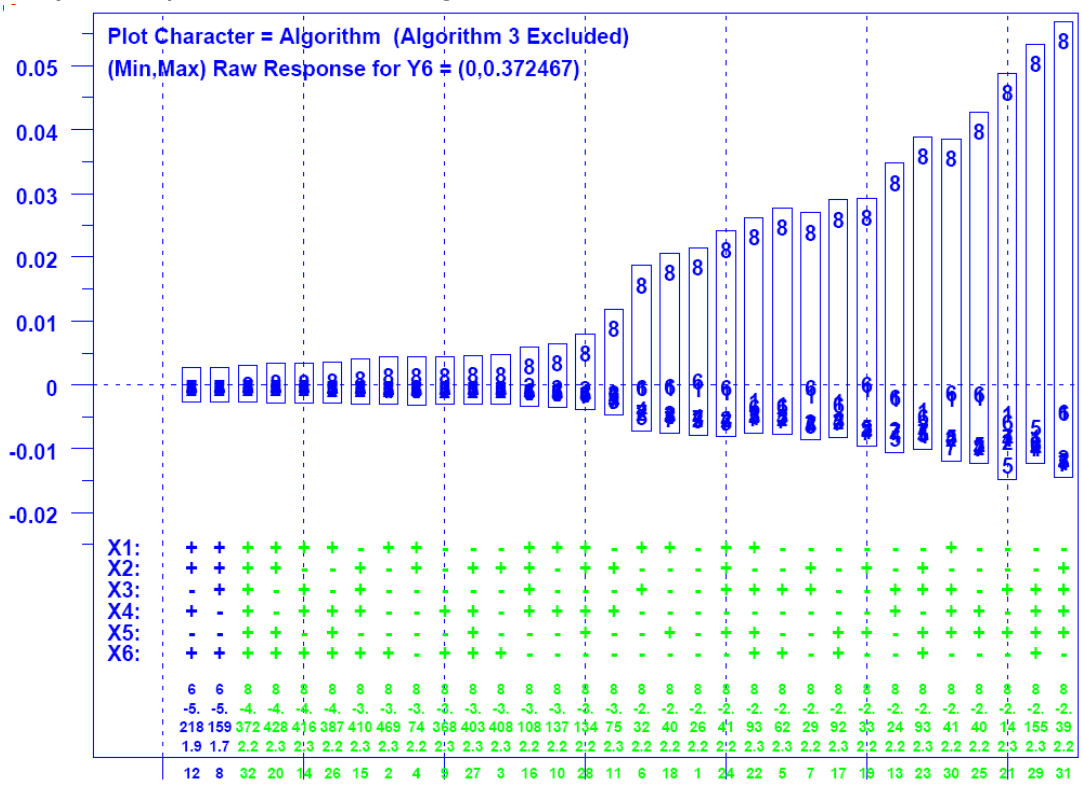


Figure 7-25. Detailed Analysis for Retransmission Rate (proportion of packets retransmitted) in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

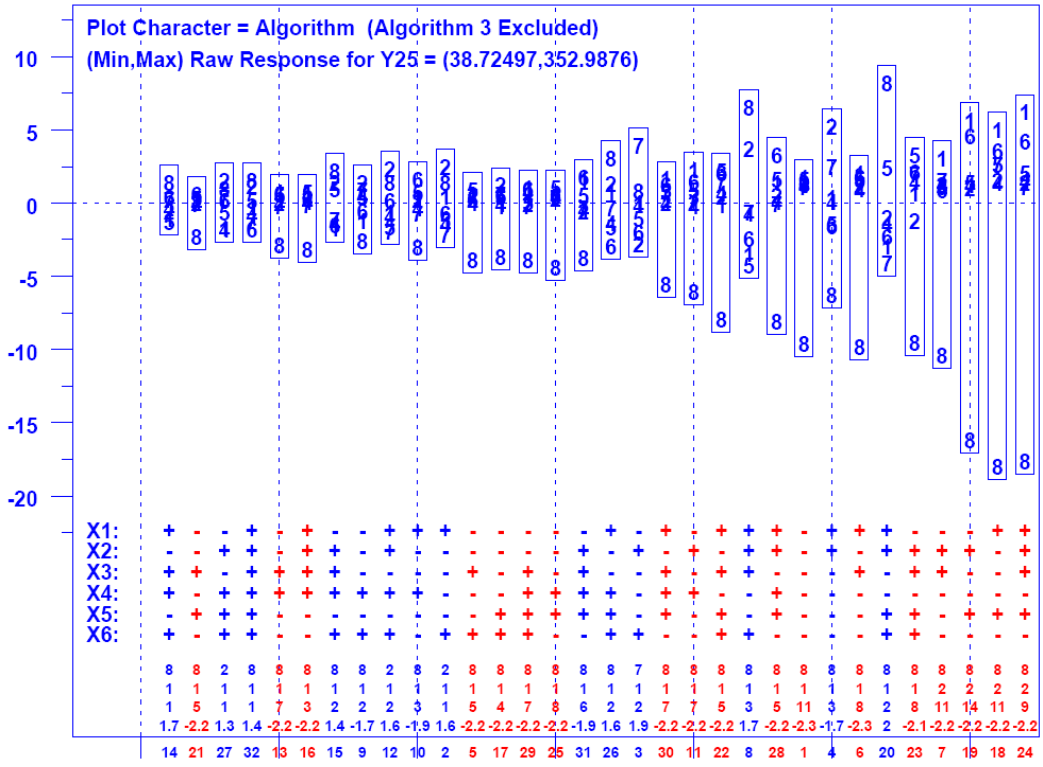


Figure 7-26. Detailed Analysis for Average Goodput (packets per second) on FN Flows in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

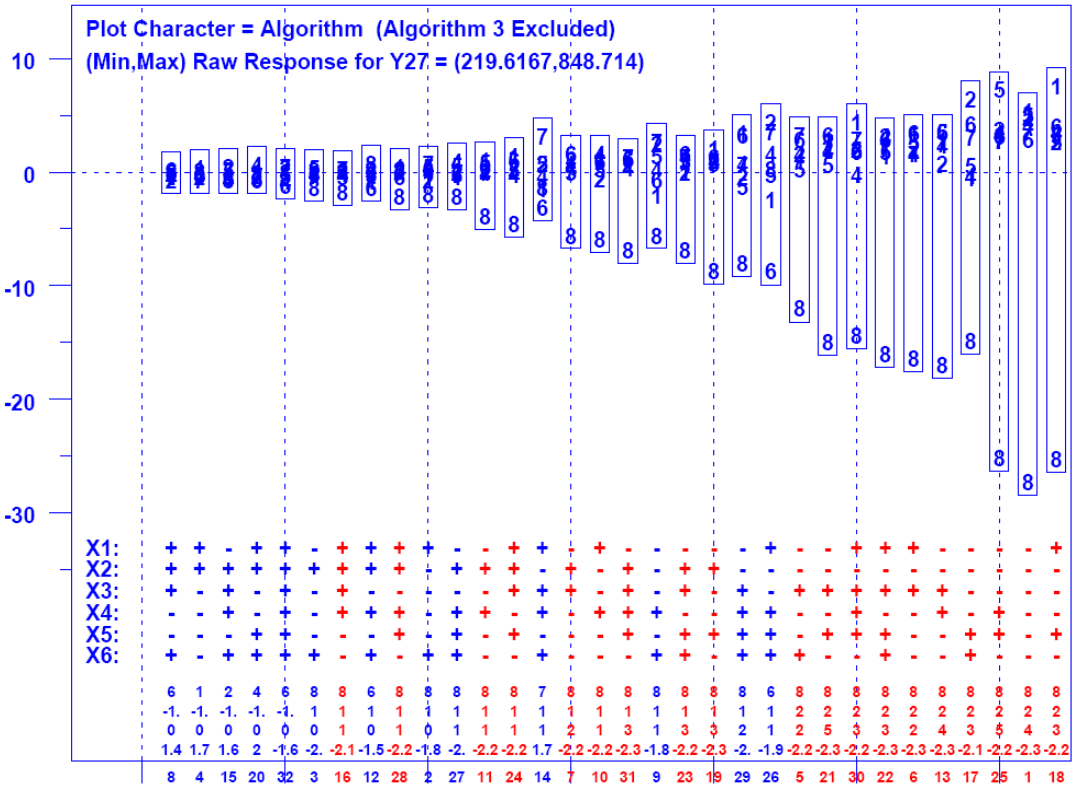


Figure 7-27. Detailed Analysis for Average FN Flow Completion Rate (flows per 200 ms) during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

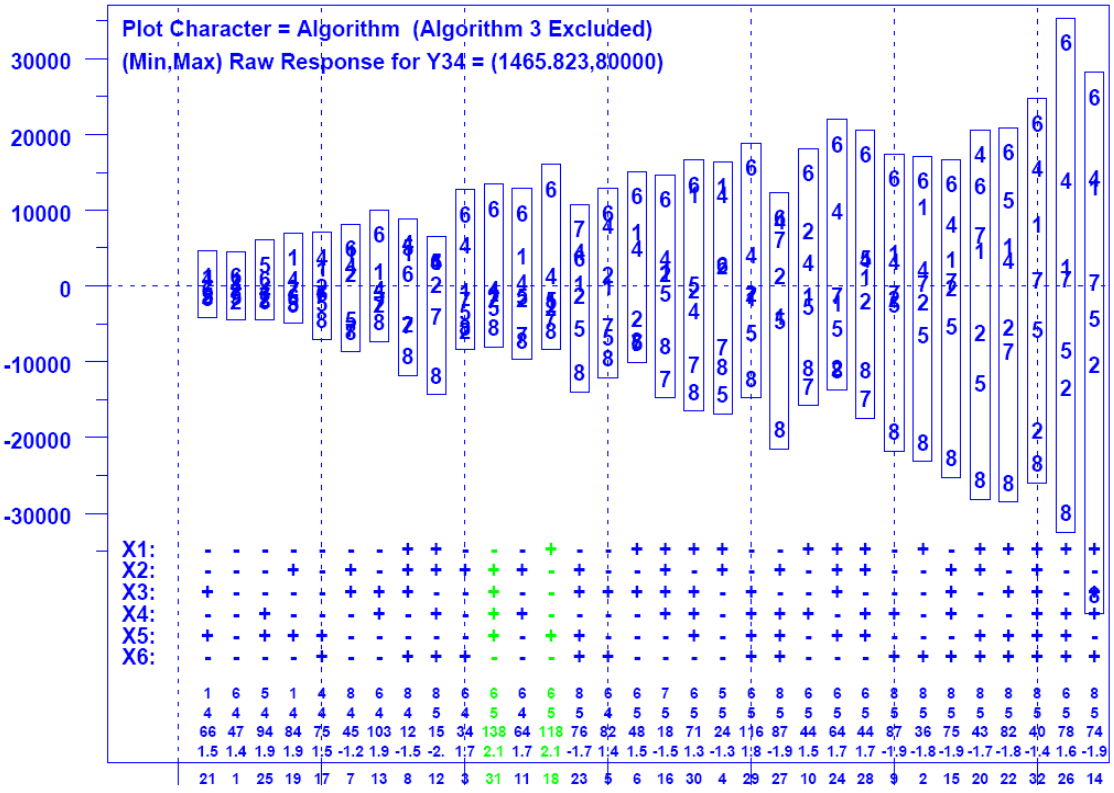


Figure 7-28. Detailed Analysis for Average Goodput (packets per second) on Long-lived Flow L2 in Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

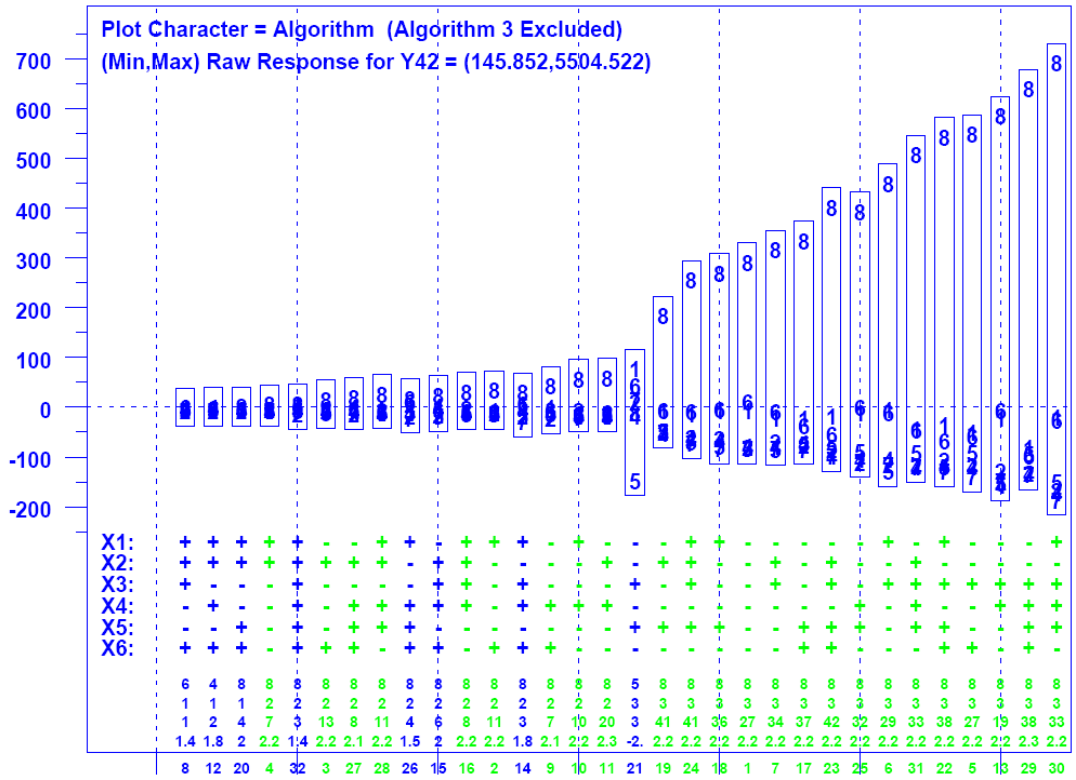


Figure 7-29. Detailed Analysis for Average Number of Connecting Flows during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

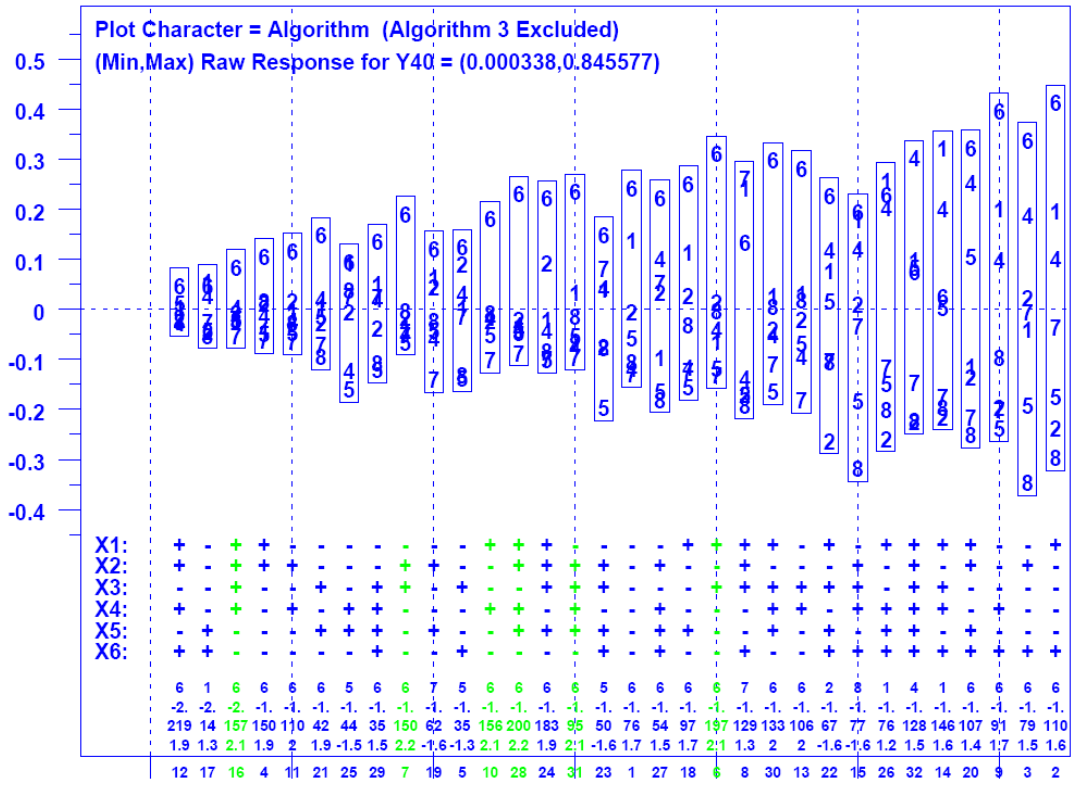


Figure 7-30. Detailed Analysis for Buffer Utilization (percentage of buffers occupied) at Router I0a during Time Period Two – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.3 Time Period Three (TP3)

During TP3 (200 ms intervals 6000 to 7500) no new jumbo file transfers are initiated on DD flows. What remains is for residual jumbo transfers to complete as the network transitions back toward normal Web traffic. The degree to which normal conditions can be restored depends upon how many jumbo transfers were created during TP2 and on how well a congestion control algorithm can recover from periods of intense congestion.

7.4.3.1 Cluster Analysis for TP3. Fig. 7-31 shows an annotated set of 32 dendrograms for TP3. Since the level of congestion stays relatively high, as residual jumbo file transfers drain from the network, algorithms 3 and 8 remain distinctive. The cluster analysis also finds algorithms 6 (Scalable TCP) and 7 (TCP Reno) to be somewhat distinctive under some conditions.

7.4.3.2 Condition-Response Summary for TP3. Fig. 7-32 gives the condition-response summary for TP3. Fig. 7-33 shows the same summary after applying a filter passing only statistically significant outliers for which the relative effect exceeds 30 %. These condition-response summaries confirm the findings from the cluster analysis: algorithms 3 and 8 stand out, along with algorithms 6 and 7 for selected responses and conditions. Figs. 7-32 and 7-33 also identify that algorithm 2 (CTCP) obtains a larger average congestion window size under selected conditions.

7.4.3.3 Analysis of Significant Responses for TP3. Guided by Figs. 7-32 and 7-33, we selected eight responses for detailed analysis, as reported in Figs. 7-34 to 7-41. Specifically, we show detailed analyses for rate of congestion window increases (y2), average packet output rate (y3), average congestion window size (y4), flow completion rate (y5), retransmission rate (y6), average goodput on the long-distance (L1), long-lived flow (y33), average buffer utilization on router C0a (y37) and average number of flows attempting to connect (y42).

7.4.3.4 Summary of Results for TP3. FAST-AT (algorithm 8) exhibits most of the same distinctive behaviors seen during TP1 and TP2. The results for TP3 also show that FAST-AT and TCP Reno (algorithm 7) both lag in recovering from the congestion of TP2. This shows up, for example, when examining detailed analyses for average packet output rate and for average goodput on the long- and moderate-distance, long-lived flows. The effect is more muted for the short-distance, long-lived flow. Scalable TCP (algorithm 6), BIC (algorithm 1) and HSTCP (algorithm 4) continue to use a higher proportion of buffers in the directly connected routers. In a previous experiment (Sec. 6.4.3), we found that CTCP exhibits a large congestion window size during easing congestion. This effect also appears in the current experiment but is somewhat attenuated.

7.4.4 Aggregated Responses (Totals)

Here we present analyses for the 28 responses collected over the entire 25-minute scenario. Recall that most of these responses are aggregated counts. Selected responses augment those counts with average values, specifically SYN rate on connected flows and goodput on completed flows. The analysis of totals examines the effects of behavioral differences when viewed over a longer period.

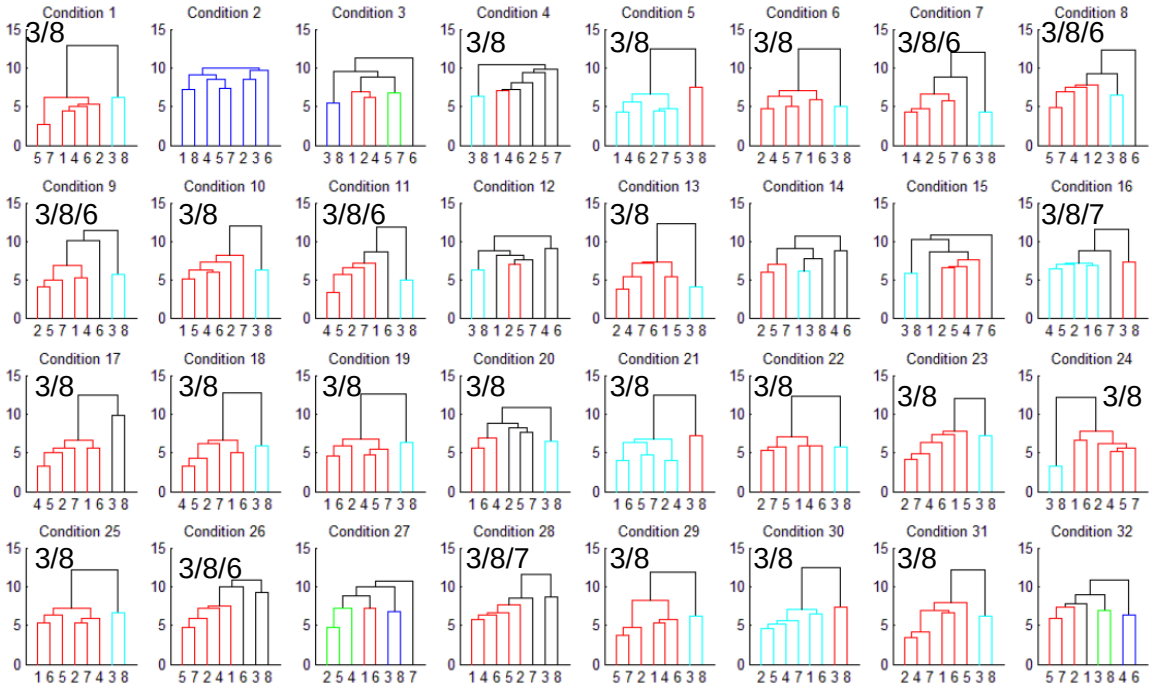


Figure 7-31. Cluster Analysis for Time Period Three – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

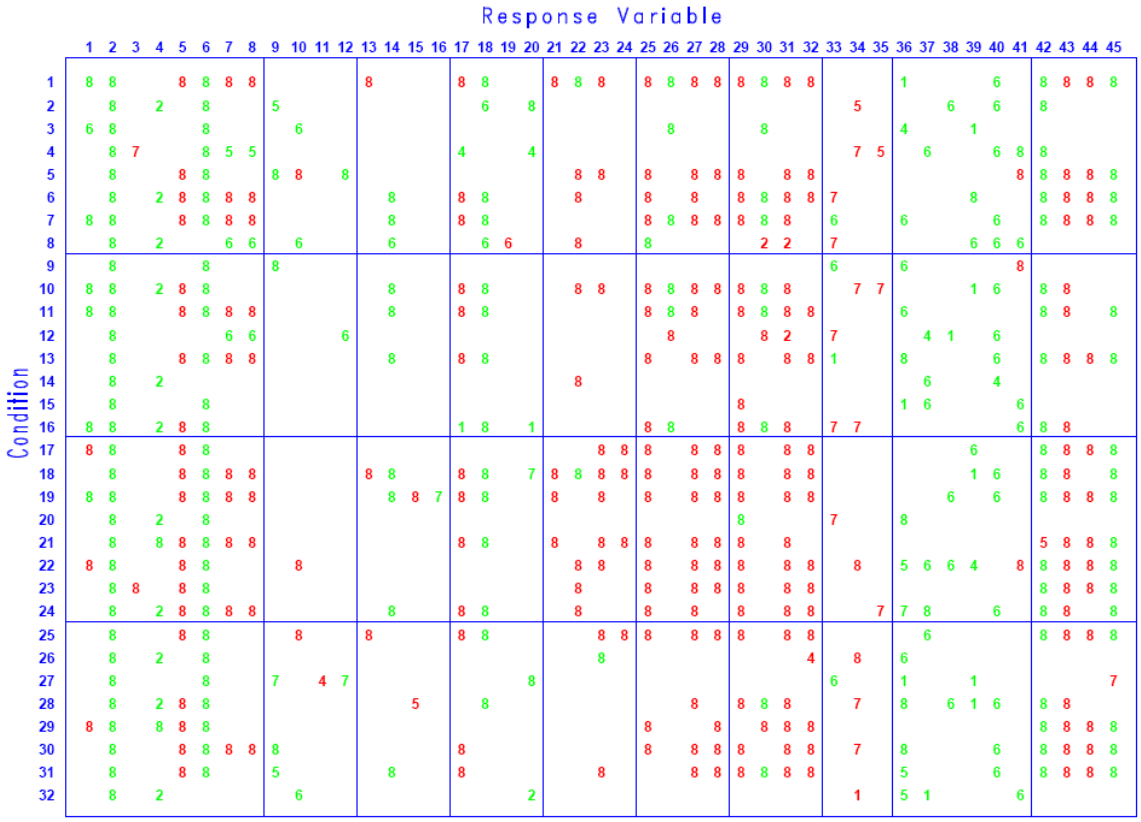


Figure 7-32. Condition-Response Summary for Time Period Three – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

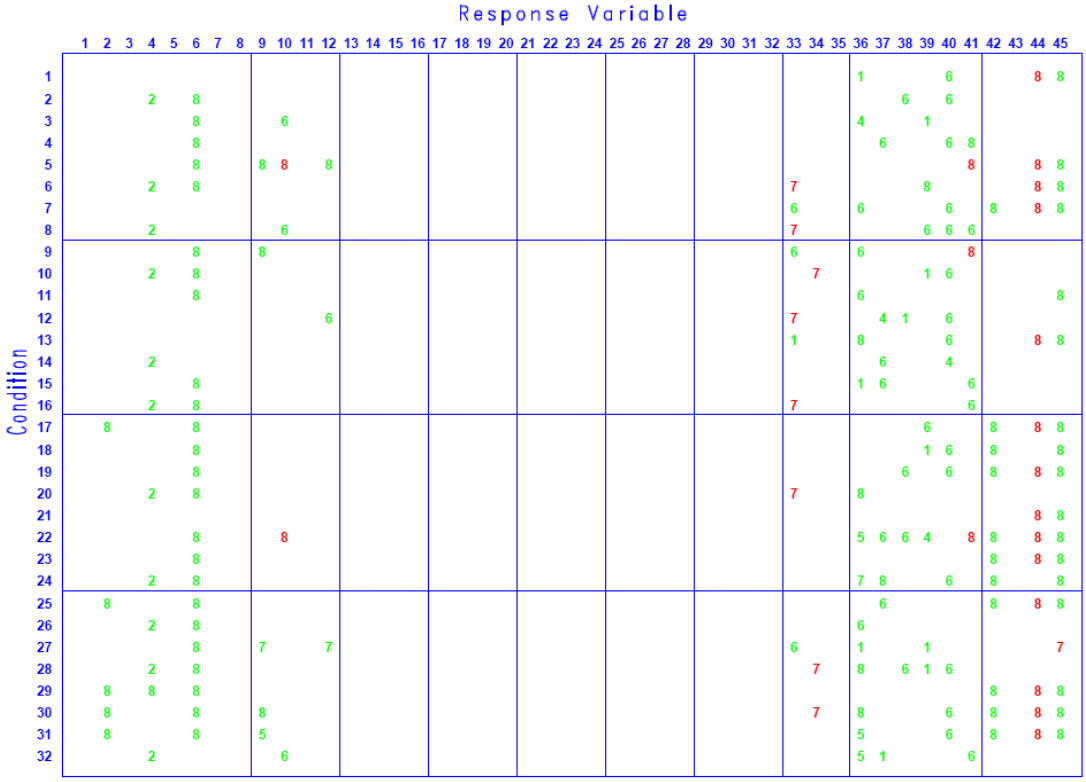


Figure 7-33. Condition-Response Summary for Time Period Three – 30% Filter Applied – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

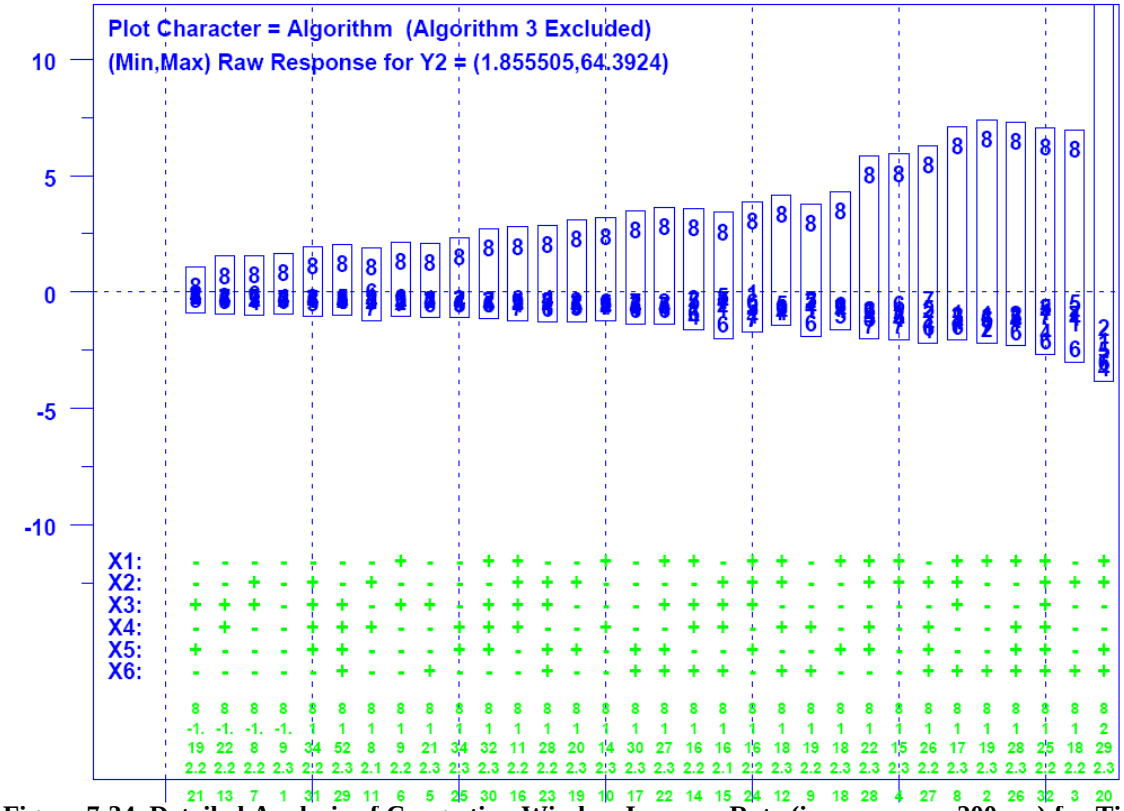


Figure 7-34. Detailed Analysis of Congestion Window Increase Rate (increases per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

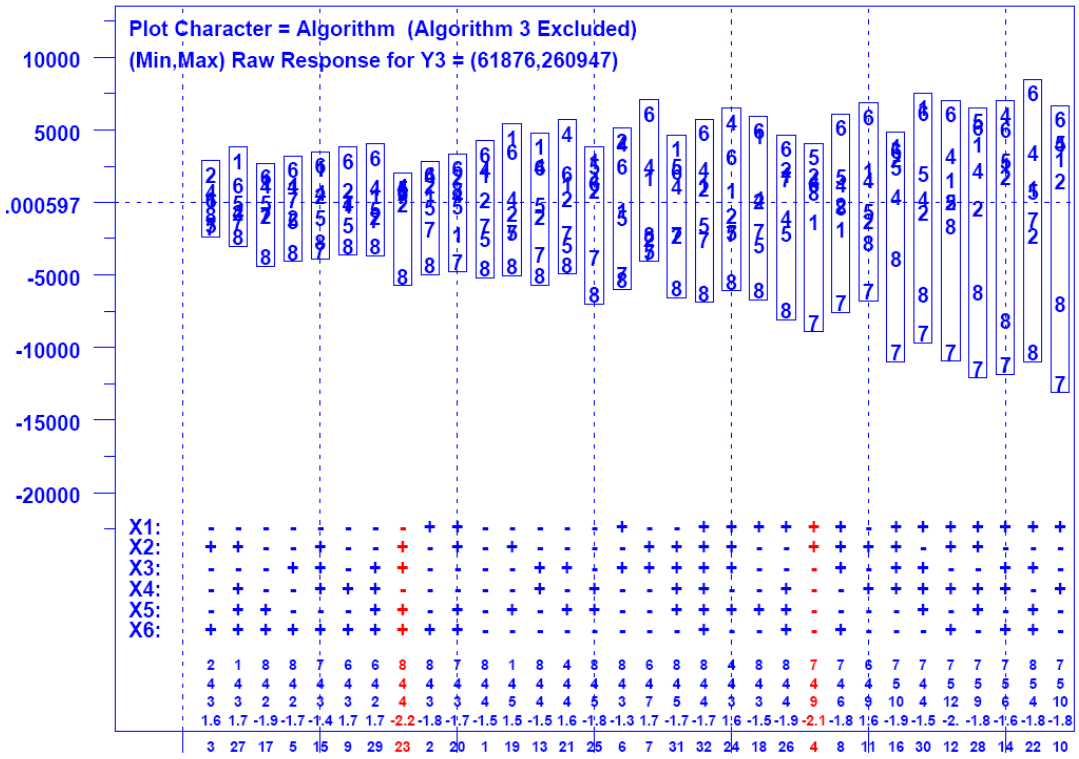


Figure 7-35. Detailed Analysis of Packet Output Rate (packets per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

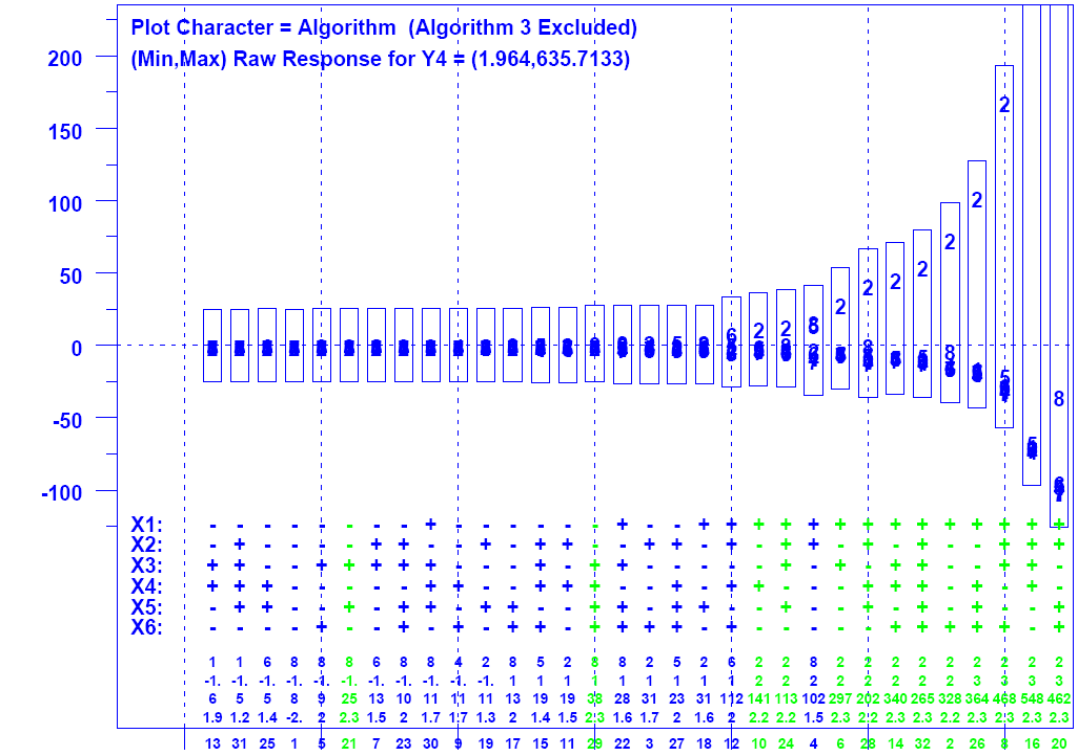


Figure 7-36. Detailed Analysis of Congestion Window Size (packets) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

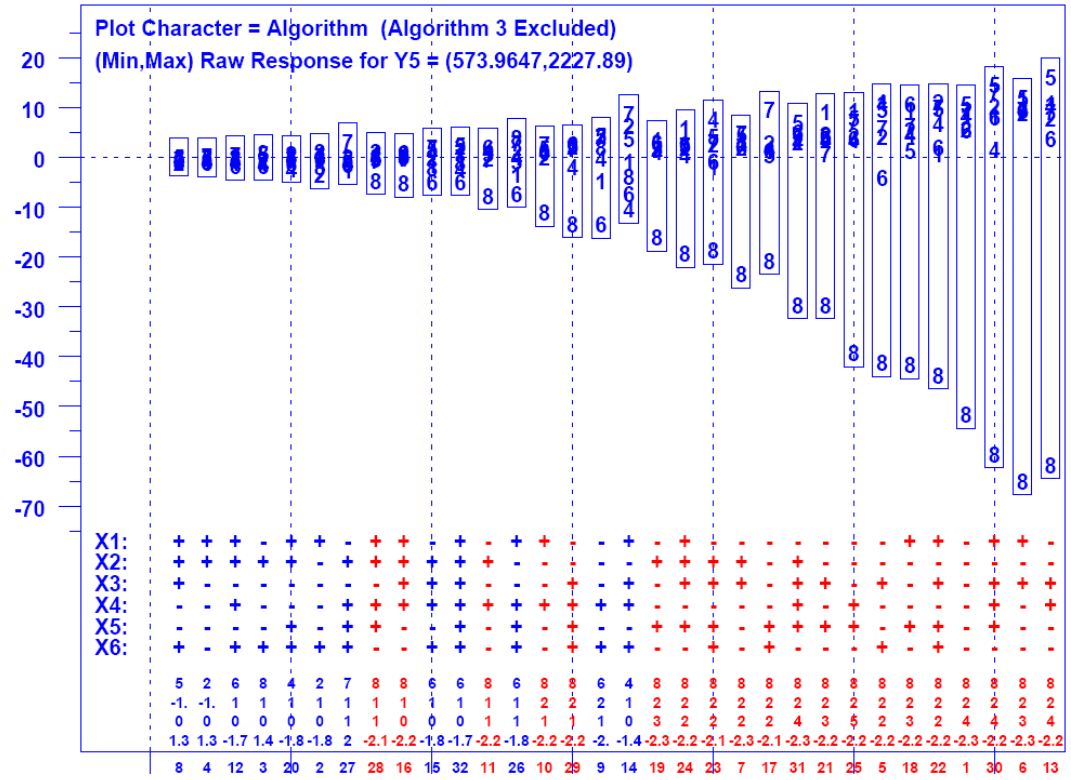


Figure 7-37. Detailed Analysis of Flow Completion Rate (flows per 200 ms) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

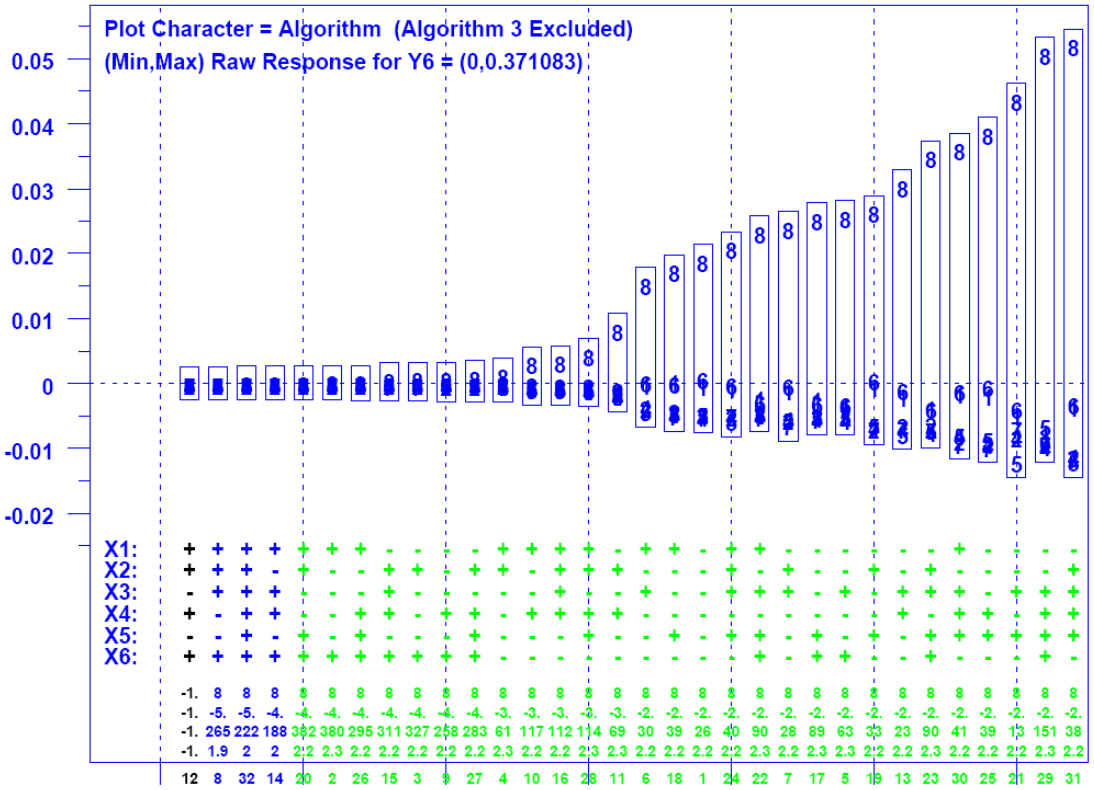


Figure 7-38. Detailed Analysis of Retransmission Rate (proportion of packets retransmitted) for Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

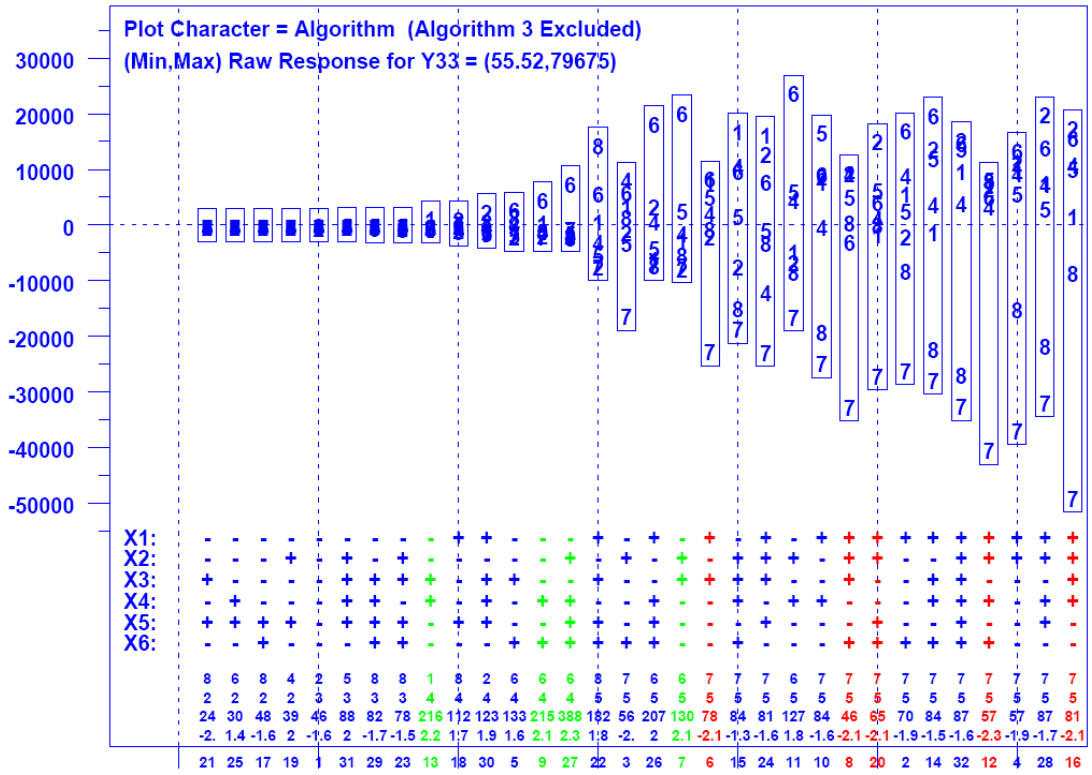


Figure 7-39. Detailed Analysis of Average Goodput (packets per second) on Long-lived Flow L1 in Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

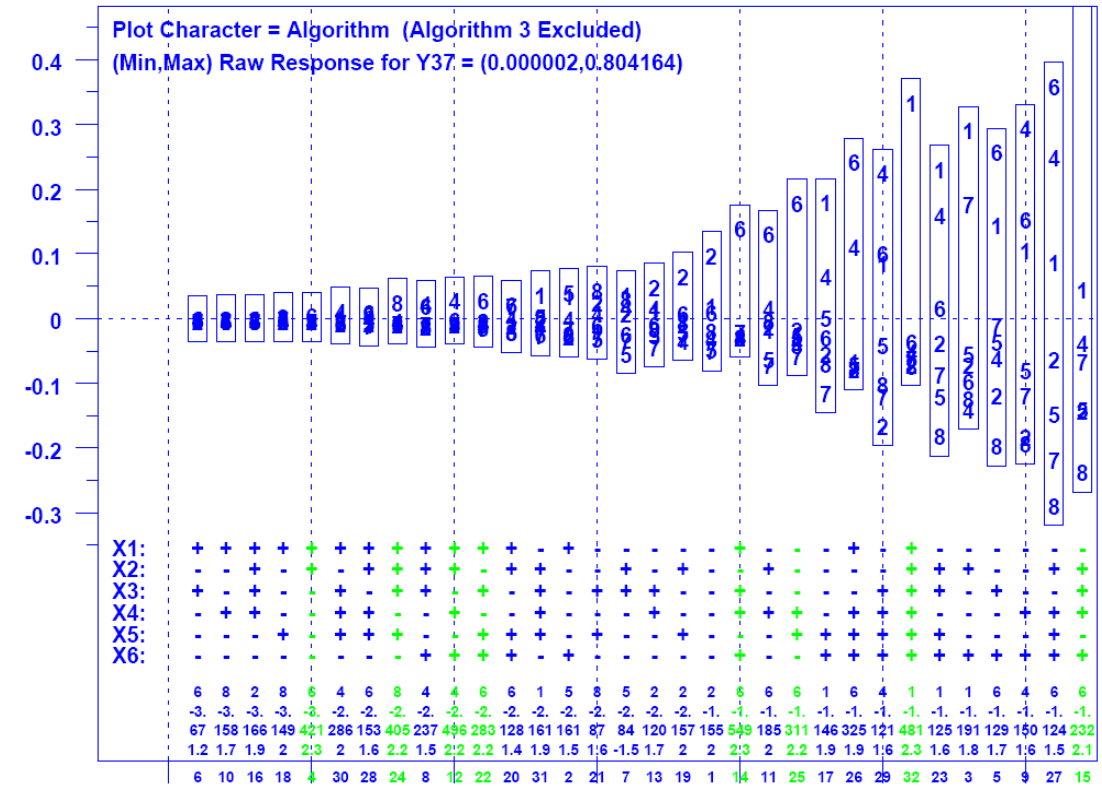


Figure 7-40. Detailed Analysis of Buffer Utilization (percentage of buffers occupied) in Router C0a during Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

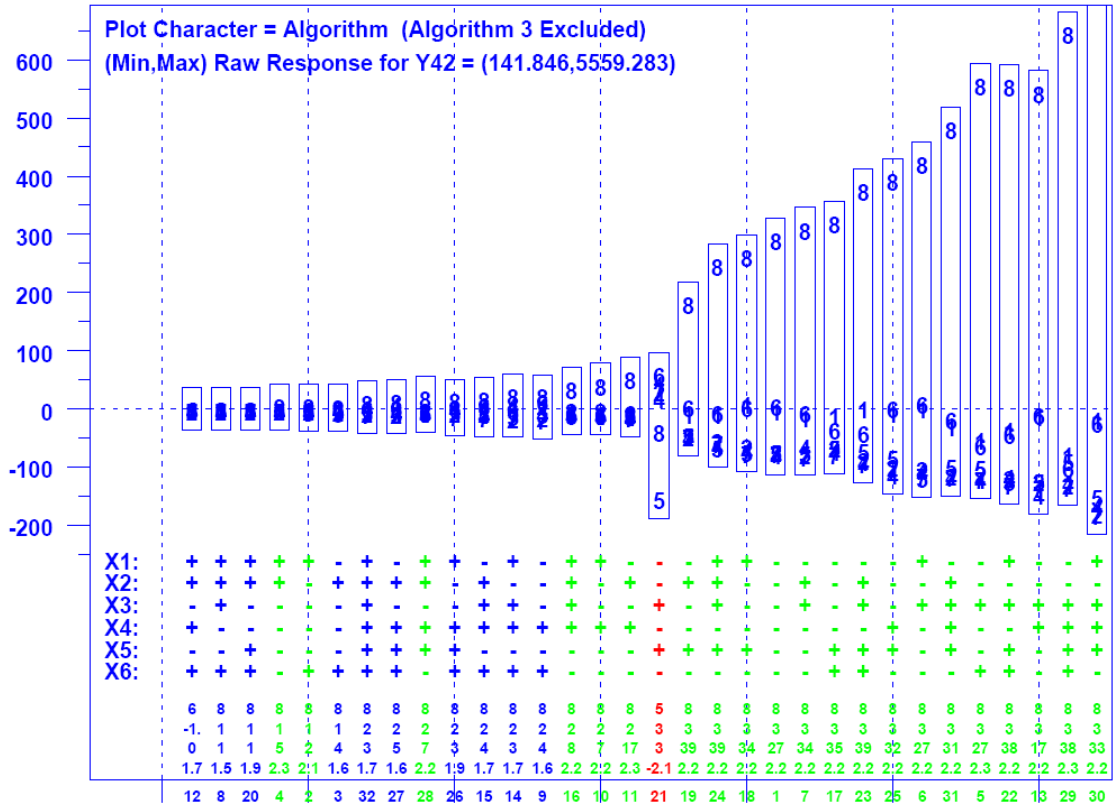


Figure 7-41. Detailed Analysis of Average Number of Connecting Flows during Time Period Three – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.4.1 Cluster Analysis for Totals. Fig. 7-42 shows a set of 32 annotated dendrograms with clustering based on the 28 aggregate responses. Similar to the cluster analyses for the three time periods, algorithms 3 and 8 appear distinctive in many (23) conditions. Also, as was true for TP2 and TP3, algorithm 6 stands out under condition 26. In addition, we note that cluster analyses for each of the time periods and the totals tend to suggest an affinity among some of the algorithms. For example, under selected conditions, algorithms 1, 4 and 6 (BIC, HSTCP and Scalable TCP) tend to cluster together and algorithms 2, 5 and sometimes 7 (CTCP, HTCP and TCP Reno) tend to cluster together.

7.4.4.2 Condition-Response Summary for Totals. Fig. 7-43, which gives the condition-response summary for the aggregate responses, proves quite revealing. First, standard TCP Reno (algorithm 7) pushes the fewest packets through the network. This occurs because TCP Reno increases its transmission rate slowly in congestion avoidance after leaving initial slow start. Second, FAST-AT (algorithm 8) provides the highest average goodput for DD, DF and FF flows (explained in Sec. 6.1), which transit very fast and fast paths, respectively. On the other hand, under most conditions, FAST-AT provides lowest average goodput for DN, FN and NN flows, which transit typical paths. Under uncongested conditions (e.g., 2, 12, 20 and 32) FAST-AT tends to provide highest throughput for DN, FN and NN flows. FAST-AT also sends more SYN packets, which

means flows have a more difficult time making connections. As a result FAST-AT connects and completes fewer flows.

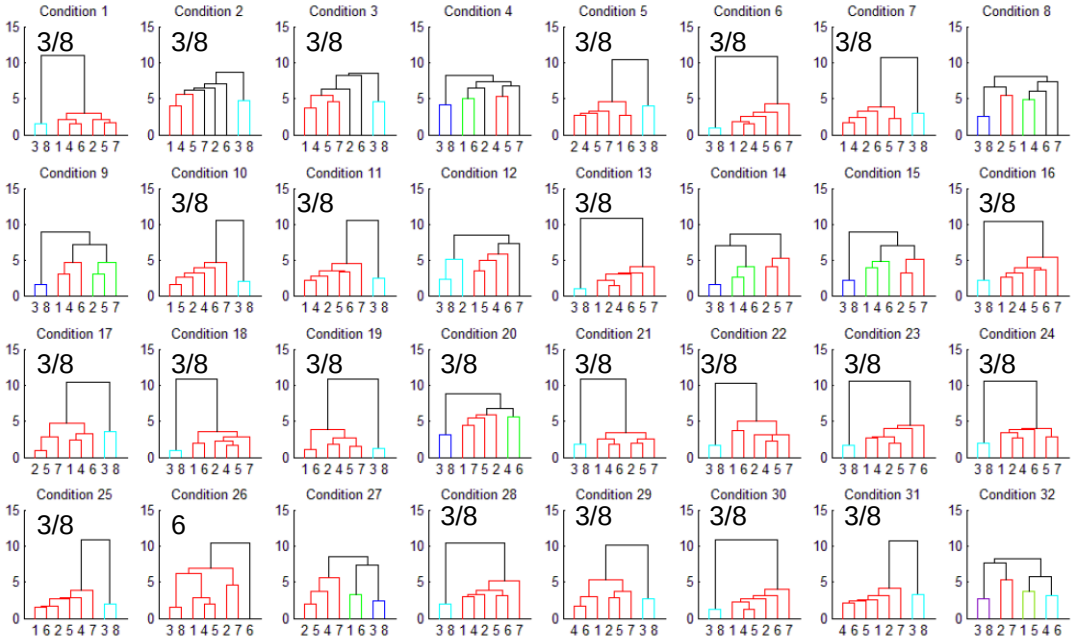


Figure 7-42. Cluster Analysis for Totals – x axis on each sub-plot gives algorithm identifier from Table 7-1 and y axis shows distance in response space between pairs of algorithms and clusters as indicated – each sub-plot Annotated to Identify Distinctive Algorithms 3/8

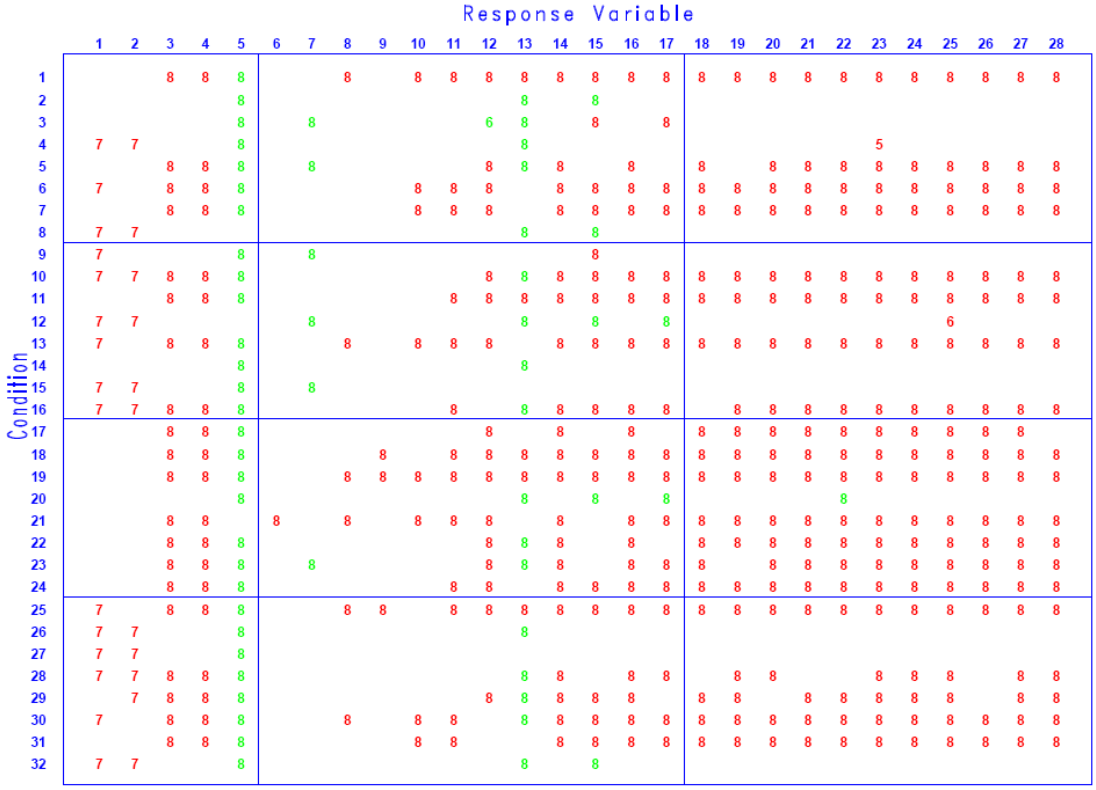


Figure 7-43. Condition-Response Summary for Totals – plot displays for each Factor Combination (row) vs. Response Variable (column) the Identifier of the Algorithm (from Table 7-1) manifesting a Statistically Significant Outlier (green high and red low)

7.4.4.3 Analysis of Significant Responses for Totals. Based on Fig. 7-43 we selected five responses for detailed analysis. Specifically, Figs. 7-44 to 7-48 provide results for aggregate packets input (T.y1) and output (T.y2), aggregate number of flows completed (T.y4), average number of SYNs sent per flow (T.y5) and average goodput on completed DD flows (T.y7).

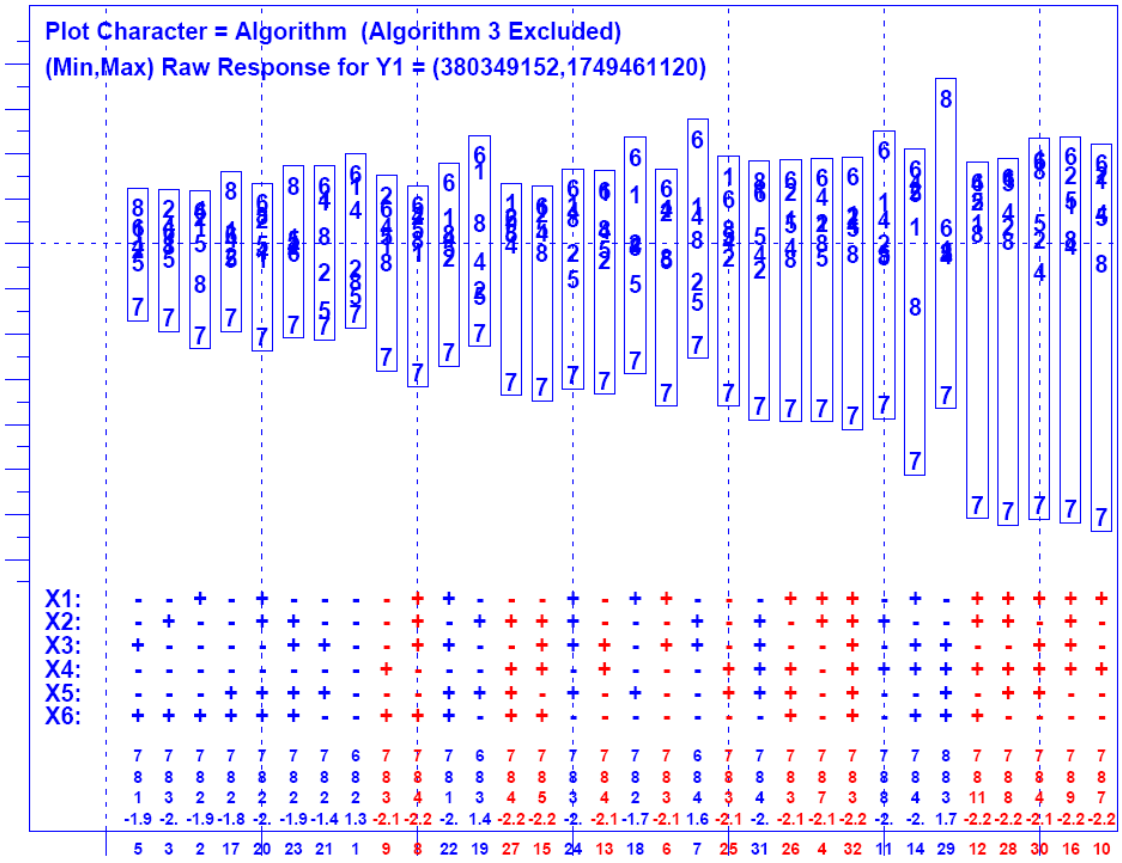


Figure 7-44. Detailed Analysis for Number of Packets Input during 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.4.4.4 Summary of Results for Totals. Under most conditions investigated in this experiment, FAST-AT connects and completes on the order of 10^5 to 10^6 fewer flows during the 25-minute scenario. For flows that complete over fast and very fast paths, FAST-AT provides higher average goodput than the other algorithms. In some cases, this could be attributed to the fact that FAST-AT has fewer active flows sharing such paths because it is harder for flows to connect. In other cases, this can be attributed to the ability of FAST-AT to quickly achieve maximum transfer rate. TCP Reno injects fewer packets into the network (and thus outputs fewer packets). This can be attributed to the slow rate of increase in the congestion window during TCP Reno congestion avoidance. The results also show another a change over the previous experiment, where FAST input more packets into the network. In this experiment we excluded FAST data from the analyses, which then revealed that FAST-AT does not input as many packets per second into the network as FAST. This appears attributable to the ability of FAST-AT to lower the α parameter under congestion, and to increase α only slowly as congestion eases.

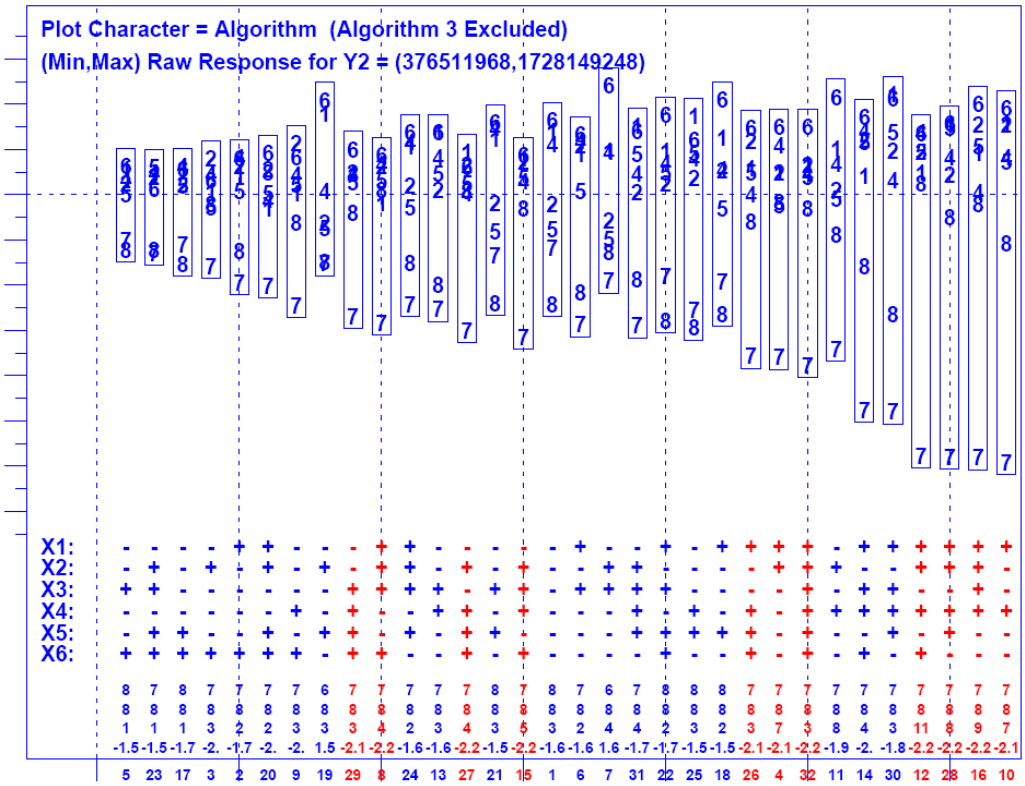


Figure 7-45. Detailed Analysis for Number of Packets Output during 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

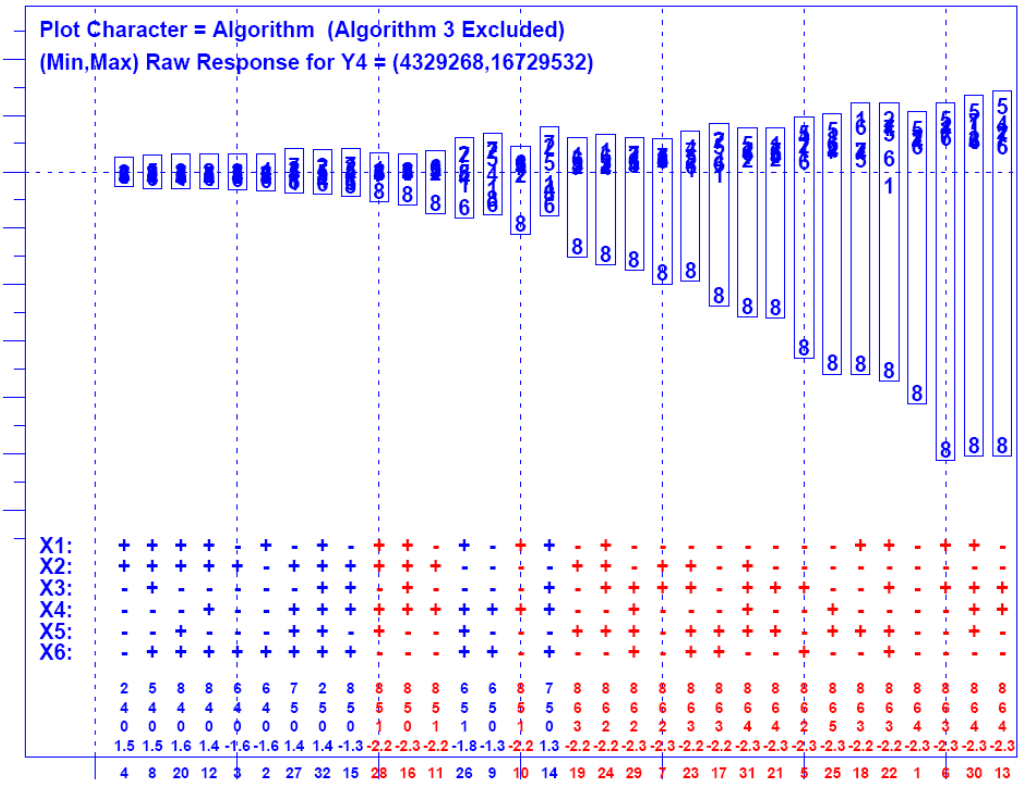


Figure 7-46. Detailed Analysis for Number of Flows Completed over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

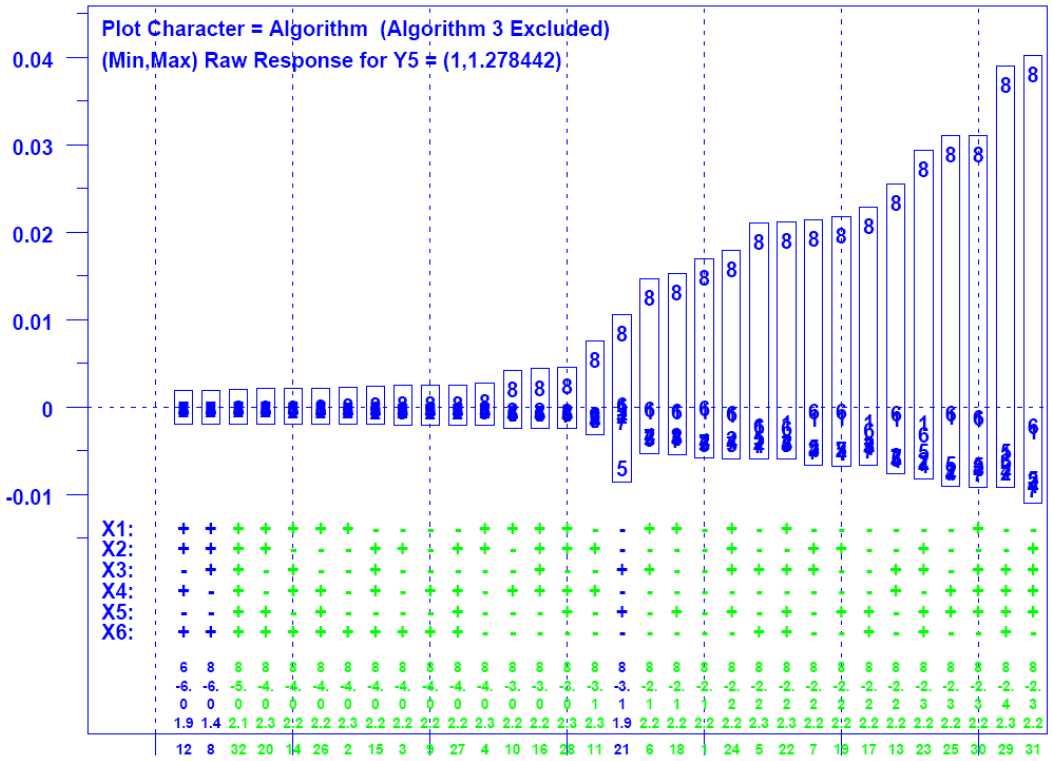


Figure 7-47. Detailed Analysis for Average SYN Rate (number of SYNs sent per flow) for Connecting Flows over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

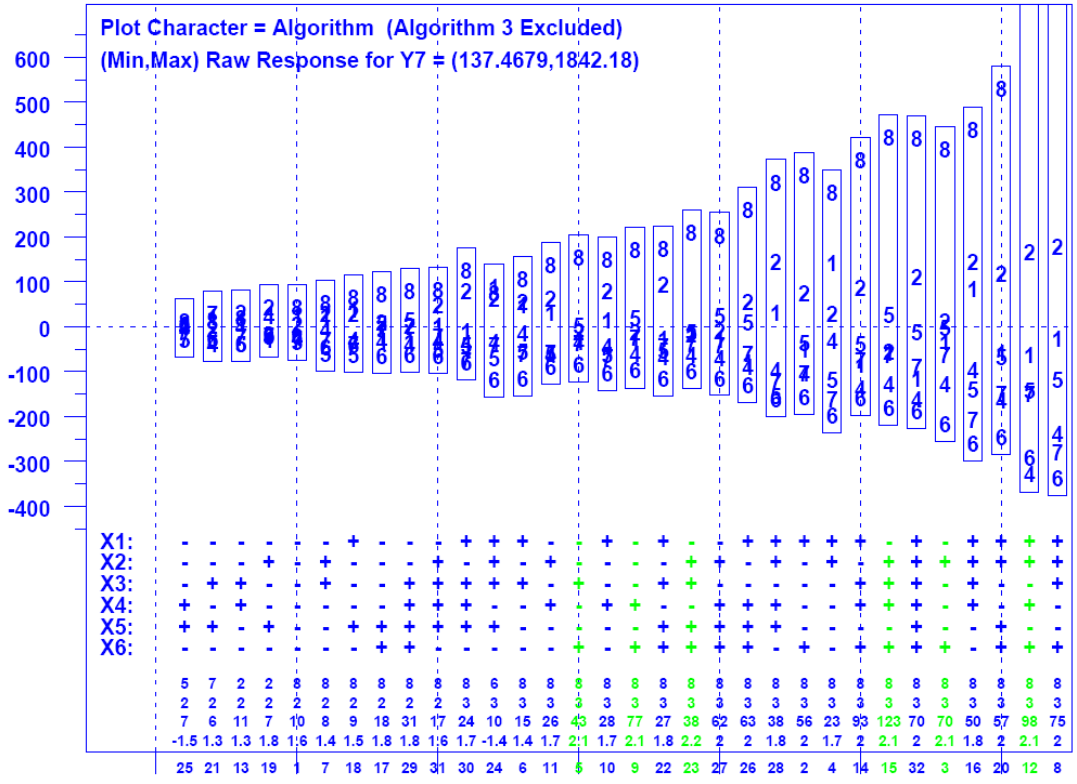


Figure 7-48. Detailed Analysis for Average Goodput (packets per second) on Completed DD Flows over 25-minute Scenario – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

7.5 Findings

We draw four main findings from our experimental results. First, all congestion control algorithms (except FAST and FAST-AT, algorithms 3 and 8, respectively) provided comparable network-wide behavior and user experience with respect to Web downloads and small file transfers over most path classes. This confirms similar findings from our previous experiment (discussed in Chapter 6) that adopted a large initial slow-start threshold. Modest differences (usually $\leq$ 20 %) do appear with respect to throughputs on DD flows. We identify and discuss these differences below in Sec. 7.5.1. Second, as discussed below in Sec. 7.5.2, when deployed network wide, alternate congestion control algorithms 3 (FAST) and 8 (FAST-AT) can produce macroscopic changes in network behavior at congested places in the topology and during congested periods. This confirms findings from our previous experiment. Third, for long-lived flows, TCP provides significantly lower throughput during TP1. While other congestion control algorithms perform better than TCP during TP1, we did detect differences among the algorithms in time taken to achieve maximum transfer rate for long-distance, long-lived flows. Differences also occur during TP3, when attempting to recover from the excessive congestion in TP2. During TP2, the congestion control algorithms provide comparable throughput across all long-lived flows. Below in Sec. 7.5.3 we discuss findings relating to long-lived flows. Fourth, under certain conditions, CTCP (algorithm 2) can drive congestion window size to substantially higher values than the other congestion control algorithms we simulated. As with our previous experiment, this behavior arises during TP3. However, as discussed below in Sec. 7.5.4, in this experiment the CTCP congestion window reaches a much lower maximum size than in our previous experiment. Finally, in Sec. 7.5.5, we identify some tendencies that were apparent but that did not achieve statistical significance.

7.5.1 Finding #1

Setting aside FAST and FAST-AT, for the experiment scenario and conditions examined in this chapter, the alternate congestion control algorithms exhibited indistinguishable macroscopic behavior and modest differences in user experience. This suggests that there was no overall advantage to be gained in switching the entire network to a particular alternate congestion avoidance scheme, nor was there any overall disadvantage in switching. (Remember we are excluding FAST and FAST-AT from this finding.) Selected users could experience somewhat higher throughputs when using alternate congestion control algorithms to complete file transfers with a size exceeding the initial slow-start threshold, but no widespread improvement in user experience should be expected.

The reasons underlying this finding are similar to the reasons identified in Sec. 6.5.1. Slow-start procedures are unaffected by alternate congestion control mechanisms, which define replacements only for the TCP congestion avoidance phase. No matter what congestion control mechanism is used, a flow commences operating in initial slow-start and switches to congestion avoidance only after a packet loss or once the initial slow-start threshold is reached. Aside from FAST, FAST-AT and TCP Reno, the alternate congestion avoidance procedures specify an activation threshold. Below that threshold, a flow adopts standard TCP congestion avoidance procedures; above that threshold the flow adopts alternate congestion avoidance procedures.

Recall from Fig. 7-1 that we simulated 32 conditions covering a range of congestion patterns, which could be classified approximately into 15 uncongested and 17 congested conditions. Condition 8 created the least congestion, while condition 21 created the most congestion. Of course, even uncongested conditions include localized congestion arising from the onset of jumbo file transfers in TP2, as well as from hot spots appearing from time-to-time at particular access routers. For example, Fig. 7-2 plots the distribution of flow states under an uncongested condition (condition 3) for algorithm 7 (TCP Reno). Note that most of the 700 or so active flows (red) in TP1 (t=3000 to t=4500) operate in initial slow start (green). This means that these active flows complete their file transfers without entering congestion avoidance. Only 20 or so (brown) flows enter congestion avoidance. Since condition 3 is uncongested, most of these 20 flows have likely entered congestion avoidance because they are larger than 100 packets, which is the initial slow-start threshold. This means that fewer than 3 % of active flows achieve any advantage from using alternate congestion avoidance procedures. This pattern also holds in TP3, after about t=6500, when the congestion (induced in TP2) abates. During the congested period (TP2) DD flows build up, as jumbo file transfers commence, and these flows contend for bandwidth, which leads to packet losses and thus to more flows operating under normal congestion control. Under such conditions, alternate congestion control algorithms do not offer any advantage. A similar story appears for congested conditions that extend over all three time periods.

Fig. 7-3 plots the distribution of flow states for TCP Reno (algorithm 7) under a congested condition (condition 5). About 60 % of the nearly 8500 active flows (red) operate in normal congestion control mode (brown), while the remaining 40 % operate in initial slow start (green). Under such congested conditions, alternate congestion avoidance procedures are not much activated. Most flows in a heavily congested network, or in heavily congested portions of a network, will be sharing paths with many other flows. For this reason, one should expect most flows to be operating within normal congestion control mode; these flows cannot achieve a large enough congestion window size (or avoid losses for long enough) to activate alternate congestion avoidance procedures. On the other hand, flows transiting very fast (DD) paths may be able to benefit from alternate congestion control procedures during periods with little congestion.

Table 7-10 reports the average, minimum and maximum per flow goodputs on DD flows (excluding long-lived flows), averaged over all conditions for each time period, for each of the eight congestion control algorithms. The table shows only modest differences in average goodput among most of the algorithms for TP1, though the average goodput lags somewhat for Scalable TCP. Note also that TCP Reno provides relatively high average goodput. Also notice that the average maximum goodput does not differ much in TP1 for most congestion control algorithms, though HSTCP seems to stand out somewhat on this metric. The average minimum goodputs are lower for BIC, HSTCP and Scalable TCP, which suggests that some flows are treated unfairly under these algorithms. On the other hand, TCP Reno provides the highest average minimum throughput, which suggests that these flows receive fairly equal treatment. During TP1 and TP2, the average, minimum and maximum goodputs provided by FAST and FAST-AT appear to be quite a bit higher than for the other algorithms, but this is due to the fact that fewer flows are actively transmitting under FAST and FAST-AT. As discussed previously, FAST flows have a more difficult time making connections. (This trait also

holds for FAST-AT.) The average goodputs among all the algorithms are relatively close in TP3. In general, for the conditions simulated, the alternate congestion control algorithms do not appear to provide a significant advantage over TCP Reno on DD flows with typical Web traffic. We consider the case of long-lived DD flows below in Sec. 7.5.3.

Table 7-10. Goodputs (pps) on DD Flows Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	174.89	189.85	207.05	172.80	183.82	164.79	194.20	201.12
	Minimum	49.48	89.62	89.25	49.97	70.90	39.05	100.63	89.25
	Maximum	378.29	345.40	377.81	431.02	374.62	363.71	357.84	356.34
TP2	Average	1545.97	1651.73	2132.79	1280.32	1433.19	1163.48	1327.92	2197.17
	Minimum	470.14	498.14	712.72	420.35	523.24	280.87	500.09	667.58
	Maximum	6173.99	5250.65	7380.92	4992.77	4781.59	4868.55	4406.78	7455.79
TP3	Average	2105.58	2077.44	2205.36	2309.45	2222.60	1998.66	2262.30	2407.78
	Minimum	322.88	254.83	372.24	262.99	289.84	317.60	291.07	400.14
	Maximum	4827.73	5627.25	5456.05	4726.75	5774.65	4684.17	5724.38	5963.79

7.5.2 Finding #2

In our previous experiment, we found (Sec. 6.5.2) that the FAST congestion control algorithm, when deployed throughout the network, produced macroscopic behavior changes at congested places in the topology and during congested periods. Further, we found that these changes could present Web-browsing users with lower average goodputs and longer connection times. The influence of these effects increased with increasing congestion. These findings suggested that deploying FAST on a wide scale could incur significant risk. Our current experiment reveals that FAST-AT (algorithm 8) shares these same undesirable traits, and for the same reasons.

FAST-AT exhibits rapid oscillations in congestion window size when a path has insufficient buffers to contain the packets that the algorithm attempts to maintain queued at a bottleneck. When a large number of flows simultaneously transit a network router, the overall effect can be to flood the router with many packets. When the number of flows is sufficient to overrun the available buffers in the router, FAST-AT flows exhibit an oscillatory behavior that can create additional congestion that causes flows to remain in oscillation for an extended time. For example, Fig. 7-49 shows the change in the congestion window for long-lived FAST-AT flow L2 during 500 measurement intervals (100 s) within TP2 under condition 21. For comparison, Fig. 7-50 gives the behavior of standard TCP Reno under the same circumstances. (The reader may wish to compare these figures with Figs. 6-60 through 6-66 from the previous experiment.)

The rapid oscillatory behavior of FAST-AT results in numerous packet losses, which leads to a large rate of congestion window increases (as shown in Figs. 7-10, 7-22 and 7-34) and to a higher retransmission rate (as shown in Figs. 7-12, 7-25 and 7-38). The higher loss rate also causes a higher SYN rate (as shown in Fig. 7-47), which leads to a larger number of flows pending in a connecting state (as shown in Figs. 7-15, 7-29 and

7-41) because flows take longer to connect. Flows also take longer to complete because a larger number of packets must be retransmitted. This effect can be seen in Figs. 7-11, 7-14, 7-24, 7-27 and 7-37, which show that FAST-AT flows have a significantly lower completion rate. The net effect of a lower completion rate appears in Fig. 7-46, which shows that FAST-AT completes many fewer flows than other algorithms, excluding FAST, over a 25-minute period.

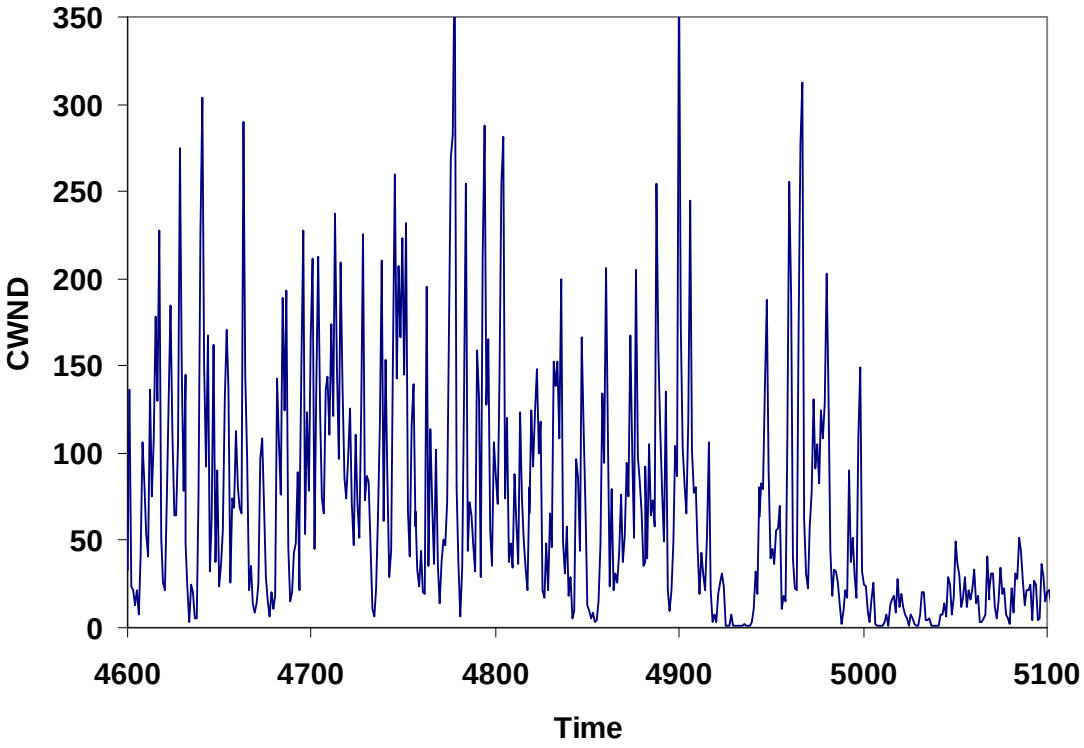


Figure 7-49. Change in Congestion Window (packets) under FAST-AT for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

In summary, a large network with many simultaneously active flows can induce congestion at various times and locations within the topology. When congestion is sufficient to induce losses, flows using the FAST-AT algorithm can enter a rapid oscillatory behavior that exacerbates congestion. As a result, the network can exhibit a higher overall loss rate with consequent increase in retransmissions. Flows can take longer to connect and complete. The number of flows completed in such a network can be significantly reduced over long time spans. Should FAST-AT be deployed throughout a network, typical Web-browsing users could experience lower average goodput on flows transiting through congested areas. These findings also apply to FAST, which showed similar behavior in the previous experiment.

7.5.3 Finding #3

The design of most alternate congestion control algorithms is motivated by a desire to improve on standard TCP congestion avoidance procedures by providing higher goodput when transmitting large files over long-delay, high-speed network paths. To investigate the degree to which alternate congestion control algorithms achieve this aim, we included

three long-lived flows in several of our experiments. The long-lived flows transmit (infinitely large) files over very fast (DD) paths between sources and receivers capable of operating at 80 x 10^3 packets per second. In the previous experiment, we found little difference in goodput among congestion control algorithms (including TCP Reno) because all long-lived flows achieved maximum transfer rate (in TP1) while operating in initial-slow start and because all long-lived flows were influenced by heavy congestion in TP2. In the current experiment, we expected TCP Reno to perform less well on long-lived flows in TP1 because the initial slow-start threshold is low (100 packets). As in the previous experiment, we also expected TCP Reno to perform less well in TP3 during recovery from the heavy congestion in TP2. Further, we expected any differences in performance among the alternate congestion control algorithms to become apparent.

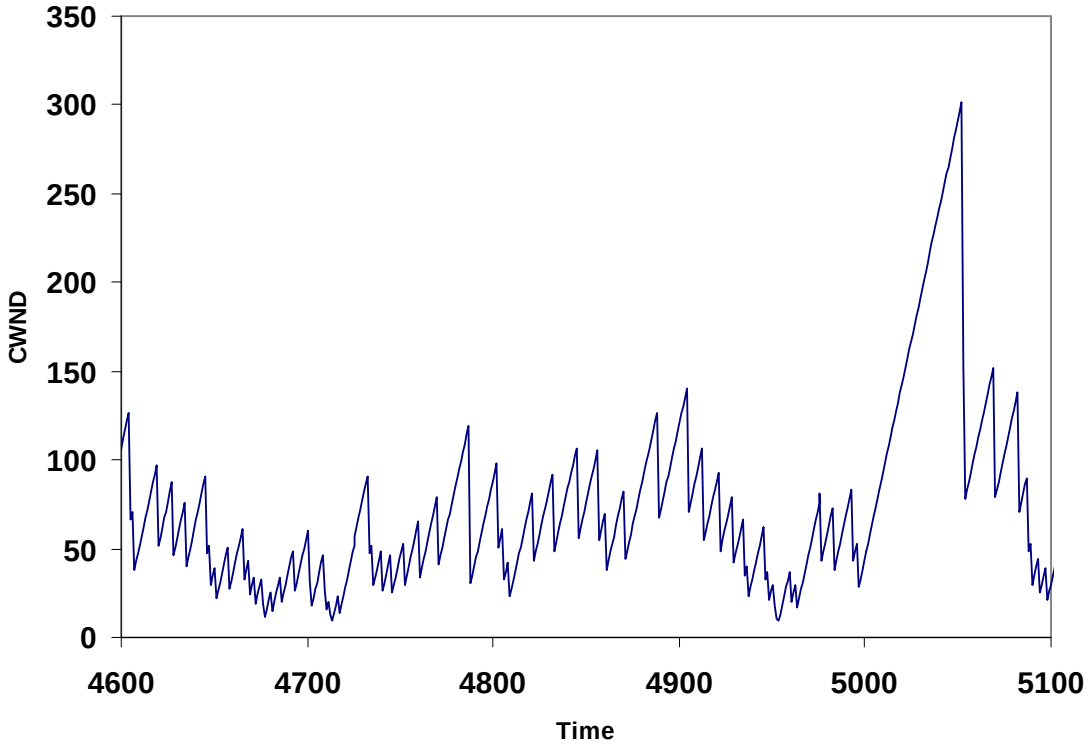


Figure 7-50. Change in Congestion Window (packets) under TCP Reno for Long-Lived Flow L2 during 500 Measurement Intervals (200 ms each) within TP2 under Condition 21

Table 7-11 reports the average, minimum and maximum goodputs (divided by 1000) provided by each congestion control algorithm for the long-distance, long-lived flow (**L1**) averaged over all 32 conditions for each of the three time periods. As expected, during TP1 the standard TCP Reno congestion control algorithm underperforms all of the alternate congestion control algorithms. This demonstrates that the alternate algorithms can provide higher goodputs than TCP when transmitting large files over long-delay, high-speed network paths, provided there is a low initial slow-start threshold. This effect is also evident for the moderate-distance (**L2**) and short-distance (**L3**) long-lived flows, as shown in Tables 7-12 and 7-13, but the effect diminishes significantly with decrease in propagation delay. The effect is also evident in Fig. 7-17. The inability of TCP

congestion control to quickly reach maximum transfer rate is also responsible for the fact that TCP Reno outputs fewer packets per measurement interval in TP1 (see Fig. 7-16). In fact, when considered across the entire 25-minute scenario, TCP Reno inputs significantly fewer packets under most conditions (see Fig. 7-44).

Table 7-11. Per Flow Goodputs (pps/1000) for Long-Lived Flow L1 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	59.822	62.497	64.857	58.513	64.617	62.087	15.859	64.840
	Minimum	44.960	34.494	37.690	43.804	47.128	48.063	6.581	38.363
	Maximum	73.204	77.051	78.899	72.038	74.466	73.150	25.151	78.899
TP2	Average	8.232	5.830	2.880	9.939	6.934	13.775	3.318	2.745
	Minimum	0.604	0.260	0.240	0.508	0.494	0.660	0.320	0.191
	Maximum	51.007	35.010	23.003	61.699	59.662	53.940	18.459	26.766
TP3	Average	23.232	23.461	27.237	22.525	22.964	27.472	6.335	16.164
	Minimum	0.104	0.087	0.050	0.087	0.110	0.117	0.087	0.056
	Maximum	79.593	79.675	76.994	79.588	79.079	78.140	37.658	78.903

Table 7-12. Per Flow Goodputs (pps/1000) for Long-Lived Flow L2 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	70.304	69.200	70.526	69.110	66.900	70.560	43.070	70.479
	Minimum	59.547	47.952	52.016	57.204	47.210	60.918	27.098	52.179
	Maximum	77.929	78.896	79.707	77.441	76.236	77.431	65.462	79.707
TP2	Average	32.860	26.996	19.188	34.727	26.716	40.243	26.732	17.222
	Minimum	2.981	1.706	1.850	3.189	1.706	2.394	2.444	1.466
	Maximum	80.000	76.982	64.945	79.986	79.984	79.979	72.793	65.099
TP3	Average	61.447	58.343	53.823	60.902	57.718	61.121	51.106	54.677
	Minimum	29.751	17.704	10.311	19.725	22.337	17.682	12.542	7.317
	Maximum	80.000	80.000	80.000	80.000	80.000	80.000	80.000	80.000

Table 7-13. Per Flow Goodputs (pps/1000) for Long-Lived Flow L3 Averaged over all 32 Conditions for Each Time Period

		BIC	CTCP	FAST	HSTCP	HTCP	Scalable	TCP	FAST-AT
TP1	Average	73.219	72.681	71.735	72.298	67.693	73.039	63.519	71.673
	Minimum	63.986	55.644	55.320	60.997	46.031	64.104	48.425	55.246
	Maximum	79.331	79.558	79.907	79.904	77.613	78.975	76.185	79.907
TP2	Average	29.485	25.822	18.281	30.809	22.005	35.495	25.882	16.238
	Minimum	2.030	1.925	1.672	1.632	1.063	1.152	1.424	1.150
	Maximum	79.985	76.669	74.230	79.672	79.181	79.997	79.685	73.810
TP3	Average	61.066	57.312	55.706	59.204	56.872	61.321	57.438	55.849
	Minimum	20.817	14.118	17.248	20.842	14.776	17.259	22.235	15.677
	Maximum	80.000	80.000	79.988	80.000	80.000	80.000	80.000	79.992

Considering TP3, we find that, on average, TCP Reno recovers significantly less well than the alternate congestion control algorithms for the long-distance (**L1**) and moderate-distance (**L2**) long-lived flows, but TCP Reno performs on a par with the alternate algorithms when recovering throughput on the short-distance (**L3**) long-lived flow. We also note that FAST-AT lags behind the other alternate congestion control algorithms when recovering on the long-distance (**L1**) long-lived flow. Further, FAST and FAST-AT tend to recover less well than other alternate congestion control algorithms on the moderate-distance (**L2**) and short-distance (**L3**) long-lived flows. In general, all congestion control algorithms provide similar goodputs across all time periods on the short-distance, long-lived flow, so differences in goodput arise only with sufficient propagation delay.

Considering TP2, Scalable TCP, BIC and HSTCP retain higher average goodputs on the long-distance (**L1**) and moderate-distance (**L2**) long-lived flows, while FAST and FAST-AT yield lower average goodputs. This tendency also appeared in the previous experiment, as discussed in Sec. 6.5.4.

In an effort to better understand similarities and differences among the congestion control algorithms, we next consider some of the detailed operations on long-lived flows, first under uncongested conditions and then under the most congested condition. We begin with the least congested condition (condition 8).

Table 7-14. Time (seconds) until Long-Lived Flows Reach Maximum Transfer Rate in TP1 for Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**39.4**	**12.6**	**4.6**
CTCP	**14.8**	**6.0**	**2.8**
FAST	**8.0**	**2.2**	**1.0**
FAST-AT	**8.0**	**2.2**	**1.0**
HSTCP	**45.6**	**15.6**	**6.0**
HTCP	**30.2**	**21.4**	**15.0**
Scalable	**33.8**	**13.6**	**6.0**
TCP	**-----**	**112.8**	**30.6**

Table 7-14 shows for each long-lived flow, under the least congested condition, the time (in seconds) taken by each congestion control algorithm to reach its maximum transfer rate of 80×10^3 packets per second. The table reveals that TCP Reno takes much longer to achieve maximum rate than the other algorithms – in fact, TCP Reno does not reach the maximum rate on the long-distance, long-lived flow (**L1**) during TP1 (or any

other time period). Table 7-14 also reveals that FAST and FAST-AT converge most quickly to maximum rate, followed by CTCP. HTCP converges fairly quickly on the long-distance, long-lived flow but lags behind the other non-TCP algorithms in achieving maximum rate on the moderate- and short-distance, long-lived flows.

Table 7-15. Time (seconds) until Long-Lived Flows Recover Maximum Transfer Rate in TP3 for Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**186.8**	**6.0**	**21.2**
CTCP	**5.2**	**2.8**	**0.6**
FAST	**18.2**	**0.0**	**1.6**
FAST-AT	**65.4**	**6.8**	**5.8**
HSTCP	**10.4**	**0.0**	**7.8**
HTCP	**26.4**	**0.0**	**3.4**
Scalable	**59.6**	**7.4**	**0.6**
TCP	**----**	**0.0**	**2.4**

Table 7-15 reports for each long-lived flow, under the least congested condition, the time (in seconds) taken by each congestion control algorithm to recover its maximum transfer rate of 80 x 10^3 packets per second after TP3 begins (at t=6000). The table suggests that BIC and FAST-AT are somewhat sluggish in recovery and that CTCP is quite good. The table is difficult to interpret, however, because the numbers seem somewhat inconsistent. For example, FAST, HTCP and TCP require no time to recover maximum transfer rate on long-lived flow **L2**. This means that, for these congestion control algorithms, flow **L2** had already recovered the maximum transfer rate before TP3 began. Perhaps better insight can be provided by examining the time series of goodput for each flow over TP2 and the beginning of TP3, specifically between t=4500 and t=6500, which covers 400 seconds (i.e., 2000 200 ms intervals). We provide eight time series (one per congestion control algorithm) for each long-lived flow.

Fig. 7-51 plots time series for goodput on long-lived flow L1. For the quickest recovering algorithm, CTCP, the plot reveals that goodput varies significantly during TP2, but goodput is generally improving after about t=5200. Further, by t=6000 CTCP has nearly regained its maximum transfer rate. Contrast this with HTCP (second quickest recovering) where goodput is trending generally down until t=6000, after which HTCP recovers quickly. Similarly, FAST shows widely varying goodput during TP2 and then

recovers quickly after *t*=6000. FAST-AT recovers less quickly because the α-parameter had been auto-tuned to 20, whereas FAST used a fixed α-parameter of 200.

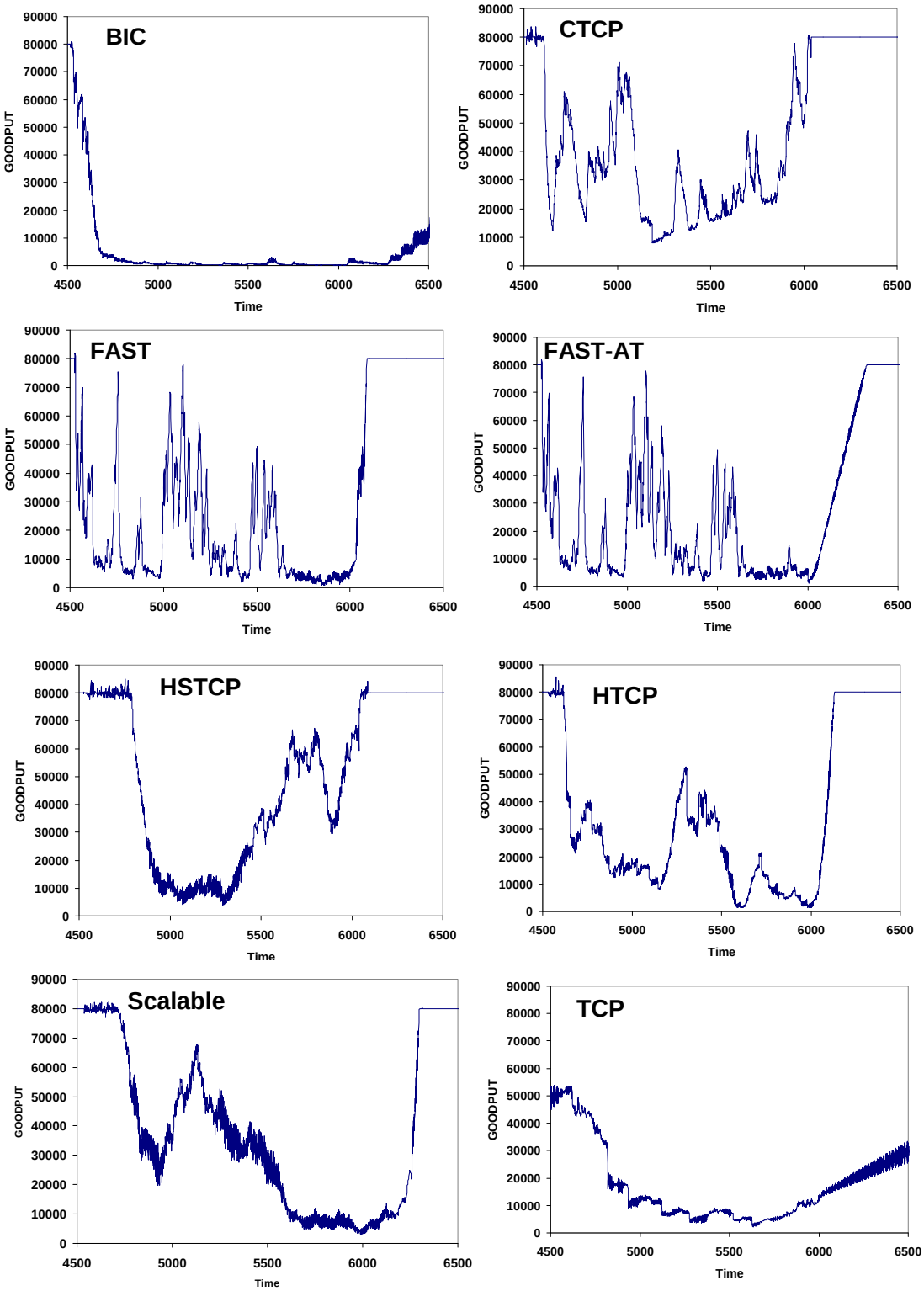


Figure 7-51. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 8 – time is in 200 ms interval since beginning of simulation

Fig. 7-51 also shows that goodput for HSTCP is generally improving after about *t*=5400, while goodput for Scalable TCP is trending generally down until *t*=6000 and then begins to recover. On the other hand, goodput under BIC reaches a very low level by *t*=4800 and does not begin to recover until *t*=6250. Goodput under TCP bottoms at *t*=5500 and then begins to recover. While BIC does eventually recover its maximum transfer rate (not shown) during TP3, TCP never achieves the maximum transfer rate on flow **L1** during any time period for condition 8. This highlights the fact that the alternate congestion control algorithms can provide substantial goodput improvement for large files on high-delay, high-speed paths.

In thinking about the results for long-lived flows recall that the long-distance flow **L1** also exhibits the highest congestion from jumbo file transfers during TP2. The moderate-distance flow **L2** suffers least congestion from jumbo file transfers during TP2 and the short-distance flow **L3** suffers congestion at a level between that of **L1** and **L2**. This factor helps explain why it takes less time in Table 7-15 to recover maximum transfer rate on flow **L2** than on flow **L3**. We continue the analysis of results in Table 7-15 with a consideration of flow **L2**.

Fig. 7-52 plots time series for goodput on flow **L2** between *t*=4500 and *t*=6500. Given that flow **L2** traverses the least congested path during TP2, we expect quick recovery during TP3, as shown in Table 7-15, where four of the congestion control algorithms (including TCP Reno) have reestablished maximum transfer rate prior to *t*=6000. The time series plots reveal different behaviors during TP2 among the four congestion control algorithms in question. HSTCP goodput never falls below the maximum transfer rate.[1] HTCP goodput briefly falls below the maximum transfer rate (around *t*=5900) and then recovers quickly. FAST goodput oscillates markedly throughout TP2 and recovers its equilibrium at the maximum transfer rate just prior to *t*=6000. FAST-AT behaves similarly to FAST but recovers its equilibrium somewhat past *t*=6000. On the other hand, TCP Reno reaches a lowest goodput of about 50×10^3 pps at about *t*=5400 and then improves steadily, reaching maximum transfer rate just prior to *t*=6000. Thus, while Table 7-15 reports similar recovery times for a number of alternate congestion control algorithms on flow **L2** under condition 8, the time series reveal behavioral differences among these algorithms.

Fig. 7-52 also shows that CTCP goodput varies somewhat like FAST (and FAST-AT) but the evident oscillations tend to be less frequent with less change in amplitude. On the other hand, Scalable TCP retains maximum goodput until nearly *t*=5500, after which its goodput drops to about 50×10^3 pps, just before *t*=6000, and recovers to 80×10^3 pps by about *t*=6200. BIC goodput drops generally to under 10×10^3 pps at about t=5400 and then rises steadily, reaching 80×10^3 pps at about *t*=6100.

We continue this analysis by considering the temporal behavior of goodput for each congestion control algorithm on the short-distance, long-lived flow **L3**. Fig. 7-53 displays the relevant time series. Recall that flow **L3** faces moderate levels of congestion during TP2 due to competing jumbo file transfers.

[1] Note that the noise around the maximum transfer rate (80×10^3 pps) is due to a measurement artifact. We measure goodput as the rate at which packets are received by the receiver. Since access routers typically operate much faster than sources and receivers, there are periods during which goodput oscillates around the maximum transfer rate due to packet clumping. This oscillation would not appear if we had instead measured the rate at which the receiver emits ACK and NAK packets in response to arriving data packets.

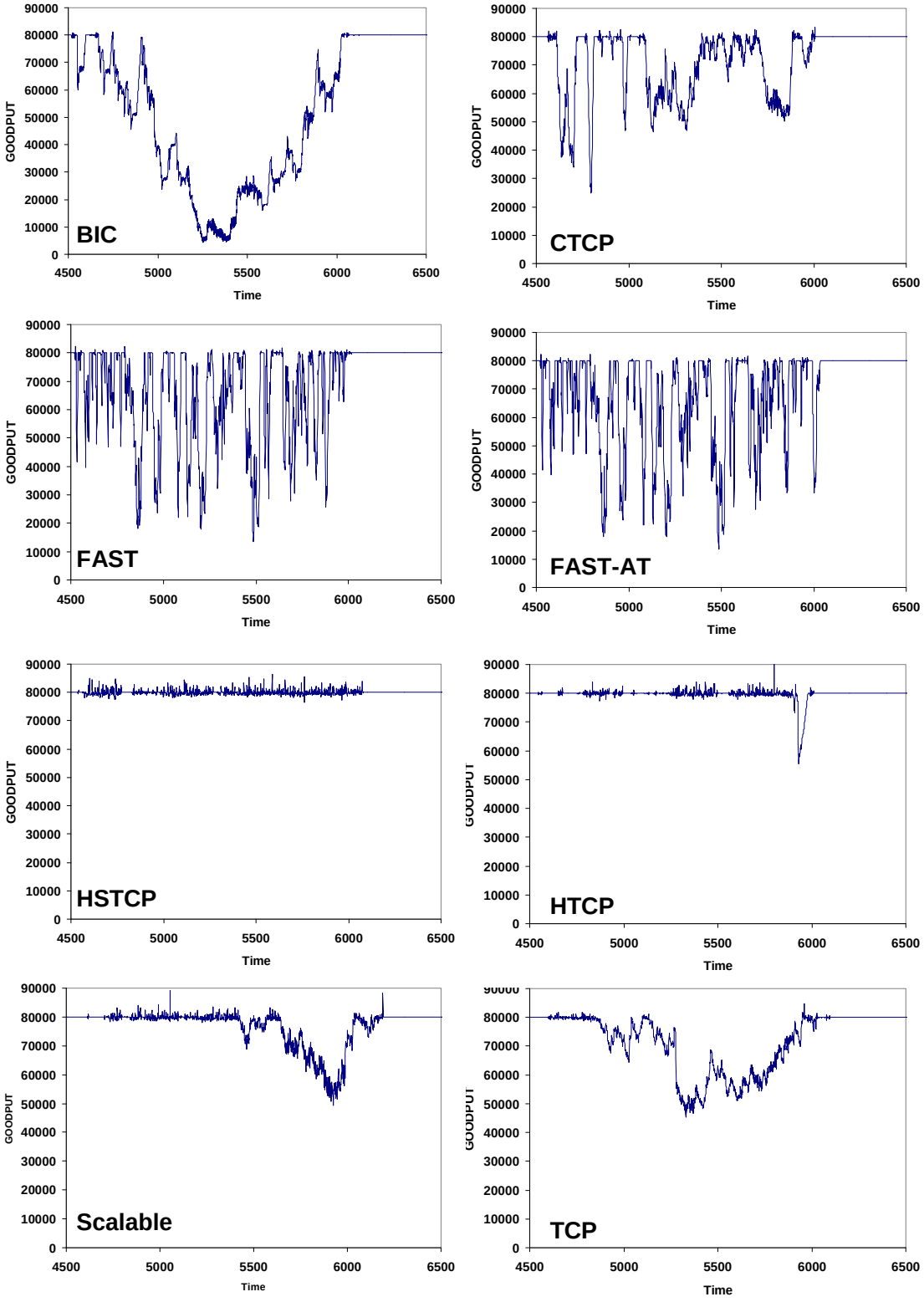


Figure 7-52. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L2 under Condition 8 – time is in 200 ms intervals since beginning of simulation

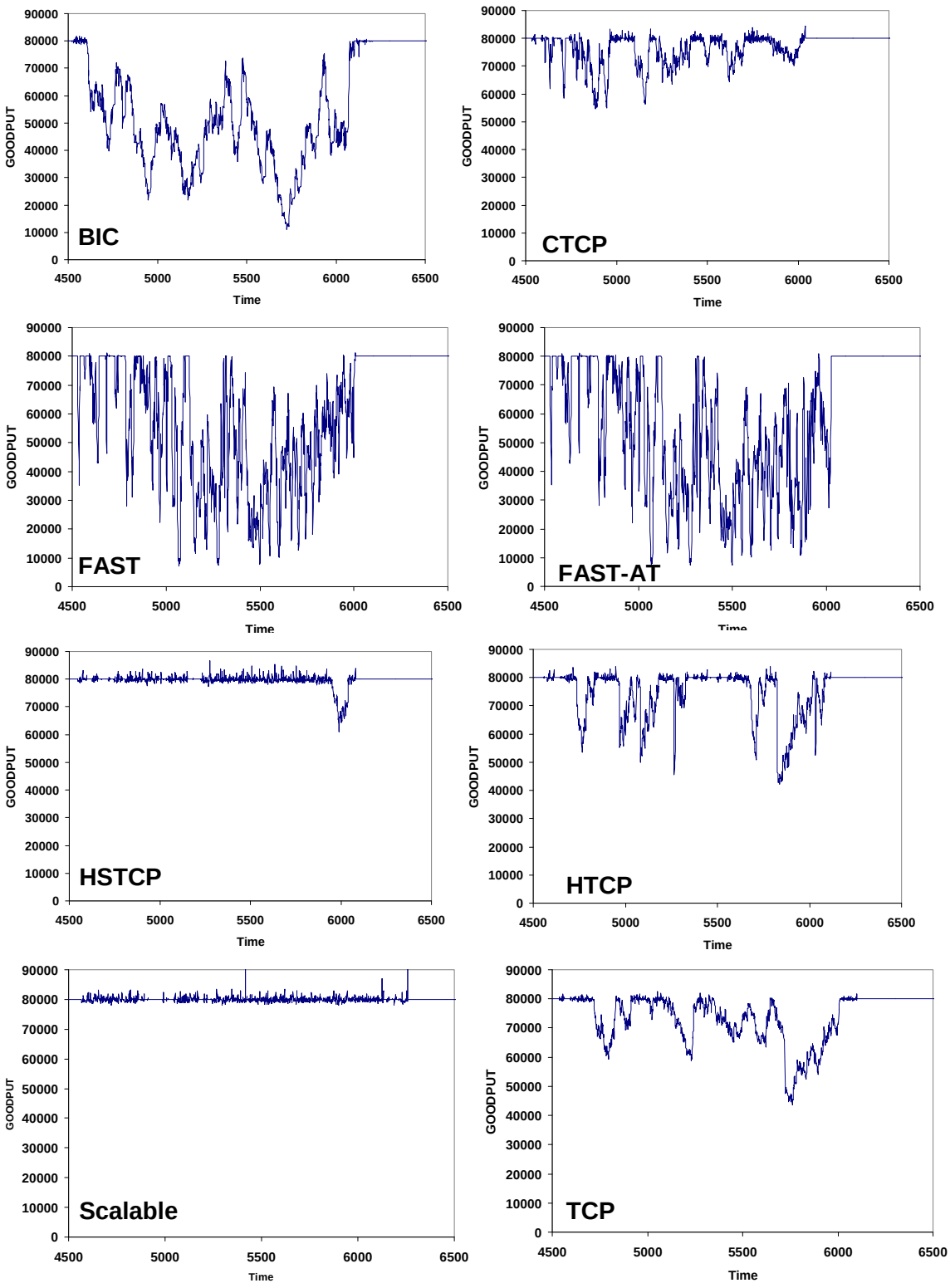


Figure 7-53. Goodput (pps) from *t*=4500 to *t*=6500 for each Congestion Control Algorithm on Long-Lived Flow L3 under Condition 8 – time is in 200 ms intervals since beginning of simulation

Fig. 7-53 shows that FAST and FAST-AT exhibit significant goodput oscillations during TP2, as also seen in Figs. 7-51 and 7-52. CTCP shows less frequent oscillations with lower amplitude. HSTCP and Scalable TCP tend to retain maximum transfer rate for most of TP2. HTCP goodput during TP2 looks very similar to goodput for TCP Reno. BIC appears to suffer two drops and recoveries in goodput during TP2.

Table 7-16 shows the average goodput (divided by 1000) under each congestion control algorithm for each long-lived flow during TP2 given condition 8. Under this condition and time period, BIC actually underperforms TCP Reno on all long-lived flows. FAST and FAST-AT provide no better goodput than TCP Reno, except in the case of the long-distance flow **L1**. We cannot generalize from looking at a single uncongested condition, so we consider information from three additional uncongested conditions: 14, 28 and 32. In this case, we limit our detailed analysis to consider only the long-distance, long-lived flow **L1**. Perhaps this additional analysis will suggest some patterns.

Table 7-16. Average Goodput (pps/1000) for Each Congestion Control Algorithm on Three Long-Lived Flows during TP2 under Condition 8

	Long-Lived Flow		
Algorithm	**L1**	**L2**	**L3**
BIC	**6.391**	**41.751**	**47.809**
CTCP	**34.488**	**69.332**	**75.435**
FAST	**18.068**	**64.444**	**53.162**
FAST-AT	**18.415**	**65.109**	**51.909**
HSTCP	**39.920**	**79.972**	**79.672**
HTCP	**24.794**	**79.845**	**73.935**
Scalable	**37.414**	**75.970**	**79.983**
TCP	**16.278**	**69.223**	**71.777**

Table 7-17 reports the lag time (in seconds) until each congestion control algorithm achieves maximum transfer rate on flow **L1** under three uncongested conditions. The relative ordering is the same as appeared in Table 7-14: FAST and FAST-AT reach maximum rate soonest, followed by CTCP, HTCP, Scalable TCP, BIC and HSTCP. TCP Reno does not achieve maximum transfer rate during TP1. The measured time lags suggest that FAST, FAST-AT, CTCP and HTCP can be grouped together as the set of algorithms providing superior quickness in attaining maximum transfer rate. Scalable TCP, BIC and HSTCP achieve less impressive quickness.

Table 7-18 reports the lag time until each congestion control algorithm recovers maximum transfer rate after TP2 on flow **L1** under uncongested conditions 14, 28 and 32. As expected, TCP Reno does not achieve maximum transfer rate during TP3. Surprisingly, perhaps, FAST-AT also does not recover to the maximum transfer rate.

This occurs because during TP2 FAST-AT auto-tunes the α-parameter from 200 to 20 to 8 as throughput falls on flow **L1**. Over the course of TP3 the α-parameter recovers as throughput rises but only reaches 20, which provides insufficient upward thrust on goodput.

Table 7-17. Time (seconds) until Long-Lived Flow L1 Reaches Maximum Transfer Rate in TP1 for Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**120.6**	**120.6**	**120.6**
CTCP	**35.2**	**35.2**	**35.2**
FAST	**26.2**	**26.2**	**26.2**
FAST-AT	**26.2**	**26.2**	**26.2**
HSTCP	**128.0**	**128.0**	**128.0**
HTCP	**41.0**	**41.0**	**41.0**
Scalable	**78.2**	**78.2**	**78.2**
TCP	**-----**	**-----**	**-----**

Table 7-18. Time (seconds) until Long-Lived Flow L1 Recovers Maximum Transfer Rate in TP3 for Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**228.6**	**186.8**	**172.2**
CTCP	**133.4**	**96.6**	**122.2**
FAST	**130.6**	**109.0**	**145.4**
FAST-AT	**-----**	**-----**	**-----**
HSTCP	**205.8**	**177.6**	**190.6**
HTCP	**145.2**	**174.2**	**125.0**
Scalable	**135.2**	**127.0**	**175.4**
TCP	**-----**	**-----**	**-----**

Among the other congestion control algorithms, Table 7-18 reveals that recovery time lags cannot be grouped as clearly as occurred for the initial lag in attaining maximum transfer rate. This makes sense because flow **L1** contends with great congestion during TP2. In general, Table 7-18 suggests that FAST and CTCP recover most quickly followed by Scalable TCP and HTCP. HSTCP and BIC appear to lag. Regarding the performance of FAST, the reader should remember that fewer flows operate simultaneously because under FAST flows have more difficulty connecting.

Table 7-19 reports average goodput (divided by 1000) on flow **L1** during TP2 under each of three uncongested conditions: 14, 28 and 32. Here, Scalable TCP, HSTCP and BIC tend to retain higher goodput under the contention of an increasing number of jumbo files during TP2. This behavior, also evident in the previous experiment, indicates that newly arriving flows have more difficulty obtaining a fair share of goodput under these three congestion control algorithms. CTCP and HTCP show some ability to retain goodput during TP2. FAST and FAST-AT appear to reduce goodput significantly in the face of increased congestion. TCP Reno did not reach high levels of goodput and so it is not surprising that it provides low goodput during TP2.

To better understand the measures reported in Tables 7-17, 7-18 and 7-19, we provide the related time series plots for each condition as Figs. 7-54 (condition 14), 7-55 (condition 28) and 7-56 (condition 32). These figures reveal that FAST and FAST-AT quickly reduce goodput on **L1** in reaction to congestion. Subsequently, as congestion clears (around t=6500) FAST quickly recovers maximum goodput. FAST-AT, on the other hand, recovers maximum goodput more slowly as the α-parameter is auto-tuned upward only every 200 s. The figures also show that TCP Reno achieves only about 25 % of the maximum transfer rate prior to TP2 and then resumes its linear increase in goodput as congestion clears. CTCP takes longer to reduce goodput in reaction to congestion but then recovers to the maximum transfer rate quickly after congestion begins to clear. HTCP shows a pattern similar to CTCP. BIC reduces goodput on flow **L1** slowly over a period of 200 s and then recovers over a period of about 100 s after congestion starts to clear. HSTCP shows a pattern similar to BIC. Scalable TCP loses goodput slowly on flow **L1** over 200 s but the minimum goodput stays higher (around 20×10^3 pps) than is the case for the other congestion control algorithms. Once congestion begins to clear, Scalable TCP recovers maximum goodput somewhat quickly (within about 80 s).

To conclude our analysis of the various congestion control algorithms performing on long-lived flows, we consider flow **L1** under the most congested condition (21). We examine performance over all three time periods. We expect to learn how the congestion control algorithms react during TP1 where they face more intense competition than was the case under less congested conditions (e.g., conditions 8, 14, 28 and 32). Under TP2 and TP3 we expect the congestion control algorithms to perform similarly because the congestion arising from jumbo files in TP2 is unlikely to clear during TP3. Table 7-20 reports the average goodput (divided by 1000) on flow **L1** for each congestion control algorithm in each of the three time periods. As expected, all algorithms provide very little goodput during TP3. The minor differences in goodput during TP2 appear due to variations in the rate at which the algorithms shed goodput in the face of intensifying congestion. These issues have been examined in earlier paragraphs. Here, we focus on TP1. To augment Table 7-20 we provide Fig. 7-57, which plots time series for goodput over all three time periods for each congestion control algorithm.

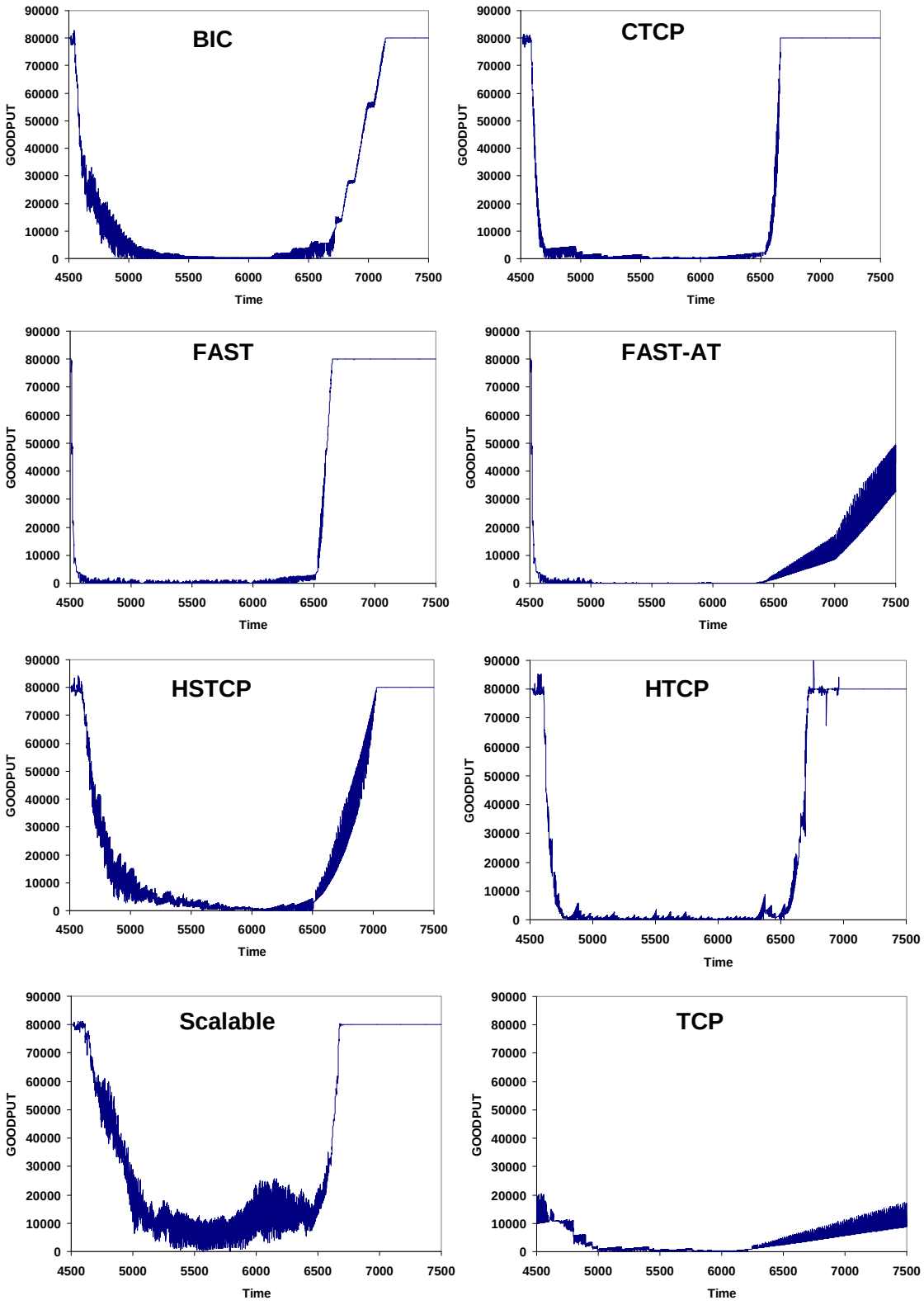


Figure 7-54. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 14 – time is in 200 ms intervals since beginning of simulation

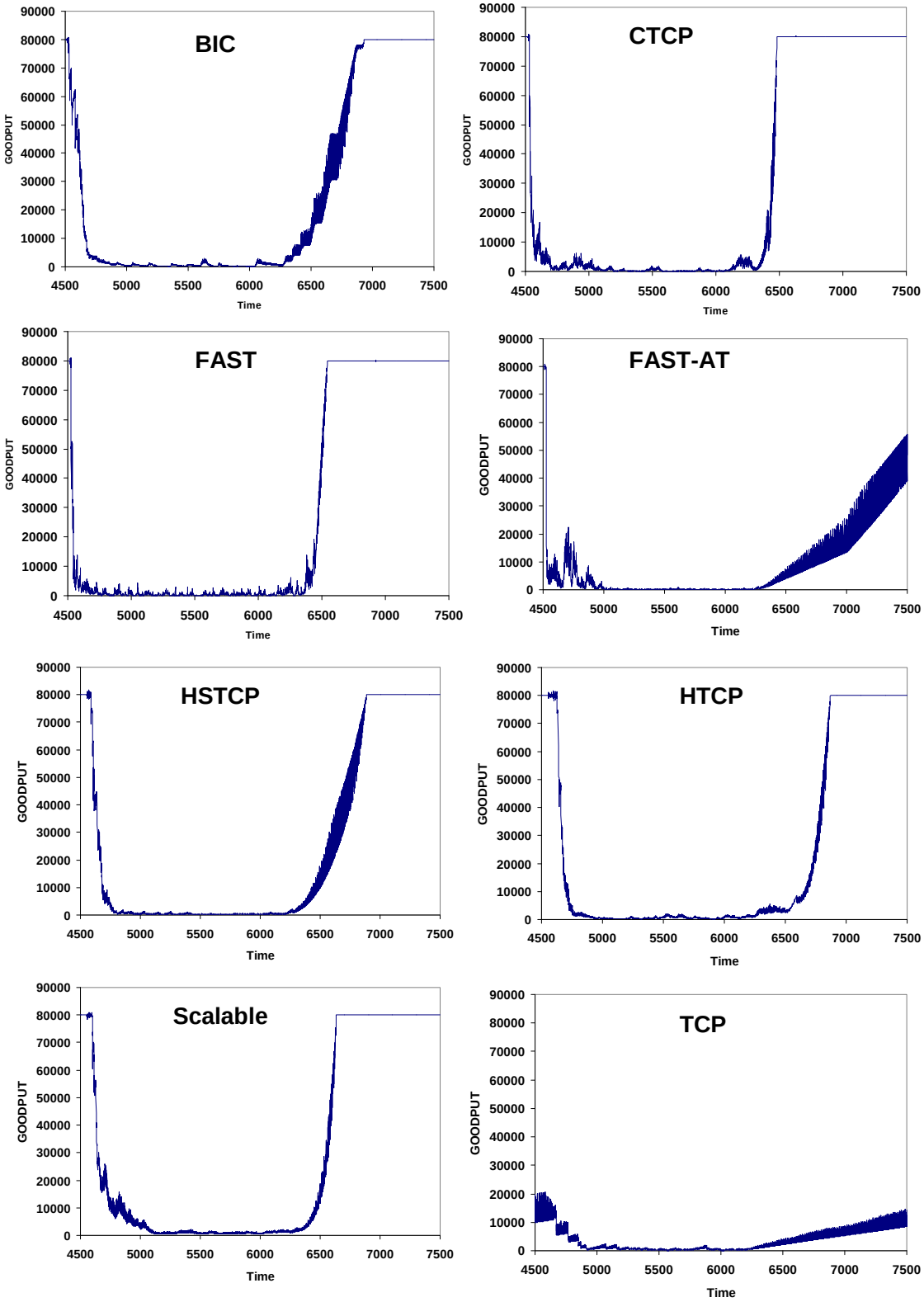


Figure 7-55. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 28 – time is in 200 ms intervals since beginning of simulation

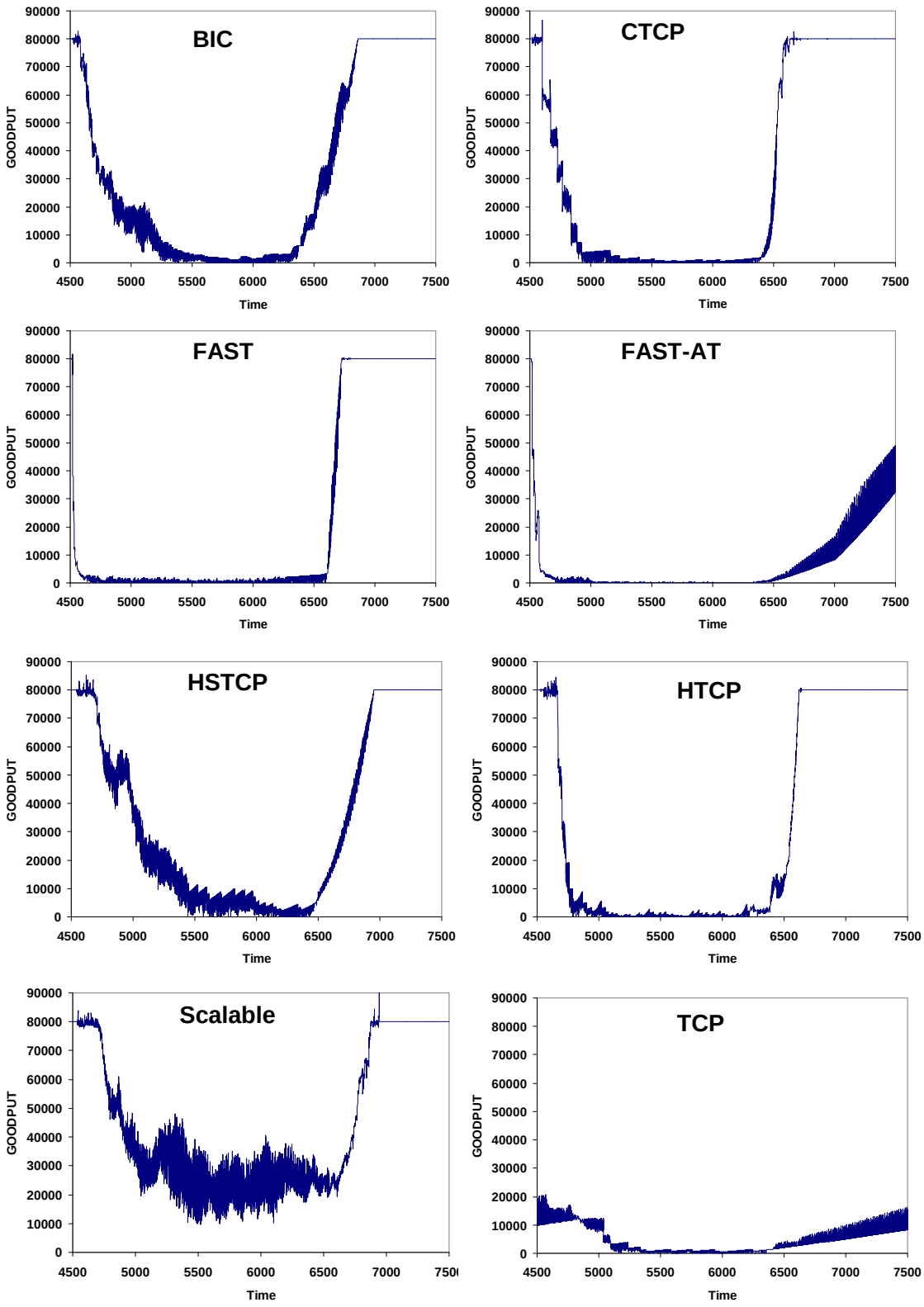


Figure 7-56. Goodput (pps) from *t*=4500 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 32 – time is in 200 ms intervals since beginning of simulation

Table 7-19. Average Goodput (pps/1000) on Long-Lived Flow L1 in Time Period 2 for Each Congestion Control Algorithm under Each of Three Uncongested Conditions

	Condition		
Algorithm	**C14**	**C28**	**C32**
BIC	**9.533**	**6.391**	**16.951**
CTCP	**7.511**	**3.435**	**13.000**
FAST	**1.709**	**2.701**	**2.272**
FAST-AT	**1.505**	**2.859**	**2.690**
HSTCP	**15.114**	**7.792**	**29.338**
HTCP	**8.635**	**9.339**	**11.943**
Scalable	**24.756**	**10.085**	**38.446**
TCP	**3.160**	**2.737**	**5.333**

Table 7-20. Average Goodput (pps/1000) on Long-Lived Flow L1 for Each Congestion Control Algorithm in Each of the Three Time Periods under Most Congested Condition 21

	Time Period		
Algorithm	**TP1**	**TP2**	**TP3**
BIC	**50.154**	**0.909**	**0.116**
CTCP	**35.950**	**0.353**	**0.130**
FAST	**39.740**	**0.398**	**0.117**
FAST-AT	**40.368**	**0.375**	**0.086**
HSTCP	**50.880**	**0.540**	**0.117**
HTCP	**47.097**	**0.759**	**0.110**
Scalable	**55.591**	**1.061**	**0.117**
TCP	**25.106**	**0.546**	**0.112**

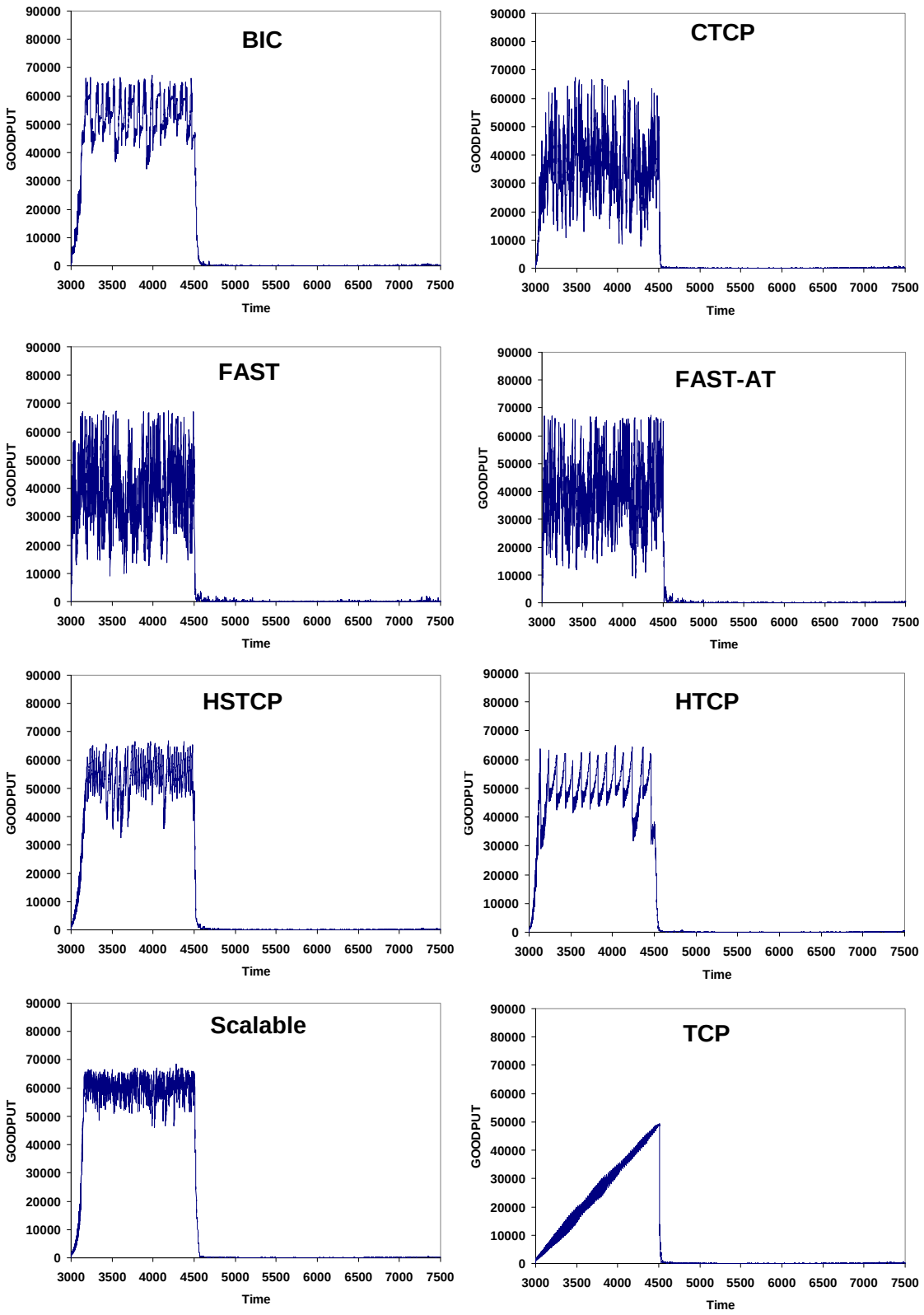


Figure 7-57. Goodput (pps) from *t*=3000 to *t*=7500 for each Congestion Control Algorithm on Long-Lived Flow L1 under Condition 21 – time is in 200 ms intervals since beginning of simulation

Given heavy congestion from competing Web traffic and document downloads during TP1 under condition 21, the ability of Scalable TCP, HSTCP and BIC to resist losing goodput allows then to achieve higher average goodput. As show in Fig. 7-57, Scalable TCP keeps goodput within the range of about 50×10^3 to 65×10^3 pps, HSTCP within the range of about 45×10^3 to 60×10^3 pps and BIC within the range of 40×10^3 to 65×10^3 pps. The higher, narrower goodput range of Scalable TCP accounts for higher average goodput. Fig. 7-57 shows that HTCP, with 4th highest average goodput, allows the range to vary between about 30×10^3 and 60×10^3 pps. CTCP, FAST and FAST-AT oscillate more frequently and with larger variation, ranging between about 15×10^3 and 65×10^3 pps during TP1. TCP Reno linearly increases over TP1 from about 5×10^3 pps at t=3000 to about 50×10^3 pps at t=4500.

Clearly, under many conditions and time periods for long-lived flows, and in the absence of a large initial slow-start threshold, the alternate congestion control algorithms provide improved goodput over TCP Reno. An exception to this occurs during TP2 when most of the algorithms cannot provide much goodput. Even in such cases, selected congestion control algorithms (Scalable TCP, HSTCP and BIC) tend to retain higher goodputs a bit longer than others. When ramping up toward maximum goodput under light to moderate Web traffic and document downloads, FAST, FAST-AT, CTCP and HTCP show some advantage in quickness over the other algorithms. When competing for goodput against heavy Web traffic and document downloads, Scalable TCP, HSTCP and BIC show some advantage in retaining higher goodput. When recovering from periods of intense jumbo file transfers, CTCP, FAST and HTCP show some advantage in recovering maximum goodput under uncongested conditions. These differences among the congestion control algorithms appear more readily under longer propagation delays, with differences shrinking along with falling propagation time.

What might be the practical implications of these findings? First, for alternate congestion control algorithms to gain a significant advantage in goodput over standard TCP, the file to be transferred must be large, the initial slow-start threshold must be low (relative to the file size), the source and receiver must both have high-speed network connections, the propagation delay must be long and the network path must be fast enough and uncongested enough to support a stream of traffic at the rate of the high-speed network connections between the source and receiver. This combination of conditions is likely to prove relatively rare within an operating network, as was the case for our simulated network. Of course, this combination of conditions is typically established (artificially) in support of attempts to show how fast a file can be transferred across a network path using a particular transport protocol. Second, any advantage for the alternate congestion control algorithms would likely be enhanced should the path show occasional packet losses due to noise (i.e., bit-error rate) or hardware malfunctions. This follows because some of the alternate congestion control algorithms (e.g., FAST, FAST-AT, CTCP and HTCP) recover more quickly from sporadic packet losses, while others (e.g., Scalable TCP, HSTCP and BIC) exhibit smaller rate reduction on a sporadic loss. Third, regular patterns of packet losses due to high congestion would limit any advantage of the alternate congestion control algorithms, excepting that Scalable TCP, HSTCP and BIC tend to retain goodput a bit longer than other algorithms in the face of congestion.

7.5.4 Finding #4

As in the previous experiment (see Sec. 6.5.3), CTCP (algorithm 2) can drive congestion window size to substantially higher values than the other congestion control algorithms we simulated. This behavior arose during TP3, as shown in Fig. 7-36, which analyzes average congestion window size. Detailed examination of the relevant time series revealed that this increase in congestion window size can be attributed solely to DD flows. The reason this occurs is the same as explained in Sec. 6.5.3. CTCP increases the delay window exponentially when no congestion had been detected and the actual congestion window is within 30 packets of the expected congestion window. This set of conditions can occur on DD flows as congestion eases at the onset of TP3.

Recall that during TP2 jumbo file transfers were initiated on DD flows, which introduced substantial congestion in directly connected access routers. At the onset of TP3 no further jumbo transfers are initiated and congestion eases as residual jumbo transfers complete. During this easing period, the congestion window on DD flows can increase – the rate of increase depends upon the level of congestion created during TP2. For example, Fig. 7-58 plots, for seven congestion control algorithms, the increase in average congestion window for DD flows during TP3 under condition 8 (most uncongested).

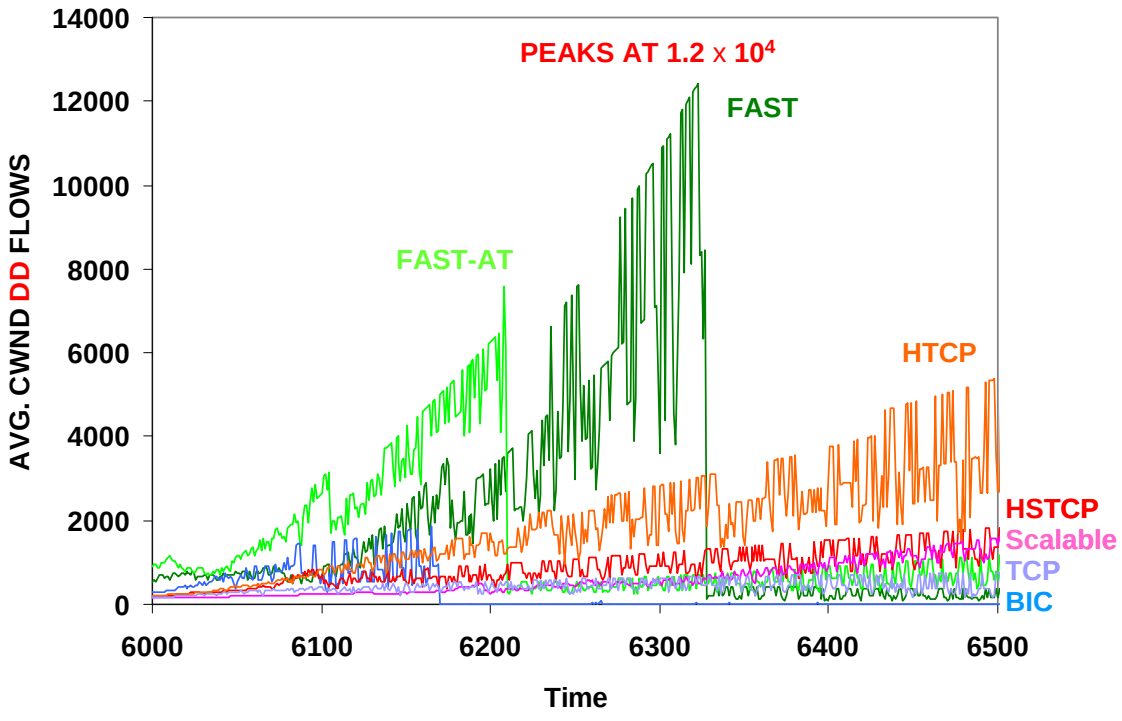


Figure 7-58. Average Congestion Window Size (packets) of DD Flows during TP3 under Condition 8 for BIC, FAST, FAST-AT, HSTCP, HTCP, Scalable TCP and TCP Reno – time is in 200 ms intervals since the beginning of the simulation

Fig. 7-58 shows that four (BIC, HSTCP, Scalable and TCP Reno) of the congestion control algorithms provide a linear increase (with a small slope) in average congestion window size, up to a maximum of about 10^3 packets. HTCP provides a similar

linear increase but with a higher slope and, thus, reaches a maximum congestion window size of about 4 x 10^3 packets. The increases for FAST-AT and FAST (which also appear approximately linear but with large slopes) peak at around 6 x 10^3 and 12 x 10^3 packets, respectively. The situation for CTCP is much different, as shown in Fig. 7-59, where under the same conditions the average congestion window size increases exponentially, reaching a peak of about 170 x 10^3 packets.

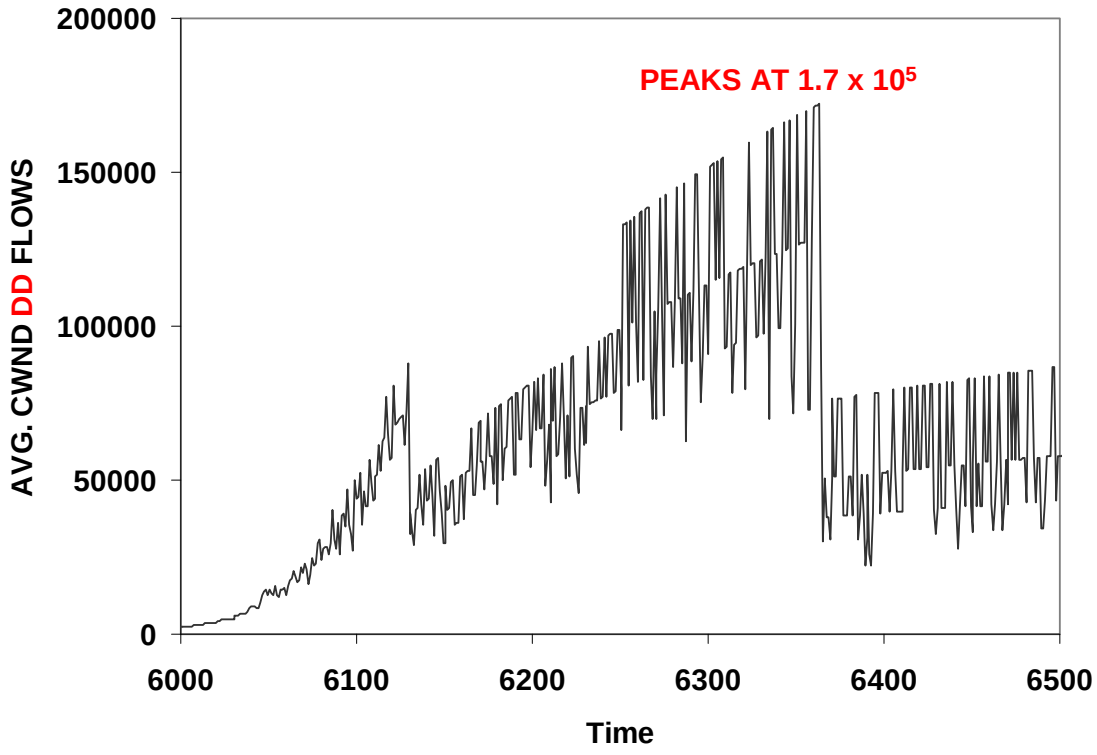


Figure 7-59. Average Congestion Window Size (packets) of DD Flows during TP3 under Condition 8 for CTCP – time is in 200 ms intervals since beginning of simulation

Overall, the congestion window size on DD flows during TP3 under condition 8 is lower for all congestion control algorithms as compared with results for the most uncongested condition (condition 12) in the previous experiment (see Figs. 6-67 and 6-68). The lower congestion window size is attributable to the order of magnitude lower network speeds used in the current experiment. While lower network speed bounds the ability of CTCP to increase the congestion window on DD flows under uncongested conditions during TP3, the general behavioral difference, first reported in Sec. 6.5.3, between CTCP and the other algorithms remains discernible.

7.5.5 Tendencies

While not consistently statistically significant across all conditions, buffer monitoring in the six directly connected access routers (B0a, C0a, E0a, F0a, I0a and K0a) revealed that Scalable TCP, BIC and HSTCP tend toward higher buffer utilizations than the other congestion control algorithms. For example, see Figs. 7-18, 7-30 and 7-40. This finding

appears consistent (at least with respect to Scalable TCP and BIC) with our measurements of buffer utilization when simulating the congestion control algorithms in a dumbbell topology (recall Sec. 5.4). This finding may also relate to tendencies of Scalable TCP to inject more packets into the network than other algorithms, to hold higher goodput on long-lived flows after the onset of jumbo file transfers in TP2 and to share goodput unfairly among competing flows.

7.6 Conclusions

In this section we described an experiment comparing alternate congestion control algorithms deployed in a scaled-down network with about an order of magnitude fewer sources and lower network speed than used in our previous experiment (described in Chapter 6). In addition, we reduced the initial slow-start threshold to a relatively low value and we added a congestion control regime: FAST with α-tuning enabled. We subjected each of eight algorithms to the same 32 conditions, which covered a range of congestion levels.

We demonstrated that FAST and FAST-AT exhibit similar influence on macroscopic network behavior and we showed that enabling α-tuning caused FAST-AT to recover less quickly than FAST when congestion eases after periods of increased contention. We also showed that, under the scenario and conditions of this experiment, the congestion control algorithms (aside from FAST and FAST-AT) exhibited indistinguishable macroscopic behavior and modest differences in experience for typical users. We showed that FAST and FAST-AT can exhibit distinctive, undesirable network-wide behavior, which grows more distinctive under increasing congestion. We also confirmed that the CTCP delay-window adjustment algorithm can lead to an exponential increase in congestion window size under particular circumstances associated with easing congestion. We identified some tendencies for Scalable TCP, BIC and HSTCP to utilize more buffers.

We were able to show that under specific, constrained circumstances the alternate congestion control algorithms can provide higher goodput than TCP Reno. For alternate congestion control algorithms to gain a significant advantage in goodput over the standard TCP algorithm, the file to be transferred must be large, the initial slow-start threshold must be low (relative to the file size), the source and receiver must both have high-speed network connections, the propagation delay must be long and the network path must be fast enough and uncongested enough to support a stream of traffic at the rate of the high-speed network connections between the source and receiver. The advantage of alternate congestion control algorithms may be expected to increase in the presence of sporadic losses. We were also able to identify some specific circumstances where particular alternate congestion control algorithms performed similarly. Under a low initial slow-start threshold and competing with typical Web traffic, FAST, FAST-AT, CTCP and HTCP tended to attain maximum transfer rate more quickly on long-lived flows than other algorithms. Under heavy congestion, Scalable TCP, BIC and HSTCP tended to retain higher goodput for a longer time on long-lived flows. Under easing congestion for long-lived flows, FAST and CTCP tended to recover maximum transfer rate more quickly than other algorithms. Overall, for long-lived flows, Scalable TCP tended to provide highest goodput but at the cost of some unfairness with respect to competing flows.

In the next two chapters, we shift our assumptions to explore alternate congestion control algorithms with richer traffic classes in a network under relatively low congestion. We simulate a network with a mix of sources, some operating under standard TCP congestion control procedures and some operating under an alternate congestion control algorithm. We consider conditions where most of the network uses standard TCP as well as conditions where most of the network uses an alternate algorithm. We also extend our traffic classes beyond Web browsing to include some proportion of downloading for larger files, such as software service packs and movies. We simulate a full hour of network operation under 32 different conditions and then compare macroscopic network behavior among the algorithms. We also investigate relative goodputs experienced by comparable flows. In Chapter 8 we examine a scaled-down network in two different cases: (1) with a high initial slow-start threshold and (2) with a low initial slow-start threshold. In Chapter 9 we examine a large, fast network with a high initial slow-start threshold. In these experiments, we aim to understand whether alternate congestion control algorithms might prove beneficial for flows with specific characteristics. We also aim to determine if particular congestion control algorithms might have deleterious effects on competing flows using standard TCP.

Chapter 8 – Comparing Congestion Control Regimes in a Heterogeneous Network

8 Comparing Congestion Control Regimes in a Heterogeneous Network

In this chapter, we investigate effects on macroscopic behavior and user experience when deploying various congestion control algorithms in a simulated, heterogeneous network, i.e., a network that includes flows operating under normal TCP congestion control procedures together with flows operating under one of seven proposed alternate congestion control algorithms, as identified in Table 8-1. Mixing alternate congestion control regimes together with standard TCP will enable us to investigate the influence of alternate congestion avoidance algorithms on the performance of TCP flows. We also introduce additional flow sizes to represent downloading movies and software updates (e.g., service packs). These file sizes augment the Web objects and document downloads used in previous experiments (Chapters 6 and 7). Here, we adopt a small-scale network, similar to that used in Chapter 7, because earlier experiments suggested that a small-scale network yields significant information while requiring fewer resources. Reducing computational cost allows us to repeat our experiments first with a large initial slow-start threshold and then with a small initial slow-start threshold. We take this step in light of the apparent significance of the initial slow-start threshold, as uncovered in earlier experiments.

Table 8-1. Alternate Congestion Control Regimes Compared

Identifier	Label	Name of Congestion Avoidance Algorithm
1	BIC	Binary Increase Congestion Control
2	CTCP	Compound Transmission Control Protocol
3	FAST	Fast Active-Queue Management Scalable Transmission Control Protocol
4	FAST-AT	FAST with α-tuning Enabled
5	HSTCP	High-Speed Transmission Control Protocol
6	HTCP	Hamilton Transmission Control Protocol
7	Scalable	Scalable Transmission Control Protocol

We exposed our simulated network to a range of congestion conditions, but we reduced overall congestion by an order of magnitude from previous experiments. We made this reduction in order to investigate behavior of the alternate congestion control algorithms under little to modest congestion, which should reveal any differences in user experience when large files are sent over fast paths between sources and receivers with high-speed network interfaces. In fact, we classified flows into groups based on four dimensions: (1) congestion control algorithm used; (2) characteristics of the network path transited; (3) minimum interface speed of the source and receiver pair; and (4) size of the transferred file. Such classification enabled us to compare relative performance among congestion control algorithms for specific flow groups. We collected and compared data representing the distribution of goodput for users of flows in each flow group.

We organize what follows into six sections. Sec. 8.1 describes the experiment design, including robustness factors, fixed factors, conditions simulated and responses measured. In describing the design, we explain how we controlled the generation of flows in each group. Sec. 8.1 also gives the domain view of the simulated conditions. Sec. 8.2 details resource requirements for simulating the experiments and outlines how we collected and summarized experiment data. Sec. 8.3 explains the data analysis approach we used to investigate experiment responses. Sec. 8.4 presents the results from both sets of experiments, that is, with a large and a small initial slow-start threshold. Sec. 8.5 discusses key findings from the results. We conclude in Sec. 8.6.

8.1 Experiment Design

We conducted these experiments within the same fixed, heterogeneous topology (see Fig. 6-1) used in previous experiments. As discussed below, we employed nine robustness factors, which define the range over which our findings apply. We fixed the remaining model parameters and then created a design template to simulate 32 conditions. We repeated the 32 simulated conditions a second time after lowering the initial slow-start threshold, so the simulations yielded two sets of results. By mixing flows using alternate congestion control algorithms together with flows using standard TCP, we can examine the relative influence of the various alternate algorithms on normal TCP flows. Such information could be useful because the Internet is unlikely to cutover all at once to an alternate congestion control algorithm, but rather will experience a transition period during which TCP flows will coexist with flows using alternate algorithms.

8.1.1 Robustness Factors and Fixed Factors

Table 8-2 specifies the robustness factors and values we used for this experiment. Robustness factors included the most significant factors identified from our sensitivity analysis (see Chapter 4): network speed (x1), propagation delay (x2), number of sources (x9), think time (x4), file sizes (x5) and buffer sizes (x3). We introduced a new factor (x6) to control distribution of files sizes. In order to sample flows in each possible flow group, we included a factor controlling the network interface speed of sources and receivers (x7). Finally, to simulate a network in transition, we included a factor (x8) determining the proportion of sources adopting the alternate congestion control algorithm (the remainder of sources adopted standard TCP congestion control procedures).

Table 8-2. Robustness Factors Adopted for Comparing Congestion Control Mechanisms

Identifier	Definition	PLUS (+1) Value	Minus (-1) Value
x1	Network Speed	1600 packets/ms	800 packets/ms
x2	Propagation Delay Multiplier	2	1
x3	Buffer Size Scaling Factor	1	0.5
x4	Think Time	7500 ms	5000 ms
x5	Average File Size for Web Objects	150 packets	100 packets
x6	Distribution for Sizing Large Files	2	1
x7	Probability of Fast Source	.7	.3
x8	Probability of Alternate Congestion-Control Algorithm	.7	.3
x9	Multiplier on Base Number of Sources (ΔU)	3	2

The parameter values for x6 indicate which of two distributions to select for the probability of various file sizes. The distribution details are given in Table 8-3. A file that is not a document (D), service pack (SP) or movie (M) is a normal Web object (WO), so the sum of **Fp**, **Sp** and **Mp** can fall below, but must not exceed, one. The size of each Web object was drawn from a Pareto distribution with an average size of x5 and a shape parameter = 1.5. The average size for the other file types were multipliers applied to the size selected for a Web object. Table 8-4 gives the details.

Table 8-3. Probability Distributions for Files of Various Sizes (Residual Files are Web Objects)

Parameter	Definition	if x6 = 2	if x6 = 1
Fp	Probability file is a Document	4×10^{-2}	2×10^{-2}
Sp	Probability file is a Service Pack	4×10^{-3}	2×10^{-3}
Mp	Probability file is a Movie	4×10^{-4}	4×10^{-4}

Table 8-4. Fixed Parameters for Sizing Files

Parameter	Definition	Value
τ	Shape parameter for Pareto distribution of file sizes	1.5
Fx	Average Document size = x5 x Fx packets	10
Sx	Average Service Pack size = x5 x Sx packets	10^3
Mx	Average Movie size = x5 x Mx packets	10^4

The probabilities shown in Table 8-3 were used to determine the size of files sent on flows, subject to constraints (explained below) intended to ensure a minimum and maximum number of flows were active simultaneously in the network for each flow group. Table 8-5 shows the dimensions used to classify flow groups.

Table 8-5. Four Dimensions Defining Flow Groups

Path Class	Interface Speed (min.)	File Type	Control Algorithm
VERY FAST	FAST	Document	BIC
FAST	NORMAL	Movie	CTCP
TYPICAL		Service Pack	FAST
		Web Object	FAST-AT
			HSTCP
			HTCP
			Scalable
			TCP Reno

One dimension of a flow group concerns path class, as described earlier in Table 6-2. A given network flow may traverse a path between a pair of (so-called D-class) access routers directly connected to backbone routers, which would yield a very fast (VF) path. Other flows may transit combinations of D-class routers and fast (so-called F-class)

access routers, which yield fast (F) paths. Any flows traversing at least one normal (so-called N-class) access router would travel on a typical (T) path. A second dimension of a flow group considers the speed with which a source-receiver pair connects to the network. A flow can operate no faster than the minimum speed of the source and receiver, which may connect at a normal speed (e.g., 100 Mbps) or fast speed (e.g., 1 Gbps). If both source and receiver have fast network connections, then the interface speed is fast (F); otherwise, the interface speed is normal (N). A third dimension of a flow group is file type, which denotes file size. Flows with smaller files (e.g., Web objects) usually achieve lower goodputs because a larger portion of the flow lifetime is spent establishing the maximum transfer rate. In fact, sufficiently short files may end before a flow even reaches the maximum achievable transfer rate on a path. The fourth dimension of a flow group identifies the congestion control algorithm used by the source that originates the flow. Since each simulation had a mix of TCP sources and alternate sources, the fourth dimension in a given experiment execution took on two values: TCP Reno and one of the remaining congestion control algorithms. Flows, originated by TCP Reno sources and alternate sources, fell into one of 24 flow groups, depending on the values for the remaining three dimensions: path class, interface speed and file type. Table 8-6 identifies these 24 flow groups.

Table 8-6. Flow Group Identifiers Assigned Based on Three-Dimensional Classification

Identifier	Path Class	Interface Speed	File Type
1	VERY FAST	FAST	Movie
2	VERY FAST	NORMAL	Movie
3	FAST	FAST	Movie
4	FAST	NORMAL	Movie
5	TYPICAL	FAST	Movie
6	TYPICAL	NORMAL	Movie
7	VERY FAST	FAST	Service Pack
8	VERY FAST	NORMAL	Service Pack
9	FAST	FAST	Service Pack
10	FAST	NORMAL	Service Pack
11	TYPICAL	FAST	Service Pack
12	TYPICAL	NORMAL	Service Pack
13	VERY FAST	FAST	Document
14	VERY FAST	NORMAL	Document
15	FAST	FAST	Document
16	FAST	NORMAL	Document
17	TYPICAL	FAST	Document
18	TYPICAL	NORMAL	Document
19	VERY FAST	FAST	Web Object
20	VERY FAST	NORMAL	Web Object
21	FAST	FAST	Web Object
22	FAST	NORMAL	Web Object
23	TYPICAL	FAST	Web Object
24	TYPICAL	NORMAL	Web Object

8.1.1.1 Constraints on Flows of Large Size. Applying probabilities associated with factor x6 (distribution for sizing larger files) could lead to two undesirable consequences: too few samples on very fast paths and too many samples on typical paths. If the probabilities of very large files, e.g., movies and service packs, were sufficiently small, then a given experiment may generate few or no large files for some rarer combinations of flow traits, e.g., flows with fast interface speeds traveling over very fast paths. On the other hand, the probabilities of very large files may also cause a simulated network to be swamped with many large files that take much time to transfer on flows with normal interface speeds traversing typical paths. In such cases, large files flowing over slow paths can accumulate in the network because each of the file transfers takes a long time to complete and the more such flows in the network, the longer each takes to complete.[1]

The problem of too few samples might be addressed by simulating longer network evolution, but the processing cost for the additional simulated time could prove prohibitive. The problem of too many samples cannot be solved by simulating longer network evolution; in fact, simulating longer evolution would increase accumulation of large files being transferred on flows transiting slow paths. For these reasons, we decided to place constraints on the generation of file types with large sizes. The aim of these constraints was to ensure a sufficient number of flow samples in each flow group, while not overwhelming the network with flows that accumulate in any particular group.

In short, using factor x6 we computed a target maximum number of active flows for each file type, other than Web objects, i.e., for movies, service packs and documents. Based on relevant factors (x7 and x8) we also computed a target minimum number of active flows for each type. During simulation, each originating flow was assigned a preliminary file type of Web object. A file size was drawn from a Pareto distribution with a specified average (x5) and shape (τ). A check was then made to see if the minimum number of movies was active on flows with matching path class, interface speed and congestion control algorithm. If not, then the flow was assigned a file type of movie and the file size was increased by the appropriate multiplier taken from Table 8-4; otherwise, a similar check was made for service pack and then, if necessary, document. If the minimum number of flows was active in all three possible flow groups (designated by a specific path class, interface speed and congestion control algorithm in combination with one of the larger file types), then a file type was selected based on the specified probability distribution (x6). If the target maximum number of flows was already active for the selected file type, then the flow remained a Web object; otherwise, the flow size was increased by the appropriate multiplier.

Computing the target maximum number of active flows for specific file types is straightforward. For example, given the total number (s) of sources in a simulation we computed the target number of active document flows as follows.

$$sDC_{MAX} \equiv \mathbf{max}\left(\mathbf{ceil}\left(s \times \mathbf{Fp}\right), \mathbf{1000}\right) \quad (1)$$

[1] In a real network the problem of too many large flows over specific paths could be ameliorated by users aborting flows observed to be running too slowly or taking too long. This would not be true for unattended flows, such as appear in typical peer-to-peer applications. The simulations used in these experiments include only unattended flows, so one cannot rely on users to abort slow flows. Note that MesoNet does include the possibility of user-attended flows in addition to unattended flows.

Here, **Fp** is taken from x6 (and related Table 8-3) and 1000 is a selected minimum for the maximum number of active document transfers desired in the simulation. Ensuring a minimum maximum enables accumulation of sufficient samples when the specified probability of document transfers is low. Similar computations can be made for movies (2) and service packs (3). Note that since these file types are larger than documents, smaller minimum maximums were chosen to prevent very large files from accumulating in the network.

$$sMV_{MAX} \equiv \mathbf{max}\left(\mathbf{ceil}\left(s \times \mathbf{Mp}\right), \mathbf{10}\right) \tag{2}$$

$$sSP_{MAX} \equiv \mathbf{max}\left(\mathbf{ceil}\left(s \times \mathbf{Sp}\right), \mathbf{100}\right) \tag{3}$$

Computing the minimum number of active flows in each flow group is somewhat more complicated. We began by selecting a target minimum for flows of each file type. We specified the target minimum as a percentage (10 % here) of the target maximum. In order to obtain sufficient samples in each flow group, we allocated the target minimum across flows based on path class, interface speed and congestion control algorithm. Table 8-7 illustrates how the target minimums were computed for document flow groups.

Table 8-7. Computing Target Minimums for Document Transfers with Combinations of Flow Traits

Path Class	Interface Speed	Control Algorithm	Minimum Number of Documents Being Sent Per Flow Group
VERY FAST	FAST	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DDflow}) \cdot x7 \cdot (\mathbf{1} - x8)\right]$
VERY FAST	NORMAL	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DDflow}) \cdot (\mathbf{1} - x7) \cdot (\mathbf{1} - x8)\right]$
VERY FAST	FAST	Alternate	$\mathbf{ceil}\left(sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DDflow}) \cdot x7 \cdot x8\right)$
VERY FAST	NORMAL	Alternate	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DDflow}) \cdot (\mathbf{1} - x7) \cdot x8\right]$
FAST	FAST	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DFflow} \vee \mathbf{FFflow}) \cdot x7 \cdot (\mathbf{1} - x8)\right]$
FAST	NORMAL	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DFflow} \vee \mathbf{FFflow}) \cdot (\mathbf{1} - x7) \cdot (\mathbf{1} - x8)\right]$
FAST	FAST	Alternate	$\mathbf{ceil}\left(sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DFflow} \vee \mathbf{FFflow}) \cdot x7 \cdot x8\right)$
FAST	NORMAL	Alternate	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DFflow} \vee \mathbf{FFflow}) \cdot (\mathbf{1} - x7) \cdot x8\right]$
TYPICAL	FAST	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DNflow} \vee \mathbf{FNflow} \vee \mathbf{NNflow}) \cdot x7 \cdot (\mathbf{1} - x8)\right]$
TYPICAL	NORMAL	TCP Reno	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DNflow} \vee \mathbf{FNflow} \vee \mathbf{NNflow}) \cdot (\mathbf{1} - x7) \cdot (\mathbf{1} - x8)\right]$
TYPICAL	FAST	Alternate	$\mathbf{ceil}\left(sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DNflow} \vee \mathbf{FNflow} \vee \mathbf{NNflow}) \cdot x7 \cdot x8\right)$
TYPICAL	NORMAL	Alternate	$\mathbf{ceil}\left[sDC_{MAX} \cdot \mathbf{0.1} \cdot \mathbf{Prob}(\mathbf{DNflow} \vee \mathbf{FNflow} \vee \mathbf{NNflow}) \cdot (\mathbf{1} - x7) \cdot x8\right]$

The computations in each row of Table 8-7 have a similar pattern. The target minimum number of active document transfers is 10 % of the target maximum (sDC_{MAX}),

but multiplied by: (a) the probability that a flow transits a given path class[2], e.g., Prob(DDflow), (b) the probability a flow connects with a particular interface speed (x7 or 1-x7) and (c) the probability a flow uses a specified congestion control algorithm (x8 or 1-x8). Similar computations can be made for movies and service packs.

In cases where the probability of a specific file type is very small, the **ceil** function on the calculations in Table 8-7 (coupled with the target maximum) ensures that the minimum number of active flows for any flow group cannot go below one. In this way, samples can always be collected for each flow group as long as the probability assigned to each file type does not equal zero.

8.1.1.2 Fixed Experiment Factors. We specified fixed values for model input parameters that were not chosen as robustness factors[3]. Table 8-8 shows the values specified for fixed network parameters. Most of these parameters remain the same as in previous experiments. The fixed network parameters defined speeds for POP routers and various access routers relative to the speed of backbone routers and also determined the speed (in packets per millisecond) for basic and fast sources and receivers. One change from previous experiments involves the buffer sizing algorithm. In the current experiment, buffers are sized using only the conventional computation (*RTT* x *C*). Variations in buffer sizes were controlled by factor x3, which specified a multiplier used to retain (x3 = 1) or halve (x3 = 0.5) the computed buffer size.

Table 8-8. Fixed Network Parameters

Parameter	Definition	Value
BBspeedup	Backbone router speed = x1xBBspeedup	2
R2	POP routers speed = x1/R2	4
R3	Access routers speed = x1/R2/R3	10
Bdirect	Directly connected access router speed = x1/R2/R3xBdirect	10
Bfast	Fast access router speed = x1/R2/R3xBfast	2
Hbase	Speed of basic sources (packets/ms)	8
Hfast	Speed of fast sources (packets/ms)	80
QszAlg	Algorithm to size buffers (in packets)	*RTT* x *C*

Table 8-9 gives fixed values assigned to parameters influencing the number and distribution of sources and receivers. The basic unit of sources allocated under routers is 100 (implying a base unit of 400 for receivers), which corresponds to our decision to simulate a small network. The base unit of sources (and receivers) is multiplied by the value for factor x9 to determine the actual number of base units for a given simulated condition. The next six parameters in Table 8-9 controlled placement of sources and receivers under specific access routers throughout the simulated topology. The probabilities listed were chosen to stimulate flow patterns consistent with a Web-centric network. Specifically, the probabilities for placing sources and receivers led to the

[2] A method for computing such probabilities was explained in Sec. 3.2.4.
[3] Recall that the values of robustness factors establish the range of variation over which any experiment conclusions can be said to hold.

distribution[4] shown in Table 8-10, where most sources were placed under fast access routers and a preponderance of receivers were placed under normal access routers. This led to a distribution of flows across flow classes with the approximate probabilities listed in Table 8-11. About 94 % of flows transited at least one N-class access router, with those flows partitioned as follows: 55 % transited FN paths, 32 % crossed NN paths and 7 % traversed DN paths.

Table 8-9. Fixed Source and Receiver Parameters

Parameter	Definition	Value
Bsources	Basic unit for sources per access router	100
P(Ns)	Probability source under normal access router	0.1
P(Nsf)	Probability source under fast access router	0.6
P(Nsd)	Probability source under directly connected access router	0.3
P(Nr)	Probability receiver under normal access router	0.6
P(Nrf)	Probability receiver under fast access router	0.2
P(Nrd)	Probability receiver under directly connected access router	0.2
sst_{INT}	Initial slow-start threshold (packets)	$2^{31}/2$ or 100

Table 8-10. Proportion of Sources and Receivers Placed under Specific Router Classes

Access Router Class	% Sources	% Receivers
Directly Connected	6	2
Fast	58	8
Normal	36	90

Table 8-11. Approximate Probability of Flows Transiting Specific Path Classes

Path Class	Flow Probability
Very Fast	1.070×10^{-3}
Fast	61.479×10^{-3}
Typical	937.451×10^{-3}

Table 8-9 also indicates the values specified for the initial slow-start threshold. In this experiment, we selected two different values: one very large and one rather modest. We ran two sets of simulations encompassing all robustness conditions, as limited by the experiment design described below in Sec. 8.1.2. For the first set of simulations we used a large initial slow-start threshold. In this case, we invoked limited slow-start where the congestion window increased exponentially up to 100 packets and then logarithmically after that. We then repeated the same simulations but with a small initial slow-start

[4] A method for computing the distribution of sources and receivers and also the probability of flows in various flow classes was explained in Sec. 3.2.4.

threshold. Repeating the simulations allowed us to assess differences among congestion control algorithms depending upon differences in initial slow-start thresholds.

The remaining fixed parameters relate to simulation control, as defined in Table 8-12. We set a simulation time step of one millisecond and chose to make measurements every 200 time steps. For each simulation run we collected 18×10^3 measurements, which equates to simulating network evolution for (18×10^3 intervals x .2 intervals/s =) 3600 s – or one hour. Differing somewhat from previous experiments, we defined individual random number streams for particular aspects of randomness within the simulation. We took this step to ensure that the experiments provided similar conditions for comparable aspects of the model when simulating different alternative congestion control algorithms. Table 8-12 gives the seeds used to initialize each random number seed. All seven seeds can be adjusted at one time by assigning a different value to parameter **RandOffset**.

Table 8-12. Fixed Simulation Control Parameters

Parameter	Definition	Value
M	Measurement Interval Size in Time Steps	200
MI	Number of Measurement Intervals Simulated	18000
MB	Number of Measurement Intervals Buffered	1500
TSD	Duration of Each Time Step in Seconds	0.001
RandOffset	Random Number Seed Offset	0
CCseed	Random Number Seed used to assign congestion-control algorithms to sources	100000
TTseed	Random Number Seed used to assign think times between flows	200000
HSseed	Random Number Seed used to assign network interface speeds to sources and receivers	300000
UPseed	Random Number Seed used to determine when a source becomes active initially	400000
WOseed	Random Number Seed used to assign basic file sizes for web objects	500000
FTseed	Random Number Seed used to assign file types (web object, document, service pack, movie)	600000
RSseed	Random Number Seed used to assign receiver for each flow started by a source	700000

8.1.2 Orthogonal Fractional Factorial Design of Robustness Conditions

Given nine robustness factors, a full factorial two-level experiment requires (2^9 =) 512 simulations. Comparing seven congestion control algorithms under 512 conditions would require (7 x 512 =) 3584 simulation runs. Repeating the experiments with a different initial slow-start threshold would double the number of simulation runs to 7168. We estimated that running all these simulations, even for a small network, would require about 150 days given the 48 processors available for our experiments. We decided to constrain our simulation cost to be no more than 10 days, which implied that we could

run only 32 conditions for each congestion control algorithm under each of two initial slow-start thresholds. This led us to select a 2^{9-4} orthogonal fractional factorial experiment design, as shown in Table 8-13. This is a resolution IV experiment design [89], which means that main effects are not confounded with each other or with any two-factor interactions, though some two-factor interactions may be confounded with each other. Given previous experiments, MesoNet simulations appear to be driven by main effects, so a resolution IV design should prove adequate for our purposes.

Table 8-13. Two-Factor 2^{9-4} Orthogonal Fractional Factorial Design Template

Factor->	x1	x2	x3	x4	x5	x6	x7	x8	x9
Condition	--	--	--	--	--	--	--	--	--
1	-1	-1	-1	-1	-1	+1	+1	+1	+1
2	+1	-1	-1	-1	-1	+1	-1	-1	-1
3	-1	+1	-1	-1	-1	-1	+1	-1	-1
4	+1	+1	-1	-1	-1	-1	-1	+1	+1
5	-1	-1	+1	-1	-1	-1	-1	+1	-1
6	+1	-1	+1	-1	-1	-1	+1	-1	+1
7	-1	+1	+1	-1	-1	+1	-1	-1	+1
8	+1	+1	+1	-1	-1	+1	+1	+1	-1
9	-1	-1	-1	+1	-1	-1	-1	-1	+1
10	+1	-1	-1	+1	-1	-1	+1	+1	-1
11	-1	+1	-1	+1	-1	+1	-1	+1	-1
12	+1	+1	-1	+1	-1	+1	+1	-1	+1
13	-1	-1	+1	+1	-1	+1	+1	-1	-1
14	+1	-1	+1	+1	-1	+1	-1	+1	+1
15	-1	+1	+1	+1	-1	-1	+1	+1	+1
16	+1	+1	+1	+1	-1	-1	-1	-1	-1
17	-1	-1	-1	-1	+1	-1	-1	-1	-1
18	+1	-1	-1	-1	+1	-1	+1	+1	+1
19	-1	+1	-1	-1	+1	+1	-1	+1	+1
20	+1	+1	-1	-1	+1	+1	+1	-1	-1
21	-1	-1	+1	-1	+1	+1	+1	-1	+1
22	+1	-1	+1	-1	+1	+1	-1	+1	-1
23	-1	+1	+1	-1	+1	-1	+1	+1	-1
24	+1	+1	+1	-1	+1	-1	-1	-1	+1
25	-1	-1	-1	+1	+1	+1	+1	1	-1
26	+1	-1	-1	+1	+1	+1	-1	-1	+1
27	-1	+1	-1	+1	+1	-1	+1	-1	+1
28	+1	+1	-1	+1	+1	-1	-1	+1	-1
29	-1	-1	+1	+1	+1	-1	-1	+1	+1
30	+1	-1	+1	+1	+1	-1	+1	-1	-1
31	-1	+1	+1	+1	+1	+1	-1	-1	-1
32	+1	+1	+1	+1	+1	+1	+1	+1	+1

To generate the experiment conditions, shown in Table 8-14, we combined the design template (from Table 8-13) with the robustness-factor values (from Tables 8-2 and 8-3). We repeated these same 32 conditions for each combination of seven alternate congestion control algorithms and two initial slow-start thresholds to yield (32 x 7 x 2 =) 448 individual simulation runs.

8.1.3 Domain View of Robustness Conditions

Changes in network speed and network size influence the domain view of our simulated network. Table 8-15 shows the simulated router speeds for this experiment, which are about an order of magnitude below speeds that might be seen in contemporary networks. Restricting **Bsources** (base number of sources) to be 100 scales the number of potentially

active flows to a level that matches the simulated network speeds. Table 8-16 shows the number of sources for each level of factor x9. The number of receivers is four times the number of sources.

Table 8-14. The 32 Simulated Conditions used to compare Each Combination of Congestion Control Algorithm and Initial-Slow Start Threshold

Factor->	x1	x2	x3	x4	x5	x6	x7	x8	x9
Condition	--	--	--	--	--	--	--	--	--
1	800	1	0.5	5000	100	0.04/0.004/0.0004	0.7	0.7	3
2	1600	1	0.5	5000	100	0.04/0.004/0.0004	0.3	0.3	2
3	800	2	0.5	5000	100	0.02/0.002/0.0002	0.7	0.3	2
4	1600	2	0.5	5000	100	0.02/0.002/0.0002	0.3	0.7	3
5	800	1	1	5000	100	0.02/0.002/0.0002	0.3	0.7	2
6	1600	1	1	5000	100	0.02/0.002/0.0002	0.7	0.3	3
7	800	2	1	5000	100	0.04/0.004/0.0004	0.3	0.3	3
8	1600	2	1	5000	100	0.04/0.004/0.0004	0.7	0.7	2
9	800	1	0.5	7500	100	0.02/0.002/0.0002	0.3	0.3	3
10	1600	1	0.5	7500	100	0.02/0.002/0.0002	0.7	0.7	2
11	800	2	0.5	7500	100	0.04/0.004/0.0004	0.3	0.7	2
12	1600	2	0.5	7500	100	0.04/0.004/0.0004	0.7	0.3	3
13	800	1	1	7500	100	0.04/0.004/0.0004	0.7	0.3	2
14	1600	1	1	7500	100	0.04/0.004/0.0004	0.3	0.7	3
15	800	2	1	7500	100	0.02/0.002/0.0002	0.7	0.7	3
16	1600	2	1	7500	100	0.02/0.002/0.0002	0.3	0.3	2
17	800	1	0.5	5000	150	0.02/0.002/0.0002	0.3	0.3	2
18	1600	1	0.5	5000	150	0.02/0.002/0.0002	0.7	0.7	3
19	800	2	0.5	5000	150	0.04/0.004/0.0004	0.3	0.7	3
20	1600	2	0.5	5000	150	0.04/0.004/0.0004	0.7	0.3	2
21	800	1	1	5000	150	0.04/0.004/0.0004	0.7	0.3	3
22	1600	1	1	5000	150	0.04/0.004/0.0004	0.3	0.7	2
23	800	2	1	5000	150	0.02/0.002/0.0002	0.7	0.7	2
24	1600	2	1	5000	150	0.02/0.002/0.0002	0.3	0.3	3
25	800	1	0.5	7500	150	0.04/0.004/0.0004	0.7	0.7	2
26	1600	1	0.5	7500	150	0.04/0.004/0.0004	0.3	0.3	3
27	800	2	0.5	7500	150	0.02/0.002/0.0002	0.7	0.3	3
28	1600	2	0.5	7500	150	0.02/0.002/0.0002	0.3	0.7	2
29	800	1	1	7500	150	0.02/0.002/0.0002	0.3	0.7	3
30	1600	1	1	7500	150	0.02/0.002/0.0002	0.7	0.3	2
31	800	2	1	7500	150	0.04/0.004/0.0004	0.3	0.3	2
32	1600	2	1	7500	150	0.04/0.004/0.0004	0.7	0.7	3

We used the same topology as in previous experiments and we simulated the same propagation delays (shown in Table 8-16). Buffer sizing was influenced by three factors: network speed (x1), propagation delay (x2) and buffer-size adjustment (x3). Table 8-17 characterizes buffer sizes for each router level under both values for factor x3.

Fig. 8-1 plots the retransmission rates for each of the 32 simulated conditions under a large initial slow-start threshold, while Fig. 8-2 plots retransmission rates under a

small threshold. In each figure, the x axis is ordered by increasing retransmission rate. Overall, the simulated conditions exhibited about two orders of magnitude reduction in congestion when compared with previous experiments: recall Figs. 6-5 and 7-1.

Table 8-15. Simulated Router Speeds

Router	PLUS (+1)	Minus (-1)
Backbone	38.4 Gbps	19.2 Gbps
POP	4.8 Gbps	2.4 Gbps
Normal Access	480 Mbps	240 Mbps
Fast Access	960 Mbps	720 Mbps
Directly Connected Access	4.8 Gbps	2.4 Gbps

Table 8-16. Number of Simulated Sources

PLUS (+1)	Minus (-1)
26.085×10^3	17.355×10^3

Table 8-17. Simulated Propagation Delays (ms)

	Min	Avg	Max
PLUS (+1)	12	81	200
Minus (-1)	6	41	100

Table 8-18. Characterization of Simulated Buffer Sizes (packets)

	x3 = 1.0			x3 = 0.5		
Router	Min	Avg	Max	Min	Avg	Max
Backbone	65.105×10^3	146.487×10^3	260.422×10^3	32.553×10^3	73.244×10^3	130.211×10^3
POP	8.138×10^3	18.311×10^3	32.553×10^3	4.096×10^3	9.155×10^3	16.276×10^3
Access	1.294×10^3	2.912×10^3	5.176×10^3	647	1.456×10^3	2.588×10^3

Using visual guidance, as shown on Figs. 8-1 and 8-2, we divided congestion conditions into six categories moving from little congestion (**C1**) to relatively high congestion (**C6**). The range of congestion conditions is similar under either large (Fig. 8-1) or small (Fig. 8-2) initial slow-start threshold. Using a high initial slow-start threshold appeared to increase overall congestion slightly, ranging from a low of 2 retransmissions per 10^4 packets to a high of about 25 per 10^3. For a small initial slow-start threshold the range goes from 4 in 10^6 to about 22 per 10^3. The number of conditions we placed in

particular categories varies slightly between the two figures. In addition, the order of the conditions varies somewhat between the two figures. Eight conditions changed categories when moving from a large to a small initial slow-start threshold. Seven of those conditions moved to a less congested category. Overall, however, the relative congestion generated by the same condition under either of the two initial slow-start thresholds appears similar.

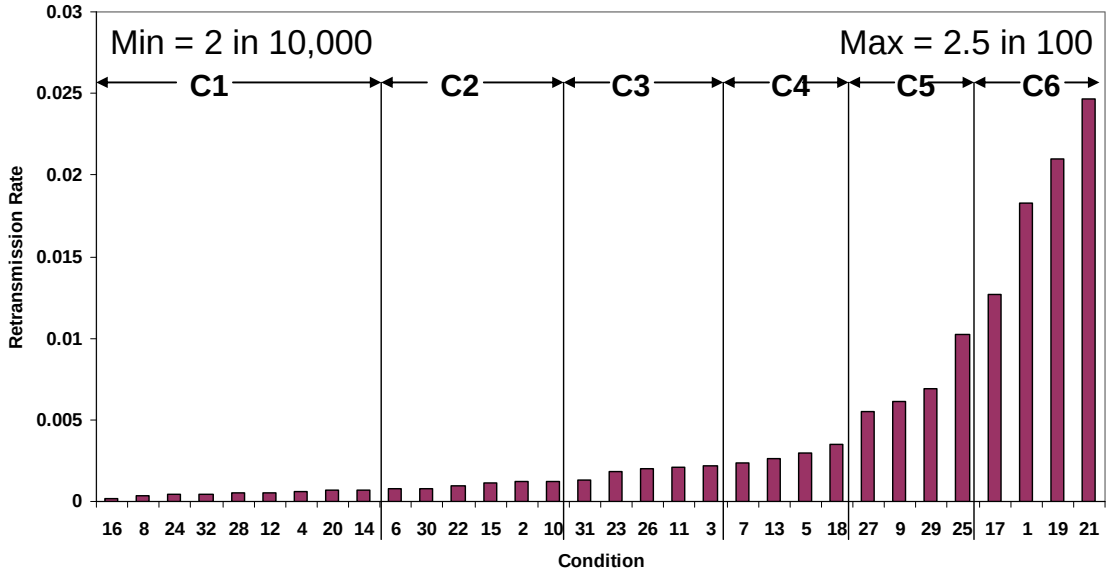


Figure 8-1. Conditions Ordered from Least to Most Congested (High Initial Slow-Start Threshold)

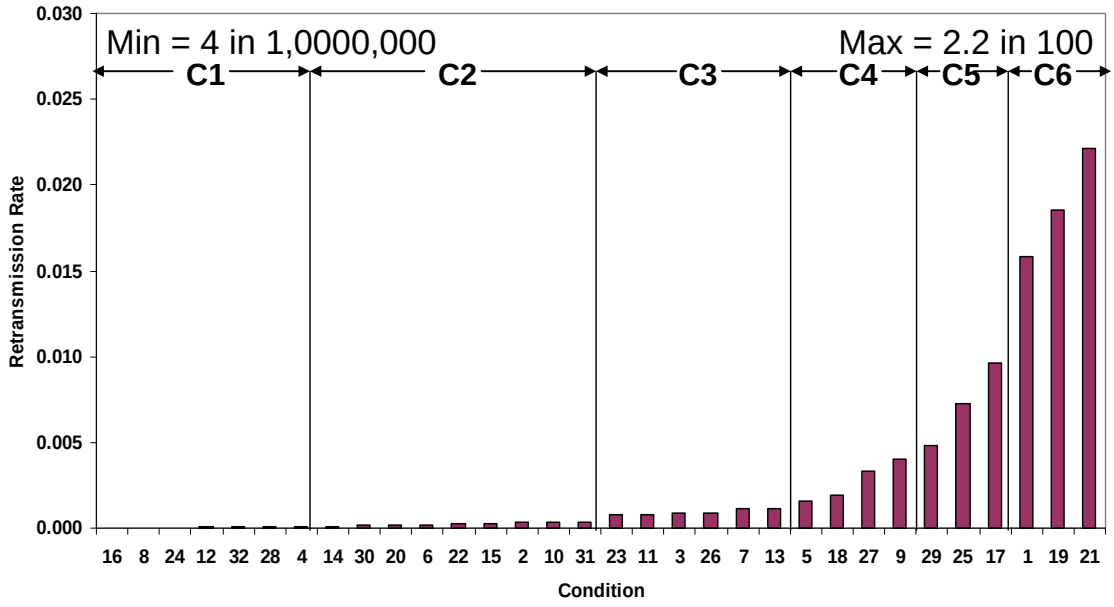


Figure 8-2. Conditions Ordered from Least to Most Congested (Low Initial Slow-Start Threshold)

To further explore the nature of congestion under the conditions simulated for this experiment, we examined six time series under each value of initial slow-start threshold. We chose one condition from each congestion class and we selected conditions that appeared in the same class under both initial slow-start thresholds. Fig. 8-3 plots related time series given a high initial slow-start threshold. Congestion increases with the following conditions: 4, 22, 26, 5, 29 and 1. The y axis indicates the number of flows in a particular state: connecting (gold) or active (red). Active flows may be operating in initial slow start (green), normal congestion avoidance (brown) or alternate congestion avoidance (blue). In these particular plots, CTCP flows were operating in the network along with flows using standard TCP congestion control procedures. The discussion considers only the relative distances between the curves on the graphs, so inability to read the axes will be immaterial. The number of active flows is generally on the order of 10^3.

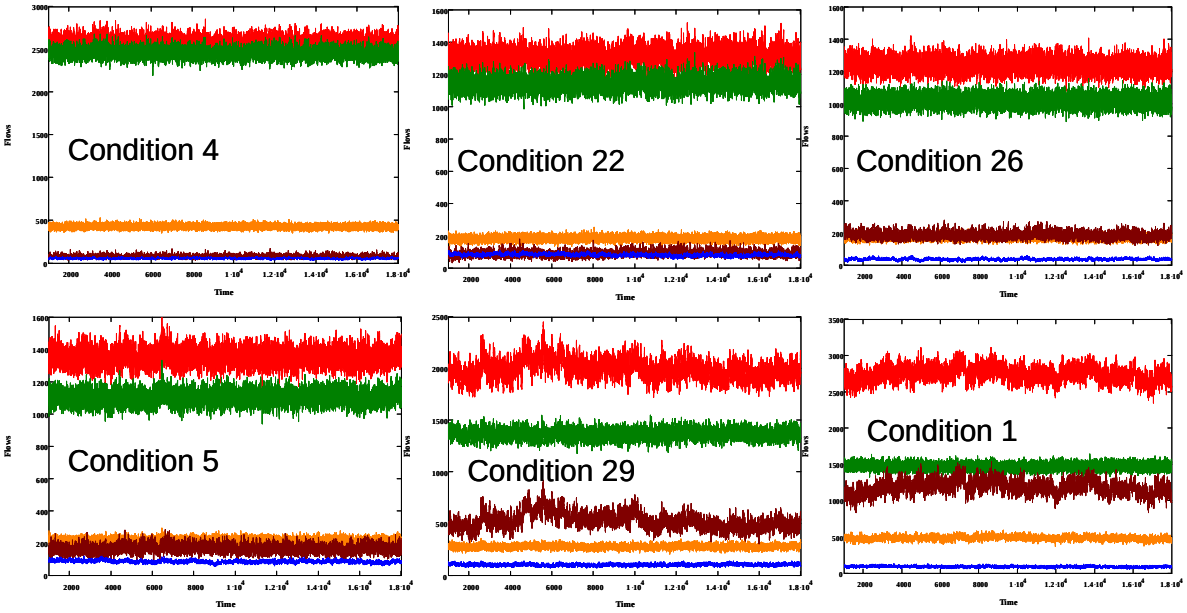


Figure 8-3. Distribution of Flow States for Six Conditions (High Initial Slow-Start Threshold) – Connecting Flows (gold) and Active Flows (red) – with Active Flows subdivided by Congestion Control Phase: Initial Slow Start (green), Normal Congestion Avoidance (brown) and Alternate Congestion Avoidance (blue)

Under the least congested condition (4), most active flows operate in initial slow-start because few losses occur. A small number of flows with larger file sizes experience sporadic losses and operate under normal or alternate congestion control procedures depending upon whether the related source implements alternate procedures and on the value of the congestion window compared against the low-window threshold. As congestion increases with condition, the relative number of active flows in initial slow-start decreases and the relative number under normal congestion control procedures increases. That is, the green and brown lines come closer together. The number of flows under alternate congestion control procedures (blue) shifts up or down slightly depending on whether a particular condition has 70 % of the sources equipped with an alternate congestion control algorithm or only 30 % so equipped.

Fig. 8-4 plots the same conditions as Fig. 8-3 but under a small initial slow-start threshold. Comparison of the figures reveals the fundamental influence of the value of initial slow-start threshold on the distribution of flow states. First, note that except for the most highly congested condition relatively fewer flows operate in initial slow-start. This stands to reason because flows must transition from initial slow-start once the threshold is reached, so relatively more flows will move to alternate or normal congestion avoidance mode. As congestion increases, the proportion of flows in initial slow-start converges with the proportion of flows in normal congestion control mode. The proportion of flows under alternate congestion control procedures shifts up or down slightly depending on whether a particular condition has more or fewer sources equipped with alternate congestion control procedures. This comparison further demonstrates that the same conditions produce similar congestion patterns no matter whether the initial slow-start threshold is large or small.

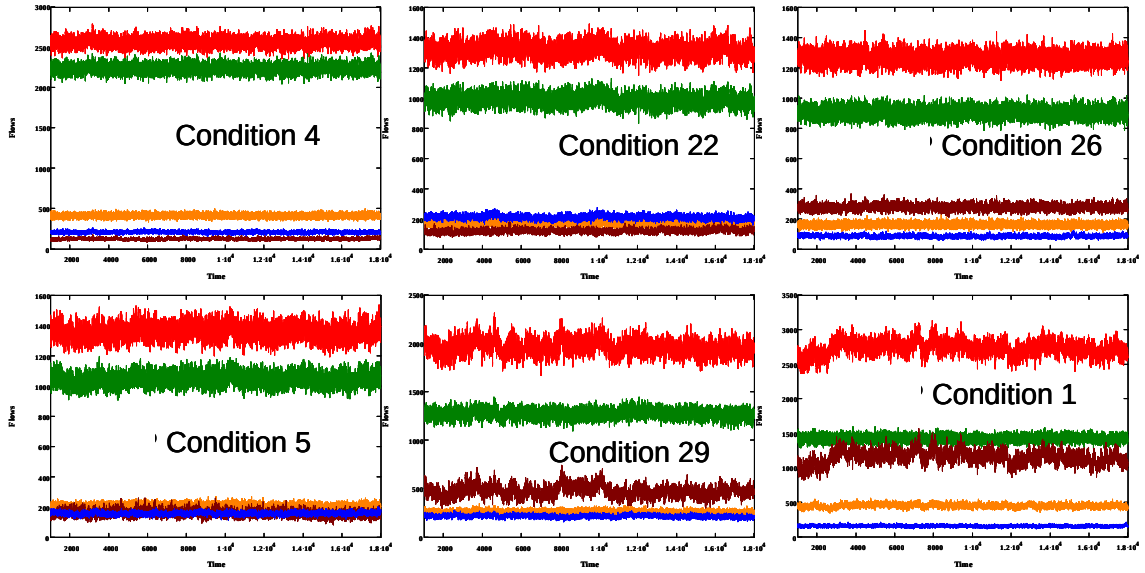


Figure 8-4. Distribution of Flow States for Six Conditions (Low Initial Slow-Start Threshold) – Connecting Flows (gold) and Active Flows (red) – with Active Flows subdivided by Congestion Control Phase: Initial Slow Start (green), Normal Congestion Avoidance (brown) and Alternate Congestion Avoidance (blue)

8.1.4 Responses Measured

As in previous experiments we measured responses in two categories: macroscopic behavior of the network and user experience. In the current experiment, however, we selected somewhat different responses in each category. Table 8-19 enumerates responses (y1 to y16) characterizing macroscopic behavior. We grouped the 16 responses into five subsets (color coded in Table 8-19) measuring: number of flows in a given state (blue); network-wide throughput in packets and flows (green); congestion window size and dynamics (yellow); congestion and delay (red); and proportion of completed flows by file type (orange). We used these responses to assess whether adopting a particular alternate congestion control algorithm alters global behavior in the simulated network.

Measuring user experience for the current experiment became more complicated than was the case for earlier experiments. First, in the current experiment we measured

user experience separately for each of the 24 flow groups identified in Table 8-6. Second, we measured 14 responses for each flow group. Table 8-20 specifies the responses – y1(u) to y14(u) – for a given flow group where all flows in that group use an alternate congestion control algorithm. We separately measured the same 14 responses in each flow group where all flows in that group use standard TCP congestion control procedures. Table 8-20 also lists this second set of 14 responses – y15(u) to y28(u). Isolating goodput on TCP flows enables us to investigate the relative influence of various alternate congestion control algorithms on the goodput of competing TCP flows. In summary, we collected 28 responses for goodput in each flow group. The first 14 responses considered only flows using alternate congestion control procedures and the second 14 responses considered only flows using TCP congestion control procedures. Classifying responses with respect to flow group and congestion control procedures allowed us to compare flows with similar traits against each other with respect to user experience. The classification also enabled us to compare user experience on flows with similar traits where one set of flows used alternate congestion control procedures and one set used TCP congestion control. Among the 14 responses for each flow group we characterized the distribution with four summary statistics (average, standard deviation, minimum and maximum) as well as nine distributional statistics (deciles) and we captured the number of flows (samples) used to create the statistics.

Table 8-19. Measured Responses Characterizing Macroscopic Network Behavior - colors indicate related responses: flow state (blue), network throughput (green), congestion window (yellow), losses and delay (red), and flows by file type (orange)

Response	Definition
y1	Average number of active flows
y2	Average number of flows in initial slow-start
y3	Average number of flows using normal congestion avoidance
y4	Average number of flows using alternate congestion avoidance
y5	Average number of flows attempting to connect
y6	Average aggregate packets output by the network every measurement interval
y7	Average number of flows completed per measurement interval
y8	Average size in packets of congestion window per flow
y9	Average number of congestion window increases per flow per measurement interval
y10	Average retransmission rate measured as proportion of packets resent
y11	Average smoothed round-trip time (ms)
y12	Aggregate number of flows completed
y13	Proportion of completed flows that were Web objects
y14	Proportion of completed flows that were document downloads
y15	Proportion of completed flows that were service-pack downloads
y16	Proportion of completed flows that were movie downloads

8.2 Experiment Execution and Data Collection

Table 8-21 compares processing and memory requirements for simulating the network when the initial slow-start threshold was high versus low. The processing time and memory demands were comparable in both cases. The demands were slightly lower when the initial slow-start threshold was low. This appears to reflect the fact that network-wide

congestion was somewhat lower when the initial slow-start threshold was not extremely high. Table 8-22 gives evidence corroborating this hypothesis. Notice that about 7 million more flows were completed in the 224 simulated hours (about 30 x 10^3 flows per hour) when the initial slow-start threshold was set low. Also notice that completing those flows required about 42 billion fewer packets. This result is consistent with lower congestion when the initial slow-start threshold was set to the lower value.

Table 8-20. Measured Responses Characterizing User Experience for Each Flow Group, inlcuding Flows using an Alternate Congestion Control Algorithm, y1(u) – y14(u), and Competing TCP Flows, y15(u) – y18(u)

Response	Definition
y1(u)	Total number of flows in group that used alternate congestion avoidance
y2(u)	Average goodput
y3(u)	Standard deviation in goodput
y4(u)	Minimum goodput
y5(u)	Maximum goodput
y6(u)	10th Percentile in goodput
y7(u)	20th Percentile in goodput
y8(u)	30th Percentile in goodput
y9(u)	40th Percentile in goodput
y10(u)	50th Percentile in goodput
y11(u)	60th Percentile in goodput
y12(u)	70th Percentile in goodput
y13(u)	80th Percentile in goodput
y14(u)	90th Percentile in goodput
y15(u)	Total number of flows in group that used standard TCP congestion avoidance
y16(u)	Average goodput
y17(u)	Standard deviation in goodput
y18(u)	Minimum goodput
y19(u)	Maximum goodput
y20(u)	10th Percentile in goodput
y21(u)	20th Percentile in goodput
y22(u)	30th Percentile in goodput
y23(u)	40th Percentile in goodput
y24(u)	50th Percentile in goodput
y25(u)	60th Percentile in goodput
y26(u)	70th Percentile in goodput
y27(u)	80th Percentile in goodput
y28(u)	90th Percentile in goodput

8.2.1 Computing Macroscopic Responses

We computed macroscopic responses in two general forms. In one form we counted events for each run over the simulated period (one hour). Specifically, for responses y12 through y16 we counted the number of completed flows and categorized each completed flow by file type. Then we computed the proportion of completed files by type (y13 to y16) as the ratio of the count by type to total flows completed.

Table 8-21. Comparing Resource Requirements for Simulating One Hour of Network Operation under 32 Conditions with High and Low Initial Slow-Start Thresholds

	Small Network with High Initial Slow-Start Threshold	Small Network with Low Initial Slow-Start Threshold
CPU hours (224 Runs)	5.857×10^3	5.639×10^3
Avg. CPU hours (per run)	26.15	25.17
Min. CPU hours (one run)	12.58	12.51
Max. CPU hours (one run)	43.97	40.94
Avg. Memory Usage (Mbytes)	196.56	194.46

Table 8-22. Comparing Flows Completed and Data Packets Sent when Simulating One Hour of Network Operation under 32 Conditions with High and Low Initial Slow-Start Thresholds

	Small Network with High Initial Slow-Start Threshold		Small Network with Low Initial Slow-Start Threshold	
Statistic	Flows Completed	Data Packets Sent	Flows Completed	Data Packets Sent
Avg. Per Condition	11.466×10^6	3.414×10^9	11.495×10^6	3.225×10^9
Min. Per Condition	7.258×10^6	2.139×10^9	7.263×10^6	2.055×10^9
Max. Per Condition	17.391×10^6	5.048×10^9	17.432×10^6	4.832×10^9
Total all Runs	2.568×10^9	$764{,}740 \times 10^9$	2.575×10^9	722.466×10^9

For each of the responses y1 through y11 we computed average values from a time series of 9000 measurements. Figure 8-5 illustrates an example of such a computation for response y10, average retransmission rate. This example was taken from simulated condition 1 in the case where CTCP was the alternate congestion control algorithm and where the initial slow-start threshold was high. Notice that we discarded the first half of the time series, which avoided startup transients. We computed the mean of the second half of the time series; in this case the mean retransmission rate was 0.018.

We organized all responses measuring macroscopic network behavior into a table, where each row contained the 16 responses under a given condition and alternate congestion control algorithm. Table 8-23 depicts the response format in the case when the initial slow-start threshold is high. We created a similar table for responses obtained

under a low initial slow-start threshold. These two tables served as the input data for our analysis of macroscopic behavior.

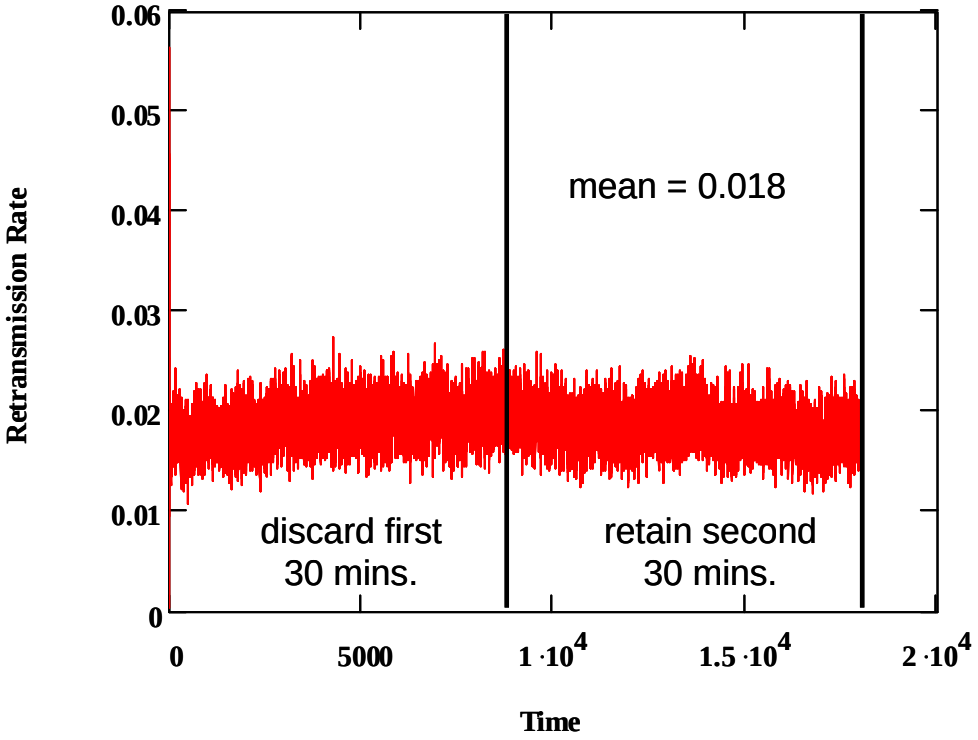


Figure 8-5. Illustration of Technique to Compute Means for Responses y1 to y11 - example for Retransmission Rate (y10), measured as the proportion of packets resent, vs. Time (200 ms intervals) under Condition 1 – CTCP – High Initial Slow-Start

Table 8-23. Data Format Summarizing Responses y1 to y16 for All Algorithms and Conditions

Algorithm	Run	y1	y2	…	y15	y16
1	1	2821.014	1475.276	…	0.000242	0.000021
1	2	1049.267	896.2793	…	0.001426	0.000059
…	…	…	…	…	…	…
1	31	1863.727	1522.075	…	0.000654	0.000036
1	32	2541.456	2215.645	…	0.001101	0.000047
…	…	…	…	…	…	…
7	1	2764.41	1471.684	…	0.000207	0.000018
7	2	1067.066	901.6619	…	0.001414	0.000061
…	…	…	…	…	…	…
7	31	1975.804	1534.182	…	0.000650	0.000038
7	32	2674.573	2213.257	…	0.001054	0.000040

8.2.2 Computing User Experience Responses

We captured user experience directly during each simulation run. The general technique was to set a threshold for a minimum number of samples prior to reporting distributional information. At each measurement interval we computed and reported distributional information for each flow group where the number of samples exceeded the threshold. At the end of the simulation we also reported distributional information for residual flows, regardless of the sample threshold. As a result of this technique we generated one output file per flow group. The format of each output file is similar to Table 8-24.

Table 8-24. Data Format Summarizing User Experience for One Flow Group – example for CTCP Flow Group 16 (Fast Path, Normal Interface Speed, Document) under Condition 1

Time	*N*	mean	stdev	min	max	10th%	20th%	30th%	40th%	50th%	60th%	70th%	80th%	90th%
736	1001	831.2	667.7	73.2	6126.7	233.8	363.8	467.8	546.8	643.1	773.4	950.2	1174.5	1636.6
1497	1000	916.4	734.5	82.9	5674.6	255.4	384.5	476.5	584.7	694.0	849.1	1052.5	1304.2	1873.0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
17454	1000	888.3	754.0	45.4	4977.2	235.9	349.7	454.2	554.7	676.6	817.0	1012.5	1298.5	1737.7
18000	696	908.8	759.9	22.1	6998.2	243.8	359.9	463.7	563.1	690.7	850.6	1040.9	1336.6	1842.0

Given information such as shown in Table 8-24, we summed the number of samples (*N*) and computed a weighted average for each of the 13 remaining statistics. For a given simulation run (specified by condition and alternate congestion control algorithm), we performed this computation for each of the 24 flow groups under the alternate congestion control algorithm and under normal TCP congestion control procedures. Thus, we summarized 48 output files (24 flow groups x two congestion control algorithms) under each simulated condition (32 x 48 = 1536 files across all conditions) for each specified alternate congestion control algorithm (7 x 1536 = 10752 files across all conditions and congestion control algorithms).

Table 8-25. Data Format Summarizing User Experience for One Flow Group under All Algorithms and Conditions

Algorithm	Run	y1(u)	y2(u)	…	y27(u)	y28(u)
1	1	23249	846.06	…	1133.338	1618.362
1	2	13800	1621.745	…	2270.871	3258.24
…	…	…	…	…	…	…
1	31	7934	856.813	…	1167.37	1687.49
1	32	15776	1003.388	…	1408.86	2063.676
…	…	…	…	…	…	…
7	1	23703	886.527	…	1156.298	1660.003
7	2	13666	1596.665	…	2264.218	3312.346
…	…	…	…	…	…	…
7	31	7954	800.124	…	1088.954	1582.971
7	32	15661	1023.458	…	1422.419	2111.122

For a given flow group, we concatenated the 14 responses on flows using alternate congestion control together with the 14 responses on flows using TCP congestion control and appended identifiers for the alternate algorithm and the condition to produce a 30 cell row for each combination, as illustrated in Table 8-25. Thus, we summarized user-experience responses into 24 files: one per flow group. Where needed to make data analysis more convenient, we concatenated all flow groups into a single file, adding a cell to each row to identify the flow group associated with the data. A single concatenated file contained (24 x 7 x 32 =) 5376 rows, one for each combination of flow group, alternate congestion control algorithm and simulated condition.

8.3 Data Analysis Approach

Most of the data analyses conducted for this experiment focused on user experience. Before explaining the techniques we applied to analyze user experience, we provide a brief summary of the single technique we applied to analyze macroscopic responses.

8.3.1 Analyzing Macroscopic Behavior

We considered each of the 16 macroscopic responses (recall Table 8-19) using a detailed analysis of the individual responses, as explained previously in Sec. 6.3.2. Here, we provide only a brief summary of the technique. Fig. 8-6 shows a sample plot displaying the analysis of retransmission rate (response y10) across all seven congestion control algorithms under the 32 conditions given a high initial slow-start threshold.

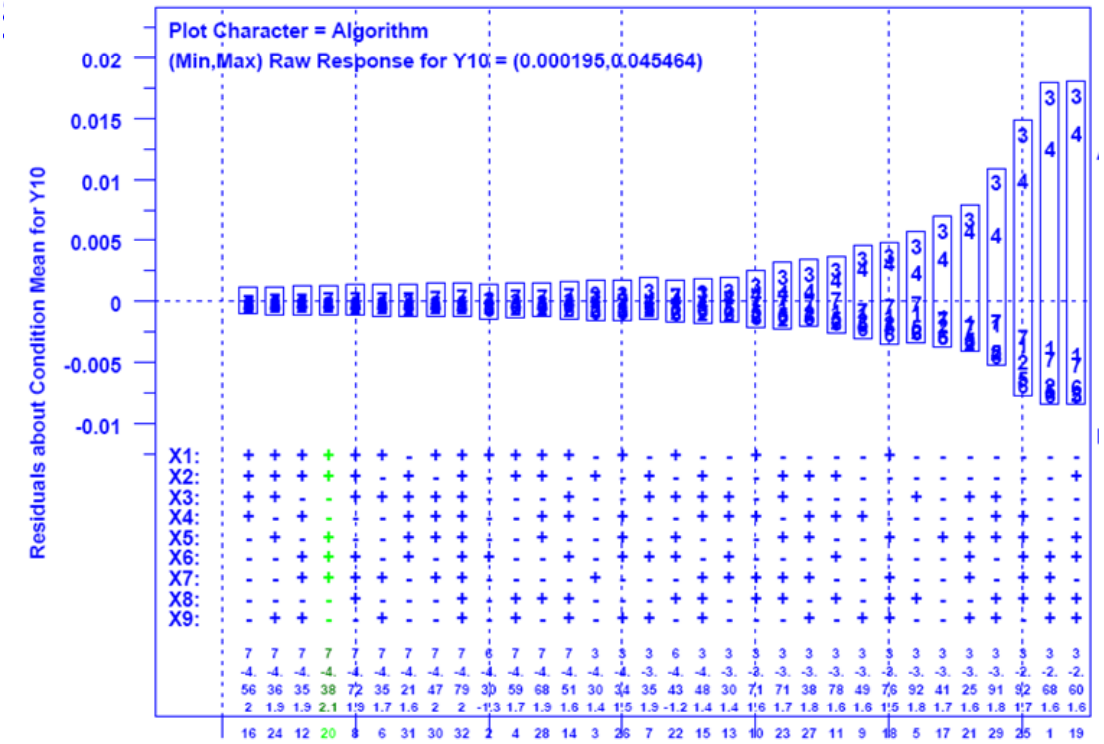


Figure 8-6. Detailed Analysis of Retransmission Rate (proportion of packets resent) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals; non-blue columns indicate statistically significant outliers, either high (green) or low (red)

For each condition, we computed the mean response and then reformulated the response for each algorithm as residuals around the condition mean by subtracting the response from the condition mean. We then sorted the conditions from the least to greatest extreme (by magnitude of the) residual and plotted the residuals (y axis) along with the factor settings associated with the related conditions (x axis). Below the factor settings we identified the algorithm exhibiting the most extreme residual. We also indicated the order of magnitude and percentage difference in the extreme residual from the mean. We applied a Grubbs' test to determine if the extreme residual represented a statistically significant difference from the mean. If the difference was statistically significant on the positive side, then we colored the column green. If significant on the negative side, we colored the column red. Otherwise, the column remains blue.

8.3.2 Analyzing User Experience

We analyzed user experience with respect to the 24 flow classes identified in Table 8-6. In each class, we considered the experience of normal TCP users and also the experience of users under a competing alternate congestion control algorithm. We measured user experience as goodput (i.e., packets received per unit of time, excluding retransmissions). While we collected distributional data for each flow group (recall Table 8-20), the analyses described in this section focus solely on mean goodput for users under alternate congestion control – y2(u) – and under standard TCP congestion control – y16(u).

We captured the average goodputs – y2(u) and y16(u) – in a tabular form, where goodputs are reported to the nearest packet per second (pps). From the table we extracted various graphs that compare goodputs of all congestion control algorithms for specific flow classes. For example, Fig. 8-7 shows two typical plots we used.

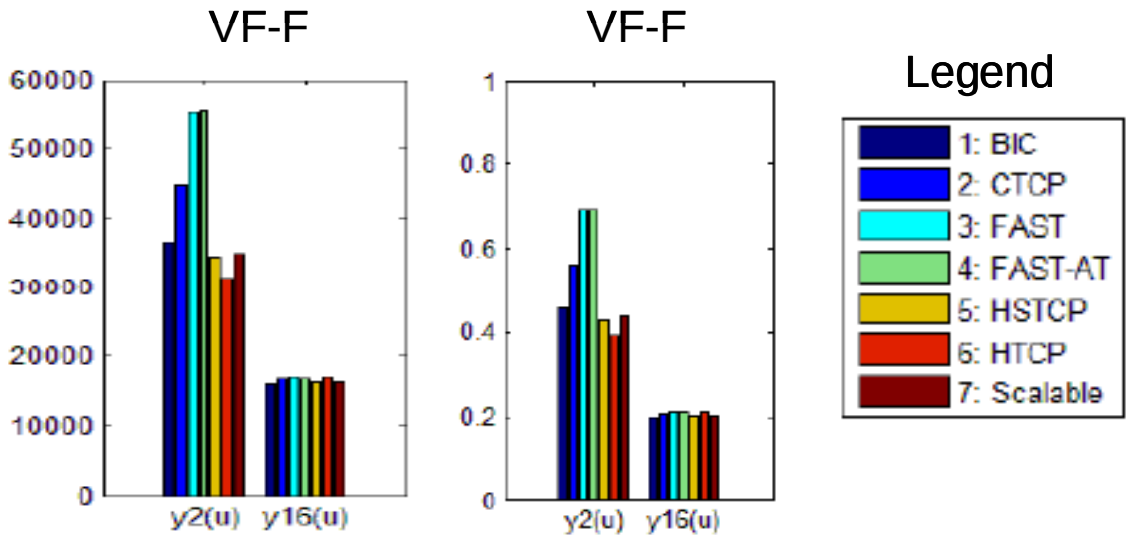


Figure 8-7. Average Goodput (packets per second and as proportion of interface speed) for Flows Using Alternate Congestion Control Algorithm – y2(u) – and Competing Flows Using TCP – y16(u) – when Transferring Movies on a Very Fast Path with a Fast Interface Speed Given a Low Initial Slow-Start Threshold. Leftmost bar graph plots raw average goodput (packets per second), while rightmost bar graph plots average goodput as a proportion of the maximum achievable transmission speed.

The legend in Fig. 8-7 shows the bar color associated with a particular alternate congestion control algorithm. When plotted in bar graphs we plot the algorithms by increasing identifier from 1 (BIC) to 7 (Scalable). Each bar graph is labeled with the path class (VF in Fig. 8-7) and interface speed (F in Fig. 8-7). The bar graphs in Fig. 8-7 plot average goodput when transferring movies over very fast paths with a fast interface speed (maximum of 80 x 10^3 pps) given a low initial slow-start threshold. The leftmost graph gives the raw average goodput (y axis) for each congestion control algorithm (one bar each). The first set of seven bars represents the goodput achieved on flows using a specific alternate congestion control algorithm. The second set of seven bars represents goodput achieved on flows using normal TCP congestion control but operating in a network where some flows use a specified, competing alternate congestion control algorithm. The rightmost graph is formulated in the same fashion except that the y axis expresses goodput as a fraction of the maximum achievable transfer rate (80 x 10^3 pps here). The leftmost graph illustrates differences in goodput among the various algorithms and also identifies differences in goodput between the alternate algorithms and normal TCP. The rightmost graph shows the degree to which the various flows were able to achieve the maximum available goodputs.

To investigate causes of variation in goodputs, we employed principal components analyses (PCA) on the average goodput data – y2(u) and y16(u) – for each of the seven alternative congestion control algorithms under all 32 conditions. For each given algorithm *a* and condition *c* we collected 24 observations for y2(u) (one per flow group) and 24 for y16(u) (one per flow group) into a 48-dimension vector: $(x_1, x_2, \ldots, x_{48})_{a,c}$ for a total of (32 x 7 =) 224 vector instances. We then conducted a PCA, as described earlier in Sec. 4.5, which yielded plots such as shown in Fig. 8-8.

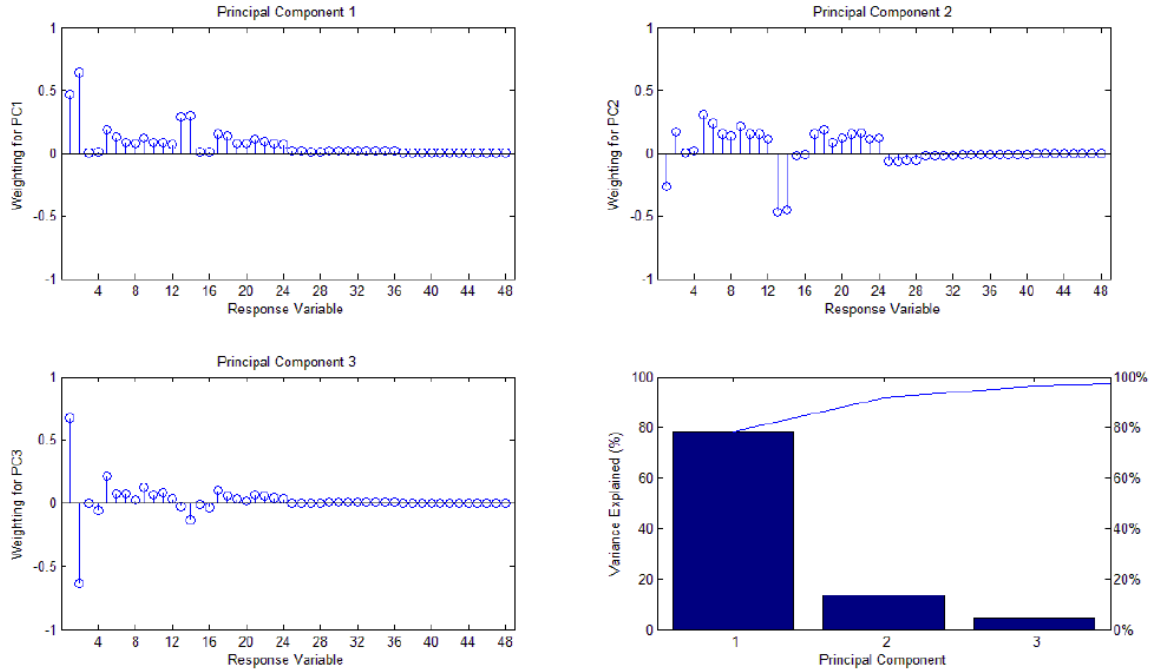


Figure 8-8. Principal Components Analysis of Goodputs given High Slow-Start Threshold – three subplots give the weight vectors for the first three PCs and one bargraph indicates the proportion of variance explained by each of the first three PCs, as well as the variance explained by a combination of the first three PCs

As Fig. 8-8 demonstrates nearly all variation in the data could be accounted for by the first three principal components (PC). We plotted pairs of PCs against one another in biplots to investigate whether specific factors caused similarity among goodputs. Fig. 8-9 gives an example of one such plot of PC1 (x axis) vs. PC 2 (y axis). The legend associates each congestion control algorithm with a particular colored symbol. Fig. 8-9 clearly shows three groups of observations (circled). Two of the groups divide into two subgroups. As explained below in Sec. 8.4.2, we analyzed factors in common among observations in each group to provide information about the causes of these groupings.

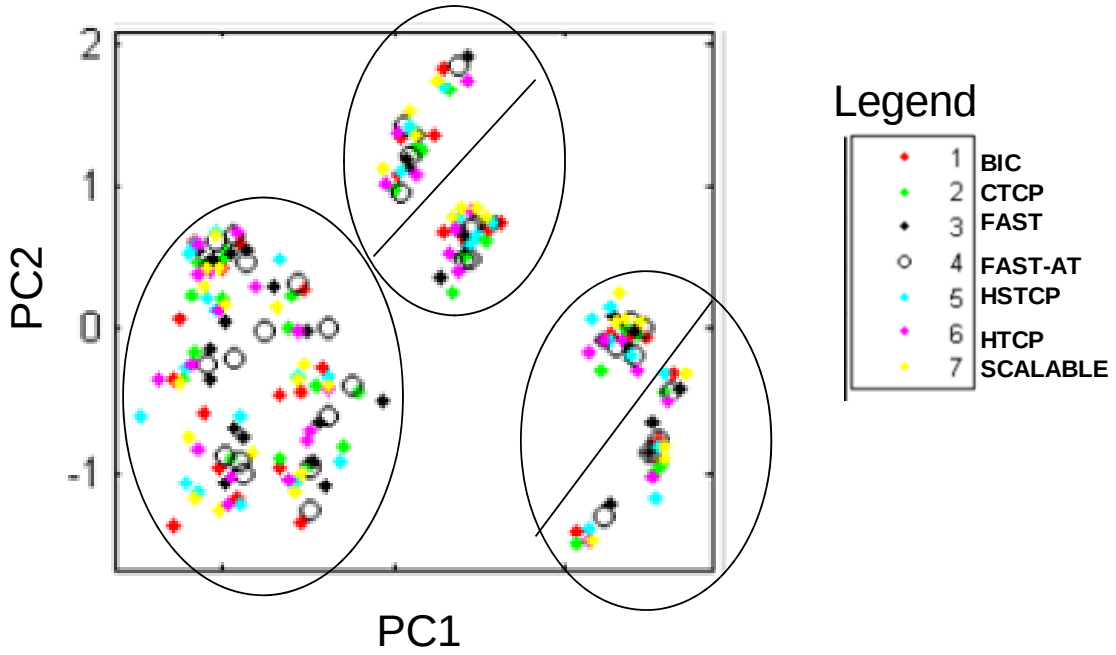


Figure 8-9. Illustration of Biplot of PC1 vs. PC2 and Related Clustering

To compare goodputs provided on normal TCP flows against goodputs provided on flows using alternate congestion control algorithms, we adopted two main techniques. First, we created plots of y2(u) vs. y16(u) for all 32 conditions for a given flow group and alternate congestion control algorithm. For example, Fig. 8-10 shows such a plot for algorithm 3 (FAST) when transferring movies over very fast paths with a fast interface speed given a high initial slow-start threshold. The figure in red (0.96632) above the plot is the computed correlation between y2(u) and y16(u). Points below the diagonal indicate cases where flows using the alternate congestion control regime achieved higher average goodput, while points above the diagonal indicate cases where TCP flows achieved higher average goodput. A strong positive correlation indicates that the trend in goodputs for all flows was linear with respect to condition.

As a second technique to compare goodput of TCP flows vs. goodput of flows using alternate congestion control algorithms, we plotted bar graphs for each condition and flow group, where each bar spans two points for each algorithm. One point represents y2(u)/1000 and one represents y16(u)/1000. If the y2(u) value is higher, then the bar is colored green. If the y16(u) value is higher, the bar is colored red. Fig. 8-11 shows a sample of such a bar graph. The bar for algorithm 4 (FAST-AT) is colored red, which shows that for this condition and flow group TCP flows achieved about 5000 pps higher

(40 p/ms – 35 p/ms = 5 p/ms) average goodput than FAST-AT flows. The specific condition (21; most congested) is reported in the lower left corner of the plot.

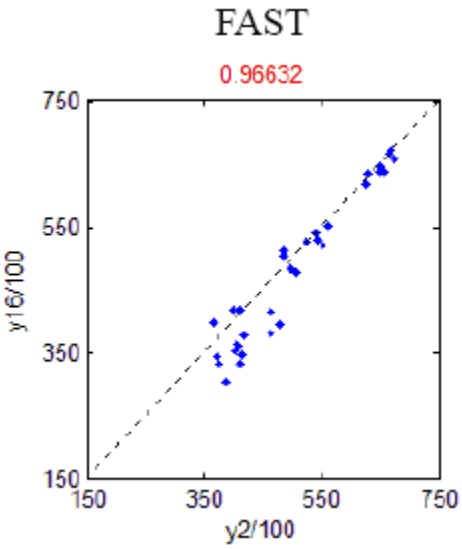


Figure 8-10. Scatter Plot of y16(u)/100 vs. y2(u)/100 for Movies Transferred over a Very Fast Path with Fast Interface Speed Given a High Initial Slow-Start Threshold; FAST Alternate Congestion Control Algorithm

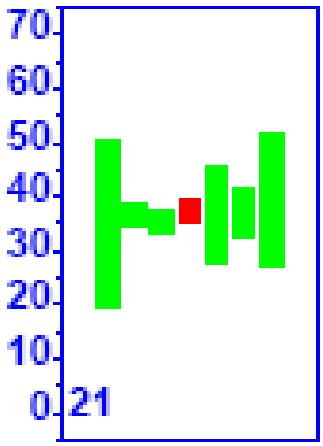


Figure 8-11. Bar Graph for Movies Transferred over a Very Fast Path with Fast Interface Speed given a High Initial Slow-Start Threshold during Condition 21 (Most Congested) – each bar is formed by plotting y16(u)/1000 and y2(u)/1000 for a Specific Alternate Congestion Control Algorithm (plotted from 1 to 7 left to right) – if a bar is red then y16(u)/1000 is plotted at the top of the bar and y2(u)/1000 is plotted at the bottom of the bar; otherwise (green bar) y2(u)/1000 is plotted at the top of the bar and y16(u)/1000 is plotted at the bottom of the bar – y axis gives goodput (packets/ms)

In addition to analyzing absolute differences in goodput among the alternate congestion control algorithms and between the alternates and normal TCP congestion control, we also analyzed the relative differences. To compare relative differences we adopted a rank analysis. For each given flow group and condition we compared the y2(u)

values among the seven alternate congestion control algorithms and ranked them from highest (7) to lowest (1). After ranking on all flow groups and conditions, we produced a rank matrix for each alternate congestion control algorithm. Fig. 8-12 shows an example of such a rank matrix. We generated similar matrices based on ranking y16(u) values among the seven alternate congestion control algorithms. The y16(u)-based ranking indicates relative goodputs achieved by TCP flows when operating concurrently with specific alternate congestion control algorithms.

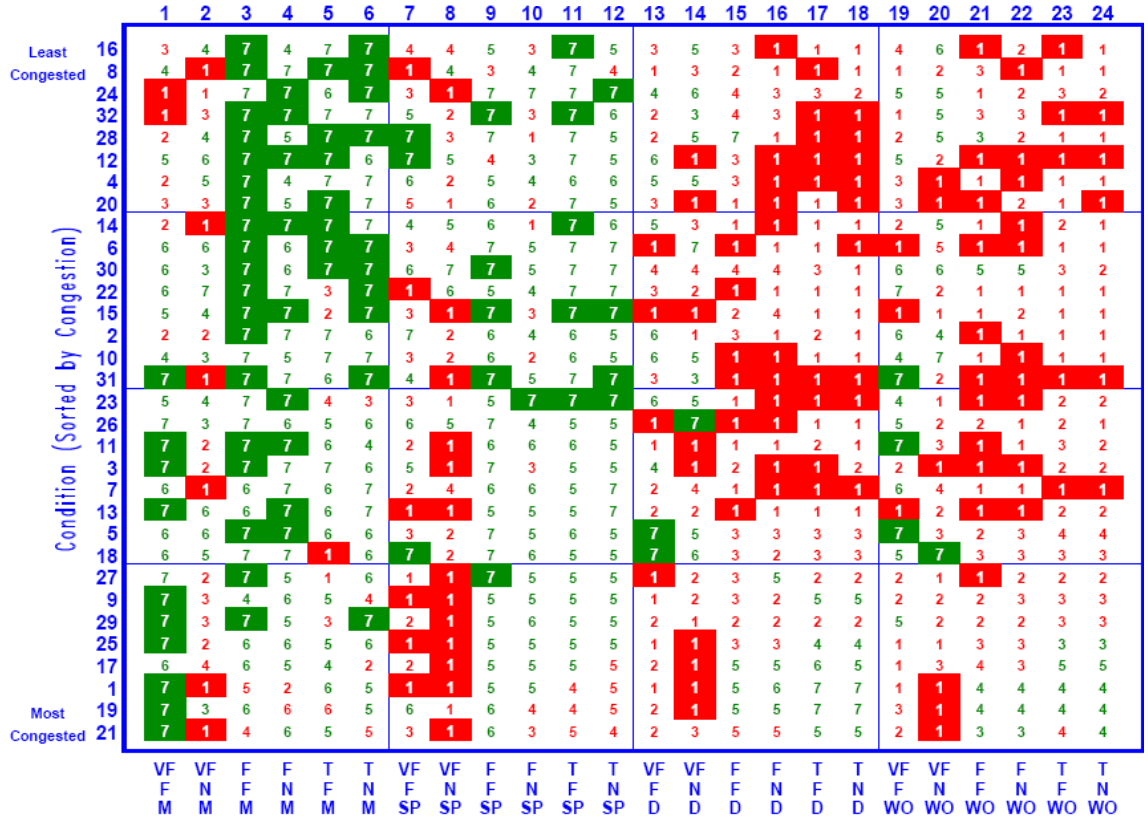


Figure 8-12. Rank Matrix for Algorithm 7 (Scalable TCP) – High Initial Slow-Start Threshold. Rank (7 high) in each cell denotes ordering of y2(u) for each condition (y axis) and flow group (x axis) – conditions are sorted from least (16) to most (21) congested and flow groups are ordered by file size – movies (M), service packs (SP), documents (D) and Web objects (WO) – and by path class – very fast (VF), fast (F), and typical (T) – within each file size and by interface speed – fast (F) or normal (N) – within each path class. Green ranks had goodput values above the condition mean, while red ranks had goodput values below the condition mean. Filled cells indicate the goodput was most extreme: either high (7 green) or low (1 red).

The matrix in Fig. 8-12 contains (24 flow groups x 32 conditions =) 768 cells, one per flow group per condition. Here the matrix reports the ranking of algorithm 7 (Scalable TCP) with respect to other alternate congestion control algorithms for response y2(u) – average goodput on flows using an alternate algorithm instead of standard TCP. To determine the rank for a given condition and flow group we order the algorithms from lowest to highest average goodput, y2(u), and then assign a integer from 1 (lowest) to 7 (highest). We also compute the mean of the seven goodputs. The rank is colored green when the value if y2(u) is above (red when below) the mean for the same condition and

flow group. If the value of y2(u) is most distance from the mean the rank is filled – green for highest (7) and red for lowest (1). A quick glance at Fig. 8-12 reveals that Scalable TCP appears to provide best goodput for larger files (movies and service packs) and worst goodput for smaller files (documents and Web objects). Given a complete set of 14 matrices, one per algorithm ranking y2(u) values and one per algorithm ranking y16(u) values, we also computed the average (and standard deviation) of the ranking for each algorithm with respect to each file type. The resulting tables (Tables 8-31 and 8-32) allowed us to succinctly compare relative ranking among the algorithms.

8.4 Results

Here, we present selected simulation results in three categories: (1) macroscopic network behavior, (2) absolute user experience and (3) relative user experience. Within each category, we first give relevant data under a high initial slow-start threshold followed by data under a low initial slow-start threshold. We present only data that reveals behavioral similarities and differences of interest.

8.4.1 Macroscopic Network Behavior

In general, the data analyses reported in this section do not reveal much in the way of statistically significant changes in macroscopic network behavior. This appears due mainly to a general lack of congestion throughout these experiments. In addition, we consider both FAST (algorithm 3) and FAST-AT (algorithm 4) together in these analyses, which reduces the statistical significance of either algorithm considered alone because both algorithms share some traits (as described previously in Chapter 7). Despite this, we could discern patterns in macroscopic network behavior with respect to some responses. In most cases, the patterns detected echo patterns seen in previous experiments, where simulated congestion tended to be much higher under most conditions. Here, we report the patterns we found informative.

8.4.1.1 High Initial Slow-start Threshold. Fig. 8-13 gives a detailed analysis of the average number of active flows under the 32 simulated conditions. Notice that in most conditions either algorithm 7 (Scalable TCP) or 3 (FAST) shows a higher number of active flows than other algorithms. This suggests that these algorithms have some number of flows that take longer to complete. Algorithm 3 exhibits the extreme value under conditions with highest congestion. This suggests that under those conditions, some FAST flows exhibit the oscillatory behavior identified in previous experiments (recall Chapter 6), which induces excessive losses and lowers goodput on affected flows. In previous experiments (see Chapter 5), Scalable TCP was found to provide significant unfairness when new flows attempt to gain bandwidth from already established flows. This occurs because Scalable TCP flows occupy significant buffer space and reduce their congestion window little on each loss, which causes affected new flows to experience a larger proportion of losses, and lower goodputs. The reader should keep these ideas in mind as additional responses are presented.

Fig. 8-14, which shows the average number of flows attempting to connect, supports the analysis from the preceding paragraph. Under conditions with higher congestion, algorithm 3 (FAST) or 4 (FAST-AT) exhibits more flows attempting to connect. Under most other conditions, Scalable (algorithm 7) exhibits a larger number of

flows attempting to connect. These behavioral differences arise as SYN packets suffer a lower rate of successful delivery, which forces affected flows to take longer to connect. Figure 8-15 further corroborates this picture by revealing that FAST completes fewer flows per interval under higher congestion and that Scalable completes fewer flows per interval under most other conditions.

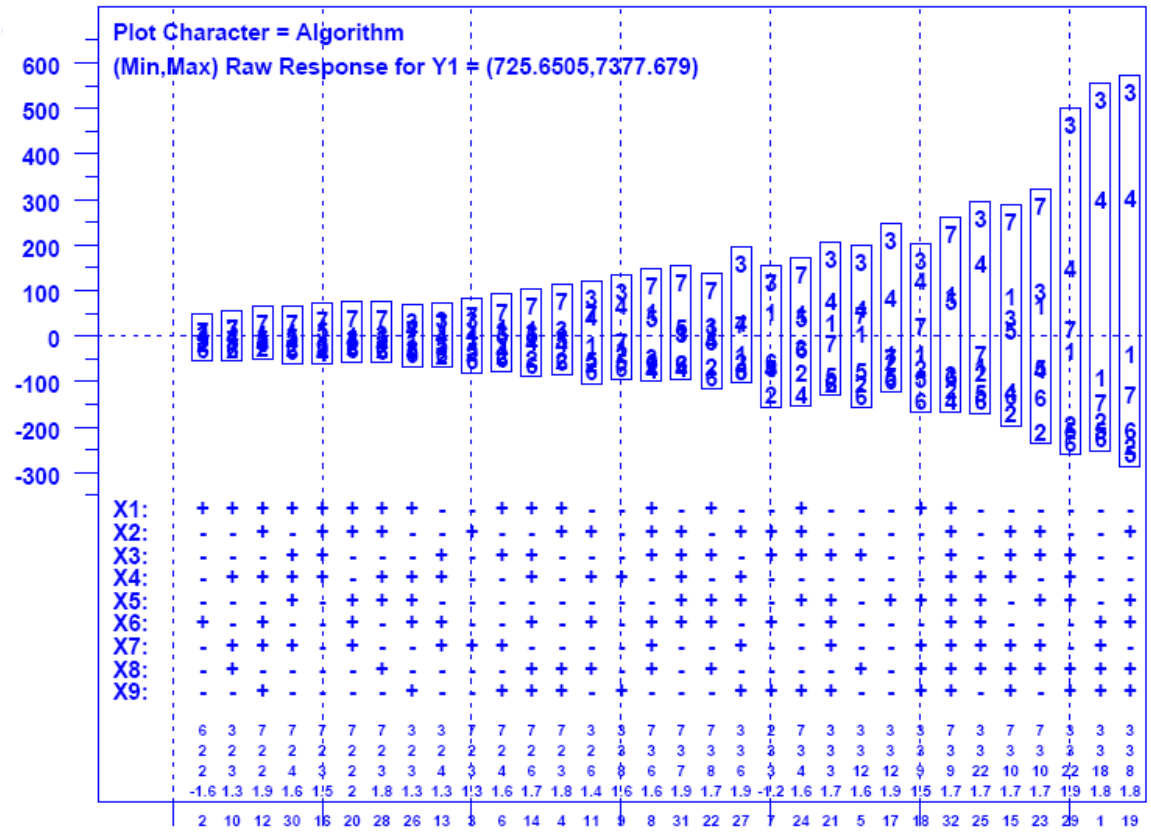


Figure 8-13. Average Number of Active Flows under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

Fig. 8-16 adds more supporting evidence. Notice that FAST and FAST-AT (algorithms 3 and 4) exhibit higher retransmission rates under conditions with higher congestion and Scalable (algorithm 7) exhibits higher retransmission rates under most other conditions. Fig. 8-17 shows that under most conditions, Scalable leads to higher average smoothed round-trip times, which supports the observation that Scalable tends to have higher buffer occupancy than other algorithms. Fig. 8-18 confirms that over an entire simulated hour, Scalable and FAST tend to complete the fewest flows. Similarly, Fig. 8-19 shows that under most conditions Scalable completes a higher proportion of flows that are small (i.e., Web objects). In the remaining conditions, either FAST or FAST-AT completes a higher proportion of flows that are Web objects. Recall that when the maximum number of flows with a given file size are already active, then newly arriving flows remain Web objects. Therefore, completing a higher proportion of flows that are Web objects implies that some larger flows (movies, service packs and documents) take longer to complete.

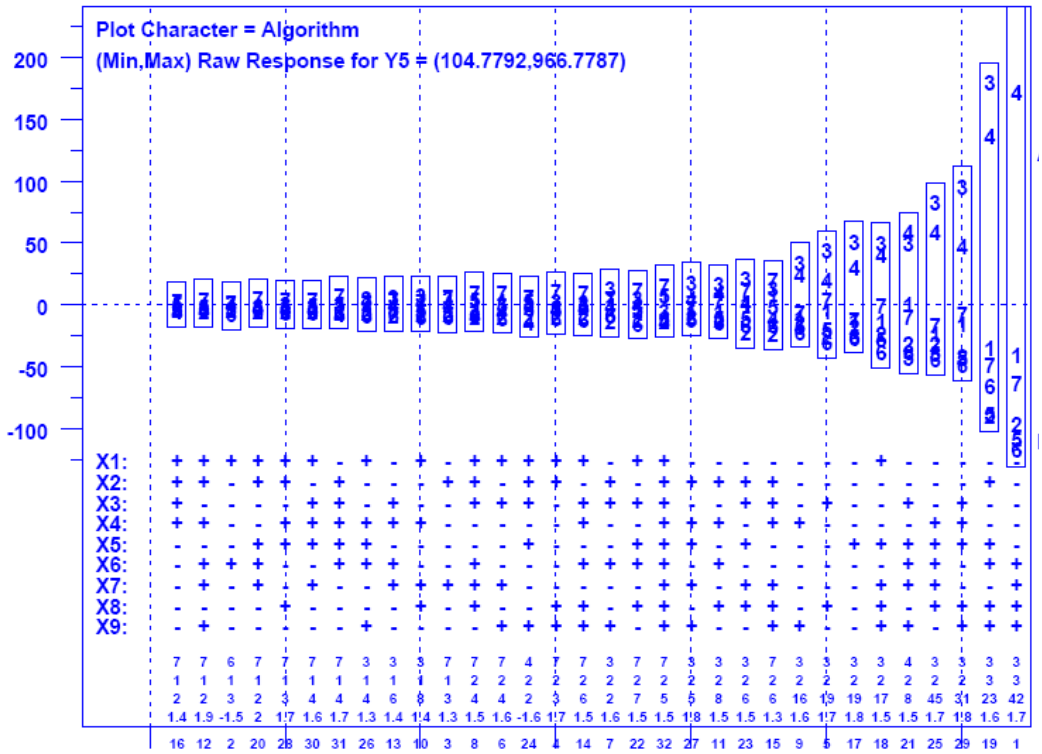


Figure 8-14. Average Number of Connecting Flows under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

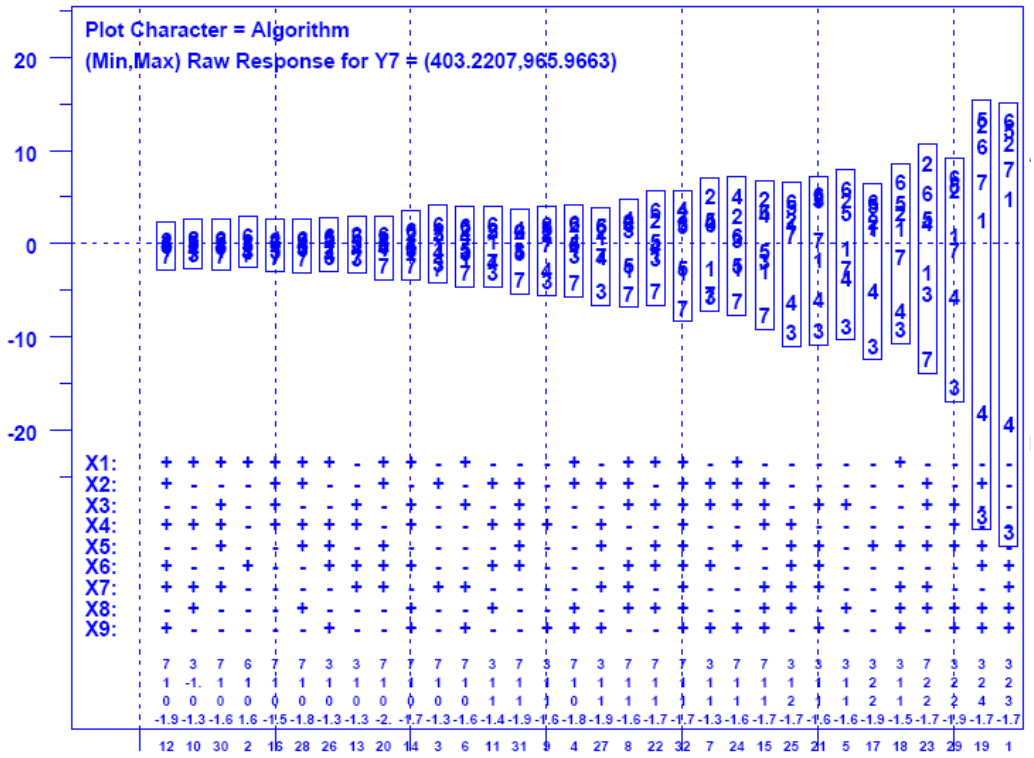


Figure 8-15. Average Rate of Flow Completion (flows per 200 ms) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

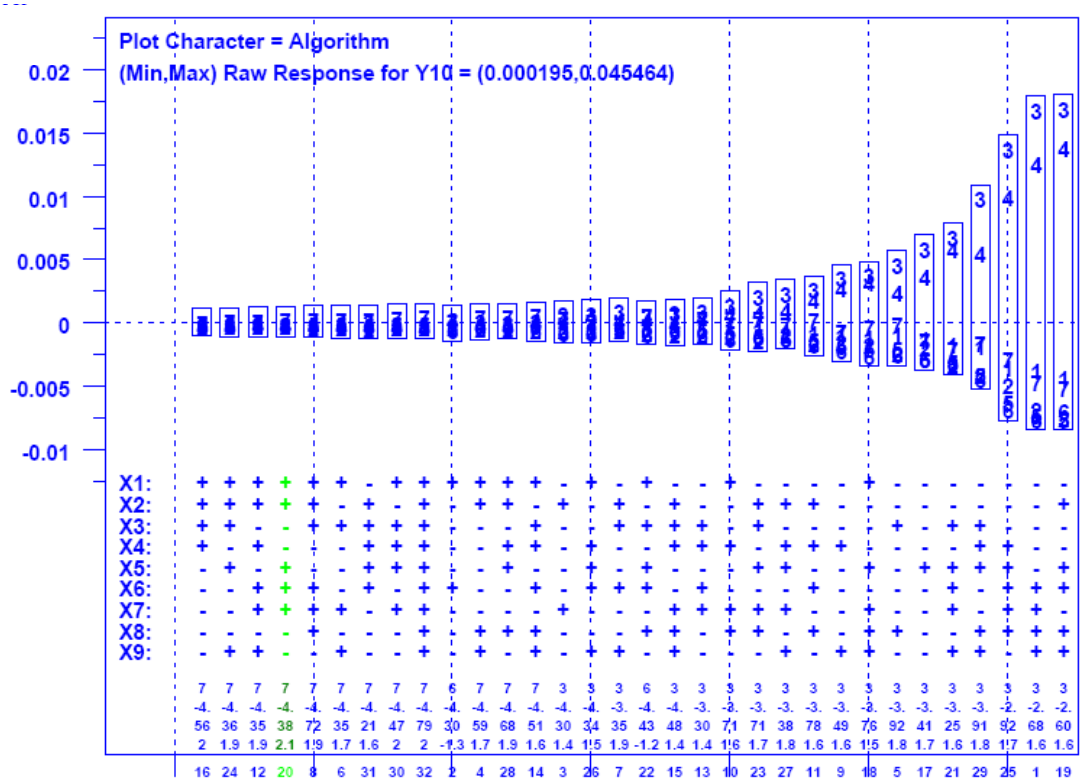


Figure 8-16. Average Flow Retransmission Rate (proportion of packets resent) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

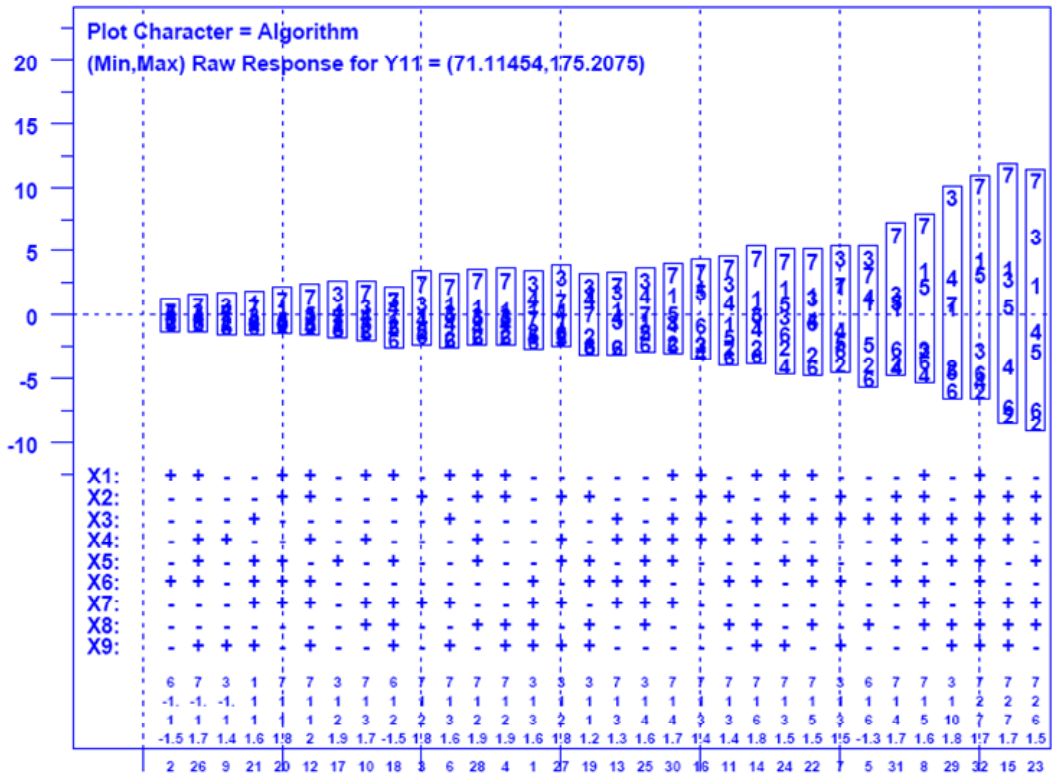


Figure 8-17. Average Smoothed Round-Trip Time (ms) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

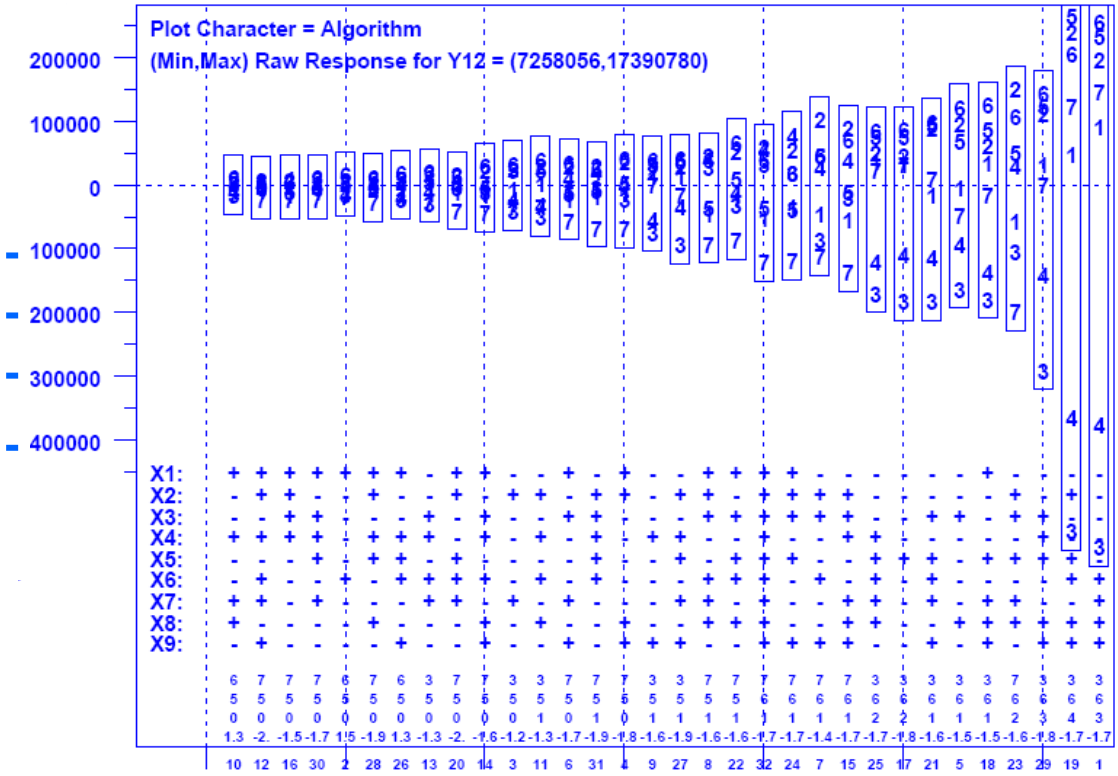


Figure 8-18. Aggregate Flows Completed under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

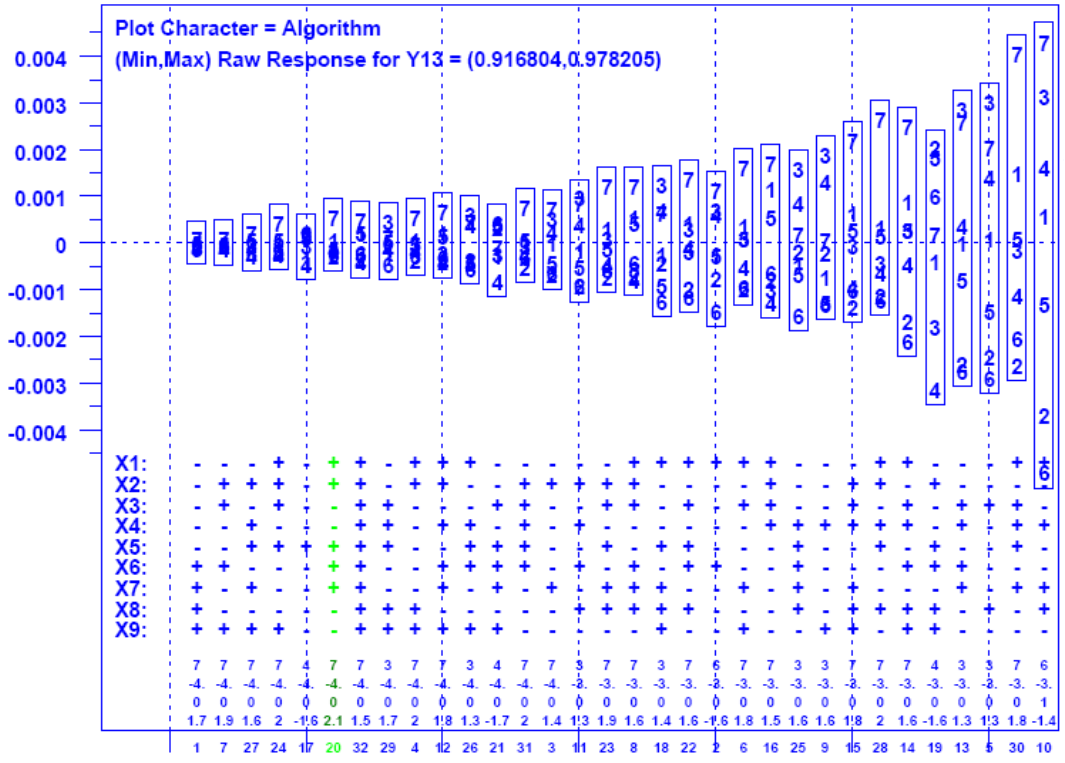


Figure 8-19. Web Objects as Proportion of Flows Completed under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

Figure 8-20 shows the propensity of CTCP (algorithm 2) to generate larger congestion window sizes on average under conditions of low congestion. This behavior was identified in previous experiments (see Chapters 6 and 7).

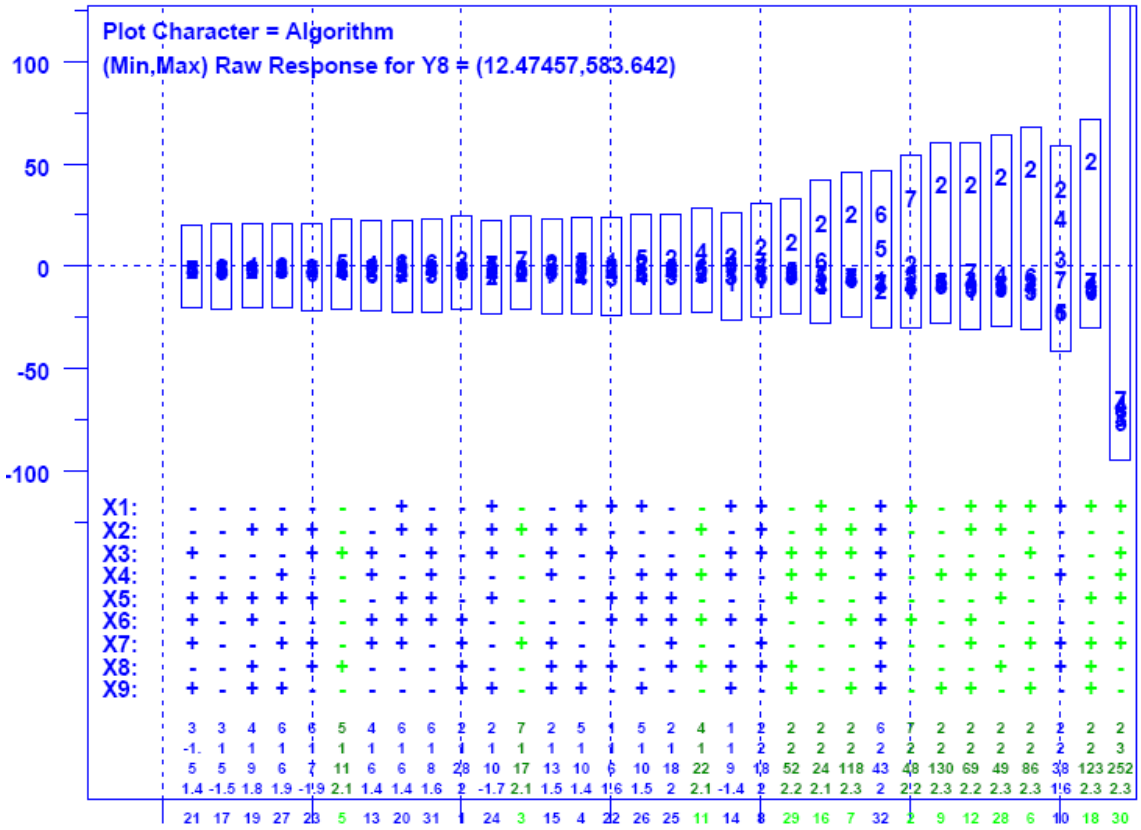


Figure 8-20. Average Flow Congestion Window Size (packets) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals – green columns indicate high statistically significant outliers

8.4.1.2 Low Initial Slow-start Threshold. Setting the initial slow-start threshold to a small value did not much alter the macroscopic behavior reported in the last section. To support this observation we give plots analogous to those shown in Fig. 8-13 to 8-20. In some cases, explained below, we did discern differences. Fig. 8-21 gives a detailed analysis of the average number of active flows under the 32 simulated conditions. As above (Sec. 8.4.1.1), in most conditions, either algorithm 7 (Scalable) or 3 (FAST) shows a higher number of active flows than other algorithms. Fig. 8-22 reveals that FAST and FAST-AT still exhibit a higher number of connecting flows under conditions of higher congestion. Comparing Fig. 8-22 with Fig. 8-24 also shows that Scalable TCP (algorithm 7) no longer exhibits a higher number of connecting flows in many conditions. This appears attributable to lowering the initial slow-start threshold. Previously, Scalable TCP and TCP Reno flows increased transmission rate to the maximum achievable using the same limited slow-start mechanism. This enabled flows to become established and presented difficulties for new flows to connect and to gain an equal congestion window size against established Scalable TCP flows. Lowering the initial slow-start threshold to 100 packets caused both standard TCP and Scalable to enter congestion avoidance (linear increase for

standard TCP; delayed exponential increase for Scalable). During the first two seconds of a flow, Scalable TCP increases its congestion window more slowly than limited slow-start. Thus, under a lower initial slow-start threshold, new (Scalable TCP) flows increased transmission rate more slowly and thus fewer packets (including SYN packets) were lost. This is supported by Fig. 8-24, which shows that Scalable TCP exhibits the highest retransmission rate in only five conditions (instead of 12 conditions as shown in Fig. 8-16).

Fig. 8-23 shows that lowering the initial slow-start threshold allows Scalable TCP to improve its flow completion rate (relative to Fig. 8-15). This occurs for the same reasons the retransmission rate is improved. Figs. 8-23 and 8-24 also show that FAST and FAST-AT continue to exhibit lower flow completion rates and higher retransmission rates under the more congested conditions.

Despite a lower initial slow-start threshold, Scalable TCP exhibits higher buffer occupancy (see Fig. 8-25) than other algorithms under 16 conditions. This effect is somewhat diminished over Fig. 8-17, where Scalable TCP had highest buffer utilization in 20 conditions. Given the delayed increase (compared to limited slow start) in congestion window for Scalable TCP, the high buffer utilization likely arises from large files. Fig. 8-26 shows that FAST (FAST-AT) and Scalable TCP still tend to complete fewer files in aggregate than other algorithms, though the effect is somewhat diminished for Scalable (relative to Fig. 8-18). The lower flow completion totals for FAST (FAST-AT) appear under the most congested conditions.

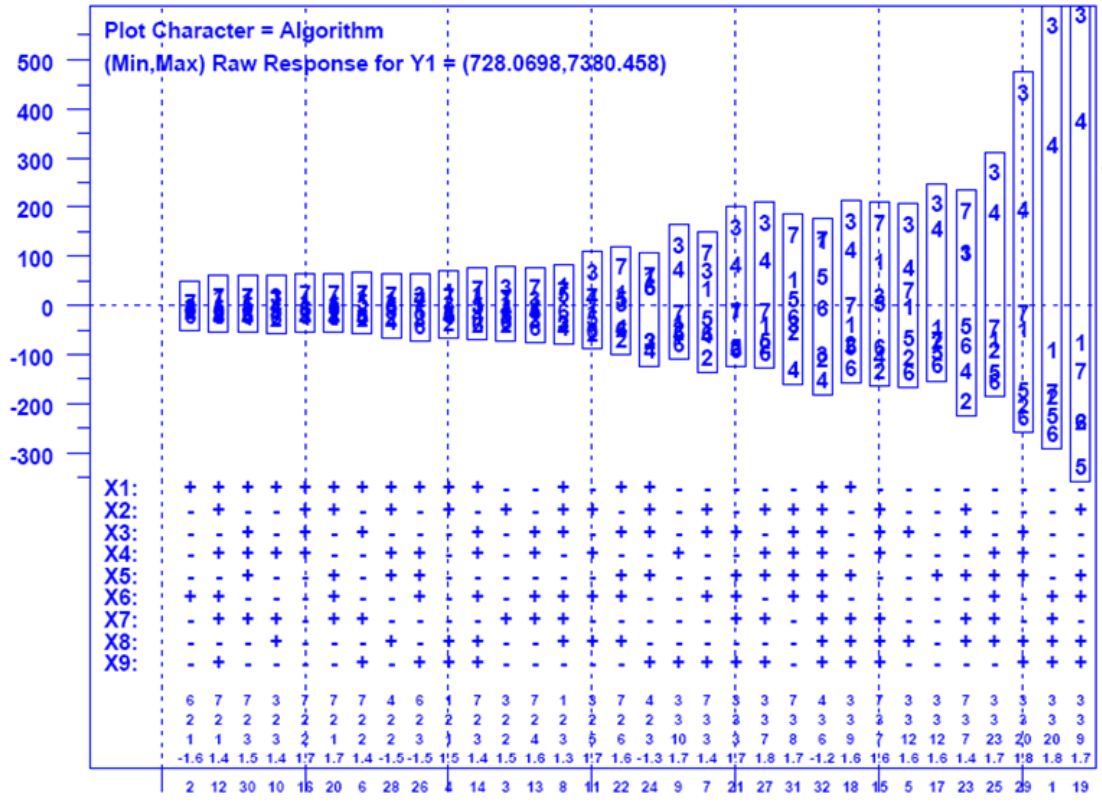


Figure 8-21. Average Number of Active Flows under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

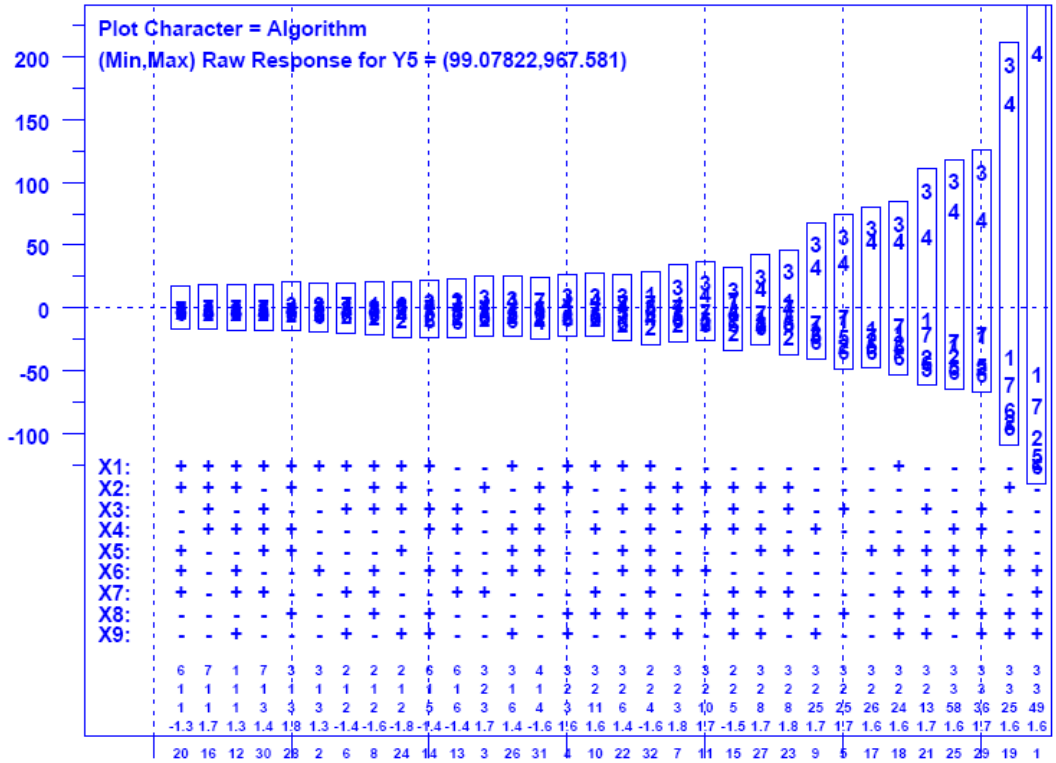


Figure 8-22. Average Number of Connecting Flows under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

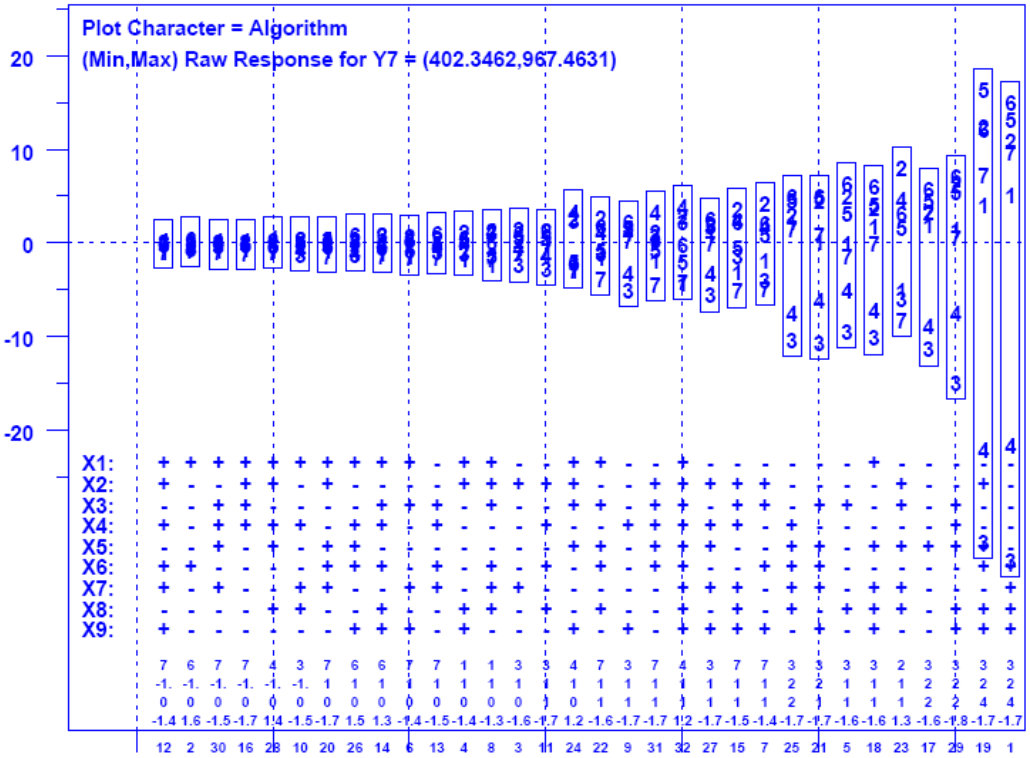


Figure 8-23. Average Rate of Flow Completion (flows per 200 ms) under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

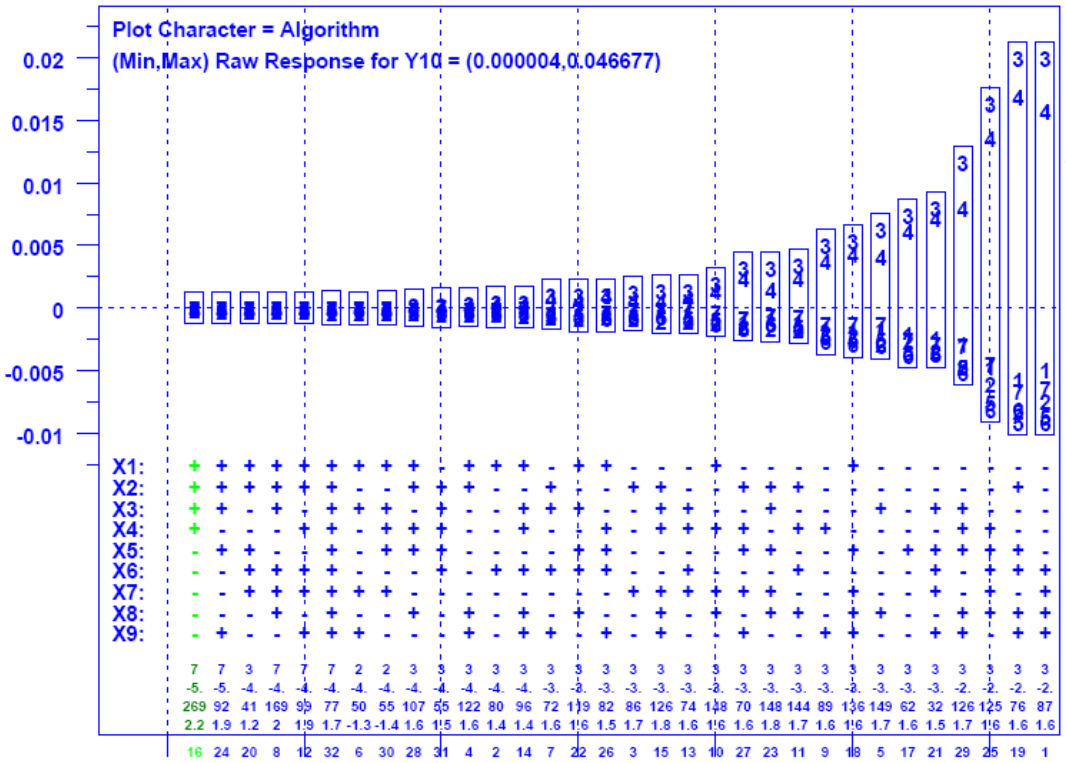


Figure 8-24. Average Flow Retransmission Rate (proportion of packets resent) under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

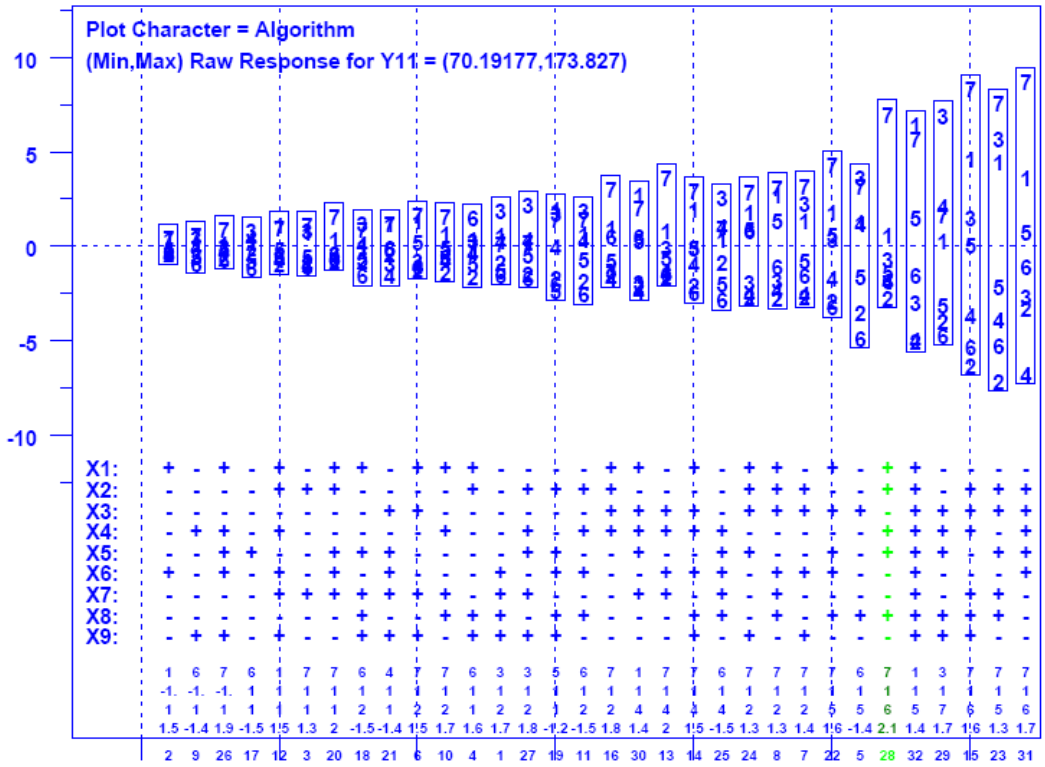


Figure 8-25. Average Smoothed Round-Trip Time (ms) under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

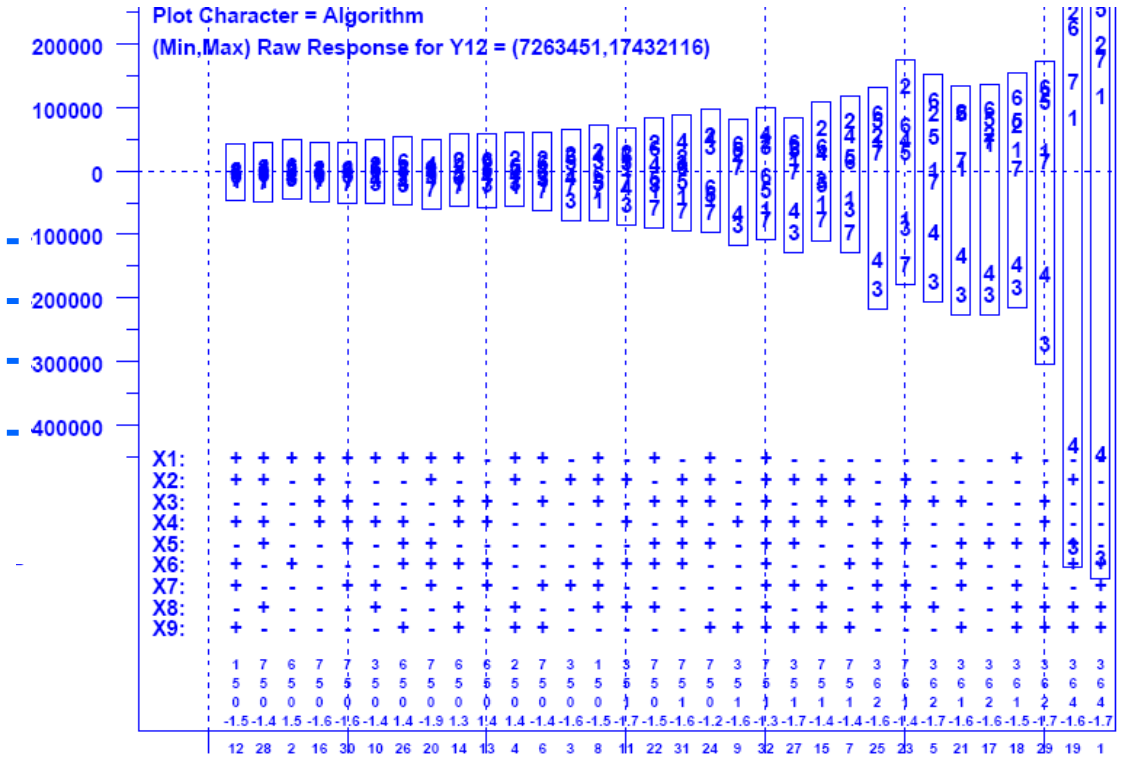


Figure 8-26. Aggregate Flows Completed under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

Fig. 8-27 shows that Scalable TCP completes a higher proportion of flows with small size (i.e., Web objects). This mirrors the result shown earlier in Fig. 8-19. Note, however, Fig. 8-27 reports that FAST (and FAST-AT) tend to complete a smaller proportion of flows with small size. This implies that FAST completes a higher proportion of flows with larger file size. As we demonstrate below (Sec. 8.4.2.2), this occurs because FAST increases transmission rate (after reaching the initial slow-start threshold) to the maximum available much more quickly than other algorithms.

Finally, Fig. 8-28 displays the previously demonstrated propensity of CTCP (algorithm 2) to increase congestion window to large sizes under low congestion. Given that a lower initial slow-start threshold leads to somewhat lower overall congestion (compared with a high threshold), one expects CTCP to stand out more in Fig. 8-28 than in Fig. 8-20. Comparing the two figures verifies this expectation.

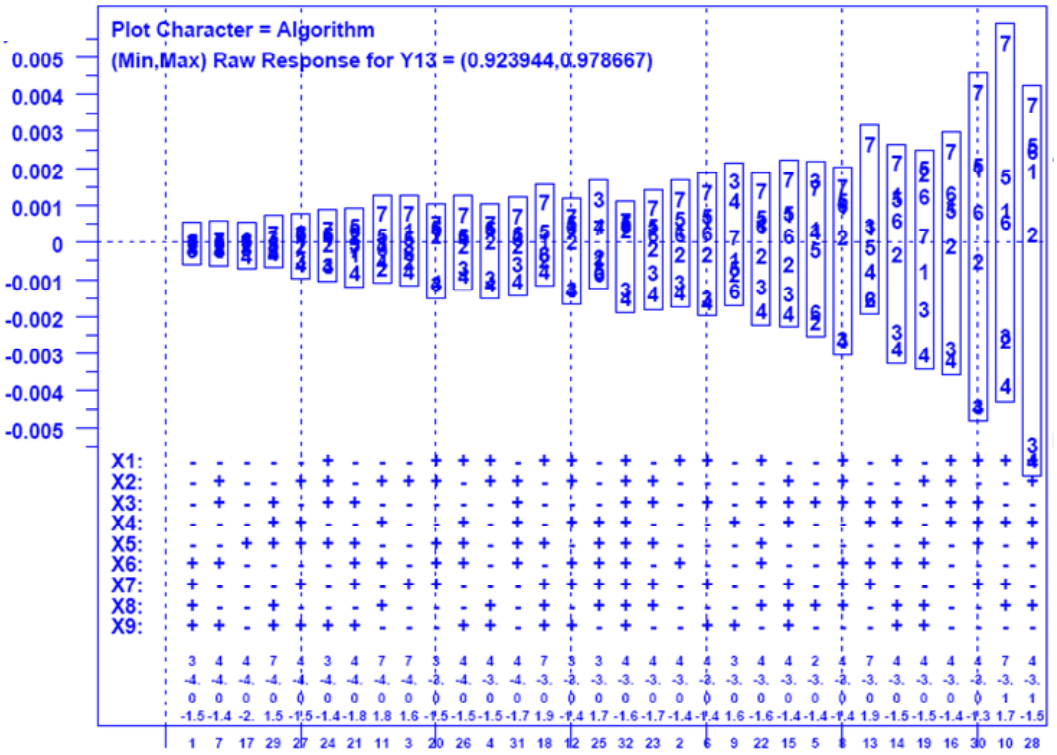


Figure 8-27. Web Objects as Proportion of Flows Completed under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals

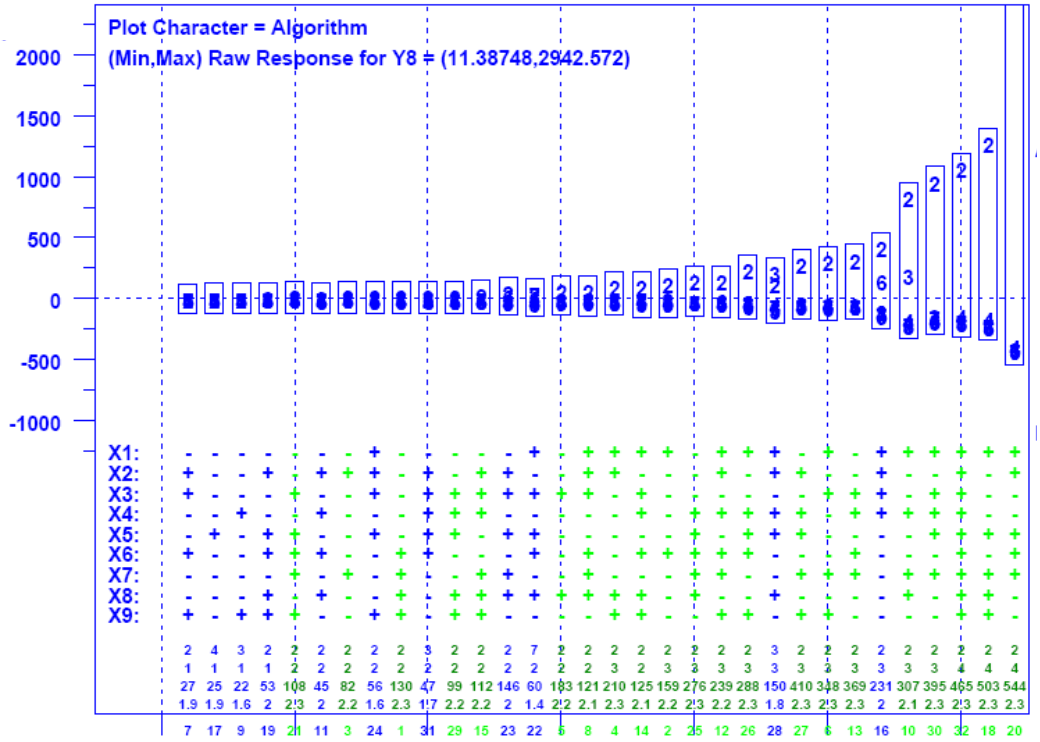


Figure 8-28. Average Flow Congestion Window Size (packets) under Low Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals – columns in green indicate statistically significant high outliers

8.4.2 Absolute User Experience

This section investigates absolute differences in user experience, which we measure as goodput in packets per second. We consider differences in goodput among users of the various alternate congestion control algorithms, as well as differences in goodput among TCP users competing with alternate congestion control algorithms. First, we compare these user experiences given a high initial slow-start threshold and then we compare them given a low initial slow-start threshold.

Table 8-26. Average Goodput (pps) per Flow Group under Each Alternate Congestion Control Algorithm (High Initial Slow-Start Threshold)

			ALTERNATE CONGESTION CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	53650	50696	50299	50325	53212	49826	54638
		N	7869	7846	7859	7819	7857	7834	7807
	F	F	8056	6451	7145	6738	7765	6295	9572
		N	4789	4291	5095	4426	4694	4359	5420
	T	F	4843	4295	5069	4424	4698	3753	5439
		N	3878	3448	4253	3527	3749	3162	4446
SP	VF	F	24911	25410	25555	25727	24675	25274	24694
		N	7262	7313	7340	7346	7242	7295	7168
	F	F	6655	6073	6563	6722	6472	5830	6935
		N	4679	4456	4934	5002	4563	4328	4801
	T	F	4870	4421	5075	5142	4617	4094	5164
		N	4053	3789	4364	4398	3876	3513	4225
D	VF	F	2008	2099	2088	2078	2025	2084	1989
		N	1800	1833	1833	1830	1787	1834	1782
	F	F	1189	1213	1175	1203	1201	1220	1174
		N	1124	1149	1113	1140	1138	1162	1111
	T	F	1308	1315	1291	1310	1313	1330	1293
		N	1254	1264	1240	1259	1261	1281	1240
WO	VF	F	366	390	378	360	427	428	378
		N	384	395	395	394	382	394	379
	F	F	255	261	250	256	258	263	252
		N	250	256	245	251	253	258	247
	T	F	308	313	301	306	312	318	306
		N	303	307	296	300	307	312	300

8.4.2.1 High Initial Slow-start Threshold. Table 8-26 summarizes the average goodput – response y2(u) – experienced by users in each of the 24 flow classes (dimensioned by file size, path quality and interface speed) under each of the seven alternate congestion control algorithms. Table 8-27 provides a similar summary of the average goodput – response y16(u) – experienced by TCP users in each of the 24 flow classes when competing with flows in each of the seven alternate congestion control algorithms. Since the tables are somewhat dense with numbers, we present this information in the form of bar graphs (Fig. 8-29 through 8-32) – one figure per file size: movie, service pack, document and Web object. (The legend for the bar graphs is shown in Fig. 8-7.) The top

row of graphs in each figure displays the average goodput in packets per second (pps), while the bottom row of graphs displays average goodput as a proportion of the maximum interface speed. When examined vertically, the first two columns of graphs consider flows transiting very fast (VF) paths, the second two columns consider flows transiting fast (F) paths and the final two columns consider flows transiting typical (T) paths. Within a given path class, the first vertical sub-column reports goodput for flows with fast (F) interface speeds (80×10^3 pps), while the second vertical sub-column reports goodput for flows with normal (N) interface speeds (8×10^3 pps). Each graph is labeled with the relevant path class and interface speed (e.g., VF-F).

Table 8-27. Average Goodput (pps) per Flow Group on TCP Flows Competing with Each Alternate Congestion Control Algorithm (High Initial Slow-Start Threshold)

			ALTERNATE CONGESTION CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	44610	47731	47899	48628	44397	47110	43551
		N	7575	7800	7811	7819	7504	7731	7206
	F	F	4730	5032	4843	5138	4770	5072	4339
		N	3146	3768	3300	3514	3301	3670	3047
	T	F	3184	3108	2956	2970	3099	3327	2994
		N	2664	2971	2478	2727	2557	2852	2459
SP	VF	F	23697	24837	24991	25068	23441	24667	23687
		N	7149	7275	7302	7307	7136	7286	6946
	F	F	5210	5582	5301	5504	5425	5709	5119
		N	3837	4159	3894	3998	3970	4144	3732
	T	F	3724	3908	3722	3772	3796	3919	3695
		N	3205	3366	3182	3224	3268	3410	3163
D	VF	F	1961	1996	2025	2027	1919	2037	1978
		N	1783	1822	1819	1818	1776	1829	1765
	F	F	1173	1205	1141	1178	1195	1221	1148
		N	1109	1142	1079	1108	1128	1152	1089
	T	F	1277	1305	1240	1264	1300	1328	1263
		N	1228	1256	1193	1212	1251	1278	1213
WO	VF	F	394	378	359	458	382	431	358
		N	378	386	385	388	377	387	373
	F	F	254	260	248	254	257	262	250
		N	249	255	243	249	253	257	246
	T	F	306	312	298	303	311	317	304
		N	302	307	293	298	306	312	299

Figs. 8-29 to 8-32 reveal some obvious points. First, differences in goodput among alternate algorithms appear more evident with the largest files (movies). Second, differences in goodput between TCP flows and competing alternate flows appear with larger files (movies and service packs) and on paths with the most congestion (Fast and Typical). In general, differences in goodput can originate from four sources: (1) the maximum transfer rate, (2) how fast a flow reaches the maximum rate, (3) file size and (4) how a flow responds to losses. Here, we ensure that all flows move toward maximum

transfer rate at the same speed (by using limited slow-start until the first packet loss). We devote each figure to only one file size. This means that any goodput differences in each figure (for Figs. 8-28 to 8-32) can result only from loss processing. In other words, how much does a flow slow its transmission rate after a loss and how quickly does it recover? We expect all alternate congestion control algorithms to improve over TCP Reno with respect to processing losses, so we expect differences to appear on congested paths and on larger flows which exhibit a larger probability of loss/recovery events. Flows transmitting small files should not experience as many loss/recovery cycles as flows transmitting large files. Similarly, flows crossing uncongested paths should not experience as many loss/recovery cycles as flows transiting congested paths.

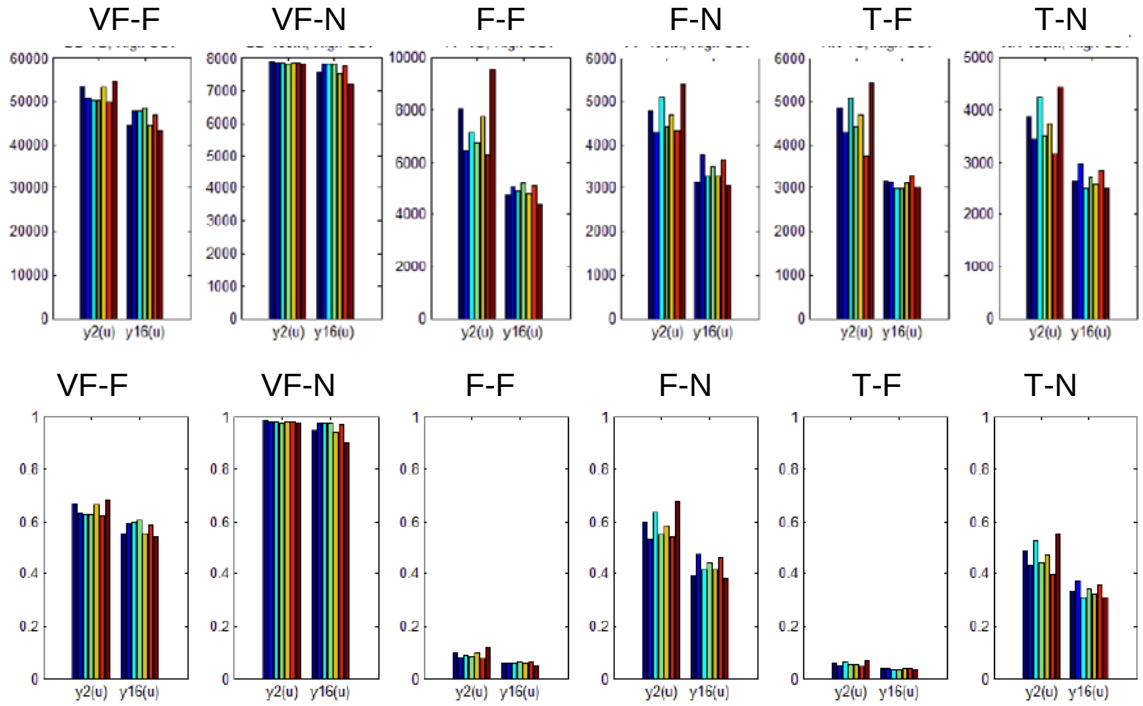


Figure 8-29. Average Goodput on Movies (High Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

Though Figs. 8-29 to 8-32 reveal some modest differences in goodput among flows groups based on congestion control algorithm, we suspected that more significant goodput variations in the data would be explained by differences in experiment conditions. To investigate, we conducted a principal components analysis (PCA) of the average goodput data across all flow groups. Fig. 8-33 plots the resulting information, which reveals three main groups: (1) a group where network speed is low (factor x1 = -1), (2) a group where network speed is high (factor x1 = +1) and propagation delay is high (factor x2 = +1) and (3) a group where network speed is high and propagation delay is low (factor x2 = -1). Each of the latter two groups could be divided into two subgroups based on average file size: (a) smaller (x5 = -1) and (b) larger (x5 = +1). No distinct collection of congestion control algorithms appears anywhere in Fig. 8-33. This suggests that most of the variation in the data under a high initial slow-start threshold arises from network speed, propagation delay and file size. The congestion control algorithm has

only a minor opportunity to affect goodput because network conditions are mainly uncongested and flows experience relatively few loss/recovery cycles.

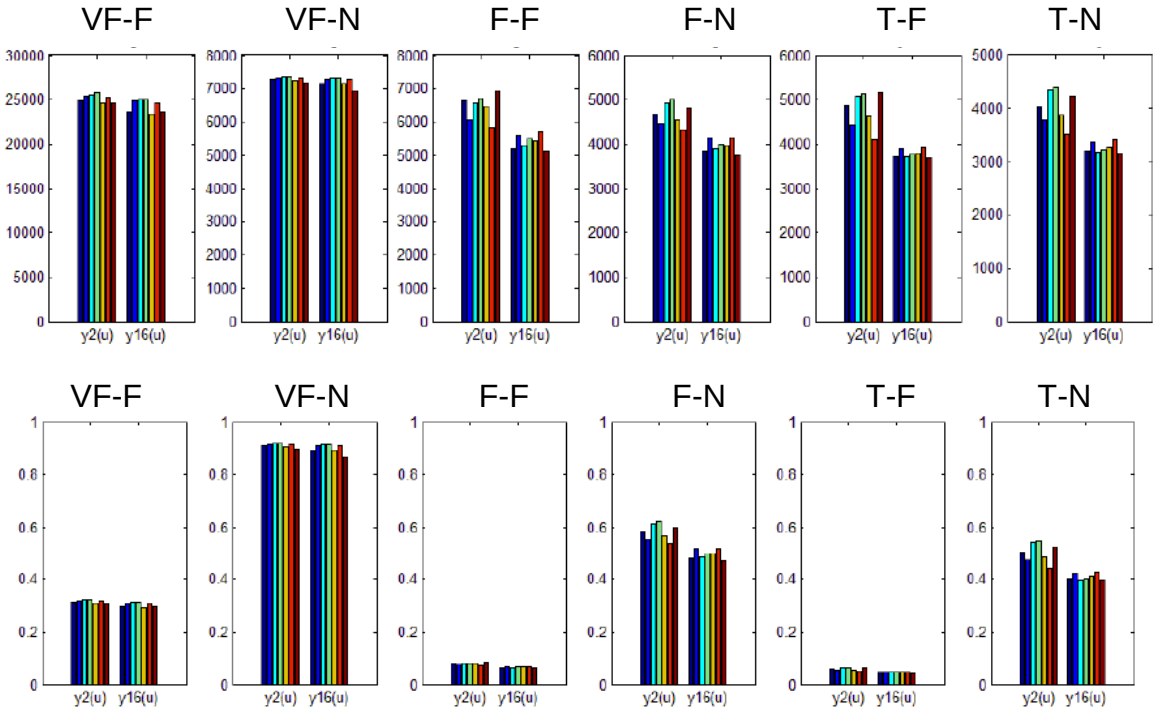


Figure 8-30. Average Goodput on Service Packs (High Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

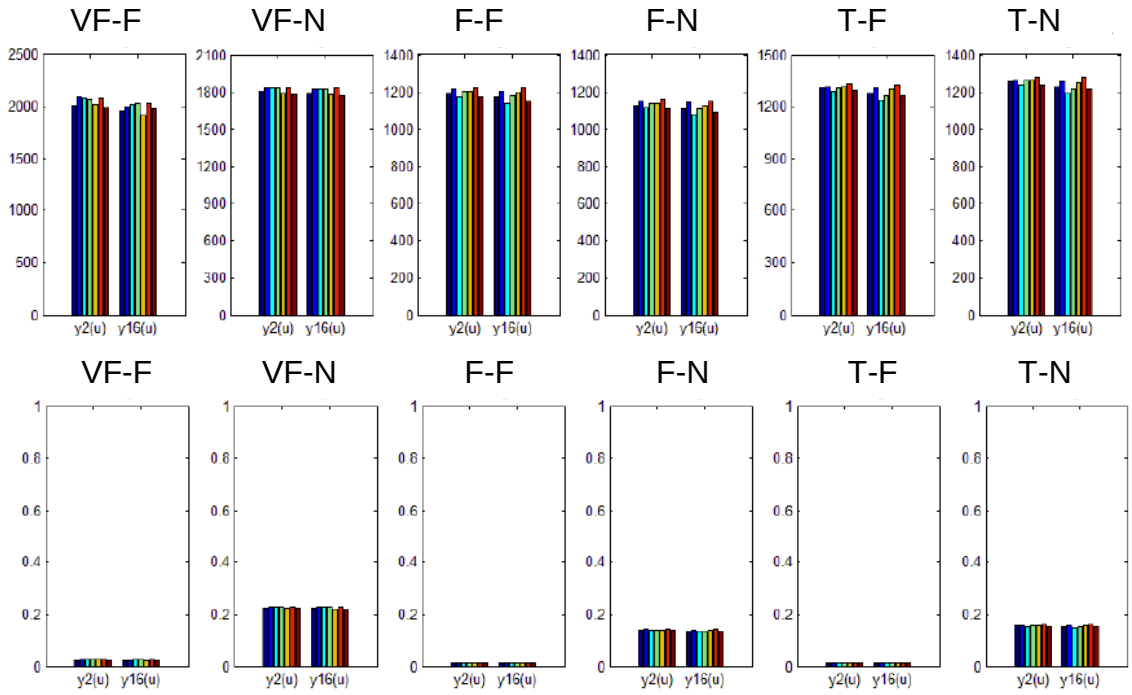


Figure 8-31. Average Goodput on Documents (High Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

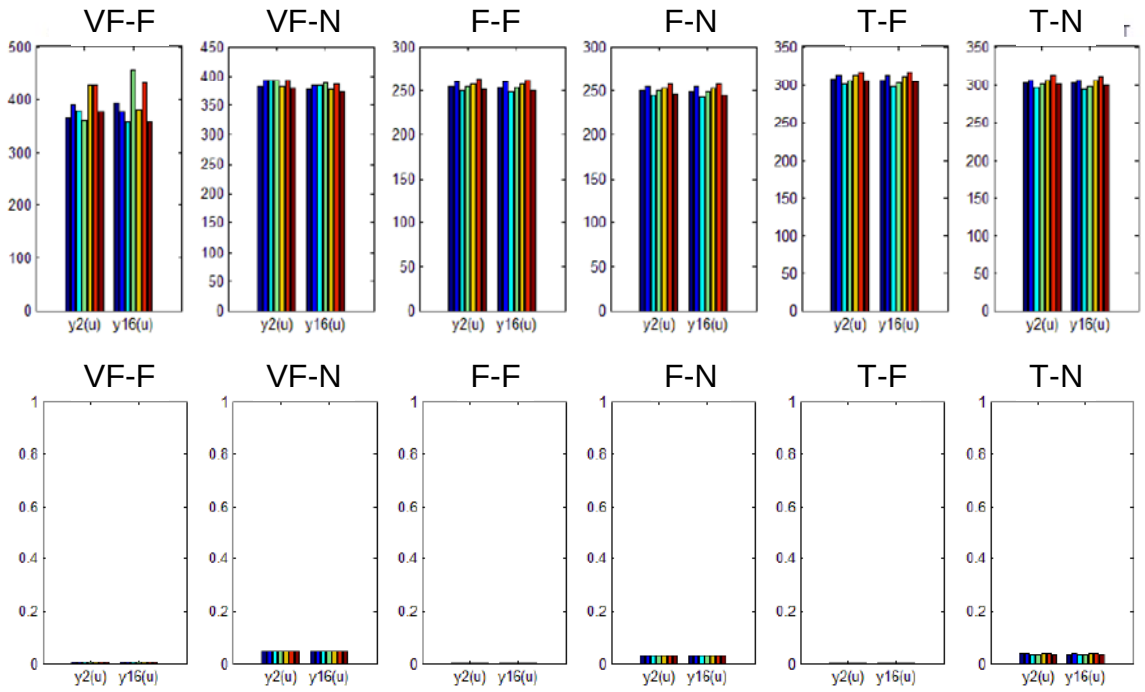
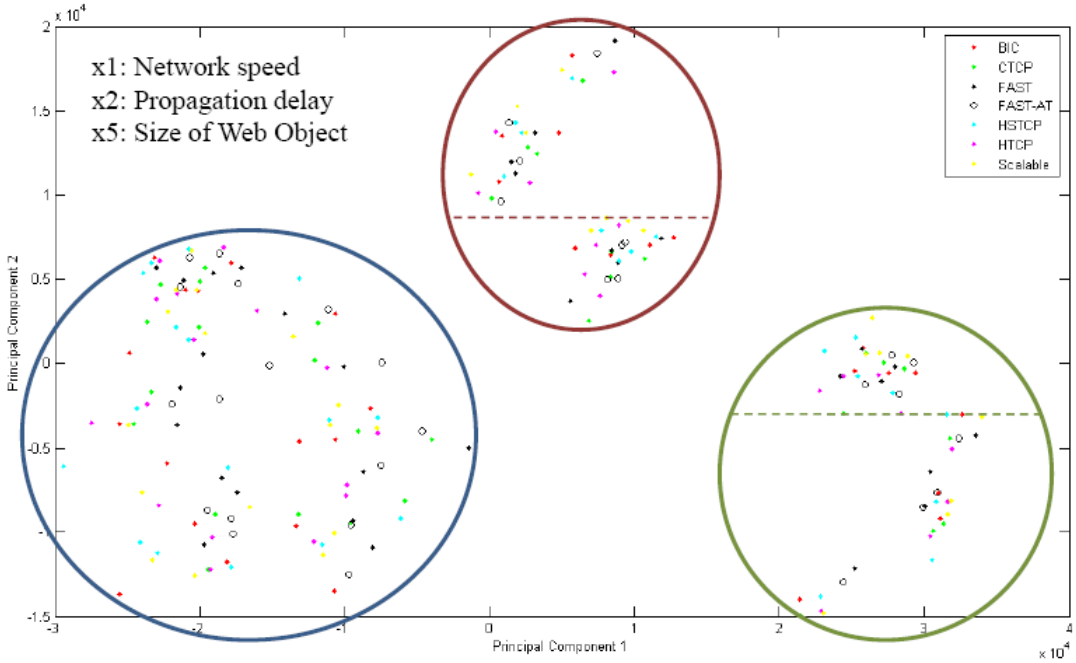


Figure 8-32. Average Goodput on Web Objects (High Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)



- Group 1: odd conditions (x1=-1)
- Group 2: conditions 4, 8, 12 ,16, 20, 24, 28, 32 (x1=1, x2=1)
  ↑[x5=-1]↓[x5=1]
- Group 3: conditions 2, 6, 10 ,14, 18, 22, 26, 30 (x1=1, x2=-1)
  ↑[x5=-1]↓[x5=1]

Figure 8-33. Principal Component 1 (x axis) vs. Principal Component 2 (y axis) from Average Goodput Data (High Initial Slow-Start Threshold)

Given that most differences in goodput arise from differences in network conditions, we can still analyze the modest goodput differences that can be attributed to congestion control algorithms. Fig. 8-34 gives seven scatter plots, each showing TCP goodput (y axis) vs. goodput (x axis) on an alternate (as labeled) congestion control algorithm for movies transferred on very fast paths with a fast interface speed. Each point represents one of the 32 simulated conditions. The diagonal would represent the case where TCP flows and alternate flows achieved identical goodput for the same condition. Points falling below the diagonal indicate flows using the alternate algorithm had higher goodput; points falling above indicate TCP flows had higher goodput. Each plot is also labeled (in red) with the computed correlation between goodput on TCP flows and alternate flows.

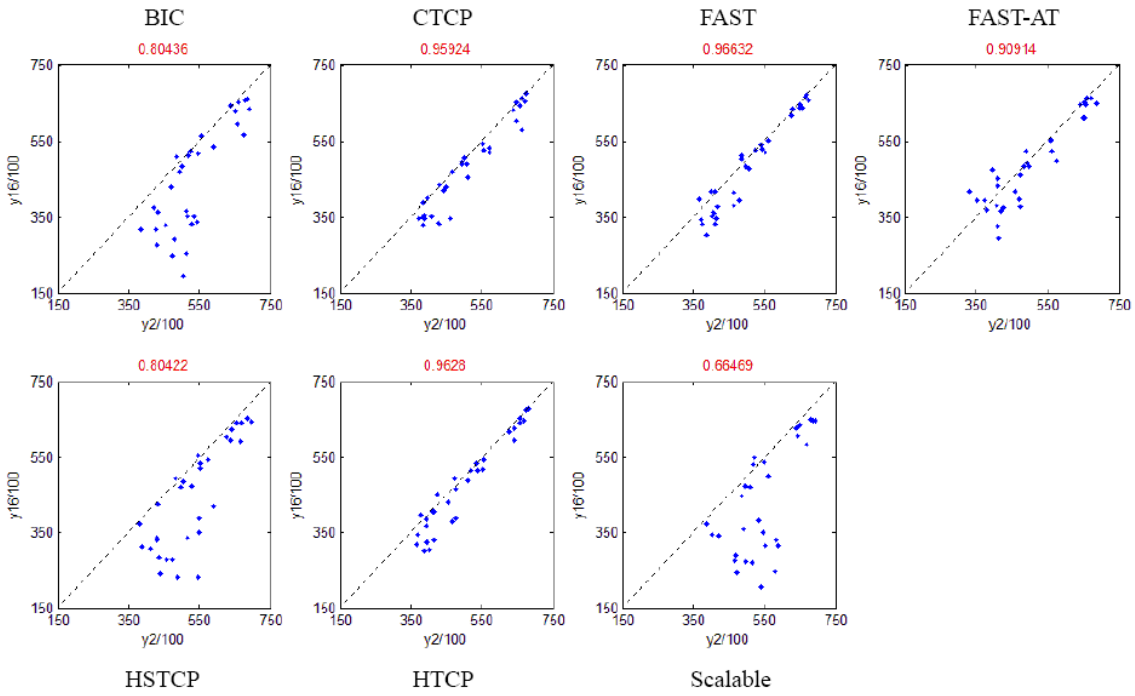


Figure 8-34. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Movies Transferred on Very Fast Paths with Fast Interfaces (High Initial Slow-Start Threshold)

Fig. 8-34 reveals that under many conditions, Scalable TCP, HSTCP and BIC flows achieve significantly higher goodputs than competing TCP flows when sending movies over very fast paths with fast interfaces. This mirrors the information shown in Fig. 8-29, which plots average goodputs, and shows that Scalable, HSTCP and BIC flows achieve higher goodputs at the expense of competing TCP flows.

Fig. 8-35 shows the specific conditions under which goodput on Scalable, HSTCP and BIC flows exceed goodput on TCP flows. Each bar graph in Fig. 8-35 represents all seven alternate congestion control algorithms under a specific condition (shown in the lower left-hand corner of each plot). The algorithms are rendered from leftmost bar to rightmost bar ordered by algorithm identifier (1-7). Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows

– y2(u). If the bar is red, y16(u) is greater; if the bar is green, y2(u) is greater. The 32 bar graphs are sorted from least to most congestion.

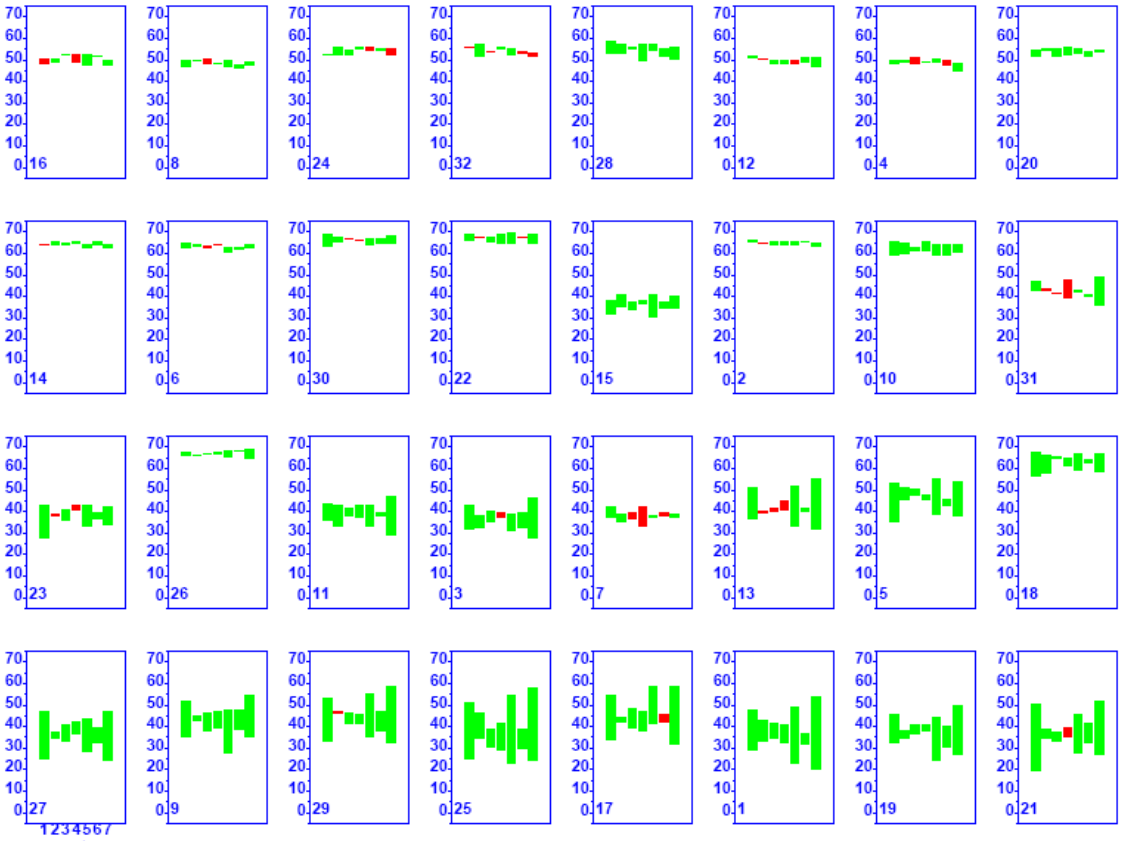


Figure 8-35. 32 Bar Graphs (one for each Simulated Condition) plotting Goodput (pps/1000) on TCP Flows vs. Non-TCP Flows for Movies Transferred on Very Fast Paths with Fast Interfaces (High Initial Slow-Start Threshold) (Each graph contains seven bars, one per congestion control algorithm, ordered left to right by algorithm identifier. Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows – y2(u). If the bar is red, y16(u) is greater; if the bar is green, y2(u) is greater. The 32 bar graphs are sorted from least to most congestion by condition, as indicated in the lower left-hand corner of each plot.)

Fig. 8-35 reveals scant differences in goodput between TCP flows and alternate flows under the 16 least congested conditions. Differences in goodput between alternate flows and TCP flows increase with increasing congestion for BIC, HSTCP and Scalable. This reveals that aspects of loss/recovery processing implemented by BIC, HSTCP and Scalable penalize TCP flows. As discussed previously, in Chapter 6, Scalable TCP (along with BIC and HSTCP) reduce congestion window size much less than TCP flows in response to a single loss, so once a Scalable flow establishes a large congestion window and related buffer space along a path, it could take many loss events to significantly reduce the flow's transmission rate. TCP flows, on the other hand, reduce the congestion window by half on each loss and thus TCP flows reduce transmission rate much faster than Scalable TCP flows.

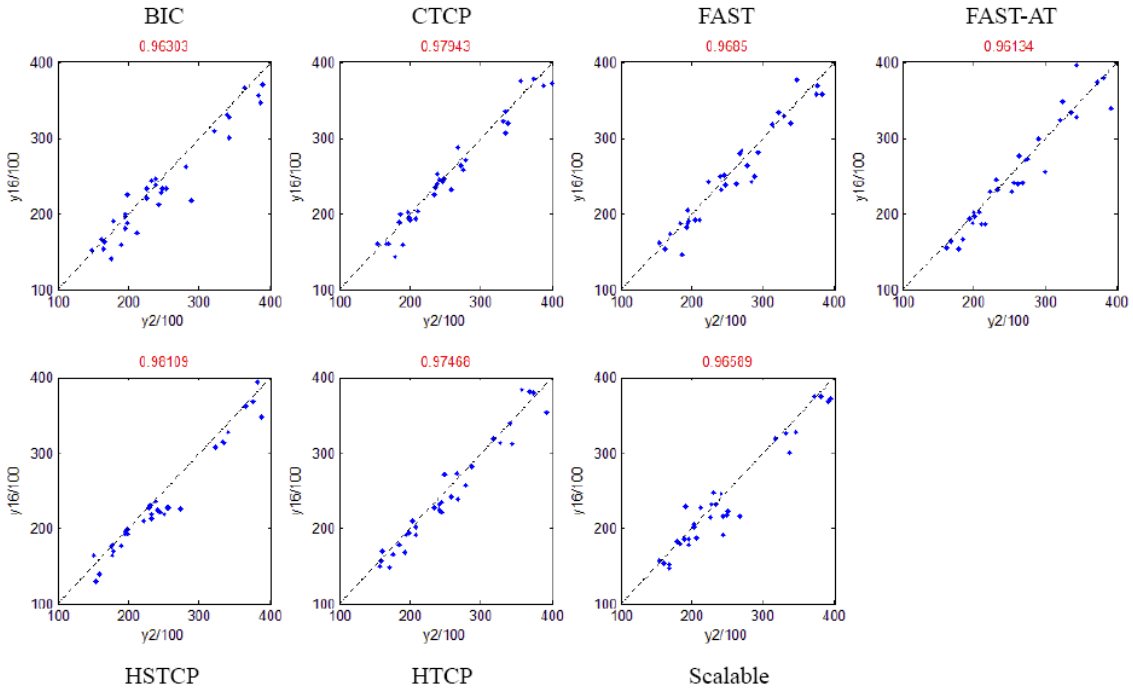


Figure 8-36. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Service Packs Transferred on Very Fast Paths with Fast Interfaces (High Initial Slow-Start Threshold)

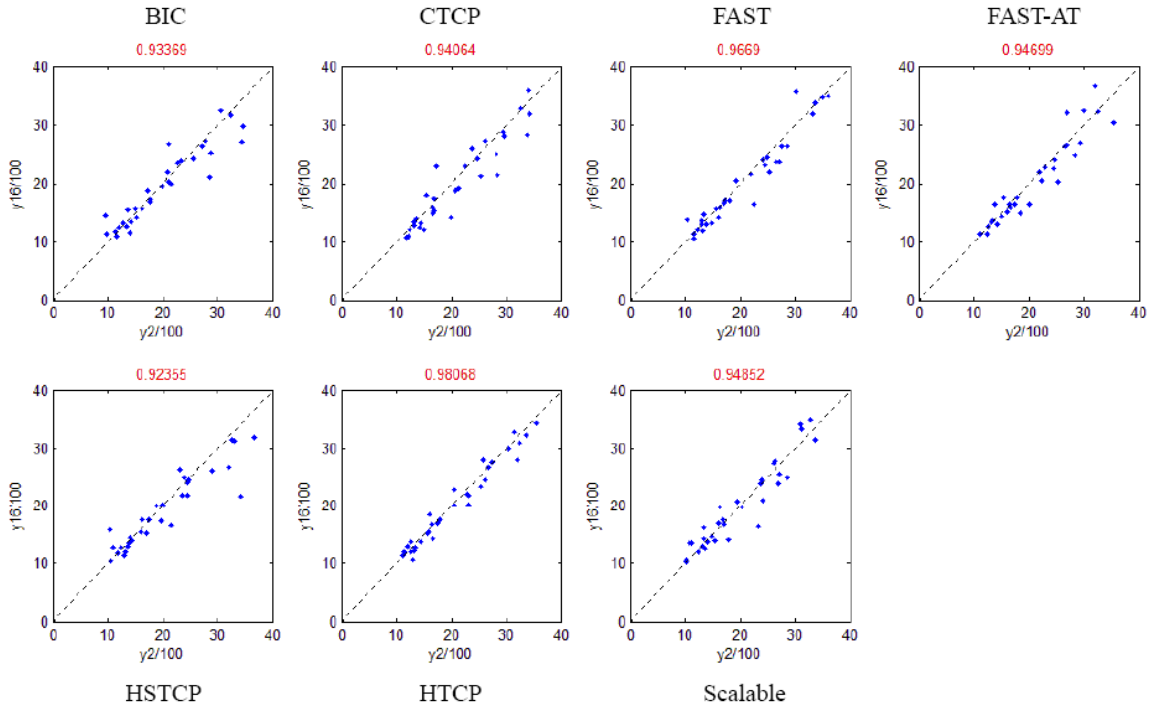


Figure 8-37. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Documents Transferred on Very Fast Paths with Fast Interfaces (High Initial Slow-Start Threshold)

The effects shown in Figs. 8-34 and 8-35 do not appear for smaller file sizes transmitted over very fast paths and fast interfaces. This is shown in Fig. 8-36 (for service packs) and Fig. 8-37 (for documents). Careful examination of Fig. 8-36 suggests a small tendency for BIC, HSTCP and Scalable to discriminate against TCP flows. The tendency exists for the reasons discussed above (with respect to movies), but the tendency is much muted because flows sending service packs have fewer opportunities to invoke loss/recovery processing. For this reason, the tendency for BIC, HSTCP and Scalable TCP to discriminate against TCP flows fades with file size (as shown in Fig. 8-37).

Table 8-28. Average Goodput (pps) per Flow Group under Each Alternate Congestion Control Algorithm (Low Initial Slow-Start Threshold)

			ALTERNATE CONGESTION CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	36750	44935	55246	55557	34448	31385	34904
		N	7736	7780	7921	7913	7537	7569	7429
	F	F	7912	6851	7169	6790	7499	6543	8336
		N	5142	4507	5144	4470	4840	4514	5297
	T	F	5217	4516	5185	4664	4840	4317	5444
		N	4270	3844	4229	3931	4013	3710	4531
SP	VF	F	13318	15578	29337	29315	10790	8897	9933
		N	6493	6765	7533	7526	5872	5759	5482
	F	F	4869	4672	6832	7023	4196	4024	4018
		N	3974	3796	5066	5139	3519	3488	3383
	T	F	4275	4045	5332	5364	3800	3504	3821
		N	3767	3580	4493	4519	3392	3215	3368
D	VF	F	1669	1682	2464	2406	1623	1589	1562
		N	1607	1653	2008	2009	1553	1546	1524
	F	F	987	1016	1300	1329	965	956	934
		N	959	997	1219	1241	939	934	911
	T	F	1147	1174	1403	1418	1126	1119	1108
		N	1120	1148	1336	1352	1102	1095	1083
WO	VF	F	431	391	423	405	384	395	392
		N	405	408	415	419	399	407	396
	F	F	253	258	261	265	255	258	251
		N	249	254	255	260	252	254	247
	T	F	310	316	313	316	314	317	311
		N	305	311	307	310	309	312	306

8.4.2.2 Low Initial Slow-start Threshold. Table 8-28 summarizes the average goodput – response y2(u) – experienced by users in each of the 24 flow classes (dimensioned by file size, path class and interface speed) under each of the seven alternate congestion control algorithms. Table 8-29 provides a similar summary of the average goodput – response y16(u) – experienced by TCP users in each of the 24 flow classes when competing with flows in each of the seven alternate congestion control algorithms. Since the tables are somewhat dense with numbers, we present this information in the form of bar graphs (Figs. 8-38 through 8-41) – one figure per file size: movie, service pack, document and Web object. (These figures are laid out in the same fashion as Figs. 8-29 through 8-32.)

Tables 8-28 and 8-29, as well as Figs. 8-38 to 8-40, show a marked increase in goodput differences among flows using alternate congestion control algorithms and between flows using alternate congestion control algorithms and flows using TCP. These increased differences must arise from reducing the initial slow-start threshold to a low value, as all other aspects of the simulations remained the same.

Table 8-29. Average Goodput (pps) per Flow Group on TCP Flows Competing with Each Alternate Congestion Control Algorithm (Low Initial Slow-Start Threshold)

			ALTERNATE CONGESTION CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	16053	16621	16951	16774	16279	16833	16228
		N	7014	7068	7065	7080	6968	6958	6857
	F	F	4532	4821	4246	4330	4651	4859	4253
		N	3286	3756	3383	3282	3542	3468	3406
	T	F	3380	3822	3098	3158	3662	3580	3451
		N	2963	3298	2780	2832	3125	3240	3028
SP	VF	F	6484	6531	6563	6494	6498	6636	6456
		N	4838	4939	4950	4959	4847	4888	4771
	F	F	2872	2936	2709	2762	2886	3037	2818
		N	2569	2717	2520	2523	2642	2704	2589
	T	F	2811	2916	2562	2627	2872	2941	2861
		N	2592	2738	2391	2444	2668	2730	2652
D	VF	F	1504	1528	1521	1521	1493	1561	1520
		N	1509	1516	1518	1514	1504	1524	1500
	F	F	919	941	899	913	939	950	920
		N	897	920	873	892	914	929	897
	T	F	1076	1098	1031	1043	1098	1113	1084
		N	1054	1076	1009	1023	1077	1091	1063
WO	VF	F	379	404	389	396	396	392	385
		N	396	397	397	397	388	396	388
	F	F	250	255	246	250	254	258	250
		N	246	251	242	246	250	253	246
	T	F	307	312	298	301	312	316	310
		N	303	308	294	297	308	312	305

Figs. 8-38 to 8-41 reveal some obvious points. First, flows using alternate congestion control algorithms often achieve much higher goodputs than flows using TCP congestion control. The differences increase with file size and with interface speed. For the smallest size (Web objects, Fig. 8-41) there is no appreciable goodput difference among flows. Second, FAST and FAST-AT flows achieve markedly higher goodputs than flows using the other alternate congestion control protocols. The ability of FAST flows to achieve higher goodputs must arise from differences in congestion window increase procedures after a flow reaches the initial slow-start threshold. Third, the tendency of Scalable TCP, BIC and HSTCP flows to discriminate against TCP flows

when competing on congested paths, though muted, is still evident, especially for the largest files (movies).

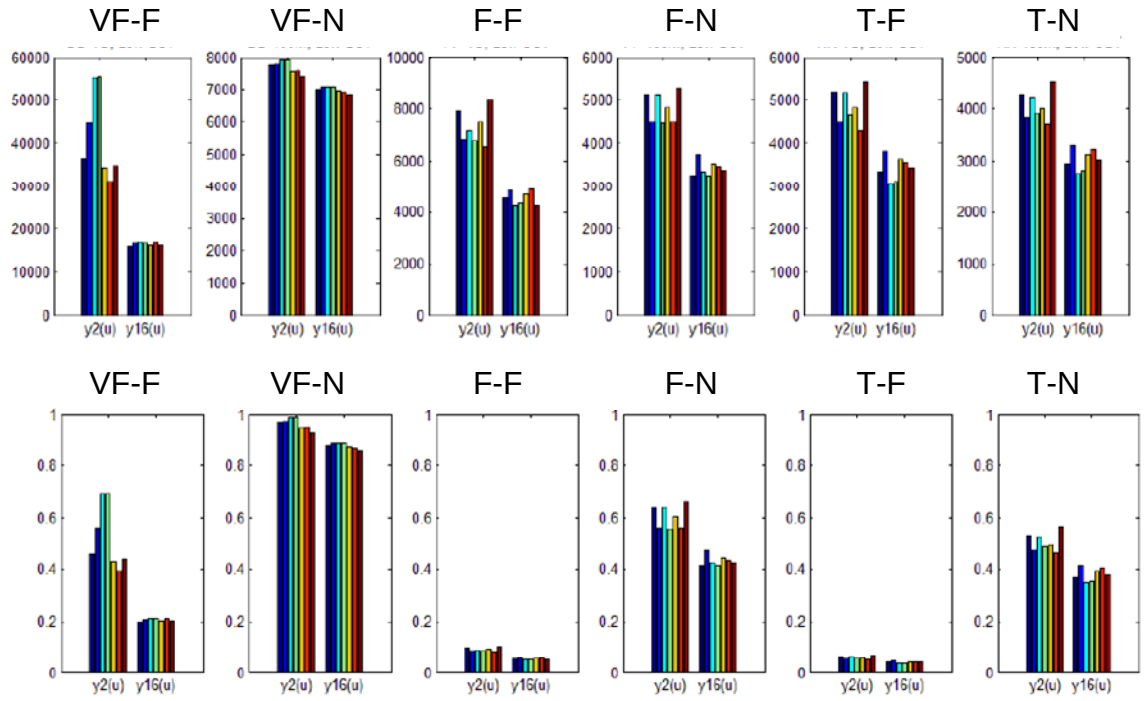


Figure 8-38. Average Goodput on Movies (Low Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

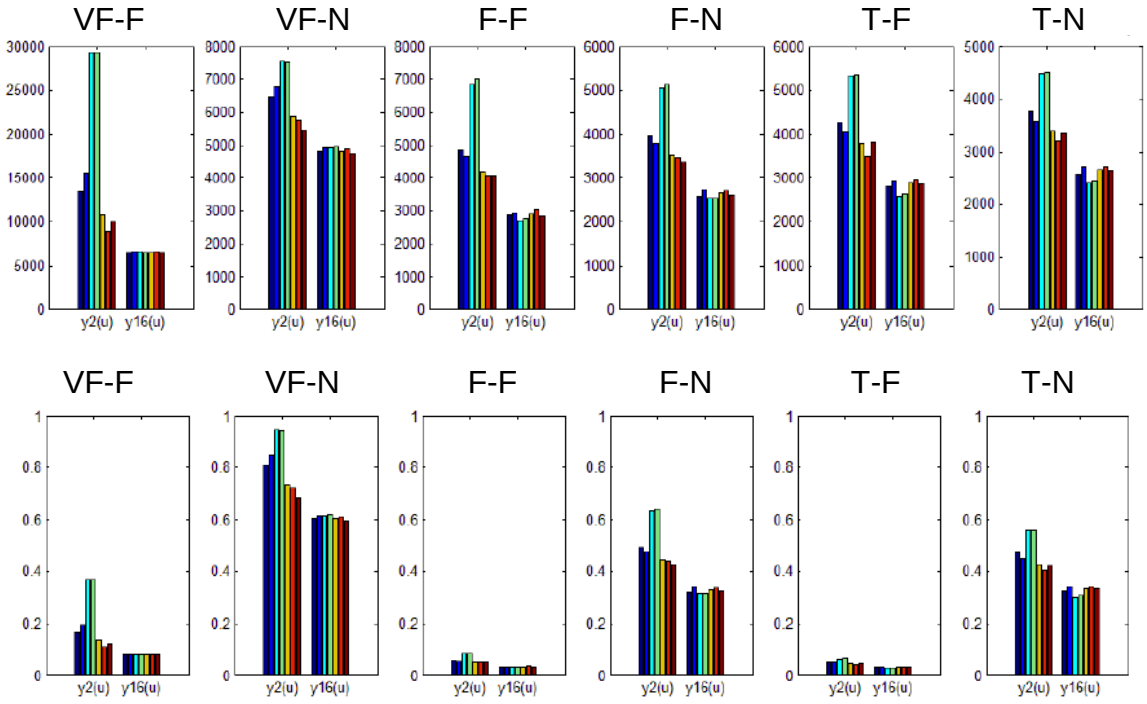


Figure 8-39. Average Goodput on Service Packs (Low Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

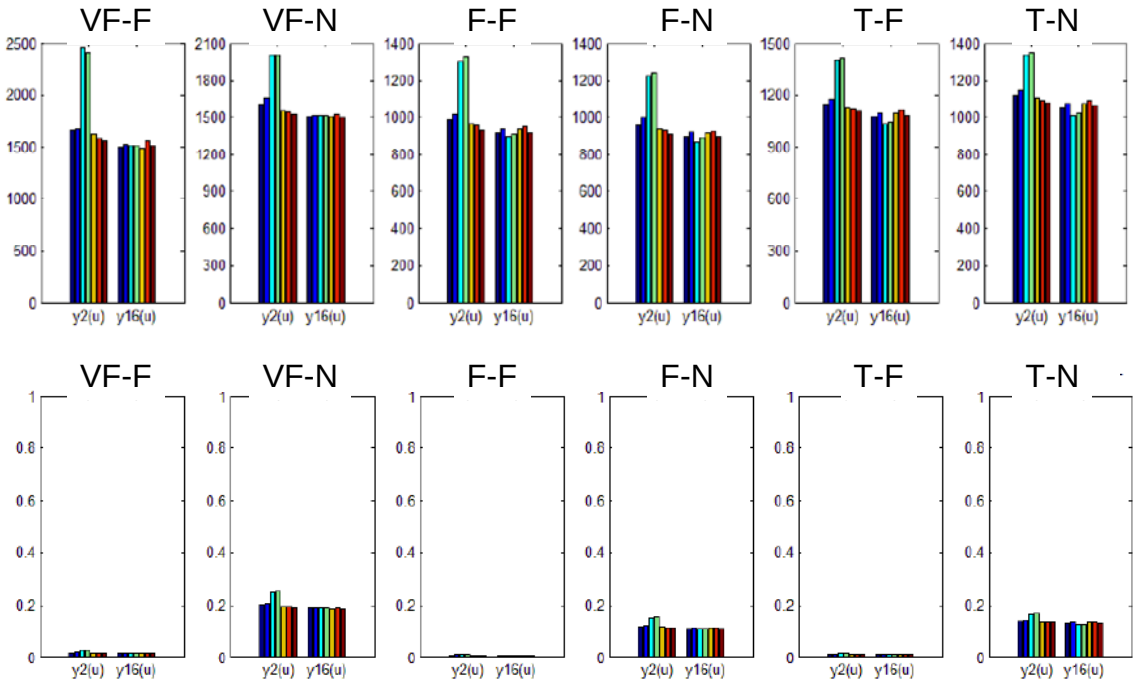


Figure 8-40. Average Goodput on Documents (Low Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

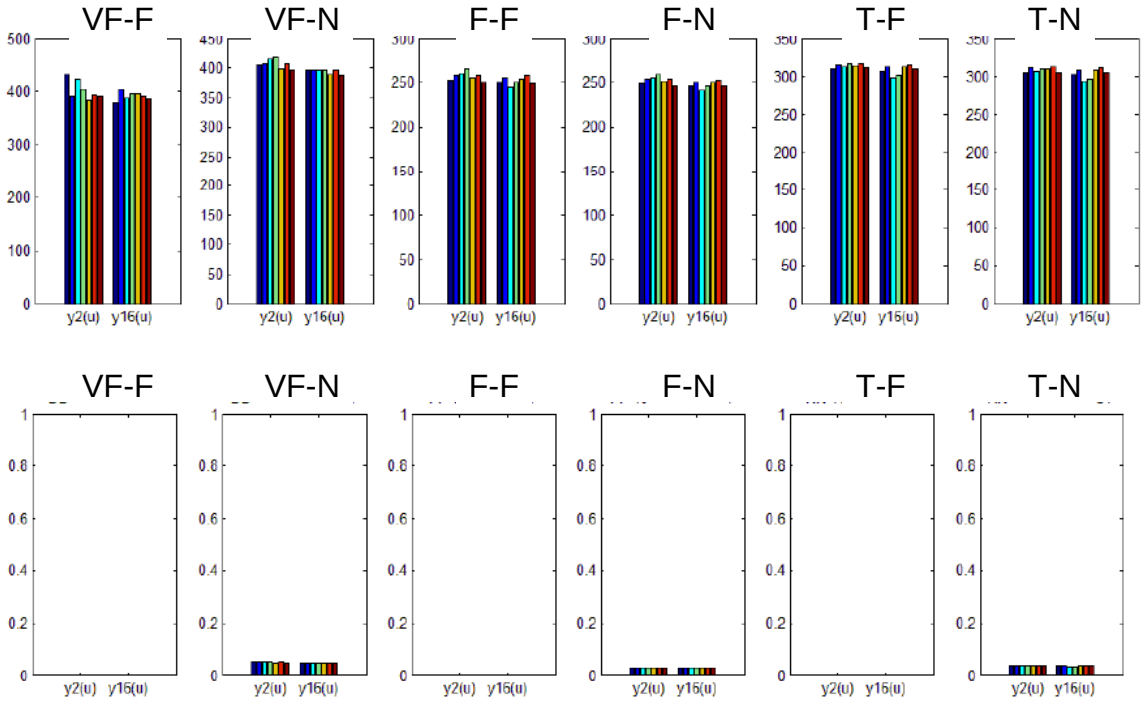
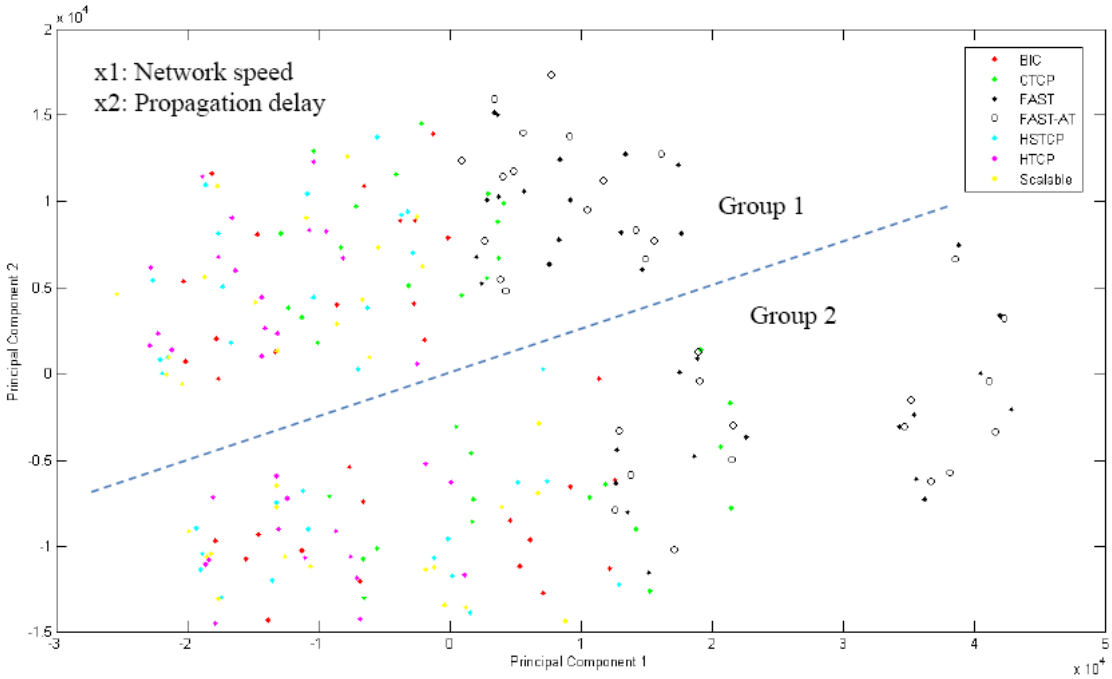


Figure 8-41. Average Goodput on Web Objects (Low Initial Slow-Start Threshold) (Top row shows raw goodput in pps and bottom row shows goodput as a proportion of interface speed)

Though Figs. 8-38 to 8-40 reveal strong differences in goodput for flow groups using FAST and FAST-AT, we wanted to investigate to what extent differences in experiment conditions drove differences in goodput. To investigate this question, we conducted a principal components analysis (PCA) of the average goodput data across all flow groups. Fig. 8-42 plots the resulting information (a biplot of the first two principal components), which reveals two main groups of points: (1) goodput when network speed was lower (x1 = -1) and (2) goodput when network speed was higher (x1 = +1). This is as expected: higher network speeds enable higher goodputs. Fig. 8-42 also reveals differences with respect to congestion control algorithm. Note that goodputs for flows using algorithm 3 (FAST) and algorithm 4 (FAST-AT) tend toward the right-hand side of the plot and there is a rightmost grouping of points associated with FAST and FAST-AT.[5] These points represent cases when network speed is high and propagation delay is low (x2 = -1). This suggests that FAST and FAST-AT can achieve significantly higher goodputs than other congestion control algorithms under such conditions.



- Group 1: odd conditions (x1=-1)
- Group 2: even conditions (x1 = 1)
  - Algorithms 3 and 4 are distinctive especially in conditions 2, 6, 10, 14, 18, 22, 26, 30 (x2 = -1)

Figure 8-42. Principal Component 1 (x axis) vs. Principal Component 2 (y axis) for Average Goodput Data (Low Initial Slow-Start Threshold)

[5] Though the data included goodput for TCP flows, differences in goodput among TCP flows was far overshadowed by differences in goodput for algorithm 3 (FAST) and algorithm 4 (FAST-AT) flows compared to flows using other alternate congestion control algorithms.

Fig. 8-43 gives seven scatter plots, each showing TCP goodput (y axis) vs. goodput on an alternate (as labeled) congestion control algorithm for movies transferred on very fast paths with a fast interface speed. Comparing Fig. 8-43 with Fig. 8-34, which gives the same information under high initial slow-start threshold, shows marked differences. Under low initial slow-start threshold, all seven alternate congestion control protocols provide much better goodput than achieved on TCP flows. This result can be attributed directly to the adoption of a low initial slow-start threshold. After reaching a congestion window size of 100 packets, the increase functions of the congestion avoidance regime of each protocol are activated. The TCP congestion avoidance regime leads to linear increase in transmission rate, while the congestion avoidance regimes in the other protocols lead to greater than linear increase. The precise increase rate depends upon the specific algorithm. Fig. 8-44 shows the degree to which goodput on flows using each alternate congestion control algorithm exceeds goodput on TCP flows for each condition when movies are transferred on very fast paths with a fast interface speed.

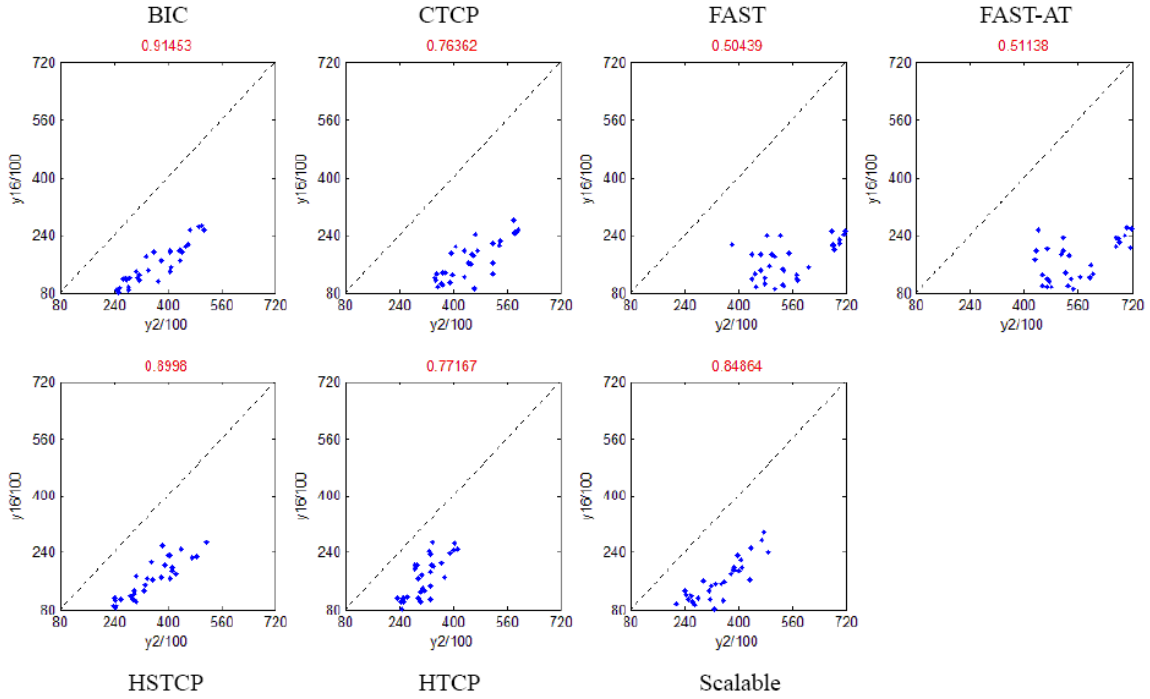


Figure 8-43. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Movies Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold)

Fig. 8-44 confirms the results in Fig. 8-43 and also reveals that flows using FAST and FAST-AT achieve higher goodput advantage over TCP flows, though the advantage diminishes somewhat with increasing congestion. This means that, in congestion avoidance, FAST increases transmission rate faster than the other congestion control algorithms. From Fig. 8-44 one can also discern that CTCP increases transmission rate second fastest. Thus, when given a low initial slow-start threshold and transferring large files at high speeds over paths with little congestion, the congestion avoidance increase procedures of the alternate protocols reach maximum transfer rate far more quickly than possible using the linear increase procedures of TCP. This general pattern also holds for

service packs (see Figs. 8-45 and 8-46) and documents (see Figs. 8-47 and 8-48). Note that for these smaller file sizes FAST and FAST-AT still achieve much higher goodputs than normal TCP, though the degree to which the other alternate congestion control algorithms outperform TCP is much diminished.

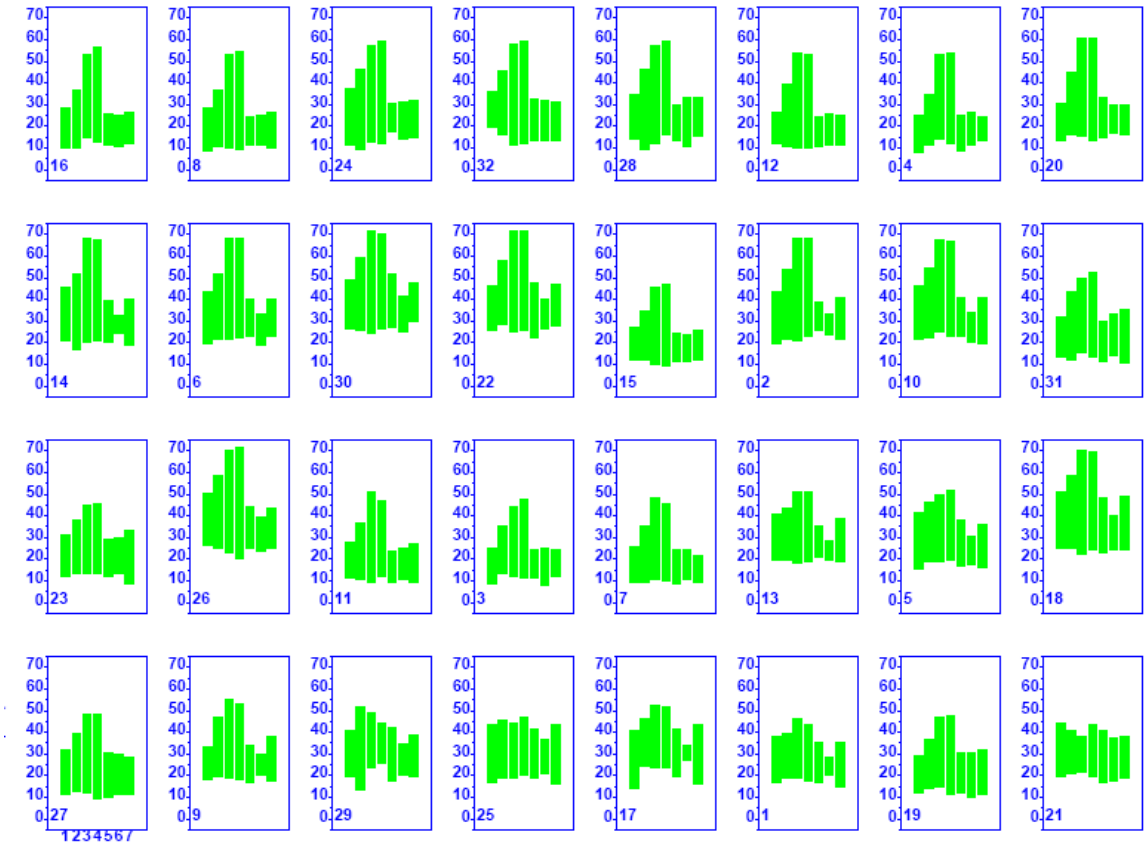


Figure 8-44. 32 Bar Graphs (one for each simulated condition) plotting Goodput (pps/1000) on TCP Flows vs. Non-TCP Flows for Movies Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold) (Each graph contains seven bars, one per congestion control algorithm, ordered left to right by algorithm identifier. Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows – y2(u).)

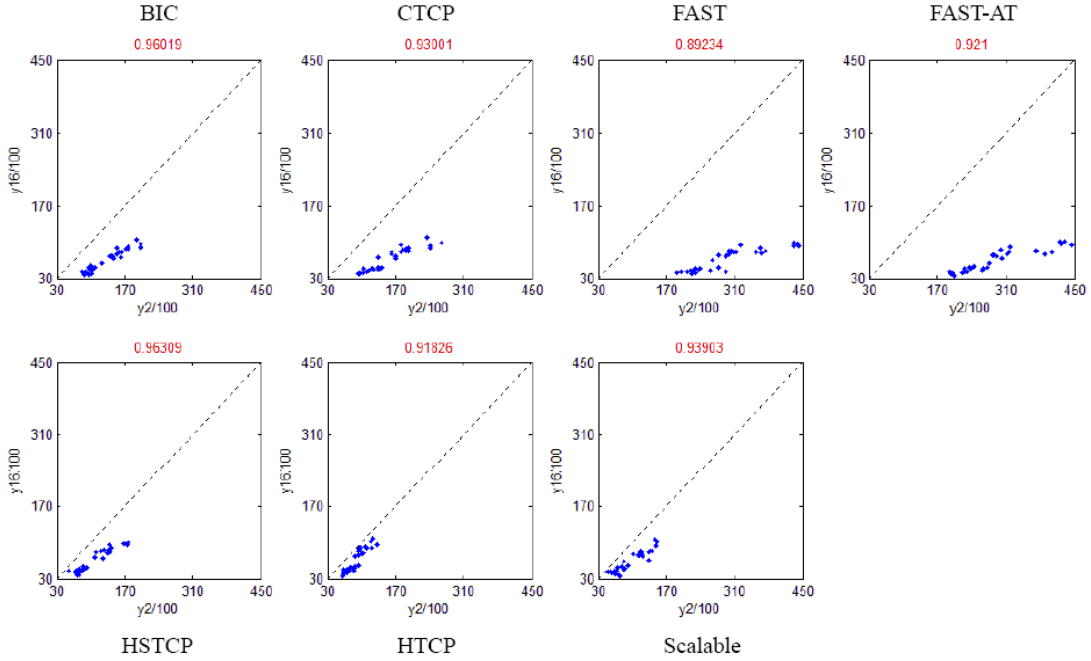


Figure 8-45. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Service Packs Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold)

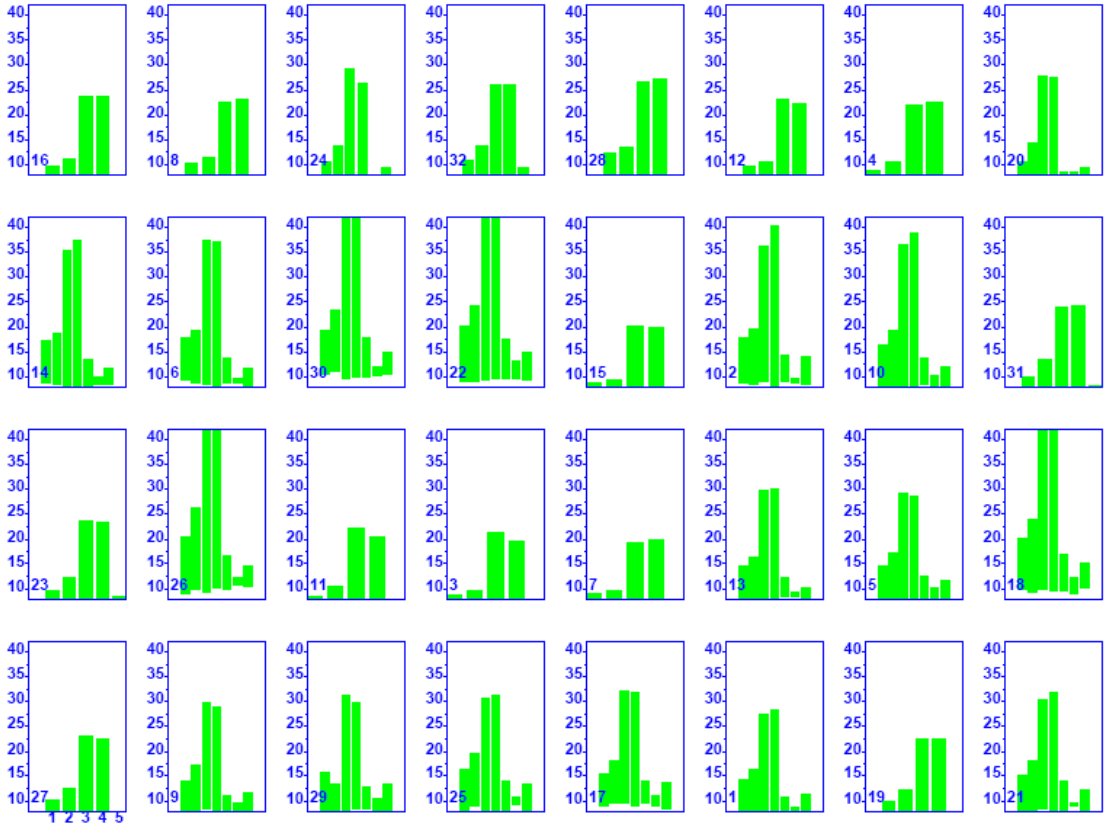


Figure 8-46. 32 Bar Graphs plotting Goodput (pps/1000) on TCP Flows vs. Non-TCP Flows for Service Packs Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold) (Each graph contains seven bars, one per congestion control algorithm, ordered left to right by algorithm identifier. Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows – y2(u).)

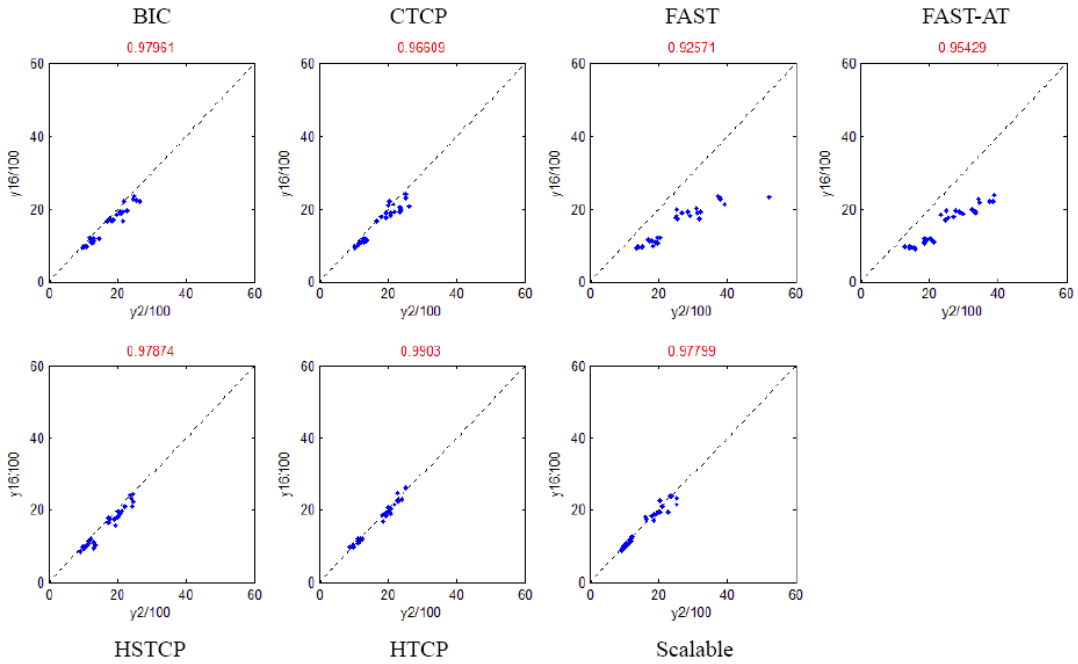


Figure 8-47. Scatter Plot of Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Documents Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold)

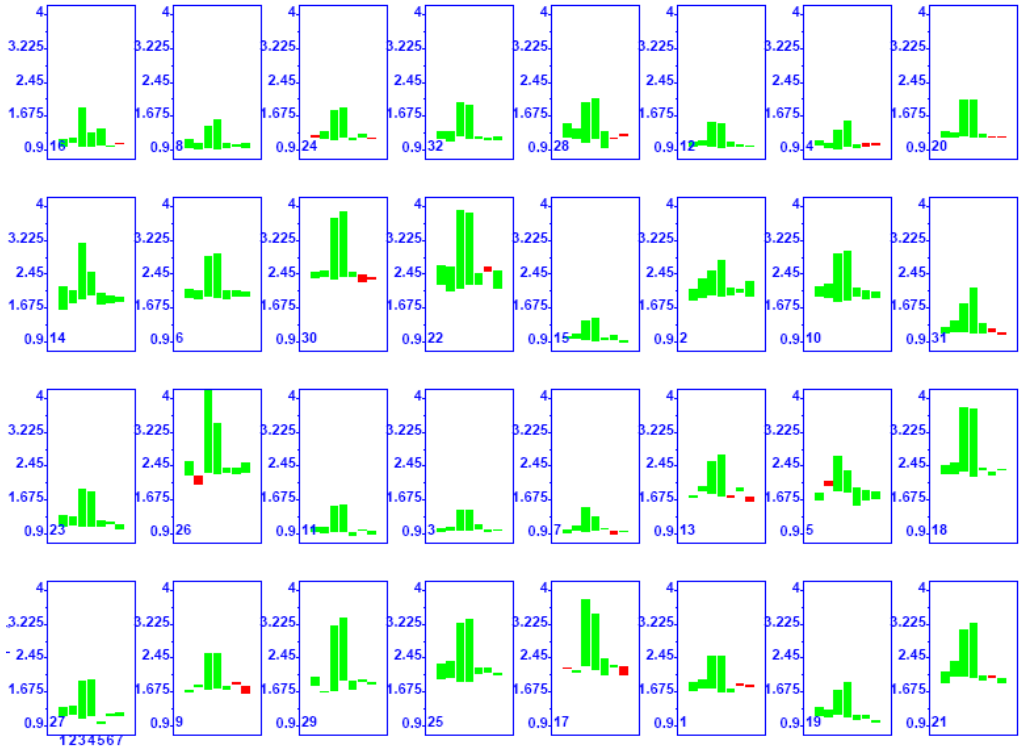


Figure 8-48. 32 Bar Graphs plotting Goodput (pps/1000) on TCP Flows vs. Non-TCP Flows for Service Packs Transferred on Very Fast Paths with Fast Interfaces (Low Initial Slow-Start Threshold) (Each graph contains seven bars, one per congestion control algorithm, ordered left to right by algorithm identifier. Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows – y2(u). If the bar is red, y16(u) is greater; if the bar is green, y2(u) is greater. The 32 bar graphs are sorted from least to most congestion by condition, as indicated in the lower left-hand corner of each plot.)

8.4.2.3 Summary of Differences in Goodput. Table 8-30 gives a summary of goodput differences as percentages for each of the 24 flow groups measured. Differences under high initial slow-start threshold (HIGH INITIAL SSTHRESH) are reported in three columns: (1) **AMONG ALTs** gives the range of percentage difference on average goodput, y2(u), between flows using the alternate congestion control algorithms with the highest and lowest average goodput; (2) **AMONG TCPs** gives the range of percentage difference on average goodput, y16(u), between TCP flows with the highest and lowest average goodput when competing with alternate congestion control algorithms; (3) **ALTs > TCPs** gives the percentage increase in average goodput, y2(u) vs. y16(u), for flows using alternate congestion control algorithms over competing TCP flows (note that in one case, given in red, TCP flows achieved higher average goodput). A similar set of three columns reports goodput differences under low initial slow-start threshold (LOW INITIAL SSTHRESH).

Table 8-30. Range of Goodput Differences (%) for Flow Groups under High and Low Initial Slow-Start Threshold (Differences are shown: among Alternate Congestion Control Algorithms, among TCP Flows Competing with Alternate Algorithms and between Alternate Algorithms and TCP Flows)

			RANGE OF GOODPUT DIFFERENCES (%)					
			HIGH INITIAL SSTHRESH			LOW INITIAL SSTHRESH		
File	Path	Interface	AMONG ALTs	AMONG TCPs	ALTs > TCPs	AMONG ALTs	AMONG TCPs	ALTs > TCPs
M	VF	F	10	11	11	45	5	60
		N	<1	8	3	6	3	9
	F	F	35	16	35	33	12	38
		N	21	20	21	16	12	30
	T	F	30	11	30	20	15	30
		N	30	17	30	20	15	25
SP	VF	F	4	6	3	70	3	60
		N	<3	5	1	30	4	20
	F	F	12	8	15	40	10	55
		N	15	10	15	35	7	35
	T	F	20	6	20	35	13	35
		N	20	7	20	30	13	30
D	VF	F	5	6	<1	40	5	20
		N	<3	4	<2	25	1	10
	F	F	4	7	<2	30	5	<2
		N	5	7	<2	25	6	12
	T	F	3	5	2	22	7	12
		N	<4	7	2	20	8	11
WO	VF	F	16	5	-1	11	8	<3
		N	<5	5	<2	5	2	<4
	F	F	5	4	<1	5	5	2
		N	4	4	<1	5	4	2
	T	F	5	6	<1	<3	6	<2
		N	5	5	<1	<3	5	<2

Under high initial slow-start threshold, all congestion control algorithms (including TCP) increase transmission rate to the available maximum using the same algorithm (limited slow-start, here), so variations in goodput result solely from differences in loss/recovery procedures among the algorithms. This means that such differences arise mainly during congestion and when transferring large files, which are likely to have more packets lost because there are more packets in the files. Under low initial slow-start threshold, TCP increases transmission rate linearly after entering congestion avoidance, while the alternate congestion control algorithms increase transmission rate more steeply. FAST and FAST-AT increase transmission rate quickest and CTCP second quickest. The advantage of a steep increase in transmission rate appears most evident for large files when transferred over fast paths experiencing little congestion. This advantage for smaller files exists mainly for FAST and FAST-AT.

Table 8-30 shows that the largest differences in average goodput occur among flows using various alternate congestion control algorithms (AMONG ALTs) and between flows using alternate algorithms and competing TCP flows (ALTs > TCP). Lesser differences in average goodput appear among TCP flows when competing with flows using alternate algorithms (AMONG TCPs). To more completely analyze differences in average goodput, we can consider the relative ranking of each alternate algorithm with respect to goodput achieved by flows using the algorithm and by TCP flows competing with the algorithm. We turn to this topic next.

8.4.3 Relative User Experience

In this section, we set aside absolute differences in average goodput and consider instead relative differences. For each simulated condition, we ranked from high (7) to low (1) the average goodput – y2(u) – provided by the seven alternate congestion control algorithms and we also computed the average goodput across all seven algorithms. We took similar steps with respect to average goodput – y16(u) – among TCP flows competing with the alternate algorithms. Armed with this information, we generated seven pairs of rank[6] matrices. One member of each pair relates to y2(u) and the other member to y16(u). (See Fig. 8-12 for a sample rank matrix). Each matrix contains (32 conditions x 24 flow groups =) 768 cells, where each cell contains the rank (of average goodput among the seven competing algorithms) for the congestion control algorithm associated with the matrix. If the rank in a cell is rendered in green, then the goodput associated with the rank was above the average goodput for all algorithms. If red, then the goodput was below the relevant average. When a highest ranked (7) cell was farther from the average goodput than the lowest ranked (1) cell, then the cell is filled in green. In the reverse case, the lowest ranked cell is filled in red.

The columns in each matrix are divided into four vertical sections that each relate to a specific file size (movie, service pack, document and Web object). Each section contains three pairs of flow groups (labeled on the x axis) ordered by path class (very fast, fast and typical). Within each flow-group pair the ordering is by interface speed (fast and normal). The matrix rows are ordered by condition (labeled on the y axis) from least (top) to most (bottom) congested. In the results below, we reproduce matrices related to

[6] The reader should keep in mind the fact that ranking forces an ordering among the congestion control algorithms without distinction to the magnitude of those differences. Absolute differences in average goodput were the subject of the preceding section (8.4.2).

high and low initial slow-start threshold. Half of the matrices show the rank in goodput for each alternate congestion control algorithm when compared against the others. The remaining matrices show the rank in goodput for TCP flows competing with each alternate congestion control algorithm when compared against TCP flows competing with the others. We reproduce these matrices to show any patterns that occur.

In addition to showing the matrices, we computed the average rank for each congestion control algorithm for each file size. Similarly, we computed the average rank for TCP flows competing with each congestion control algorithm for each file size. We also determined the standard deviation in rank for each alternate congestion control algorithm, across all files sizes and considering both y2(u) and y16(u). We report these averages and standard deviations in summary tables (Tables 8-31 and 8-32). We use the information from the summary tables to generate scatter plots of average rank (x axis) vs. standard deviation in rank (y axis), which reveal differences in relative user experience among the seven alternate congestion control algorithms.

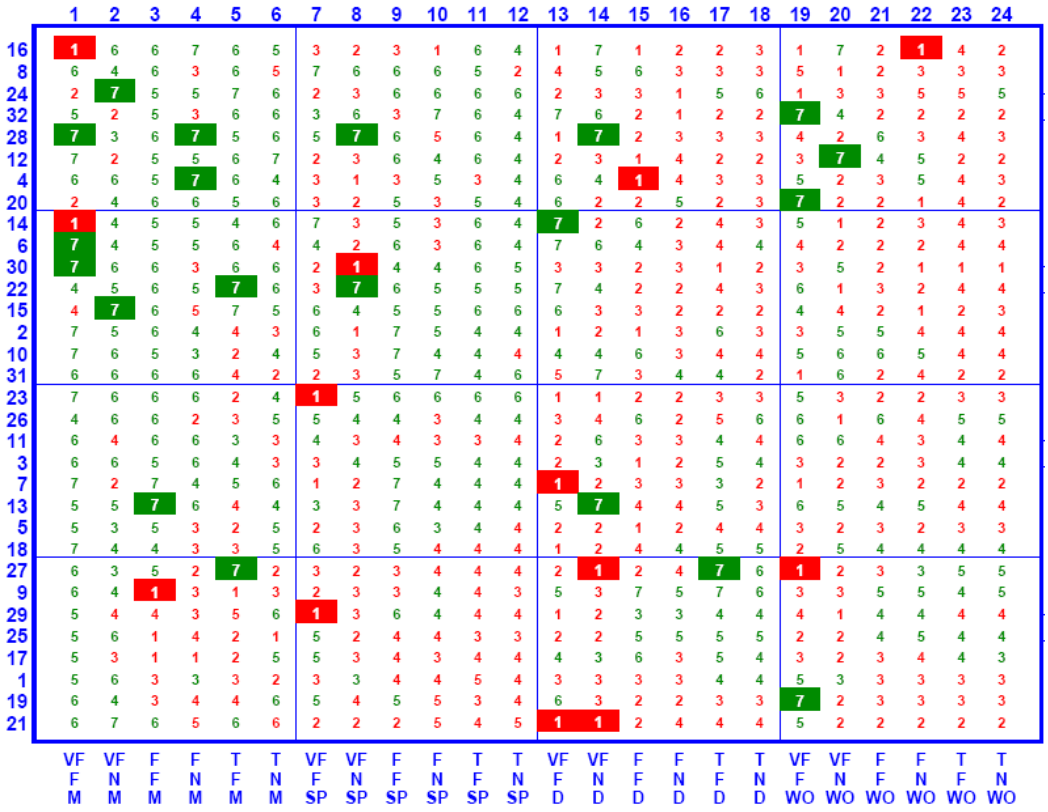


Figure 8-49. Goodput Rank Matrix – y2(u) – BIC (High Initial Slow-Start Threshold) Rank (7 high) in each cell denotes ordering of y2(u) for each condition (y axis) and flow group (x axis) – conditions are sorted from least (16) to most (21) congested and flow groups are ordered by file size – movies (M), service packs (SP), documents (D) and Web objects (WO) – and by path class – very fast (VF), fast (F), and typical (T) – within each file size and by interface speed – fast (F) or normal (N) – within each path class.

8.4.3.1 High Initial Slow-start Threshold. Figs. 8-49 through 8-55 show the ranking matrices for y2(u) under a high initial slow-start threshold. The related matrices for y16(u) are given in Figs. 8-56 through 8-62. Table 8-31 summarizes the rankings.

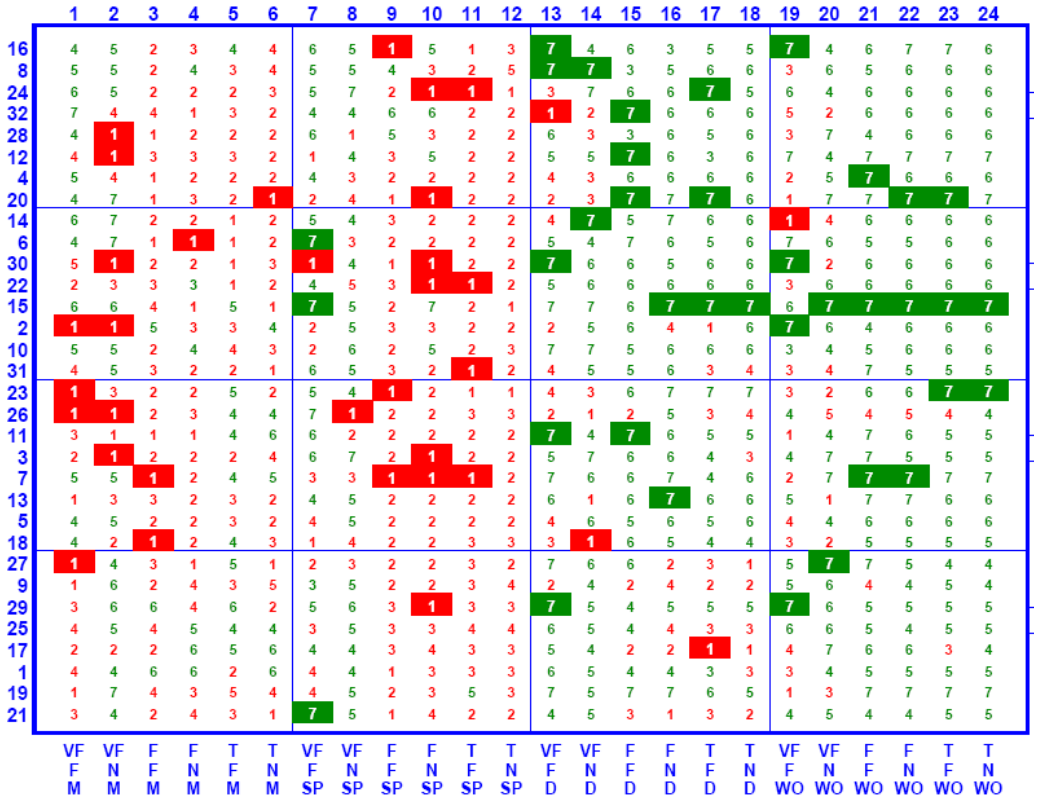


Figure 8-50. Goodput Rank Matrix – y2(u) – CTCP (High Initial Slow-Start Threshold)

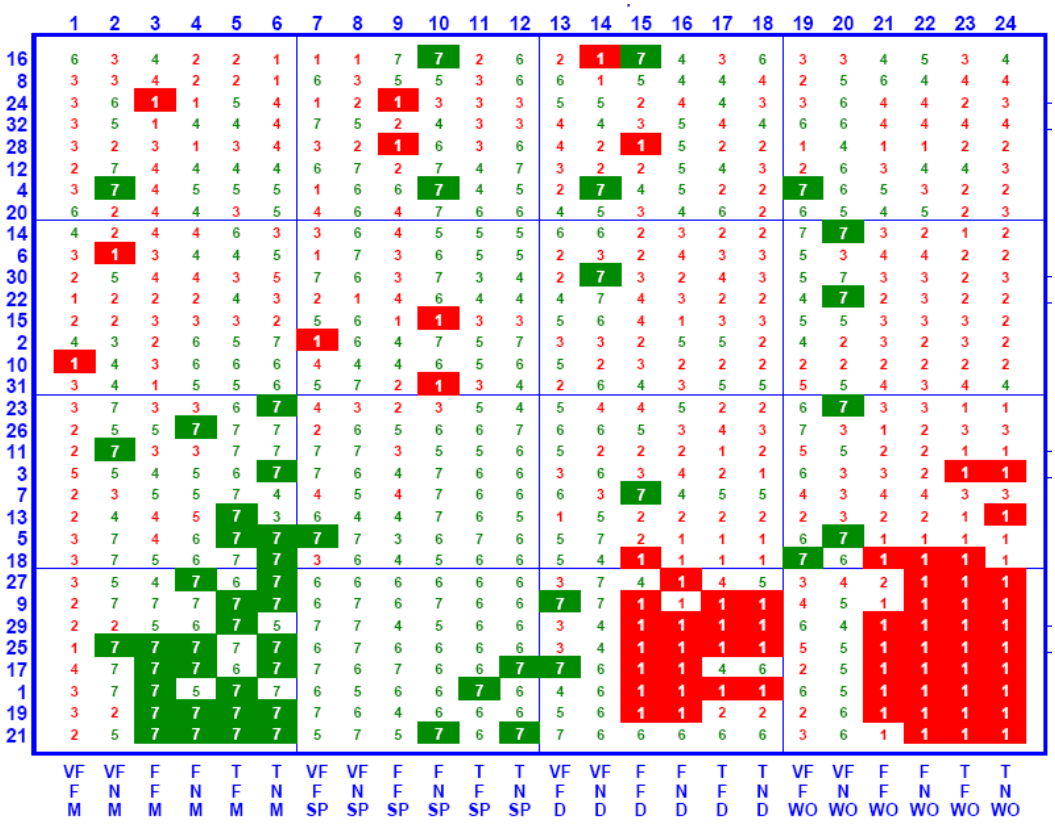


Figure 8-51. Goodput Rank Matrix – y2(u) – FAST (High Initial Slow-Start Threshold)

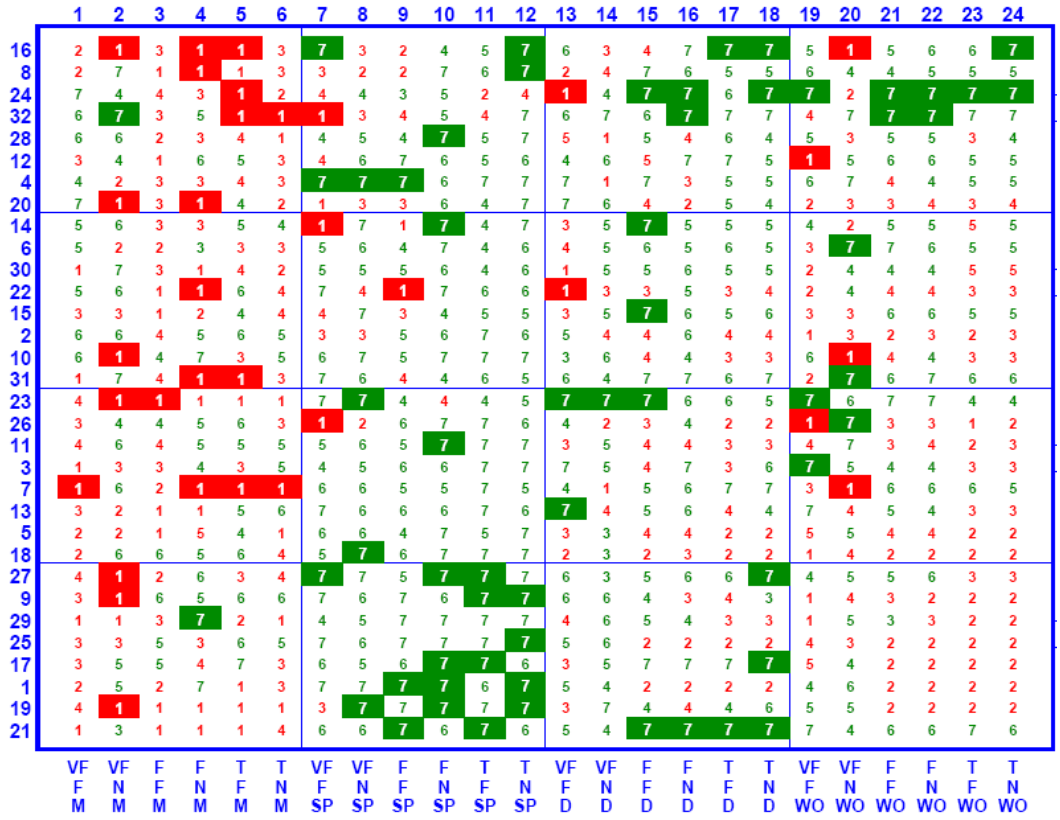


Figure 8-52. Goodput Rank Matrix – y2(u) – FAST-AT (High Initial Slow-Start Threshold)

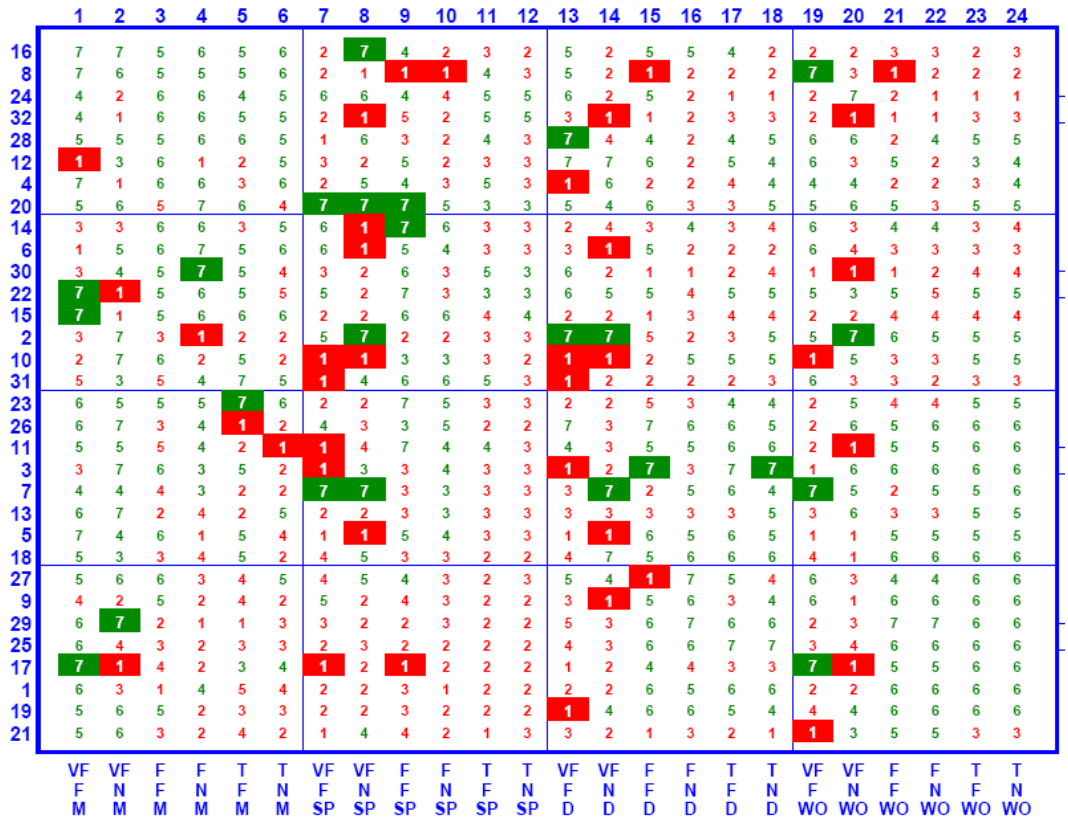


Figure 8-53. Goodput Rank Matrix – y2(u) – HSTCP (High Initial Slow-Start Threshold)

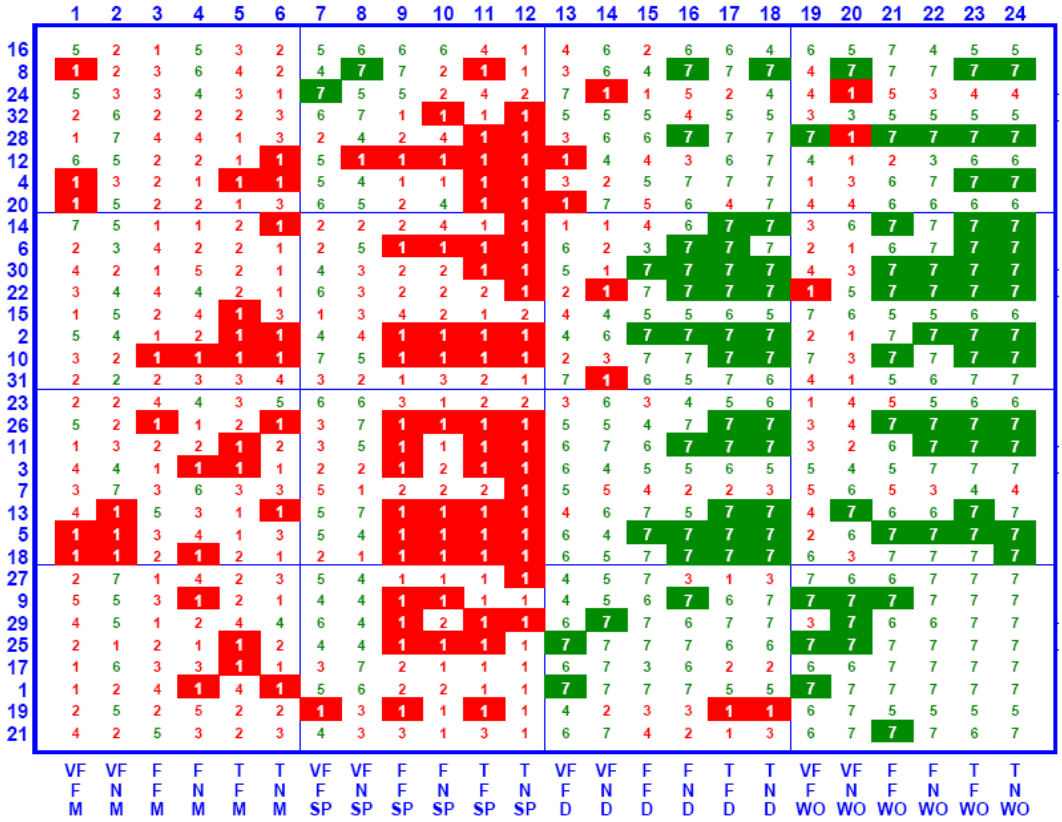


Figure 8-54. Goodput Rank Matrix – y2(u) – HTCP (High Initial Slow-Start Threshold)

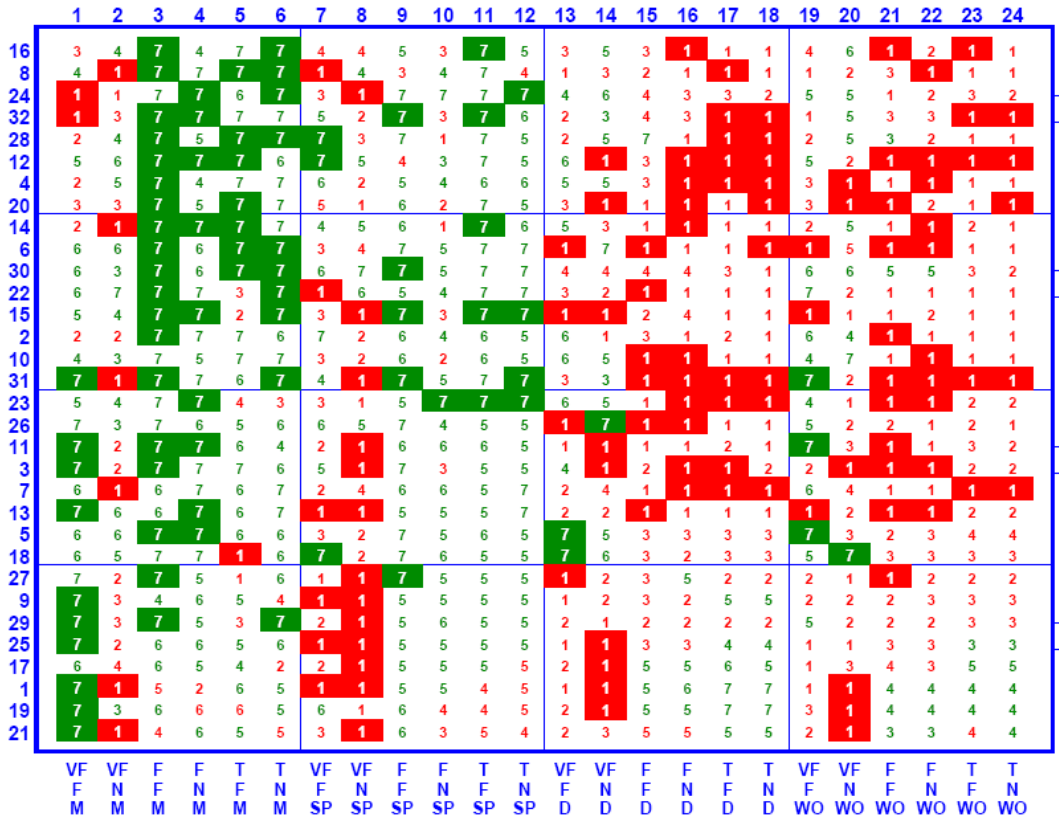


Figure 8-55. Goodput Rank Matrix – y2(u) – Scalable TCP (High Initial Slow-Start Threshold)

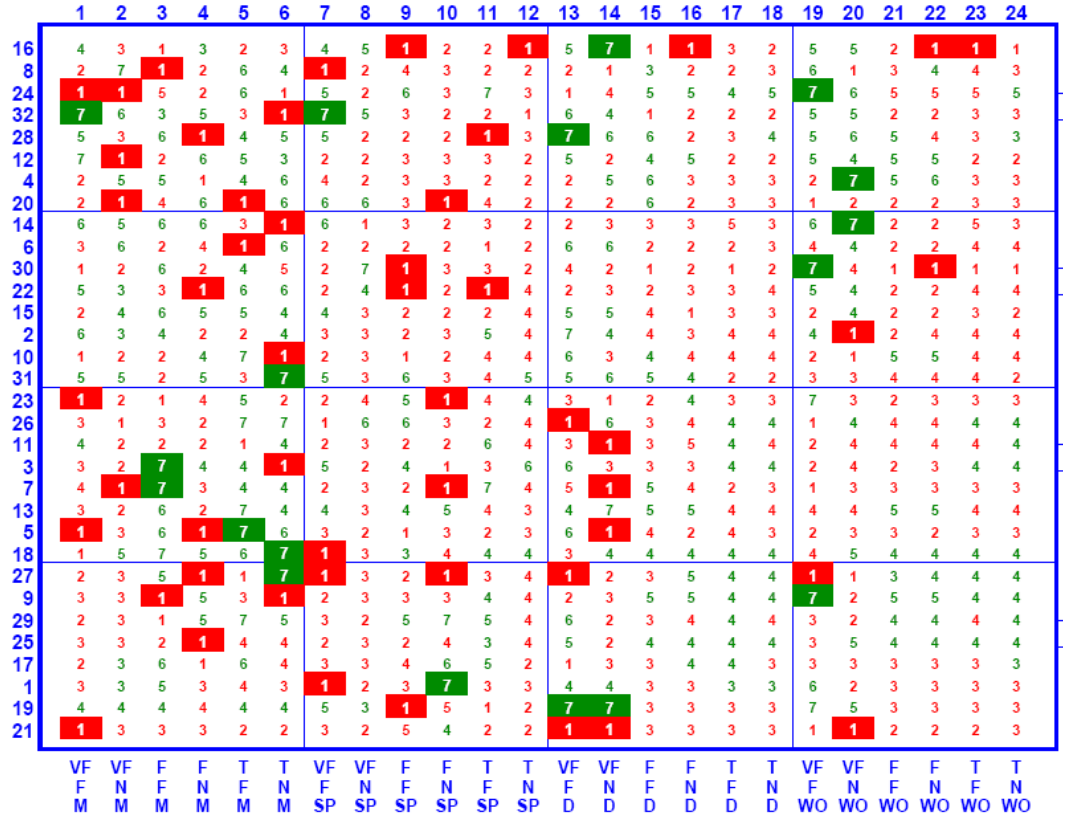


Figure 8-56. TCP Goodput Rank Matrix – y16(u) – BIC (High Initial Slow-Start Threshold)

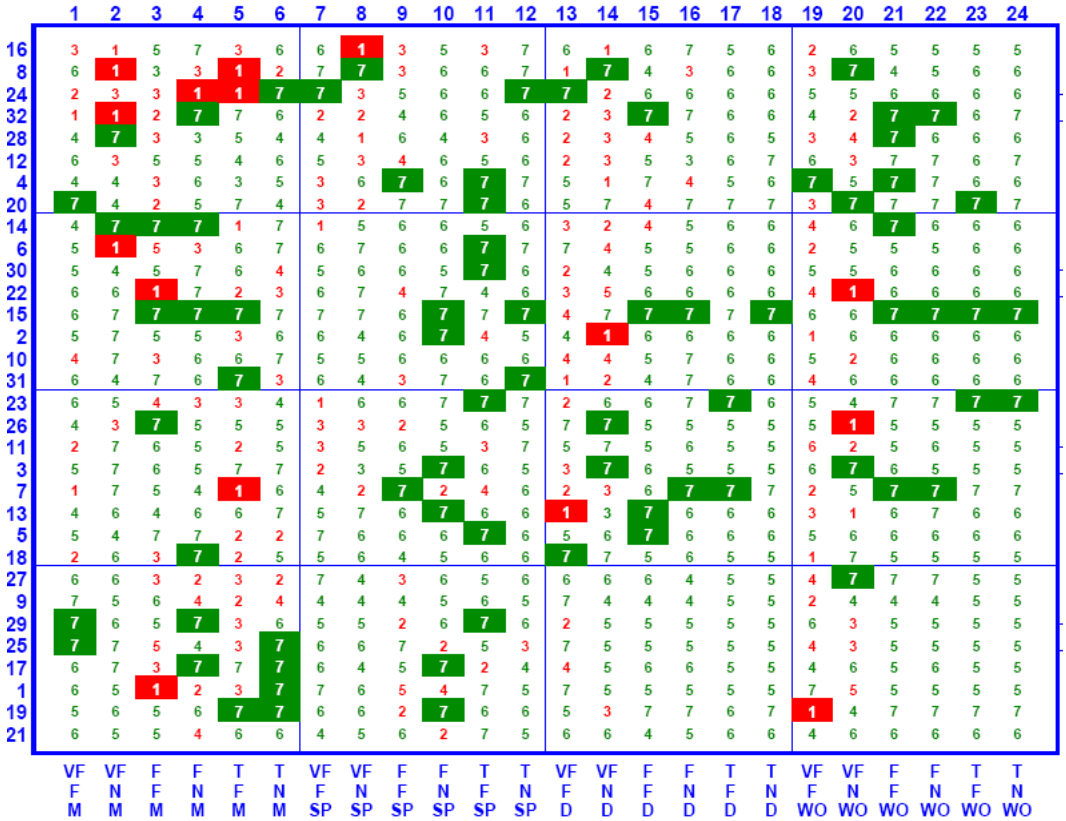


Figure 8-57. TCP Goodput Rank Matrix – y16(u) – CTCP (High Initial Slow-Start Threshold)

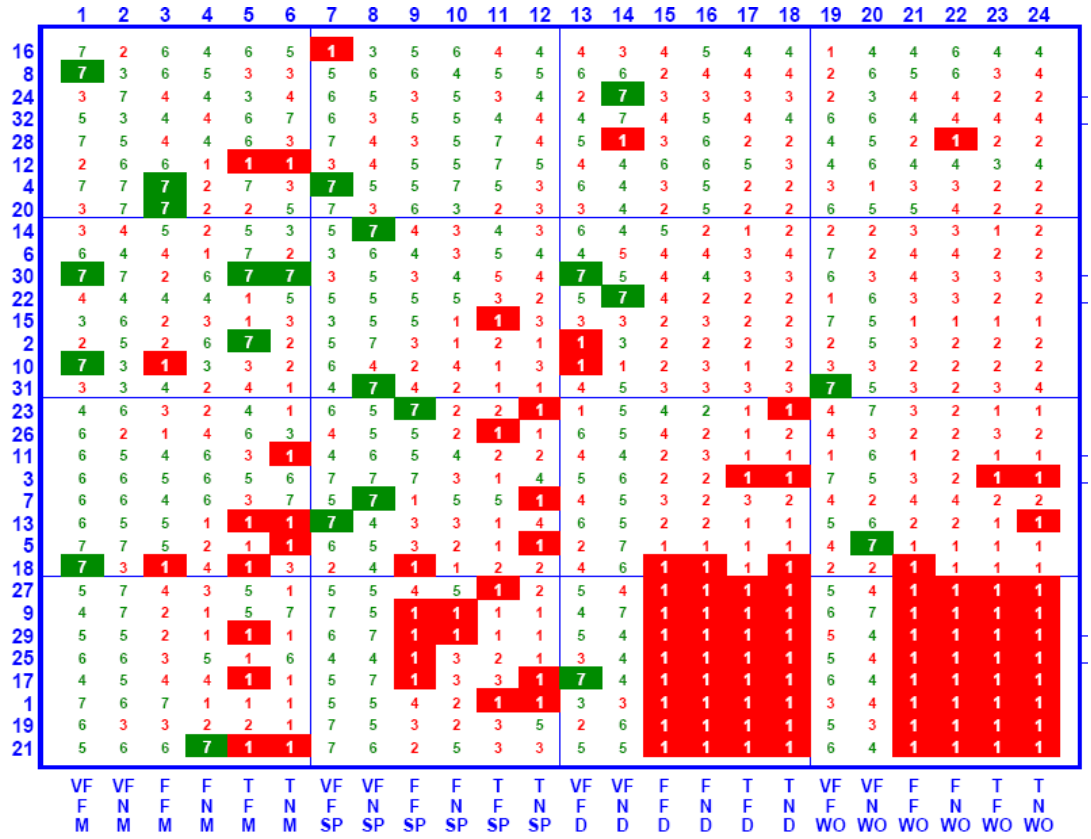


Figure 8-58. TCP Goodput Rank Matrix – y16(u) – FAST (High Initial Slow-Start Threshold)

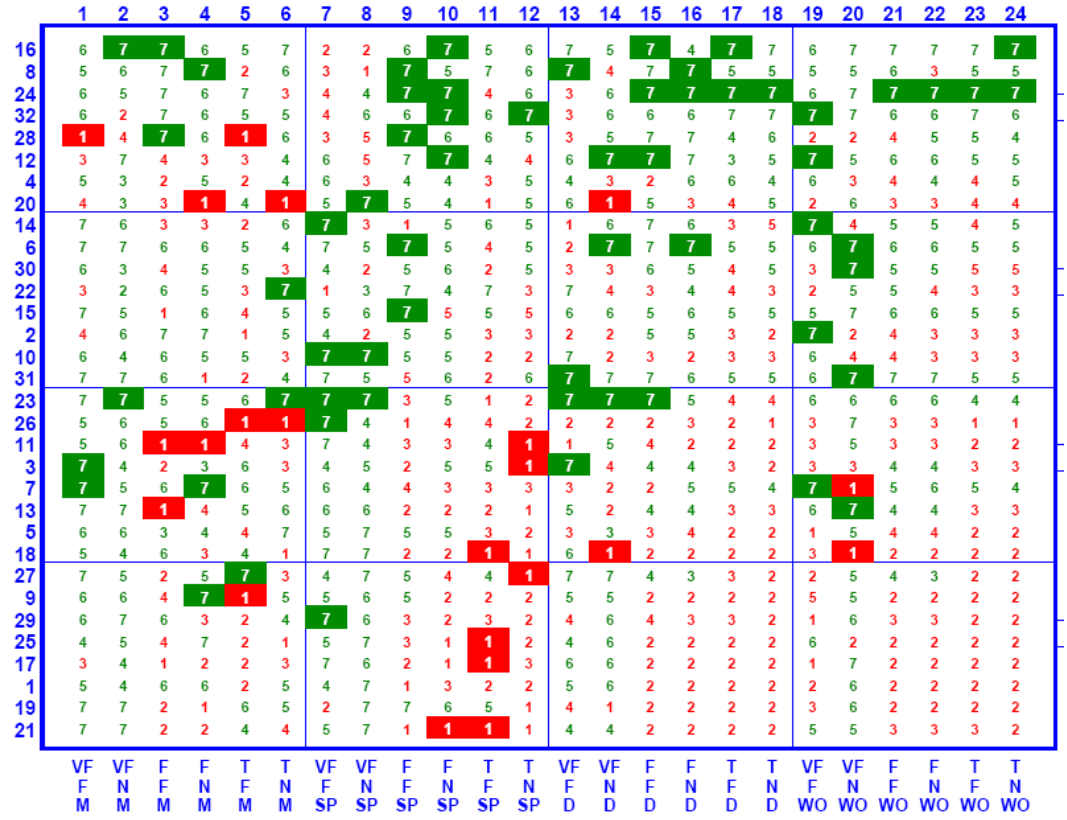


Figure 8-59. TCP Goodput Rank Matrix – y16(u) – FAST-AT (High Initial Slow-Start Threshold)

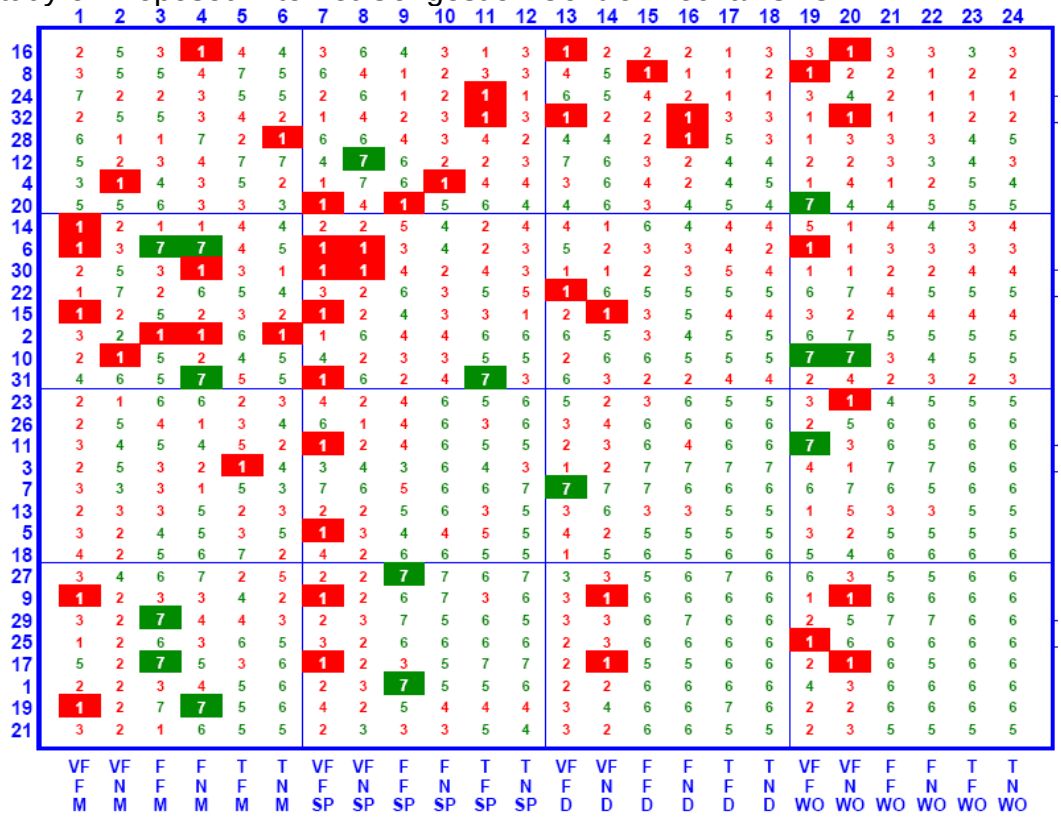


Figure 8-60. TCP Goodput Rank Matrix – y16(u) – HSTCP (High Initial Slow-Start Threshold)

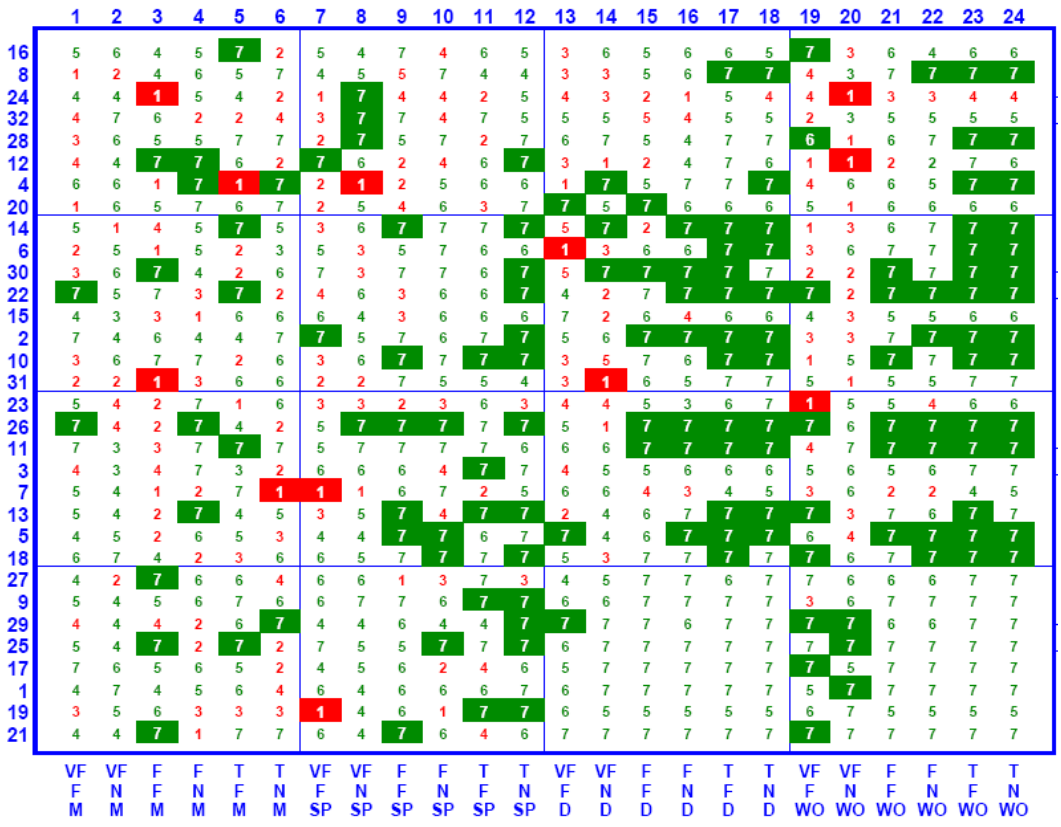


Figure 8-61. TCP Goodput Rank Matrix – y16(u) – HTCP (High Initial Slow-Start Threshold)

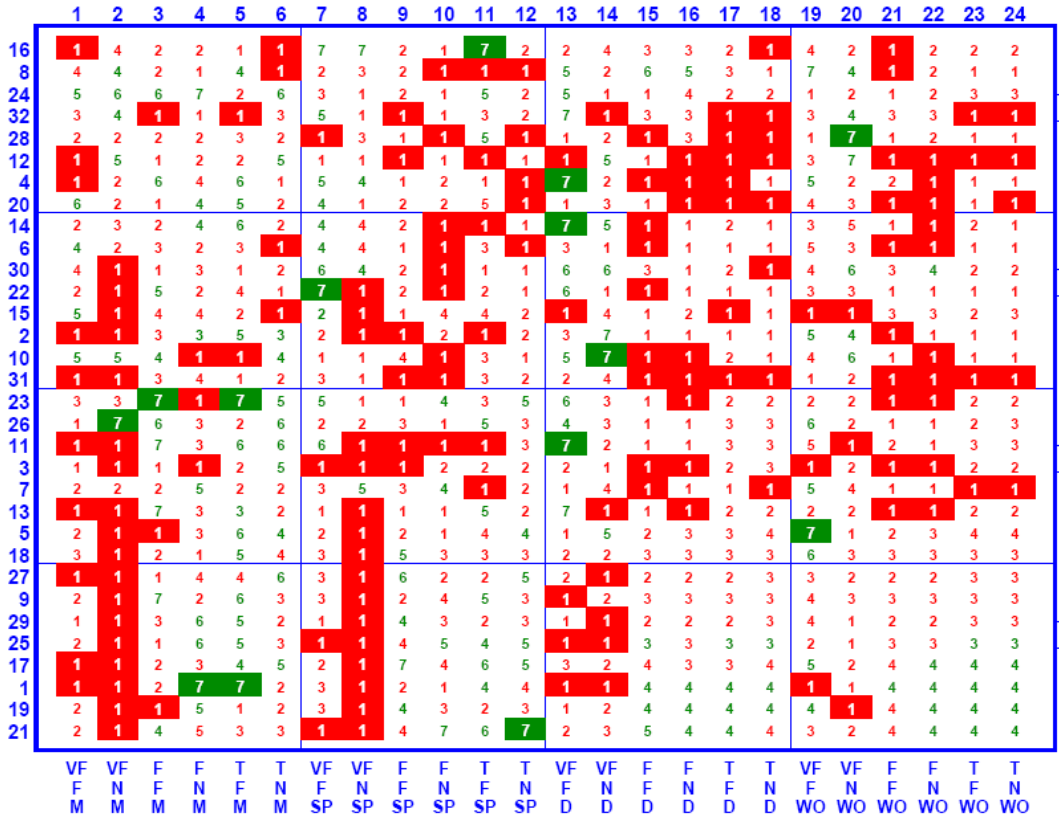


Figure 8-62. TCP Goodput Rank Matrix – y16(u) – Scalable TCP (High Initial Slow-Start Threshold)

Table 8-31. Summary Average and Standard Deviation in Goodput and TCP Goodput Rank for All Congestion Control Algorithms (High Initial Slow-Start Threshold)

		BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
y2(u)	M	4.73	3.14	4.46	3.32	4.32	2.59	5.44
	SP	4.15	3.01	4.98	5.58	3.26	2.42	4.61
	D	3.42	4.94	3.40	4.68	3.84	5.18	2.55
	WO	3.37	5.32	3.03	4.10	4.10	5.68	2.40
	Avg.	3.92	4.10	3.97	4.42	3.88	3.96	3.75
y16(u)	M	3.57	4.78	4.01	4.59	3.61	4.53	2.91
	SP	3.09	5.25	3.78	4.26	3.83	5.29	2.51
	D	3.46	5.20	2.96	4.13	4.15	5.70	2.39
	WO	3.44	5.37	2.84	4.16	4.04	5.66	2.45
	Avg.	3.39	5.15	3.40	4.28	3.91	5.30	2.56
y2(u) & y16(u)	Avg.	3.66	4.63	3.69	4.35	3.90	4.63	3.16
	Std.	0.53	0.98	0.77	0.64	0.34	1.37	1.19

The matrices and summary table give some impressions regarding relative goodput for flows operating under various congestion control algorithms as well as for

competing TCP flows. Keep in mind that these observations relate only to a high initial slow-start threshold, so the main differences must be attributable to how congestion control algorithms react to losses. First, HTCP and CTCP, followed by FAST-AT, appear to interfere least with goodput on competing TCP flows. On a loss, these protocols reduce congestion window to the same extent as TCP. Of course, so does FAST. FAST-AT can be less aggressive than FAST when recovering from congestion because the α parameter can be driven down, which causes FAST-AT to recover less forcefully. More aggressive recovery by FAST can induce higher losses from which TCP flows recover with a linear increase in congestion window. Second, Scalable TCP provides significant goodput on large files but interferes with TCP flows. BIC shows traits similar to Scalable but with lower magnitude. HSTCP provides moderate goodputs and interferes only moderately with TCP flows. HTCP and CTCP provide relatively high goodputs on smaller files, while not interfering much with TCP flows. HTCP interferes less with TCP flows than does CTCP, but HTCP also provides substantially lower relative goodput on larger files.

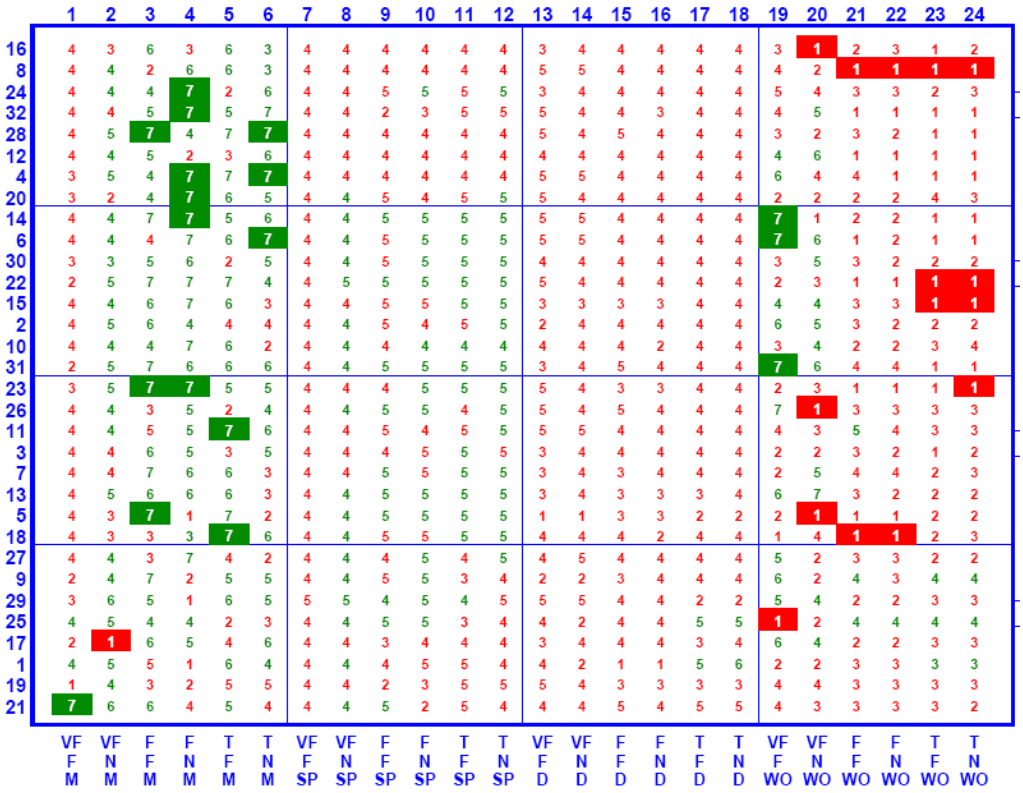


Figure 8-63. Goodput Rank Matrix – y2(u) – BIC (Low Initial Slow-Start Threshold)

8.4.3.2 Low Initial Slow-start Threshold. When the initial slow-start threshold is low, differences in relative goodput appear not only due to loss/recovery processing but also due to the rate at which flows discover the maximum available transmission rate. For this reason, all alternate congestion control protocols provide substantially better goodput than standard TCP. Despite this fact, appropriate analyses can still discern differences in relative goodput among alternate congestion control protocols as well as among competing TCP flows. Figs. 8-63 through 8-69 show the ranking matrices for y2(u) under

a low initial slow-start threshold. The related matrices for y16(u) are given in Figs. 8-70 through 8-76. Table 8-32 summarizes the rankings.

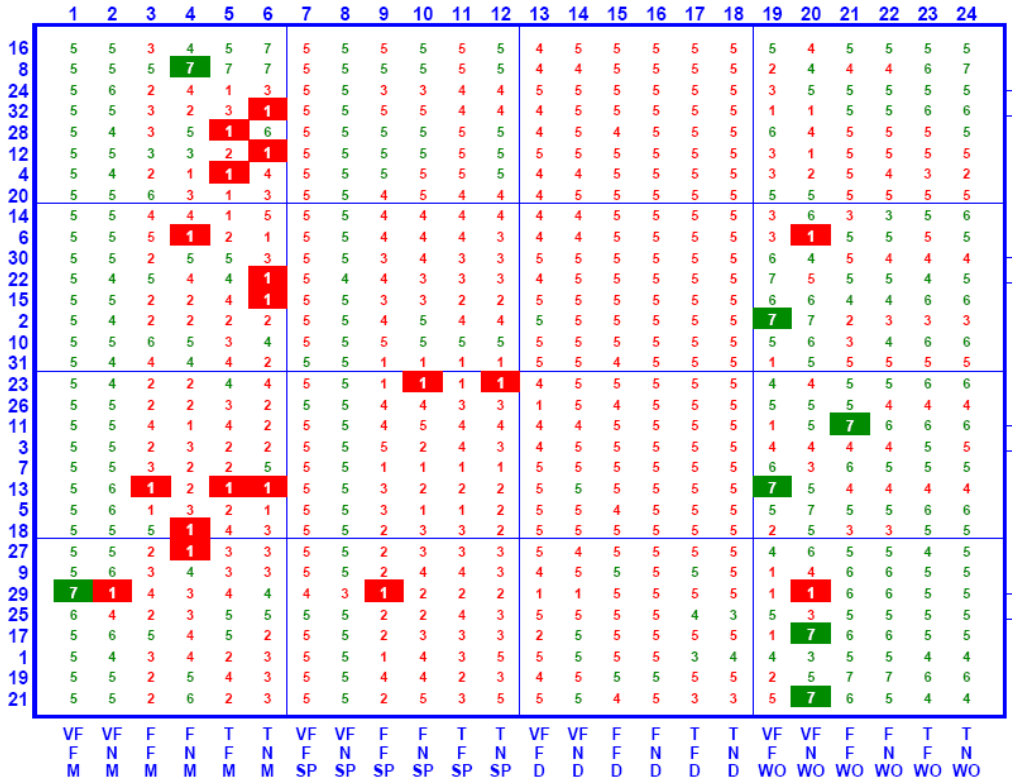


Figure 8-64. Goodput Rank Matrix – y2(u) – CTCP (Low Initial Slow-Start Threshold)

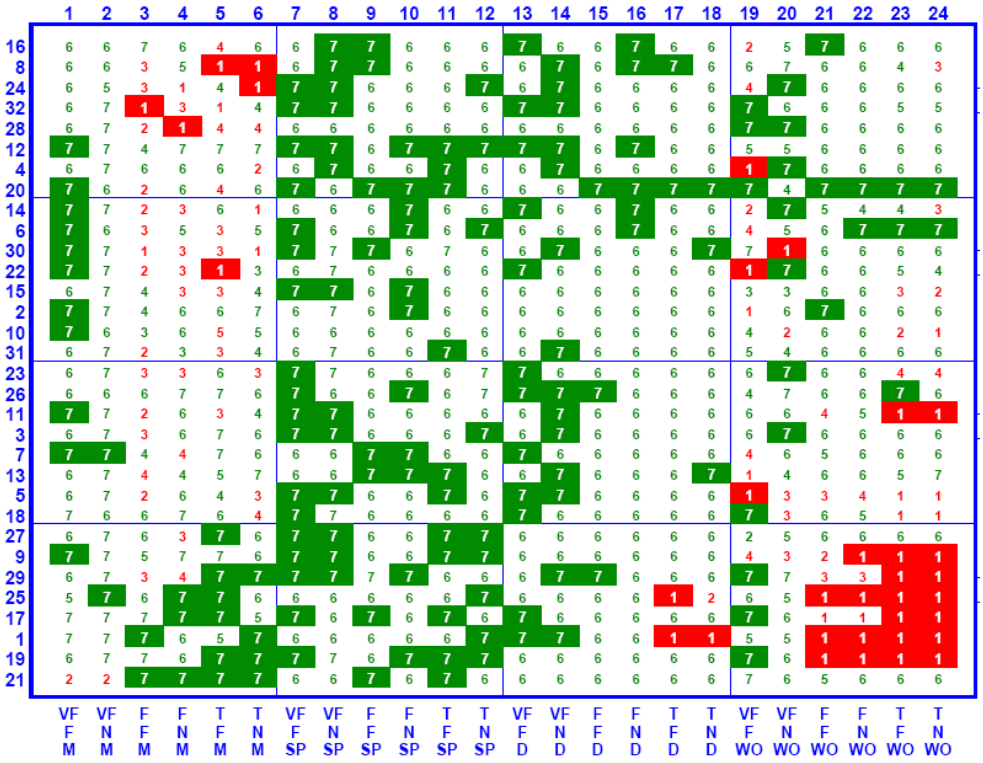


Figure 8-65. Goodput Rank Matrix – y2(u) – FAST (Low Initial Slow-Start Threshold)

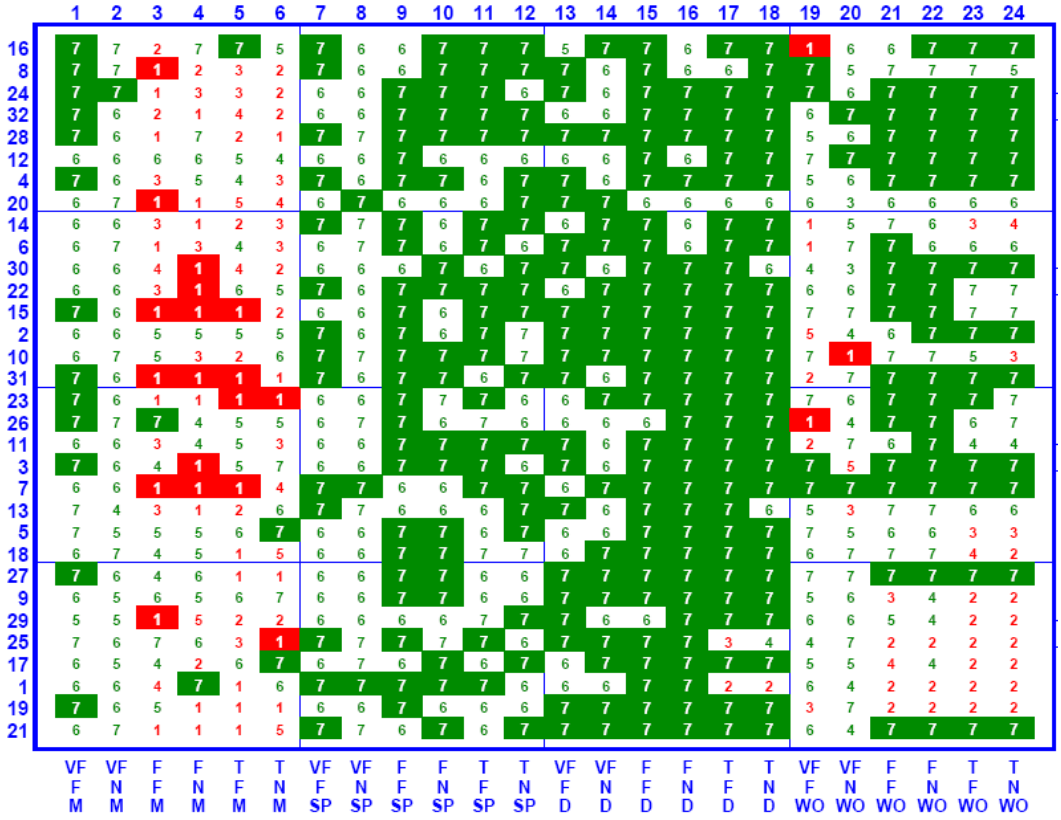


Figure 8-66. Goodput Rank Matrix – y2(u) – FAST-AT (Low Initial Slow-Start Threshold)

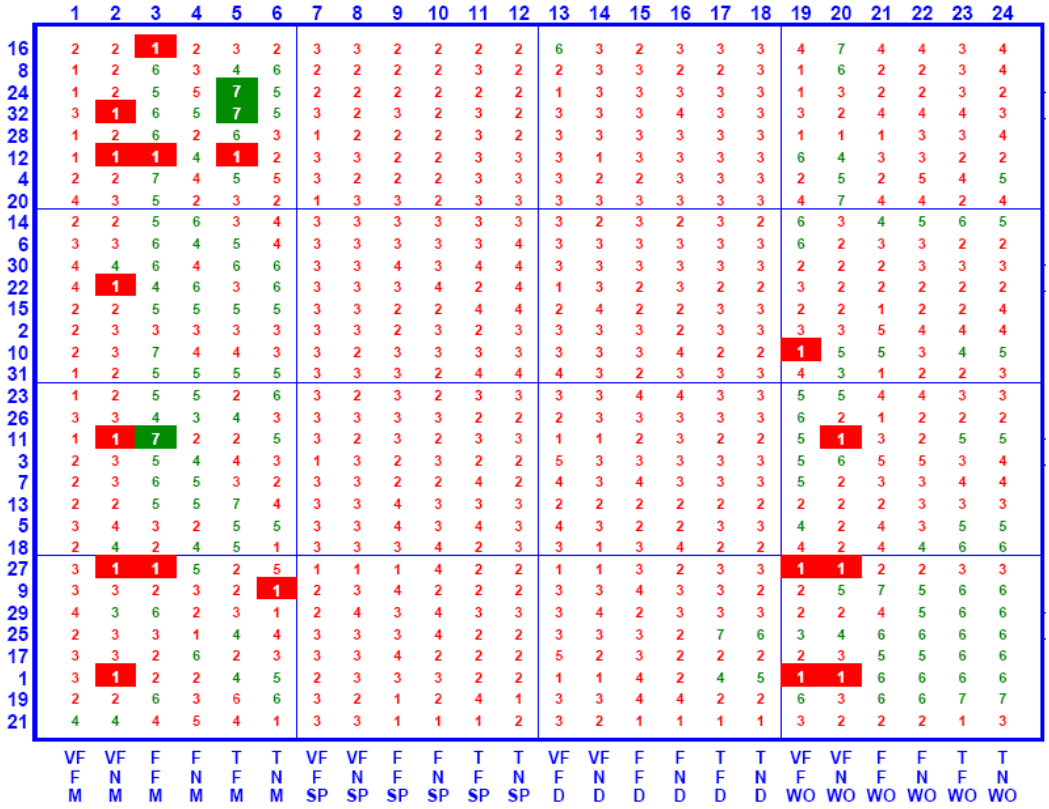


Figure 8-67. Goodput Rank Matrix – y2(u) – HSTCP (Low Initial Slow-Start Threshold)

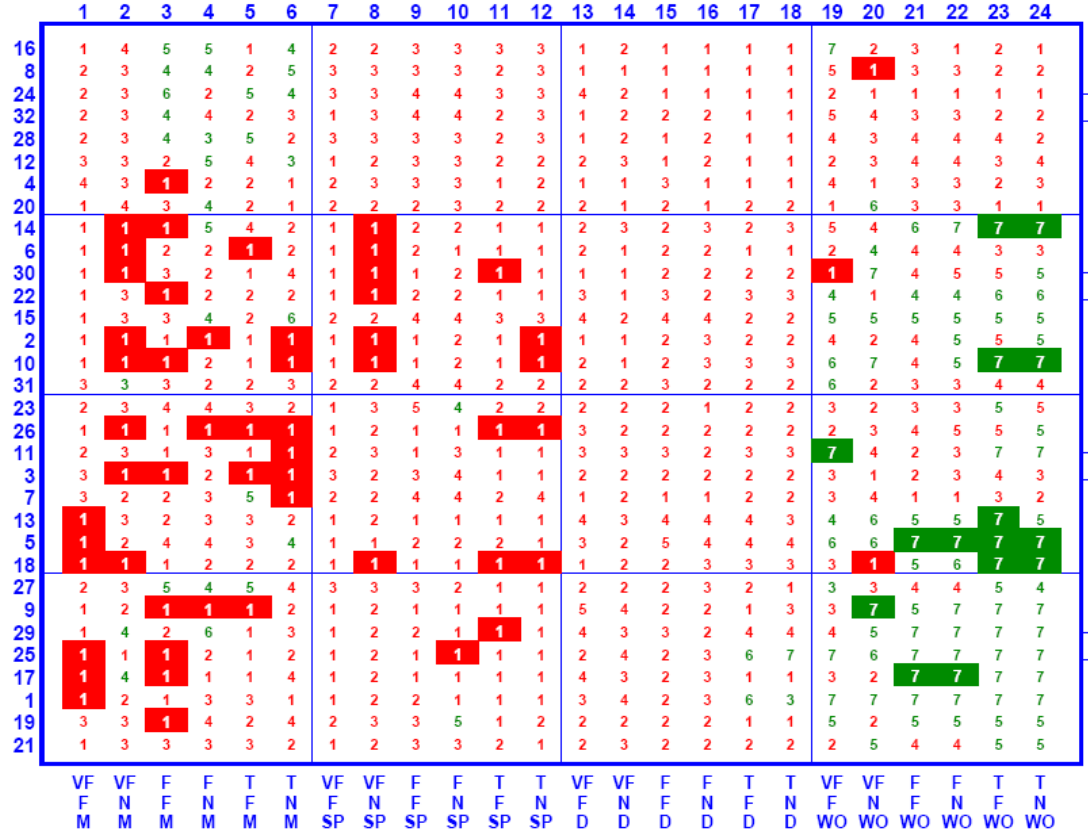


Figure 8-68. Goodput Rank Matrix – y2(u) – HTCP (Low Initial Slow-Start Threshold)

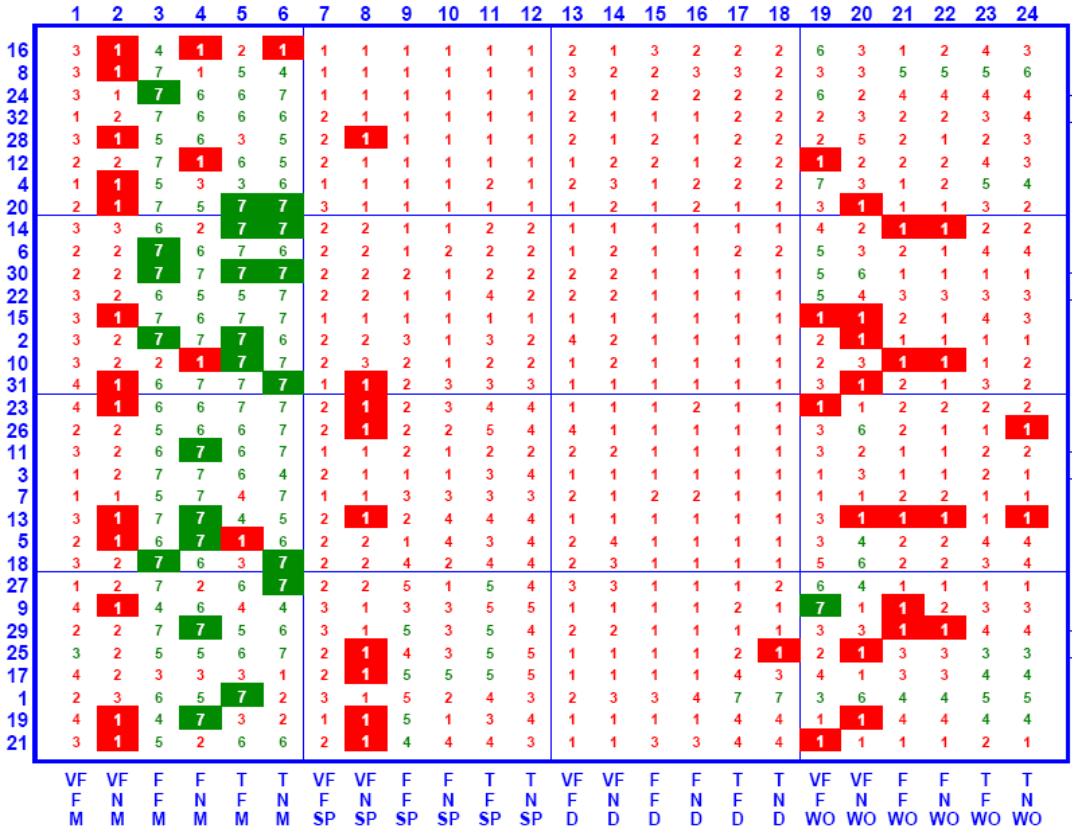


Figure 8-69. Goodput Rank Matrix – y2(u) – Scalable TCP (Low Initial Slow-Start Threshold)

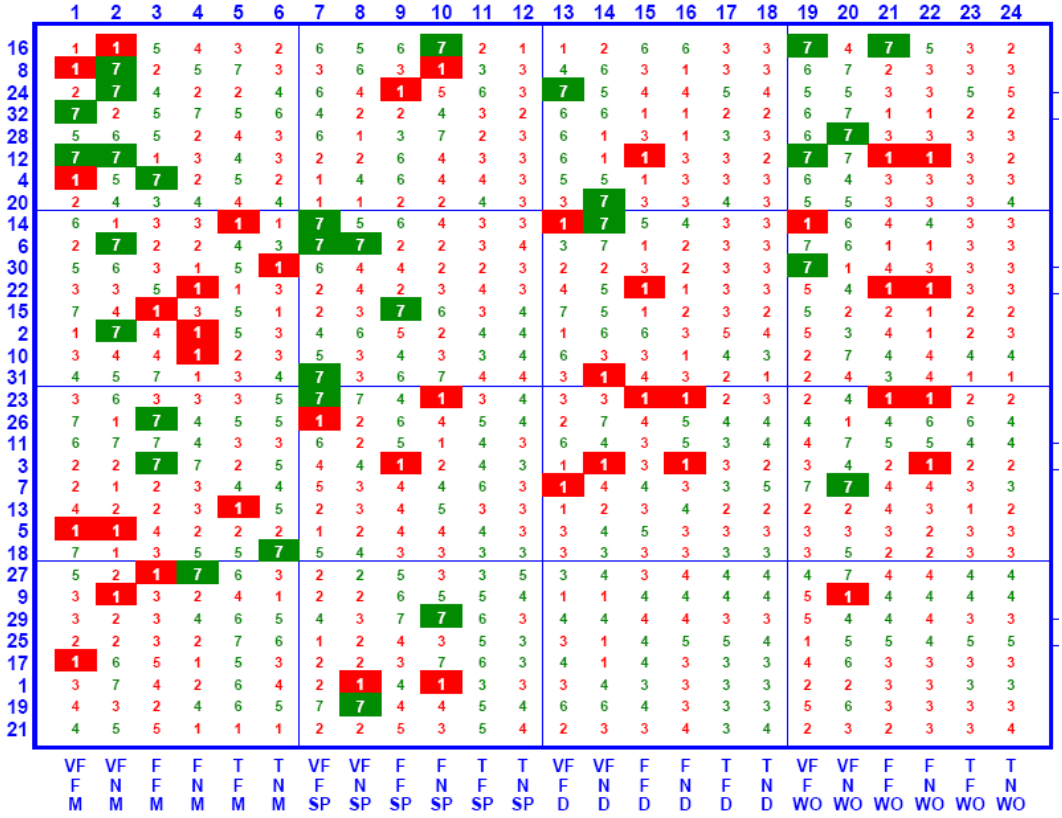


Figure 8-70. TCP Goodput Rank Matrix – y16(u) – BIC (Low Initial Slow-Start Threshold)

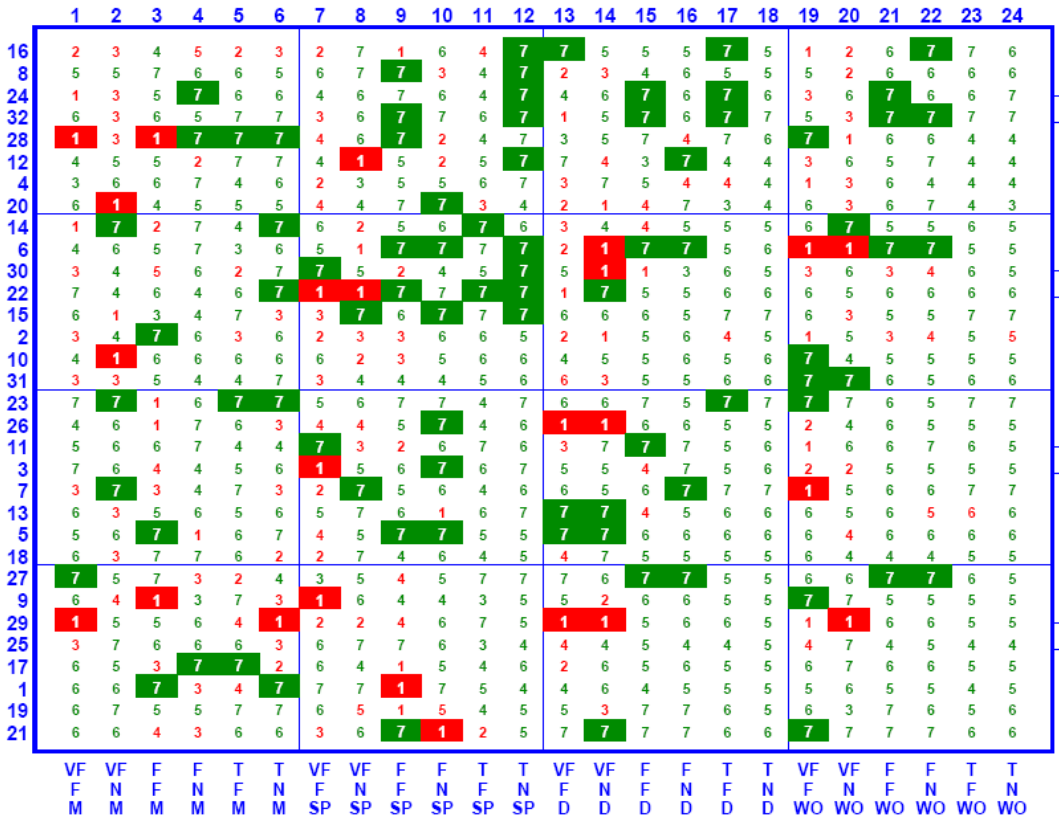


Figure 8-71. TCP Goodput Rank Matrix – y16(u) – CTCP (Low Initial Slow-Start Threshold)

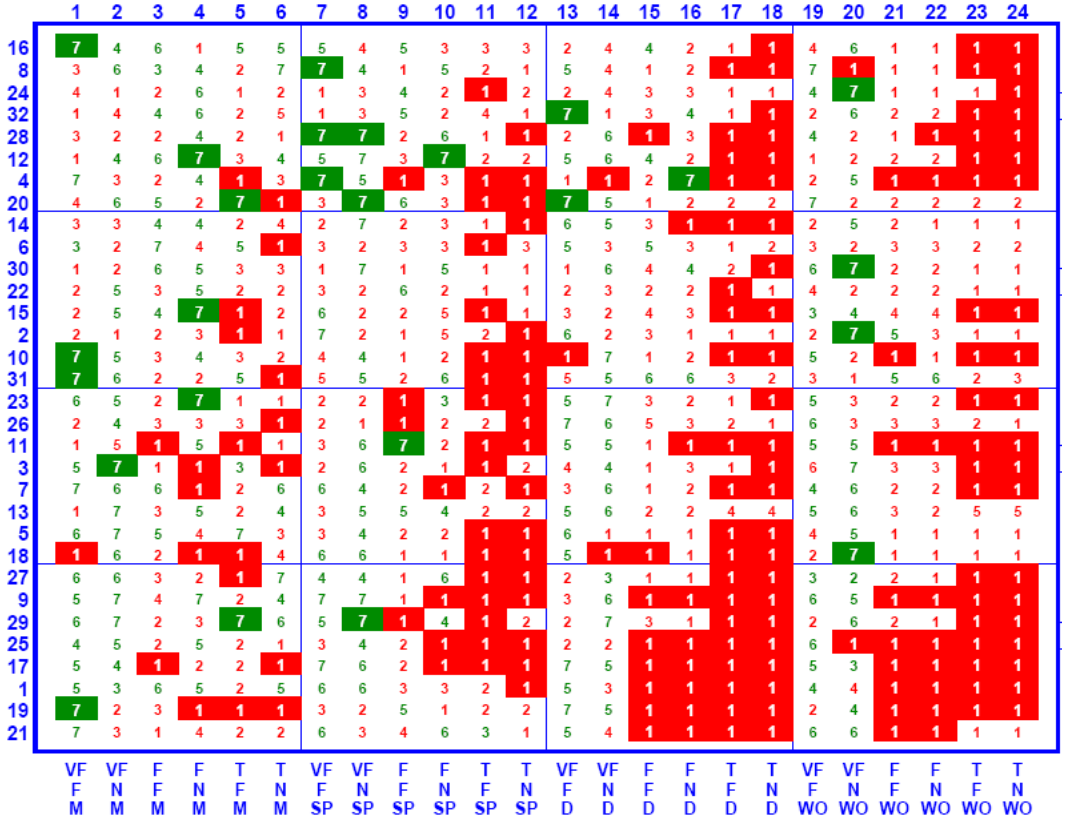


Figure 8-72. TCP Goodput Rank Matrix – y16(u) – FAST (Low Initial Slow-Start Threshold)

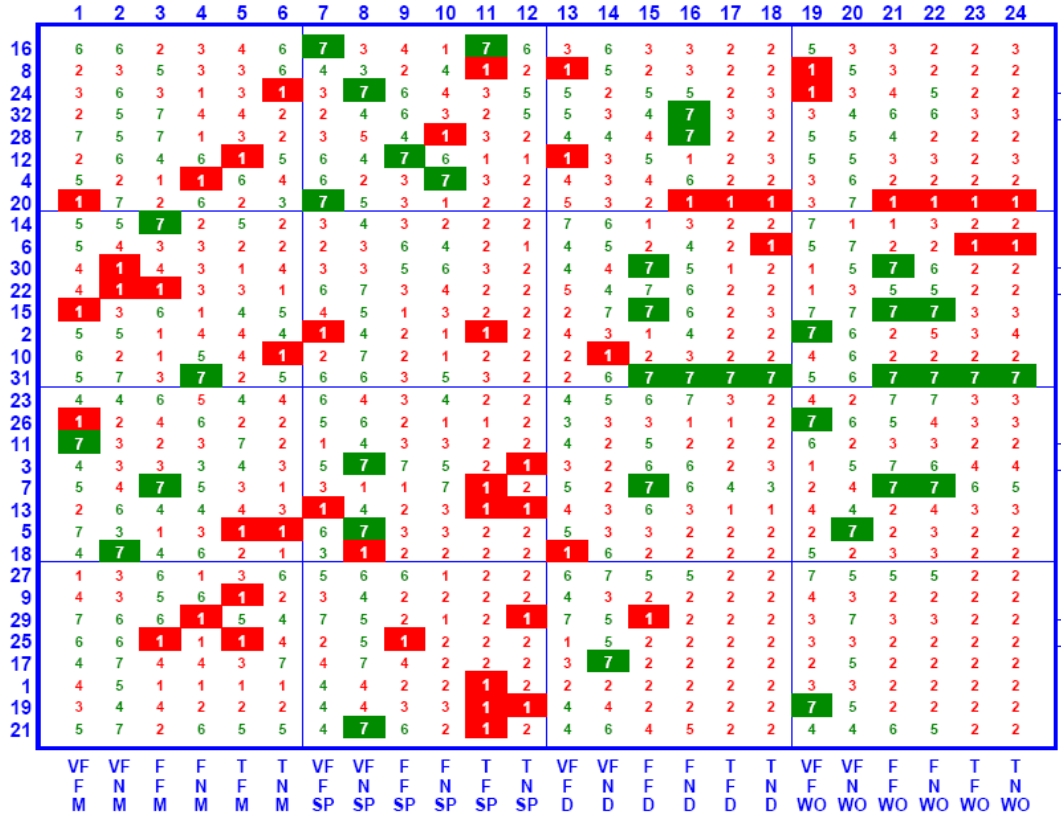


Figure 8-73. TCP Goodput Rank Matrix – y16(u) – FAST-AT (Low Initial Slow-Start Threshold)

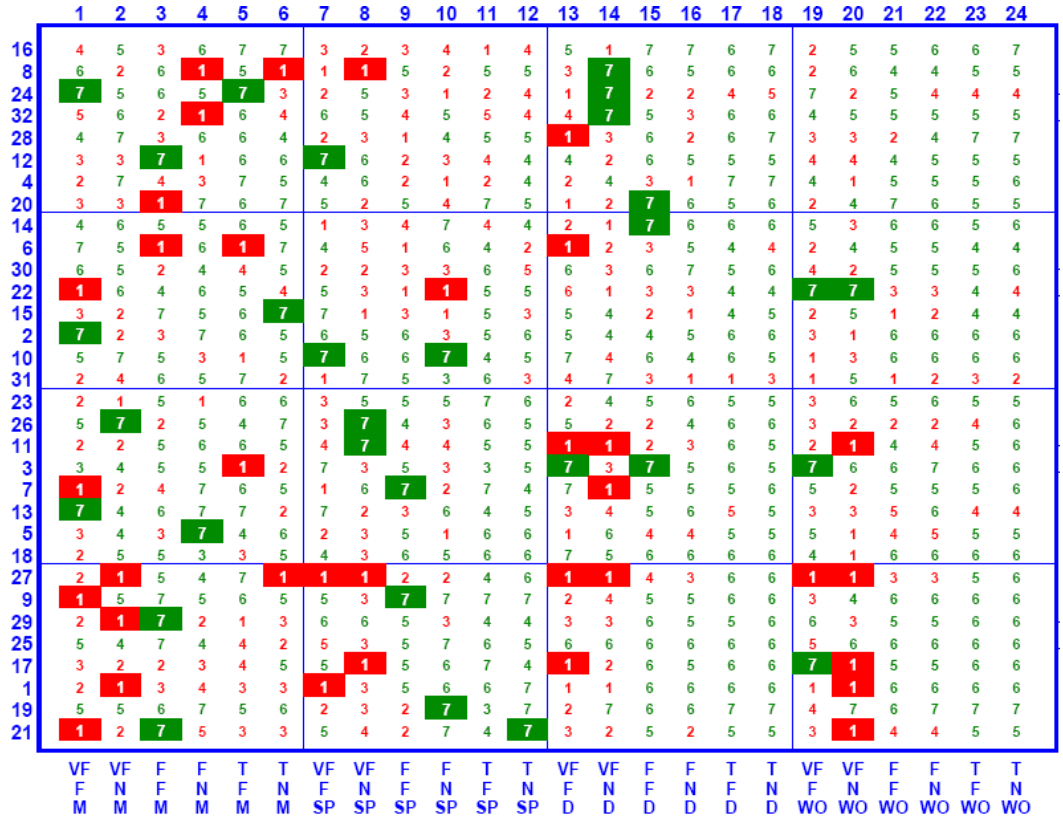


Figure 8-74. TCP Goodput Rank Matrix – y16(u) – HSTCP (Low Initial Slow-Start Threshold)

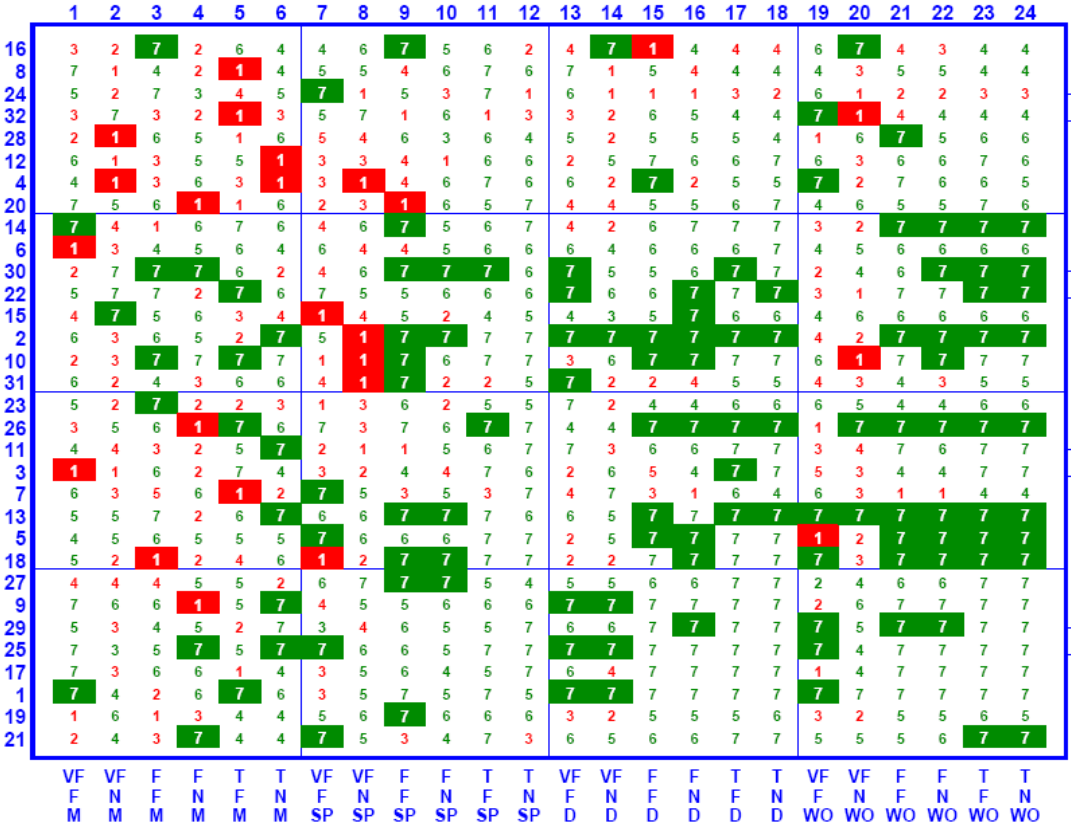


Figure 8-75. TCP Goodput Rank Matrix – y16(u) – HTCP (Low Initial Slow-Start Threshold)

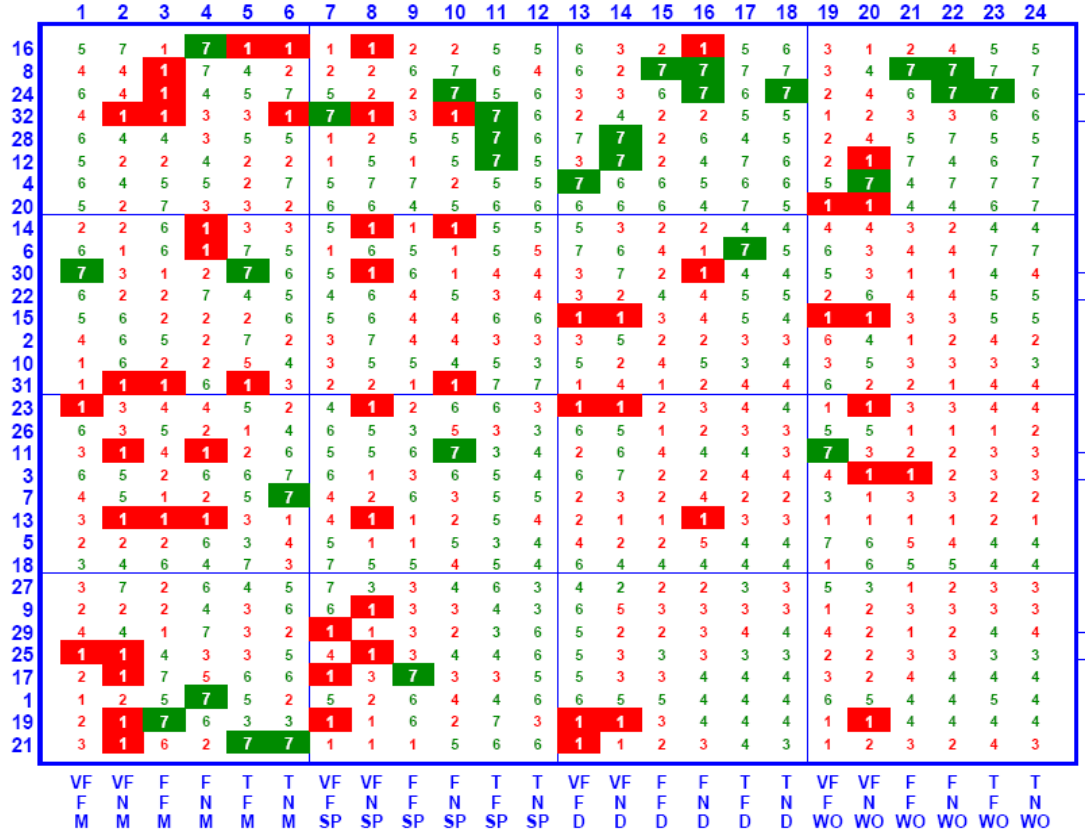


Figure 8-76. TCP Goodput Rank Matrix – y16(u) – Scalable TCP (Low Initial Slow-Start Threshold)

Table 8-32. Summary Average and Standard Deviation in Goodput and TCP Goodput Rank for All Congestion Control Algorithms (Low Initial Slow-Start Threshold)

		BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
y2(u)	M	4.60	3.70	5.24	4.31	3.46	2.36	4.32
	SP	4.36	3.89	6.42	6.58	2.66	1.94	2.16
	D	3.85	4.74	6.11	6.70	2.79	2.21	1.61
	WO	2.74	4.63	4.68	5.58	3.54	4.28	2.55
	Avg.	3.89	4.24	5.61	5.79	3.11	2.70	2.66
y16(u)	M	3.61	4.90	3.54	3.64	4.32	4.32	3.68
	SP	3.71	4.98	2.89	3.20	4.23	5.02	3.97
	D	3.28	5.11	2.54	3.29	4.52	5.47	3.79
	WO	3.50	5.18	2.40	3.50	4.51	5.35	3.56
	Avg.	3.52	5.04	2.84	3.41	4.39	5.04	3.75
y2(u) & y16(u)	Avg.	3.71	4.64	4.23	4.60	3.75	3.87	3.21
	Std.	0.59	0.55	1.60	1.47	0.75	1.47	0.97

Figs. 8-63 through 8-76 and Table 8-32 reveal the key differences in relative goodput, under low initial slow-start threshold, among flows using alternate congestion

control protocols and also among competing TCP flows. First, FAST and FAST-AT provide highest relative goodputs due largely to very quick increase in transmission rate after reaching the initial slow-start threshold. On the other hand, the quick increase can lead to losses, from which TCP flows recover linearly. Thus, FAST interferes most with TCP flows. FAST-AT interferes somewhat less than FAST because, under sustained congestion, FAST-AT flows do not increase transmission rate as quickly as FAST flows. Second, Scalable TCP and BIC flows still interfere significantly with TCP flows – the reasons are as discussed earlier. In addition, Scalable flows see significant goodput only on the largest files. This occurs because Scalable TCP increases transmission rate steeply only after some period of delay. The largest files last long enough for Scalable TCP to reach the steep increase in transmission rate. Third, CTCP and HTCP are least disruptive to the throughput of competing TCP flows. CTCP still does better than HTCP in providing goodput on flows running alternate congestion control procedures. Contrasts in relative goodput between flows using alternate congestion control and TCP flows account for the large standard deviations in rank exhibited by FAST, FAST-AT and HTCP.

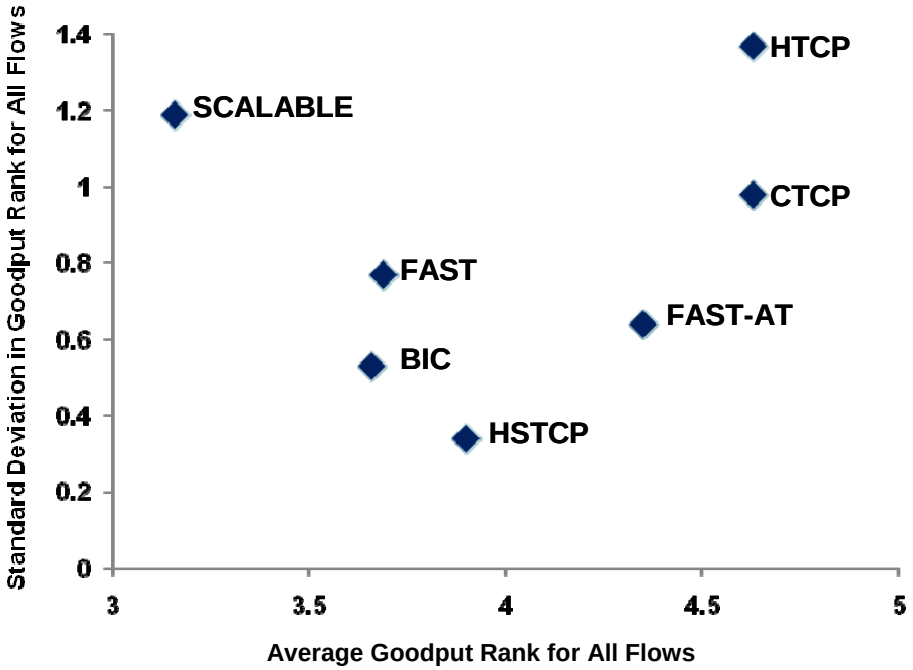


Figure 8-77. Average (x axis) vs. Standard Deviation (y axis) in Goodput Rank (High Initial Slow-Start Threshold)

8.4.3.3 Summary of Differences in Relative Goodput. To summarize differences in relative goodputs we plot the average goodput rank (x axis) against the standard deviation in goodput rank (y axis) under high (Fig. 8-77) and low (Fig. 8-78) initial slow-start thresholds for each alternate congestion control regime. The average and standard deviations consider goodput rank on flows using an alternate congestion control regime and also on competing TCP flows. In such a plot, the ideal congestion control regime would appear in the lower right-hand corner – high average rank in goodput applied evenly to all competing flows. Where alternate congestion control regimes provide

equally high average rankings, one should prefer the regime with lower standard deviation in rank.

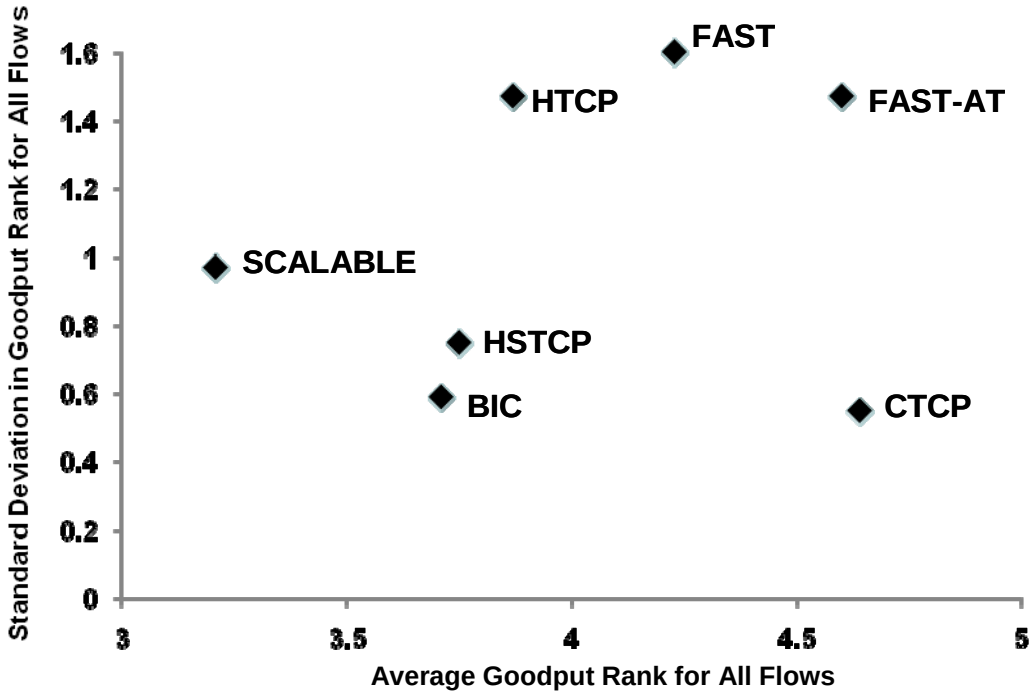


Figure 8-78. Average vs. Standard Deviation in Goodput Rank (Low Initial Slow-Start Threshold)

Fig. 8-77 and 8-78 show CTCP to be best with respect to relative goodput rank. CTCP provides the highest average rank (over 4.5) and the lowest standard deviation among high ranking alternatives (e.g., HTCP in Fig. 8-77 and FAST-AT in Fig. 8-78). Further, Scalable TCP is worst with respect to relative goodput rank and BIC is second worst. HSTCP ranks in the middle. HTCP ranks well with respect to average goodput under large initial slow-start threshold, but ranks significantly less well under low initial slow-start threshold. Further, HTCP exhibits a high standard deviation, interfering relatively little with TCP flows, while underperforming other alternate congestion control algorithms with respect to large files. The relative performance of FAST-AT might be considered second best, though due to its rapid increase in transmission rate (under low initial slow-start threshold) FAST-AT can induce losses in TCP flows, which recover only linearly. FAST induces more losses in TCP flows than FAST-AT, and also benefits from the same (as FAST-AT) rapid increase in transmission rate when the initial slow-start threshold is low.

8.5 Findings

This experiment considered a range of files sizes (movies, service packs, documents and Web objects) being transferred across a relatively uncongested network, where some (fast and typical) paths experienced more congestion than others (very fast paths) and where some flows could achieve a maximum rate of 80×10^3 pps, while others were constrained (by the interface speed of a sender or receiver) to at most 8×10^3 pps. Flows using TCP congestion control were mixed with flows using one of seven alternate congestion control

algorithms. In general, under these conditions, goodput experienced on flows is influenced by three main factors (when ignoring network speed and delay): (1) quickeness with which the maximum transfer rate is achieved, (2) file size and (3) packet losses and related recovery procedures. The 32 conditions simulated in this experiment were run twice: once with a high and once with a low initial slow-start threshold. Under a high threshold, all flows used the same algorithm (limited slow-start) to find the maximum transfer rate. In such cases, only file size and packet loss/recovery procedures served to distinguish goodput among the various algorithms investigated. Under a low threshold, flows discovered the maximum transfer rate using techniques associated with the specific algorithms. In such cases, the quickness with which a flow could reach maximum transfer rate is the largest factor distinguishing among goodput.

8.5.1 Finding #1

Under low congestion, choice of initial slow-start threshold significantly influenced goodput differences between TCP flows and flows running alternate congestion control algorithms. Given a high threshold, all flows discovered the maximum available transmission rate using the same slow-start algorithm. In such cases, goodput differences between TCP flows and flows running alternate algorithms were diminished greatly, depending only on differences associated with loss/recovery procedures. Loss/recovery procedures played a larger role with bigger files (more packets mean more losses) and in congested areas and conditions (more simultaneous flows lead to more losses). Given a low threshold, all alternate algorithms yielded superior performance to standard TCP due to TCP's linear rate of increase in transmission toward the maximum rate. FAST and FAST-AT, which showed the quickest increase to the maximum transmission rate, benefited most from a low initial slow-start threshold and exhibited significantly higher goodputs (than the other algorithms) for all but the smallest files. CTCP achieved the second fastest pace of increase to maximum rate.

8.5.2 Finding #2

With increasing losses, due to large file size or path congestion, goodputs were distinguished mainly by loss/recovery procedures. Scalable TCP, BIC and HSTCP do not decrease their transmission rate as much as the other algorithms when a loss is detected. This means that already established flows continue to transmit at higher rates, inducing losses in newer flows, and also in ongoing TCP flows, which cut their transmission rate in half on each loss. Thus, under congested conditions, Scalable TCP, BIC and HSTCP interfered most with competing TCP flows. On the other hand, FAST, FAST-AT, CTCP and HTCP reduce transmission rate by half on a loss, which mirrors the reduction of TCP flows. Of course, FAST (and sometimes FAST-AT) subsequently increases transmission rate quickly to recover from the loss, while CTCP increases transmission rate second most quickly. HTCP delays for one second without another packet loss before increasing transmission rate more than linearly, so HTCP lagged somewhat in recovering from losses.

The ability of FAST to rapidly increase transmission rate on loss recovery was somewhat of a double-edged sword. Increased rate of transmission by competing FAST flows could induce additional losses in TCP flows, which recover at only a linear rate. Thus, under such circumstances, FAST could interfere markedly with TCP flows. FAST-

AT includes the ability to reduce the α parameter, so when congestion is significant FAST-AT flows recover less aggressively than FAST flows. For this reason, FAST-AT interfered somewhat less with standard TCP flows.

8.5.3 Finding #3

Overall, CTCP provided the best balance in relative goodput achieved on all flows. CTCP interfered little (but HTCP interfered least) with TCP flows and proved second best (after FAST/FAST-AT) at providing goodput to flows using alternate congestion control procedures. FAST-AT disrupted TCP flows somewhat more than HTCP and CTCP, while providing nearly the best goodput to flows using alternate procedures.

8.5.4 Finding #4

As seen in earlier experiments, this experiment showed that use of some alternate congestion control protocols altered selected macroscopic characteristics of the network. Here, however, the characteristic changes were, in general, not statistically significant. We attribute this to two main factors: (1) overall congestion levels were kept much lower than in previous experiments (e.g., Chapters 6 and 7) and (2) FAST and FAST-AT, which have similar characteristics, where not separated in the analyses, which (as discussed in Chapter 7) tended to reduce the statistical significance that might be attributed to either algorithm considered without the other. In general, the current experiments confirmed (as seen previously in Chapters 6 and 7) that FAST and FAST-AT tend to increase retransmission rate under higher congestion. Thus, more flows are pending in the connecting state and fewer flows complete per unit of time. In addition, Scalable TCP tends to increase buffer occupancy throughout the network. As discussed in Sec. 8.4.1, this can also lead to higher losses and increased retransmission rates, to more flows pending in the connecting state and to fewer flows completing per unit time. At lower congestion levels, Scalable TCP performed worse on these metrics than FAST. At higher congestion levels, FAST performed worse. Finally, we found again in this experiment that CTCP can exhibit a much higher average congestion window size than other congestion control algorithms. The increase appears more prominent under lower levels of congestion.

8.6 Conclusions

In this section, we described an experiment to investigate effects on macroscopic behavior and user experience when deploying various congestion control algorithms in a simulated, heterogeneous network, i.e., a network that includes flows operating under normal TCP congestion control procedures together with flows operating under one of seven alternate congestion control algorithms. Mixing each alternate congestion control regime together with standard TCP enabled us to investigate the influence of alternate congestion avoidance algorithms on the performance of TCP flows, as might prove important during a period of transition from TCP toward adoption of an alternate congestion control regime. Under half of the test conditions more flows operated with TCP, as might be typical in earlier stages of transition to an alternate congestion control regime, while under the remaining test conditions more flows operated with an alternate congestion control regime, as might be typical in later stages of transition. We also introduced additional flow sizes to represent downloading movies and software service

packs. These file sizes augmented the Web objects and document downloads used in previous experiments. We adopted a small-scale network because earlier experiments suggested that a small network yields significant information while requiring fewer resources. Reducing computational cost allowed us to repeat our experiments first with a high initial slow-start threshold and then with a low initial slow-start threshold. We took this step in light of the apparent significance of the initial slow-start threshold, as uncovered in earlier experiments (Chapters 6 and 7).

We demonstrated that, under the conditions simulated, and setting aside network speed and delay, goodput experienced on flows is influenced by three main factors: (1) quickness with which the maximum transfer rate is achieved, (2) file size and (3) packet losses and related recovery procedures. We showed that adopting a high initial slow-start threshold throughout the network allowed all flows to reach maximum transfer rate at the same speed, which substantially reduced goodput differences among TCP flows and flows using alternate congestion control algorithms. With a high threshold, only loss/recovery procedures distinguished goodput among congestion control algorithms. We found that on a loss, Scalable TCP, BIC and HSTCP reduced transmission rate less than other algorithms, causing Scalable TCP, BIC and HSTCP to interfere more with TCP flows under congested conditions. While CTCP, FAST and FAST-AT (and sometimes HTCP) halved their transmission rate on a loss, FAST (and sometimes FAST-AT) where able to increase transmission rate at the quickest pace, followed by CTCP. The pace of increase of HTCP was much less. Under heavy congestion, FAST-AT was less aggressive in recovering from losses than was FAST.

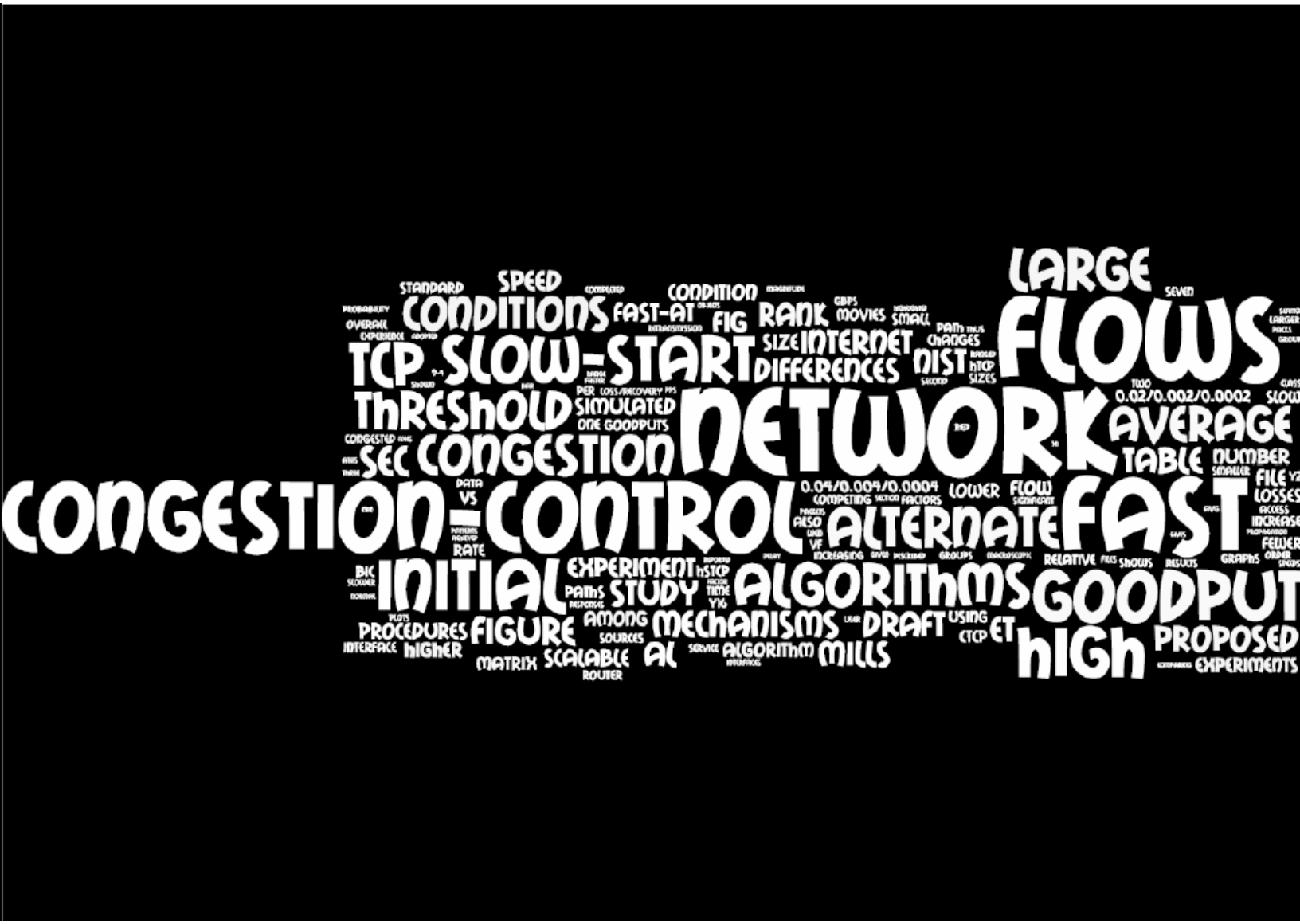
We showed that under a low initial slow-start threshold all of the alternate congestion control algorithms reached the maximum transmission rate much more quickly than TCP, which was limited to a linear rate of increase. FAST (and FAST-AT) increased transmission rate most quickly, followed by CTCP. Scalable TCP increased transmission rate least quickly during a flow's initial period before achieving a steep rate of increase, so under a low initial slow-start threshold, Scalable achieved substantial goodputs only on large files. Differences in the speed of increase in transmission rate among the other congestion control algorithms (BIC, HSTCP and HTCP) did not appear significant. We found that CTCP gave the best balance in goodput among all flows, but FAST and FAST-AT flows achieved the highest goodputs when all flows used a low initial slow-start threshold.

We were also able to confirm some network-wide results from earlier experiments, where FAST and FAST-AT exhibited higher retransmission rates, more pending flow connections and fewer flows completing. In addition, we found that, under high initial slow-start threshold, Scalable TCP could also exhibit such undesirable network-wide properties.

In the next section, we revisit the results from this section by rerunning the experiment on a larger (10 times more sources) and faster (10 times higher capacity) network. The substantial increase in computational requirements arising from this increase in network size and speed will limit us to consider only one setting for initial slow-start threshold. We chose the high initial slow-start threshold in order to focus on differences in loss/recovery processing. We expect that the larger network will experience substantially less congestion under most conditions. Given the findings from the current section, we suspect a less congested network, where all flows use a high

initial slow-start threshold, will yield a narrowing of differences in goodput among the alternate congestion control algorithms. Fewer losses should mean that the algorithms have fewer opportunities to invoke their loss/recovery behaviors.

Chapter 9 – Comparing Congestion Control Regimes in a Large, Fast, Heterogeneous Network



9 Comparing Congestion Control Regimes in a Large, Fast, Heterogeneous Network

In this chapter, we repeat the fundamental experiment design and data analyses described in the previous chapter (Chapter 8), while increasing the scale (speed and size) of the simulated network by one order of magnitude. Because increasing network scale will also increase the computational resources needed for executing the experiments, we decided to limit the simulations to cases where the initial slow-start threshold is set high. We made this choice in order to focus on the loss/recovery aspects of the alternate congestion control algorithms.

Table 9-1. Comparison of Experiment with Congestion Control Algorithms in a Small Network (Chapter 8) vs. Experiment in a Large, Fast Network (Chapter 9)

Characteristic	Chapter 8 High SST	Chapter 9 High SST
Network Size (sources)	17.455×10^{-3} & 26.085×10^{-3}	174.6×10^{-3} & 261.792×10^{-3}
Backbone Speed (Gbps)	19.2 & 38.4	192 & 384
Packet Loss Rate	1×10^{-4} to 1×10^{-2}	2×10^{-9} to 2×10^{-2}
Initial Slow-Start Threshold	$2^{32}/2$ packets	$2^{32}/2$ packets
Alternate Congestion Control Algorithms & Associated Identifiers	1-BIC, 2-CTCP, 3-FAST, 4-FAST-AT, 5-HSTCP, 6-HTCP, 7-Scalable	1-BIC, 2-CTCP, 3-FAST, 4-FAST-AT, 5-HSTCP, 6- HTCP, 7-Scalable
Ratio (%) of Sources using Alternate Congestion-Control to Standard TCP Congestion-Control	30:70 & 70:30	30:70 & 70:30
Scenario	60 min. – 96-98% Web objects; 2-4% document transfers; smaller number of service-pack and movie downloads	60 min. – 96-98% Web objects; 2-4% document transfers; smaller number of service-pack and movie downloads

Table 9-1 highlights in red differences from the relevant experiment reported in Chapter 8. As indicated, we compared the seven congestion control algorithms under the same mix of sources with the same traffic patterns as used in Chapter 8. We also set the initial slow-start threshold to a high value and simulated network operation for one hour. As the table shows, we increased the number of sources and network speed tenfold. One ramification of increasing network speed is to extend the range of congestion conditions, as measured by packet-loss rate. Specifically, the experiment conditions in Chapter 9 led to five orders of magnitude lower congestion for the least congested case. Note, however, that the experiments in both Chapters 8 and 9 have the same order of losses under the condition with highest congestion. To the extent that faster network speeds permit lower congestion, and thus fewer losses, we expected the performance of the alternate

algorithms to become closer to each other, and to the performance of standard TCP congestion control. This follows from the fact that in these experiments only loss/recovery procedures can distinguish the alternate algorithms from each other and from TCP. Fewer losses equate to fewer chances to distinguish among the various congestion control algorithms.

9.1 Changes in Experiment Design

Except as described in this section, we adopted the same parameter settings used for the experiment reported in Chapter 8. Below, we discuss the few changes we made in robustness factors and fixed factors and we report the resulting experiment conditions. We then describe how these few changes affected the domain view of the experiment conditions. We close with a recap of responses recorded.

9.1.1 Changes in Robustness Factors and Fixed Factors

Table 9-2 specifies the robustness factors and values we used for this experiment. Highlighted in red are the only changes from Chapter 8 – we multiplied the network speed settings by 10. Table 9-3 identifies (in red) the only change we made to the fixed factors used in Chapter 8. We multiplied the base number of sources by 10. These two changes led to the desired order of magnitude increase in network speed and size.

Table 9-2. Robustness Factors Adopted for Comparing Congestion Control Mechanisms (Changes from Chapter 8 highlighted in red)

Identifier	Definition	PLUS (+1) Value	Minus (-1) Value
x1	Network Speed	16000 p/ms	8000 p/ms
x2	Propagation Delay Multiplier	2	1
x3	Buffer Size Adjustment Factor	1	0.5
x4	Think Time	7500 ms	5000 ms
x5	Average File Size for Web Objects	150 packets	100 packets
x6	Distribution for Sizing Large Files	2	1
x7	Probability of Fast Source	.7	.3
x8	Probability of Alternate Congestion-Control Algorithm	.7	.3
x9	Multiplier on Base Number of Sources (ΔU)	3	2

Table 9-3. Key Fixed Factors Adopted for Comparing Congestion Control Mechanisms (Change from Chapter 8 highlighted in red)

Parameter	Definition	Value
Bsources	Basic unit for sources per access router	1000
P(Ns)	Probability source under normal access router	0.1
P(Nsf)	Probability source under fast access router	0.6
P(Nsd)	Probability source under directly connected access router	0.3
P(Nr)	Probability receiver under normal access router	0.6
P(Nrf)	Probability receiver under fast access router	0.2
P(Nrd)	Probability receiver under directly connected access router	0.2
sst_{INT}	Initial slow-start threshold (packets)	$2^{31}/2$

9.1.2 Changes in Orthogonal Fractional Factorial Design of Robustness Conditions

Increasing network speed caused the experiment conditions to change only with respect to a single factor (x1). The resulting 32 experiment conditions are shown in Table 9-4.

Table 9-4. Two-Level 2^{9-4} Orthogonal Fractional Factorial Design (Changes from Chapter 8 highlighted in red)

Factor->	x1	x2	x3	x4	x5	x6	x7	x8	x9
Condition	--	--	--	--	--	--	--	--	--
1	8000	1	0.5	5000	100	0.04/0.004/0.0004	0.7	0.7	3
2	16000	1	0.5	5000	100	0.04/0.004/0.0004	0.3	0.3	2
3	8000	2	0.5	5000	100	0.02/0.002/0.0002	0.7	0.3	2
4	16000	2	0.5	5000	100	0.02/0.002/0.0002	0.3	0.7	3
5	8000	1	1	5000	100	0.02/0.002/0.0002	0.3	0.7	2
6	16000	1	1	5000	100	0.02/0.002/0.0002	0.7	0.3	3
7	8000	2	1	5000	100	0.04/0.004/0.0004	0.3	0.3	3
8	16000	2	1	5000	100	0.04/0.004/0.0004	0.7	0.7	2
9	8000	1	0.5	7500	100	0.02/0.002/0.0002	0.3	0.3	3
10	16000	1	0.5	7500	100	0.02/0.002/0.0002	0.7	0.7	2
11	8000	2	0.5	7500	100	0.04/0.004/0.0004	0.3	0.7	2
12	16000	2	0.5	7500	100	0.04/0.004/0.0004	0.7	0.3	3
13	8000	1	1	7500	100	0.04/0.004/0.0004	0.7	0.3	2
14	16000	1	1	7500	100	0.04/0.004/0.0004	0.3	0.7	3
15	8000	2	1	7500	100	0.02/0.002/0.0002	0.7	0.7	3
16	16000	2	1	7500	100	0.02/0.002/0.0002	0.3	0.3	2
17	8000	1	0.5	5000	150	0.02/0.002/0.0002	0.3	0.3	2
18	16000	1	0.5	5000	150	0.02/0.002/0.0002	0.7	0.7	3
19	8000	2	0.5	5000	150	0.04/0.004/0.0004	0.3	0.7	3
20	16000	2	0.5	5000	150	0.04/0.004/0.0004	0.7	0.3	2
21	8000	1	1	5000	150	0.04/0.004/0.0004	0.7	0.3	3
22	16000	1	1	5000	150	0.04/0.004/0.0004	0.3	0.7	2
23	8000	2	1	5000	150	0.02/0.002/0.0002	0.7	0.7	2
24	16000	2	1	5000	150	0.02/0.002/0.0002	0.3	0.3	3
25	8000	1	0.5	7500	150	0.04/0.004/0.0004	0.7	1	2
26	16000	1	0.5	7500	150	0.04/0.004/0.0004	0.3	0.3	3
27	8000	2	0.5	7500	150	0.02/0.002/0.0002	0.7	0.3	3
28	16000	2	0.5	7500	150	0.02/0.002/0.0002	0.3	0.7	2
29	8000	1	1	7500	150	0.02/0.002/0.0002	0.3	0.7	3
30	16000	1	1	7500	150	0.02/0.002/0.0002	0.7	0.3	2
31	8000	2	1	7500	150	0.04/0.004/0.0004	0.3	0.3	2
32	16000	2	1	7500	150	0.04/0.004/0.0004	0.7	0.7	3

9.1.3 Changes in Domain View of Robustness Conditions

Changes in speed and size influence the domain view of our simulated network, as reported in Tables 9-5 through 9-7, where changes from the experiment in Chapter 8 are highlighted in red. Table 9-5 shows simulated router speeds for this experiment, which are comparable to speeds that might be seen in contemporary networks. Increasing **Bsources** (base number of sources) to 10^3 scales the number of potentially active flows to a level that matches the simulated network speeds. Table 9-6 shows the number of sources for each level of factor x9 (multiplier on **Bsources**). The number of receivers is four times the number of sources. We used the same topology, including propagation delays, as in previous experiments. Buffer sizing is influenced by three factors: network speed (x1), propagation delay (x2) and buffer-size adjustment factor (x3). Table 9-7 characterizes buffer sizes for each router level under both values for factor x3.

Table 9-5. Simulated Router Speeds

Router	PLUS (+1)	Minus (-1)
Backbone	384 Gbps	192 Gbps
POP	48 Gbps	24 Gbps
Normal Access	4.8 Gbps	2.4 Gbps
Fast Access	9.6 Gbps	7.2 Gbps
Directly Connected Access	48 Gbps	24 Gbps

Table 9-6. Number of Simulated Sources

PLUS (+1)	Minus (-1)
261.792×10^3	174.6×10^3

Table 9-7. Characterization of Simulated Buffer Sizes

	x3 = 1.0			x3 = 0.5		
Router	Min	Avg	Max	Min	Avg	Max
Backbone	651×10^3	1.464×10^6	2.604×10^6	326×10^3	732×10^3	1.302×10^6
POP	81×10^3	183×10^3	325×10^3	41×10^3	92×10^3	163×10^3
Access	13×10^3	29×10^3	52×10^3	6.5×10^3	14.6×10^3	25.9×10^3

Fig. 9-1 plots the retransmission rates for each of the 32 simulated conditions. The x axis is ordered by increasing retransmission rate. Using visual guidance, we divided congestion conditions into six categories moving from little congestion (**C1**) to relatively high congestion (**C6**). Except for the highest congestion category (**C6**), the simulated conditions exhibit several orders of magnitude reduction in congestion when compared with the experiments in Chapter 8 (recall Figs. 8-1 and 8-2).

To further explore the nature of congestion under the conditions simulated for this experiment, we examined six time series. We chose one condition from the middle of each congestion class. Fig. 9-2 plots related time series. We selected the following conditions, one from each congestion class C1 through C6: 4, 6, 31, 7, 29 and 19. The y axis indicates the number of flows in a particular state: connecting (gold) or active (red). Active flows may be operating in initial slow start (green), normal congestion avoidance (brown) or alternate congestion avoidance (blue). In these particular plots, CTCP flows were operating in the network along with flows using standard TCP congestion control procedures. The discussion considers only the relative distances between the curves on the graphs, so inability to read the axes will be immaterial. The number of active flows generally appears to be on the order of 10^4.

Under the least congested condition (4), nearly all active flows operate in initial slow-start, and few losses occur. In general, as congestion increases with condition, the relative number of active flows in initial slow-start decreases and the relative number under normal congestion avoidance procedures increases. That is, the green and brown

lines come closer together.[1] The number of flows under alternate congestion avoidance procedures (blue) shifts up or down slightly depending on whether a particular condition has 30 % or 70 % of the sources equipped with an alternate congestion control algorithm.

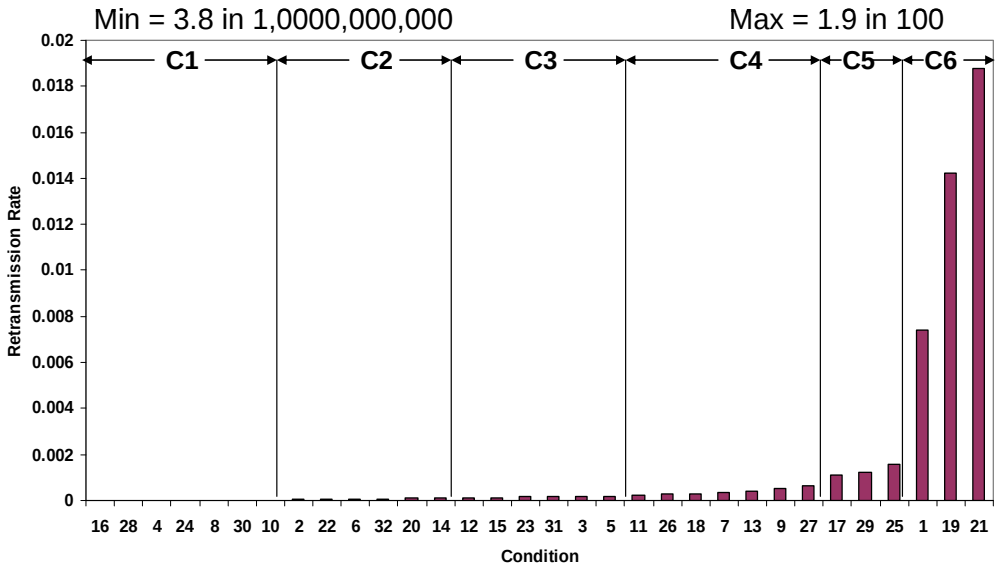


Figure 9-1. Conditions Ordered Least to Most Congested under High Initial Slow-Start Threshold

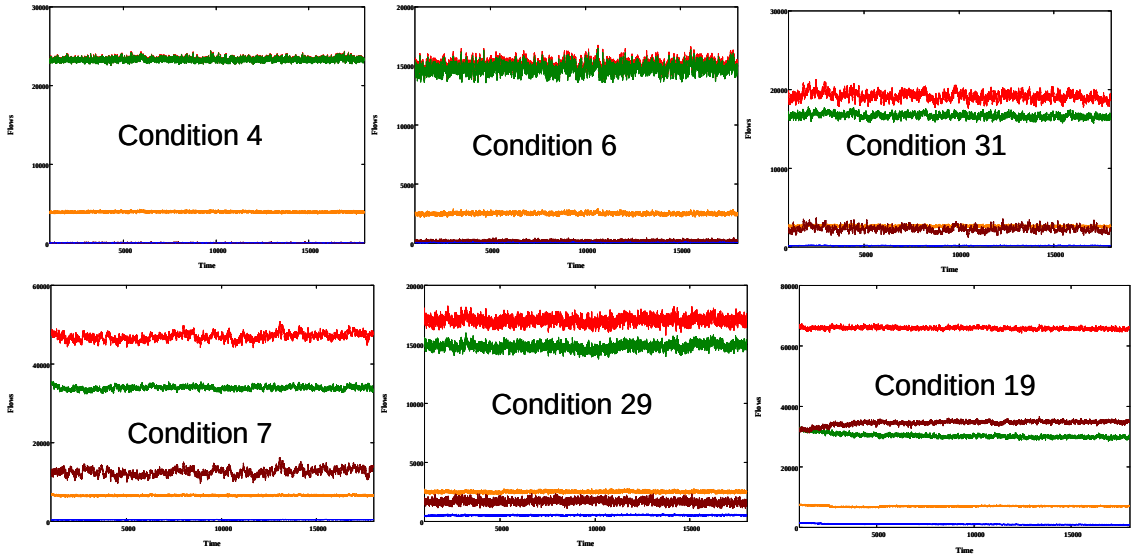


Figure 9-2. Distribution of Flow States for Six Conditions with Increasing Congestion (Flows are either connecting (gold) or sending (red) and sending flows may be in one of three congestion control states: initial slow start (green), normal congestion avoidance (brown) or alternate congestion avoidance (blue)

[1] Note that this trend is not monotonic – the green and brown lines move farther apart as condition advances from 7 to 29. We attribute this to the fact that condition 29 is the only condition among conditions 31, 7, 29 and 19 that has a lower probability of larger file sizes. This means more files can complete in initial slow start under condition 29, than under the other three conditions.

9.1.4 Responses Measured

We measured the same responses for this experiment as we measured for the experiments discussed in Chapter 8. We measured 16 responses characterizing macroscopic behavior of the network and 28 responses representing user experience in each of 24 flow groups. Refer to back Sec. 8.1.4 for a definition of the responses.

9.2 Experiment Execution and Data Collection

Table 9-8 compares resource requirements for simulating the large, fast network against resource requirements for simulating the smaller network used in Chapter 8. Simulating the large, fast network over (7 algorithms x 32 conditions =) 224 runs, required about 11 processor years, compared with only 2/3 of a processor year for simulating the same number of runs given the smaller network. Scaling up the network by an order of magnitude led to increasing computation requirements by a factor of 16 or so. Table 9-9 shows that the increase in the number of packets sent and flows simulated was approximately linear (i.e., tenfold). The higher than linear increase in computation requirements can be attributed to extra processing time associated with managing larger event lists. Since we collected data in the same form as described in Sec. 8.2.2, increasing the scale of the simulation did not increase the amount of data collected.

9.3 Data Analysis Approach

We used the same data analysis approach described in Sec. 8.3. We focused mainly on user experience in each of 24 flow classes (recall Table 8-6), where we investigated both absolute and relative differences. We examined macroscopic data with detailed analyses for each of the 16 responses, applying a Grubbs' test to residuals about the mean associated with each of the 32 conditions.

Table 9-8. Comparing Resource Requirements for Simulating a Small Network (from Chapter 8) and a Large, Fast Network (from Chapter 9)

	Small, Slow Network with High Initial Slow-Start Threshold	Large, Fast Network with High Initial Slow-Start Threshold
CPU hours (224 Runs)	5.857×10^3	94.355×10^3
Avg. CPU hours (per run)	26.15	421.23
Min. CPU hours (one run)	12.58	203.04
Max. CPU hours (one run)	43.97	739.04
Avg. Memory Usage (Mbytes)	196.56	2.392×10^3

Table 9-9. Comparing Number of Simulated Flows and Packets for a Small Network (from Chapter 8) and a Large, Fast Network (from Chapter 9)

	Small, Slow Network with High Initial Slow-Start Threshold		Large, Fast Network with High Initial Slow-Start Threshold	
Statistic	**Flows Completed**	**Data Packets Sent**	**Flows Completed**	**Data Packets Sent**
Avg. Per Condition	11.466×10^6	3.414×10^9	116.317×10^6	33.351×10^9
Min. Per Condition	7.258×10^6	2.139×10^9	72.945×10^6	21.069×10^9
Max. Per Condition	17.391×10^6	$5.048, \times 10^9$	175.948×10^6	50.932×10^9
Total all Runs	2.568×10^9	764.740×10^9	26.055×10^9	7.471×10^{12}

9.4 Results

In this section, we present selected simulation results in three categories: (1) macroscopic network behavior, (2) absolute user experience and (3) relative user experience. We present only data that reveals behavioral similarities and differences of interest. In some cases, we compare results with results obtained from one of the experiments in Chapter 8. Specifically, we compare results under a high initial slow-start threshold.

9.4.1 Macroscopic Network Behavior

In general, as we found in the earlier experiment (Chapter 8), the data analyses reported in this section do not reveal much in the way of statistically significant changes in macroscopic network behavior. This appears due mainly to the general lack of congestion throughout the experiment conditions. As in the results from Chapter 8, we consider both FAST (algorithm 3) and FAST-AT (algorithm 4) together, which reduces the statistical significance of either algorithm considered alone because both algorithms share some traits (as described previously in Chapter 7). Despite the lack of statistical significance, we could discern patterns in macroscopic network behavior with respect to some responses. In most cases, the patterns detected here echo patterns seen in Chapter 8 under a high initial slow-start threshold. The patterns appeared less distinct in the current experiments because overall levels of congestion were much lower across most of the 32 simulated conditions. We report the patterns we found informative.

Fig. 9-3 shows the average number of flows attempting to connect. In the six conditions with highest congestion (17, 29, 25, 1, 19 and 21), FAST and FAST-AT had more flows pending in the connecting state than other algorithms. This was especially so for the three most congested conditions. This result is consistent with results from our other experiments, which showed that FAST and FAST-AT led flows to take longer to connect in the face of significant congestion. Most conditions in the current experiment did not lead to significant congestion, but where significant congestion existed FAST and FAST-AT induced more losses in SYN packets. In addition, as shown in Fig. 9-4, under highly congested conditions, FAST and FAST-AT induced higher retransmission rates. Fig. 9-4 also mirrors results in Fig. 8-36 – under conditions of lower congestion, Scalable TCP induced more losses and retransmissions than other algorithms. Comparing Fig. 9-4 with Fig. 8-36 shows that Scalable TCP induced more losses under more conditions in Fig. 9-4. This should be expected because the current experiment has significantly lower congestion under most conditions than was the case for the previous experiment (Chapter 8).

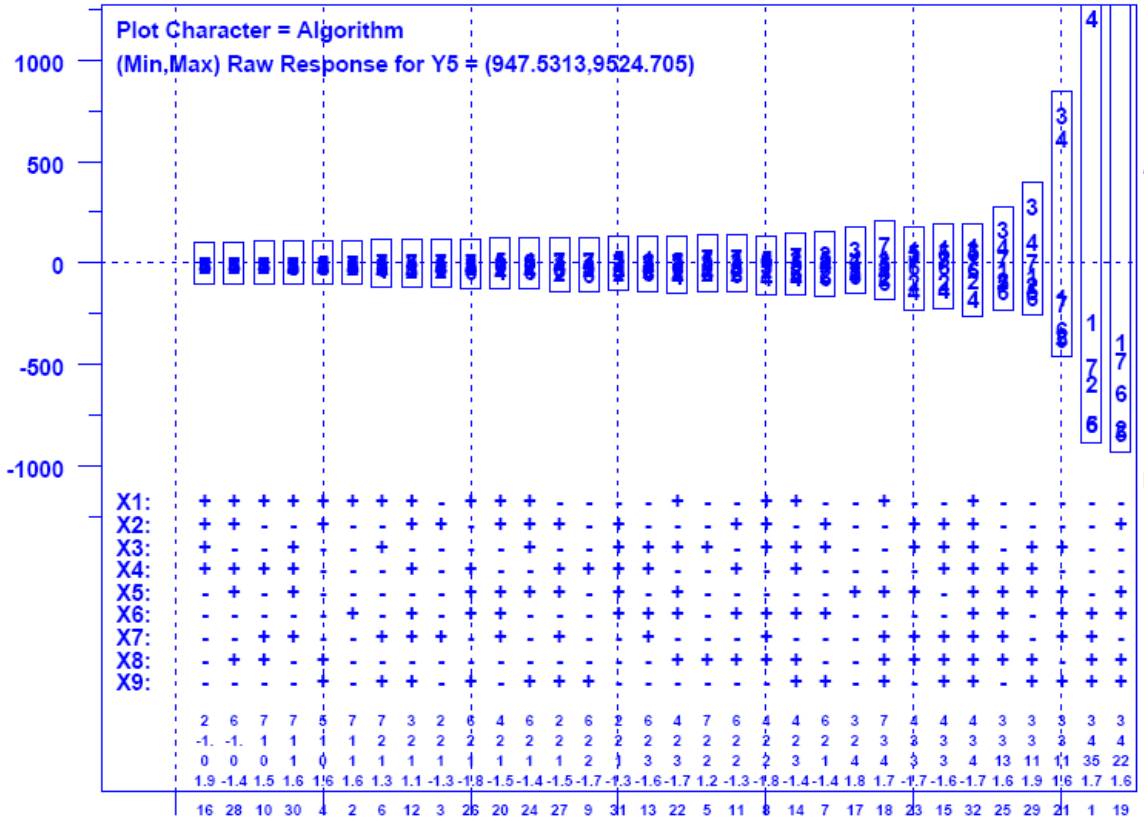


Figure 9-3. Average Number of Connecting Flows under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals.

Fig. 9-5 shows that FAST and FAST-AT completed substantially fewer flows per measurement interval under the three most congested conditions (1, 19 and 21). As shown in Fig. 9-6, the lower flow completion rate for FAST and FAST-AT under severe congestion (conditions 1, 19 and 21) resulted in millions fewer completed flows over the entire simulated hour.

Fig. 9-7 shows that Scalable TCP had a tendency to incur longer smoothed round-trip times, which resulted from larger network packet queues. This echoes results from the previous experiment (Chapter 8), where Scalable TCP round-trip times could be 2-10 ms higher on average than those of other algorithms. Fig. 9-8 shows that, under Scalable TCP, a higher proportion of completed flows were Web objects. Note, however, that the differences in proportion were quite small (most on the order of 10^{-4}). The case with respect to movie transfers is shown in Fig. 9-9. In more than half the simulated conditions, under all algorithms the same proportion of files transferred were movies (highlighted in black in Fig. 9-9). In the remaining conditions, differences were on the order of 10^{-6}. Overall, the differences in proportion of flows completed were very small. We attribute this to the fact that conditions generally exhibited little congestion.

Finally, as shown in Fig. 9-10, CTCP achieved a significant increase in average congestion window. This characteristic also appeared in pervious experiments. The higher network speed available in the current experiment enabled CTCP to achieve a

more substantial advantage in average congestion window than reported for the slower network used in Chapter 8 (see Fig. 8-40).

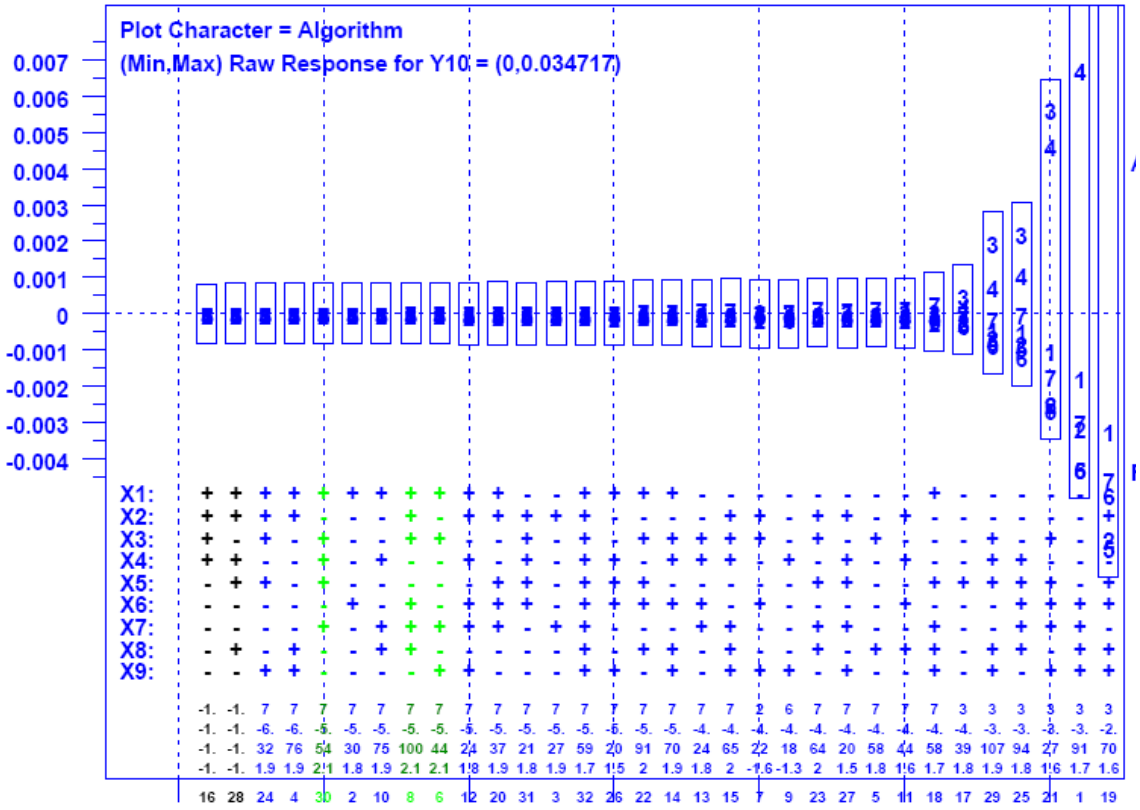


Figure 9-4. Average Retransmission Rate (proportion of packets resent) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Columns highlighted in green indicate significant outliers on the high side, columns highlighted in black indicate no numeric difference measured among the congestion control algorithms and blue columns mean that differences among the congestion control algorithms were not statistically significant.

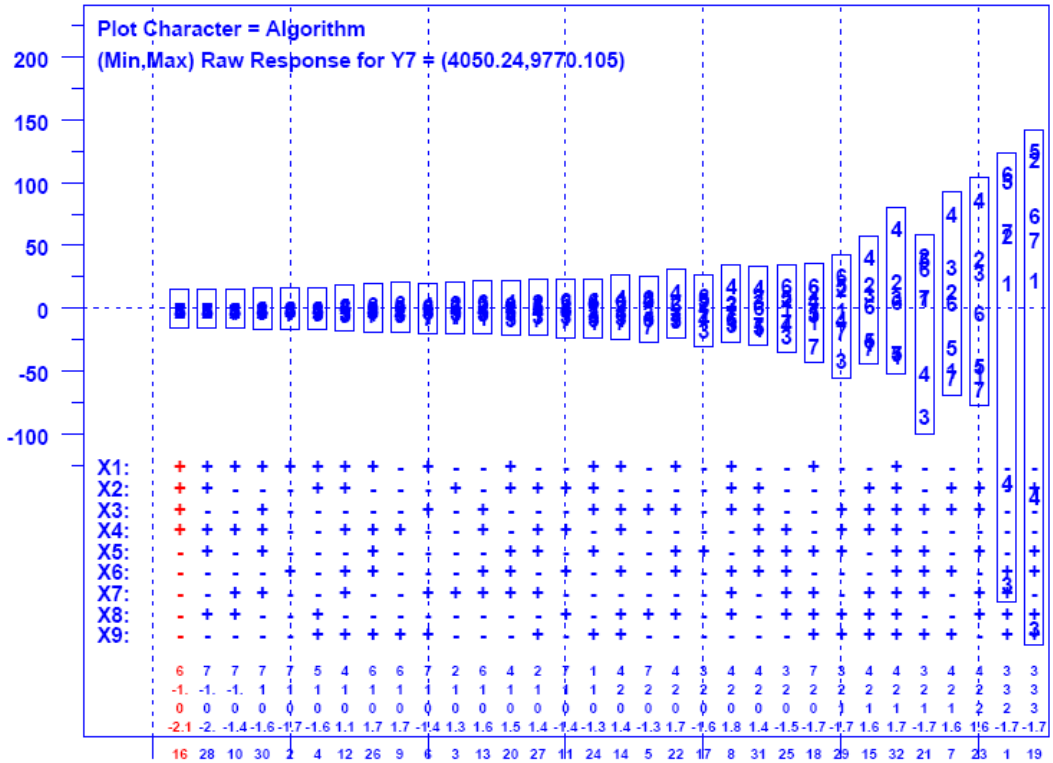


Figure 9-5. Average Flow Completion Rate (flows per 200 ms) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Column highlighted in red denotes a statistically significant outliner on the low side.

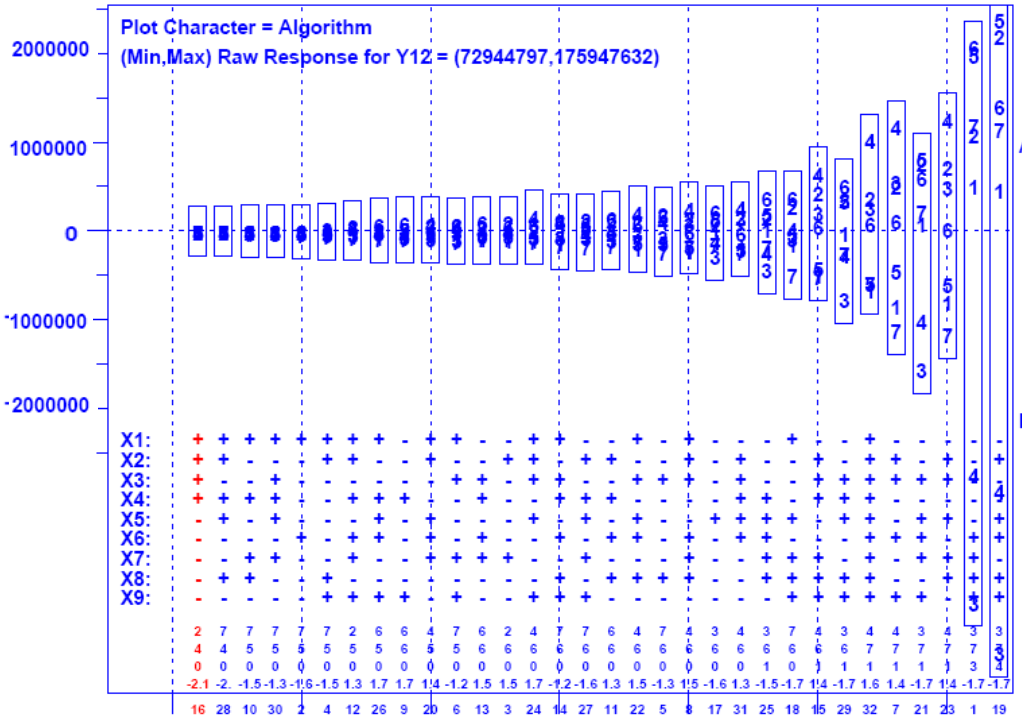


Figure 9-6. Aggregate Flows Completed under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Column highlighted in red denotes a statistically significant outlier on the low side.

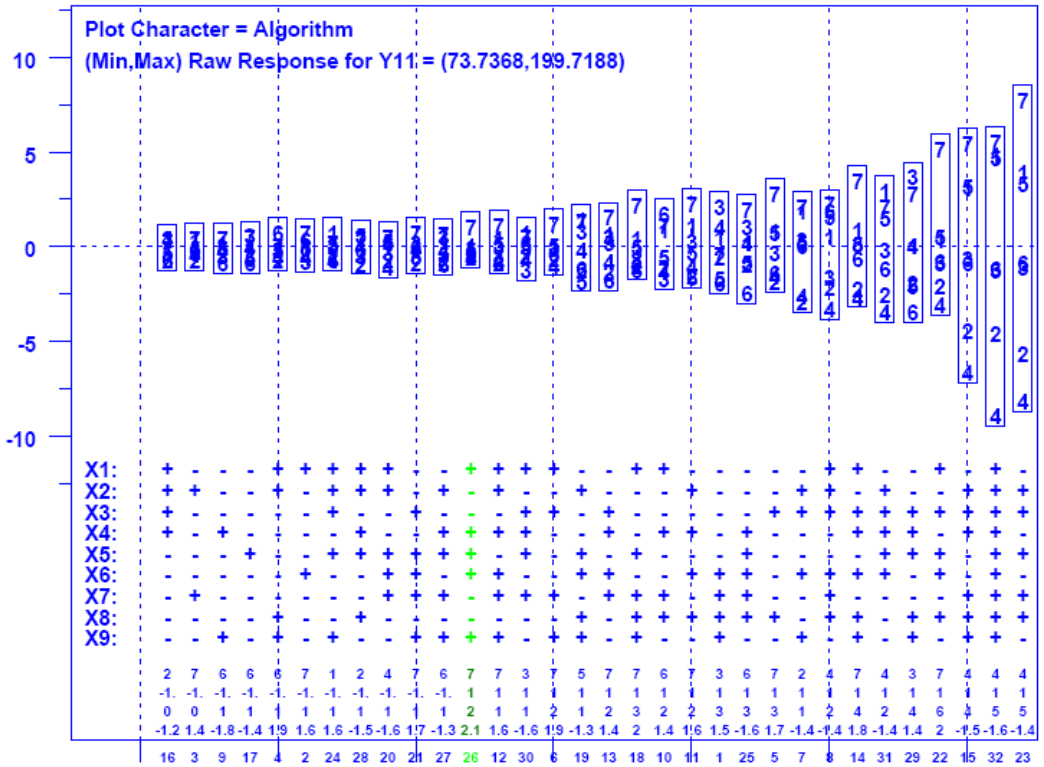


Figure 9-7. Average Smoothed Round-Trip Time (ms) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Column highlighted in green denotes a statistically significant outlier on the high side.

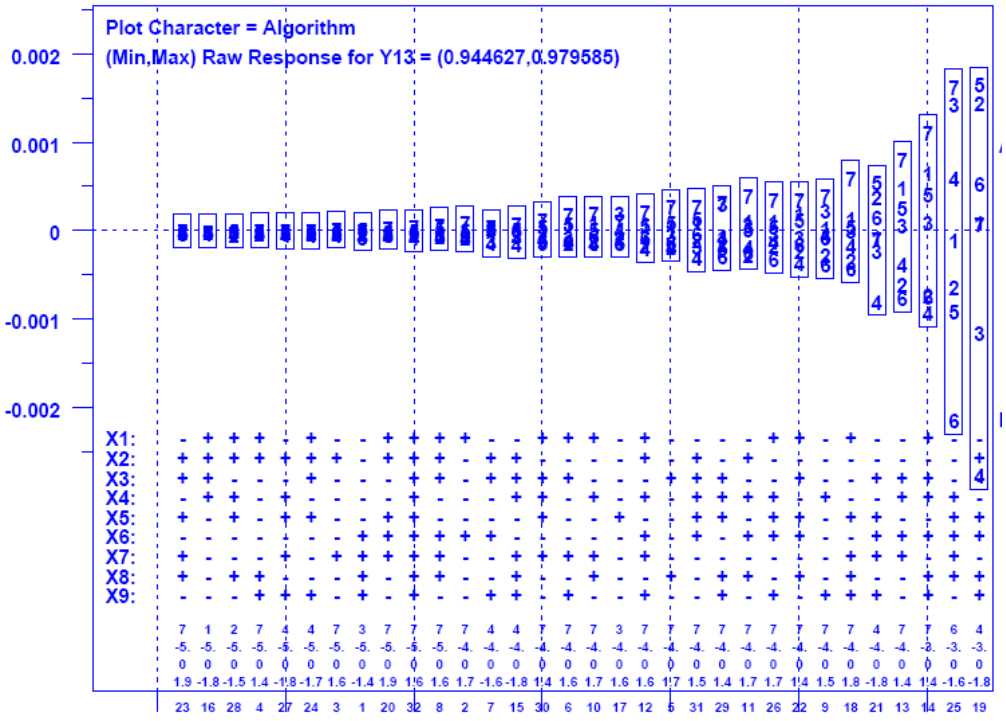


Figure 9-8. Web Objects as Proportion of Flows Completed under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals.

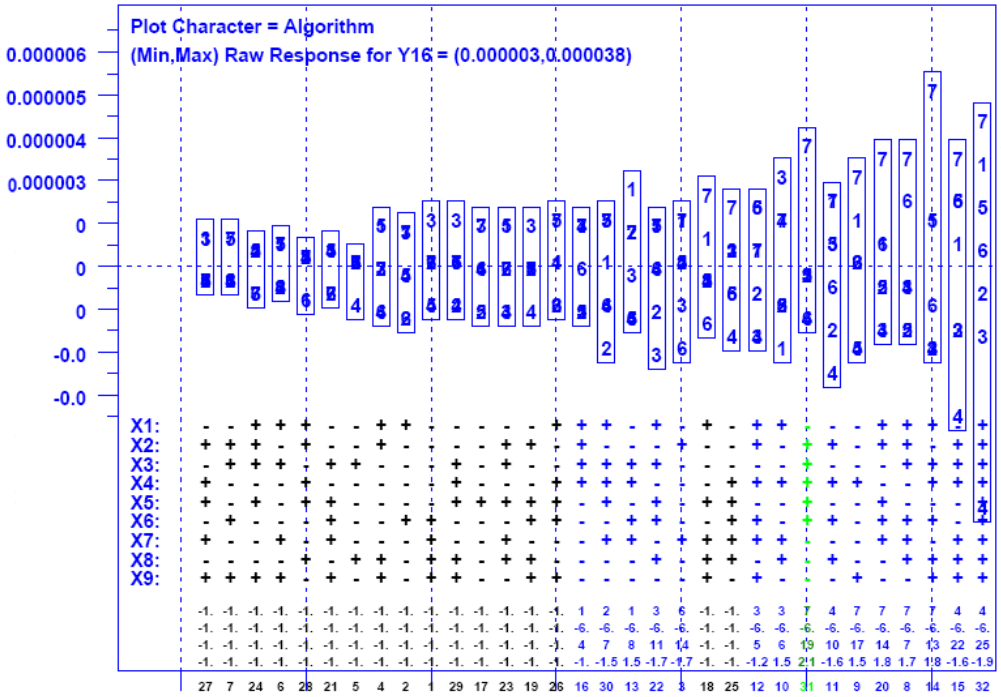


Figure 9-9. Movies as Proportion of Flows Completed under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Column highlighted in green indicates significant outlier on the high side, columns highlighted in black indicate no numeric difference measured among the congestion control algorithms.

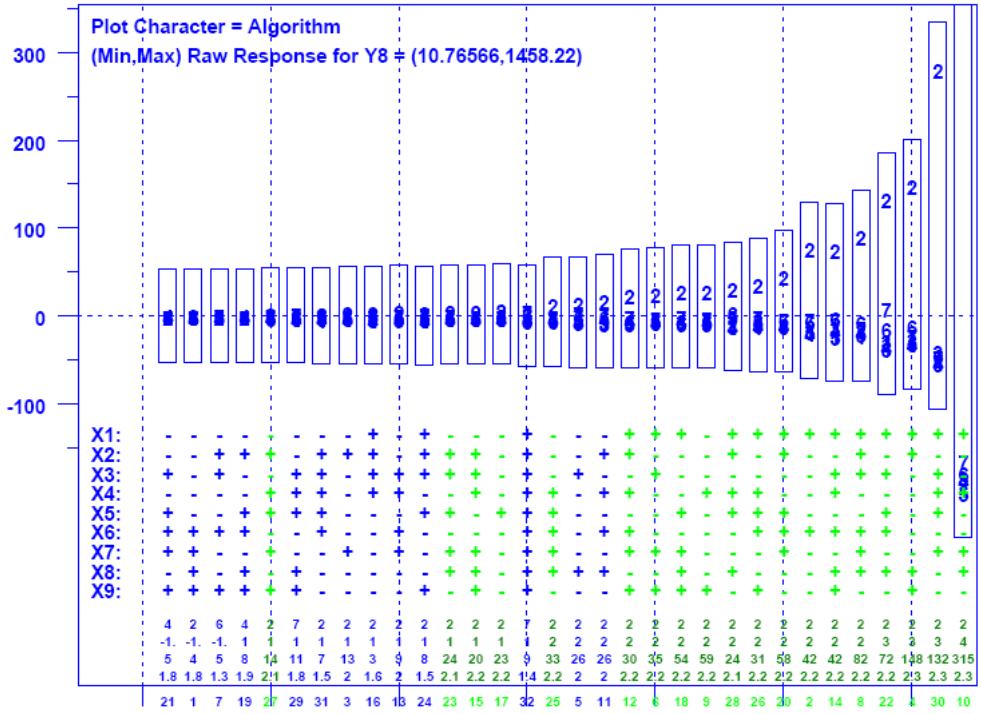


Figure 9-10. Average Flow Congestion Window Size (packets) under High Initial Slow-Start Threshold – y axis gives residuals around the mean value for each condition and x axis gives conditions ordered by increasing range of residuals. Columns highlighted in green indicate significant outliers on the high side.

9.4.2 Absolute User Experience

Table 9-10 summarizes the average goodput – response y2(u) – experienced by users in each of the 24 flow groups (dimensioned by file size, path class and interface speed) under each of the seven alternate congestion control algorithms. Table 9-11 provides a similar summary of the average goodput – response y16(u) – experienced by TCP users in each of the 24 flow groups when competing with flows in each of the seven alternate congestion control algorithms.

Table 9-10. Average Goodput (pps) per Flow Group under Each Alternate Congestion Control Algorithm for a Large, Fast Network with High Initial Slow-Start Threshold – file sizes include movies (M), service packs (SP), documents (D) and Web objects (WO); path classes include very fast (VF), fast (F) and typical (T); interface speeds include fast (F) and normal (N)

			ALTERNATE CONGESTION-CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	60728	60595	60778	60745	60574	60665	60319
		N	7920	7923	7933	7919	7929	7928	7930
	F	F	32495	29893	29981	29094	31177	29745	34822
		N	6793	6236	6865	6603	6798	6609	7091
	T	F	30155	26632	27177	26672	29328	26451	32629
		N	6669	6258	6674	6478	6643	6266	6954
SP	VF	F	30551	30876	30568	30628	30719	30594	30646
		N	7440	7429	7434	7431	7430	7431	7427
	F	F	16381	16273	16426	16675	16327	16250	16920
		N	6174	6029	6302	6338	6111	6012	6222
	T	F	17772	17262	17692	17794	17607	17151	18160
		N	6226	6112	6393	6424	6174	6047	6295
D	VF	F	2377	2375	2368	2372	2360	2360	2377
		N	1942	1939	1941	1946	1944	1949	1937
	F	F	1434	1438	1427	1438	1440	1449	1436
		N	1257	1264	1252	1262	1260	1268	1257
	T	F	1774	1786	1770	1786	1782	1795	1781
		N	1513	1527	1514	1526	1519	1532	1516
WO	VF	F	448	450	451	457	449	451	451
		N	424	424	425	424	426	425	426
	F	F	278	279	276	279	279	280	278
		N	267	268	266	268	268	269	267
	T	F	348	351	347	349	350	352	349
		N	332	335	331	333	334	336	333

Since the tables are somewhat dense with numbers, we present this information in the form of bar graphs (Fig. 9-11 through 9-14) – one figure per file size: movie (M), service pack (SP), document (D) and Web object (WO). (The legend for the bar graphs is

shown in Fig. 8-27.) The top row of graphs in each figure displays the average goodput in packets per second (pps) achieved in a large, fast network with a high initial slow-start threshold, while, for comparison, the bottom row of graphs displays average goodput achieved in a smaller, slower network with high initial slow-start threshold (as reported previously in Sec. 8.4.2.1). When examined vertically, the first two columns of graphs consider flows transiting very fast (VF) paths, the second two columns consider flows transiting fast (F) paths and the final two columns consider flows transiting typical (T) paths. Within a given path class, the first vertical sub-column reports goodput for flows with fast (F) interface speeds (80×10^3 pps), while the second vertical sub-column reports goodput for flows with normal (N) interface speeds (8×10^3 pps). Each column of graphs is labeled with the relevant path class and interface speed (e.g., VF-F).

Table 9-11. Average Goodput (pps) per Flow Group on TCP Flows Competing with Each Alternate Algorithm for a Large, Fast Network with High Initial Slow-Start Threshold – file sizes include movies (M), service packs (SP), documents (D) and Web objects (WO); path classes include very fast (VF), fast (F) and typical (T); interface speeds include fast (F) and normal (N)

			ALTERNATE CONGESTION-CONTROL ALGORITHM						
File	Path	Interface	BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
M	VF	F	60576	60463	60718	60754	60813	60454	60627
		N	7926	7930	7924	7930	7931	7935	7932
	F	F	27158	27312	27828	28014	27842	28280	27318
		N	5962	6005	5975	6017	6109	5961	5847
	T	F	24246	25378	24303	24974	24830	25232	24859
		N	5809	6017	5841	5899	5864	5887	5770
SP	VF	F	30852	30692	30657	30714	30819	30767	30944
		N	7439	7440	7443	7448	7438	7446	7440
	F	F	15578	16188	15827	16046	15741	15986	15758
		N	5787	5922	5779	5834	5828	5893	5728
	T	F	16620	16966	16812	16936	16801	16973	16717
		N	5795	5975	5847	5895	5852	5955	5743
D	VF	F	2383	2383	2386	2367	2391	2380	2378
		N	1950	1947	1943	1947	1946	1947	1960
	F	F	1433	1439	1421	1430	1440	1443	1433
		N	1254	1264	1242	1256	1260	1268	1253
	T	F	1766	1784	1754	1771	1779	1794	1773
		N	1507	1527	1498	1512	1518	1532	1509
WO	VF	F	456	451	451	452	451	455	453
		N	426	428	427	427	427	427	427
	F	F	278	280	276	279	279	281	279
		N	267	269	265	268	268	270	268
	T	F	348	351	346	349	350	353	349
		N	332	335	330	333	334	336	333

Figs. 9-11 through 9-14 reveal two main points. First, in a larger, faster network, flows for large files (movies and service packs) over fast interfaces (80 x 10^3 pps) achieve significantly higher average goodputs than similar flows in a smaller, slower network. Second, average goodputs achieved by competing TCP flows in a larger, faster network appear closer to average goodputs achieved by competing TCP flows in a smaller, slower network. These two points appear due to generally reduced congestion in the larger, faster network. Recall that under a high initial slow-start threshold any goodput differences result from loss/recovery processing because all flows use the same algorithm to accelerate to the initial maximum transfer rate. Lower overall congestion leads to fewer losses per flow, which means that all flows achieve higher goodputs and that alternate congestion control algorithms have fewer opportunities to invoke their loss/recovery procedures.

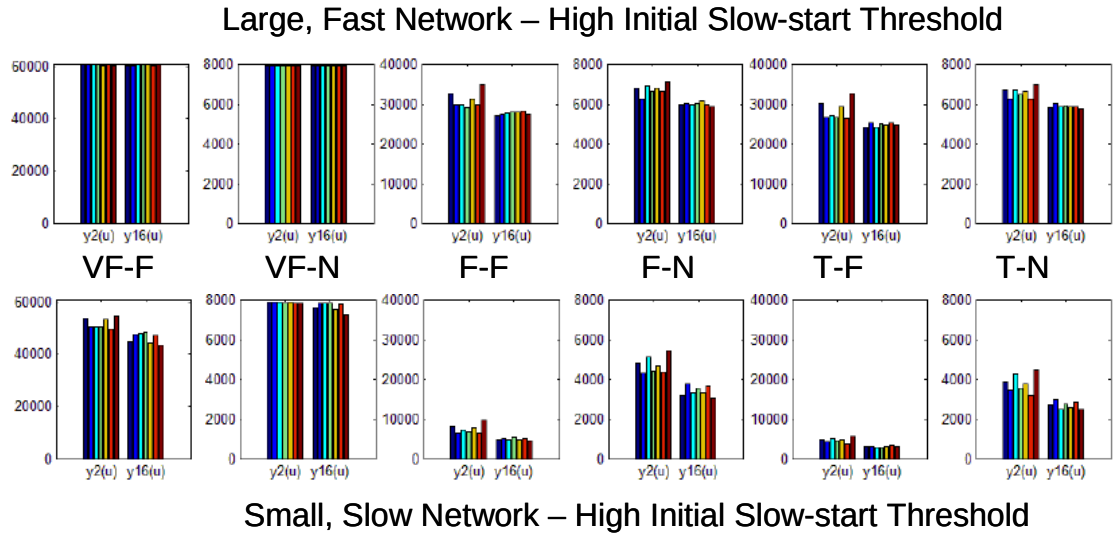


Figure 9-11. Average Goodputs (pps) on Movies under Combinations of Path Class and Interface Speed (Large Fast Network vs. Small Slow Network)

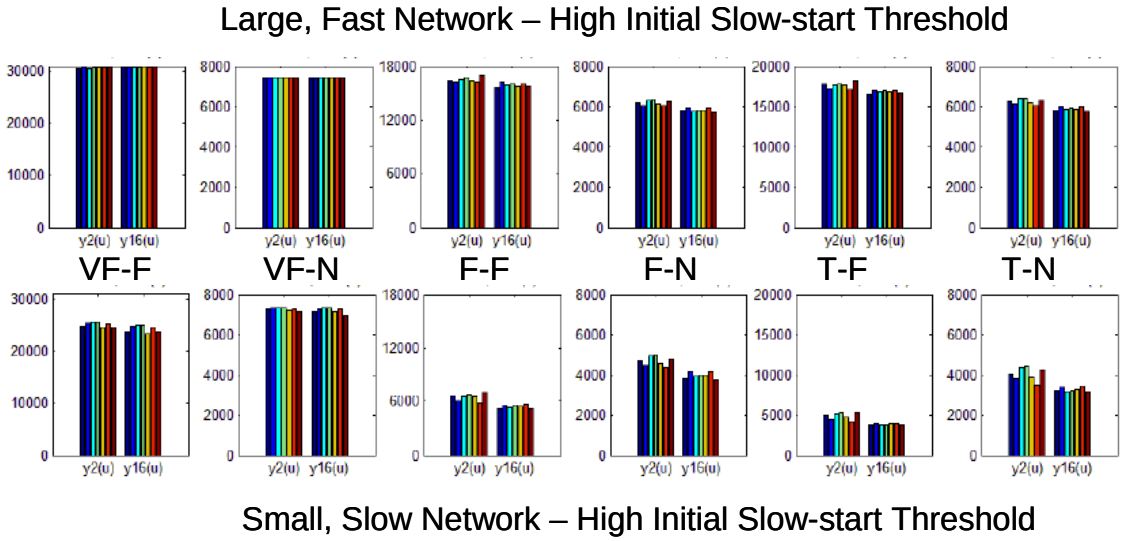


Figure 9-12. Average Goodputs (pps) on Service Packs under Combinations of Path Class and Interface Speed (Large Fast Network vs. Small Slow Network)

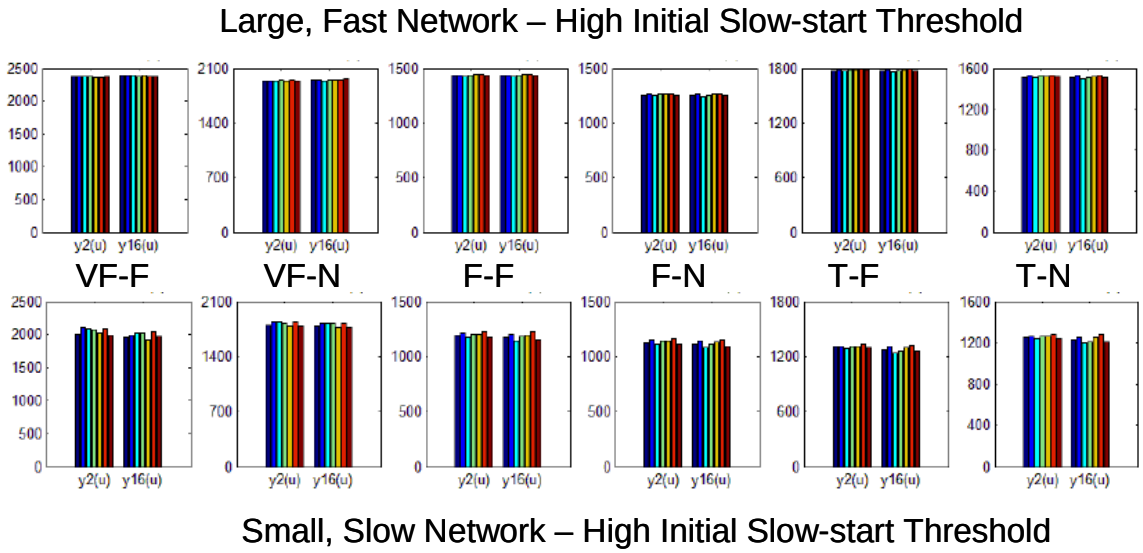


Figure 9-13. Average Goodputs (pps) on Documents under Combinations of Path Class and Interface Speed (Large Fast Network vs. Small Slow Network)

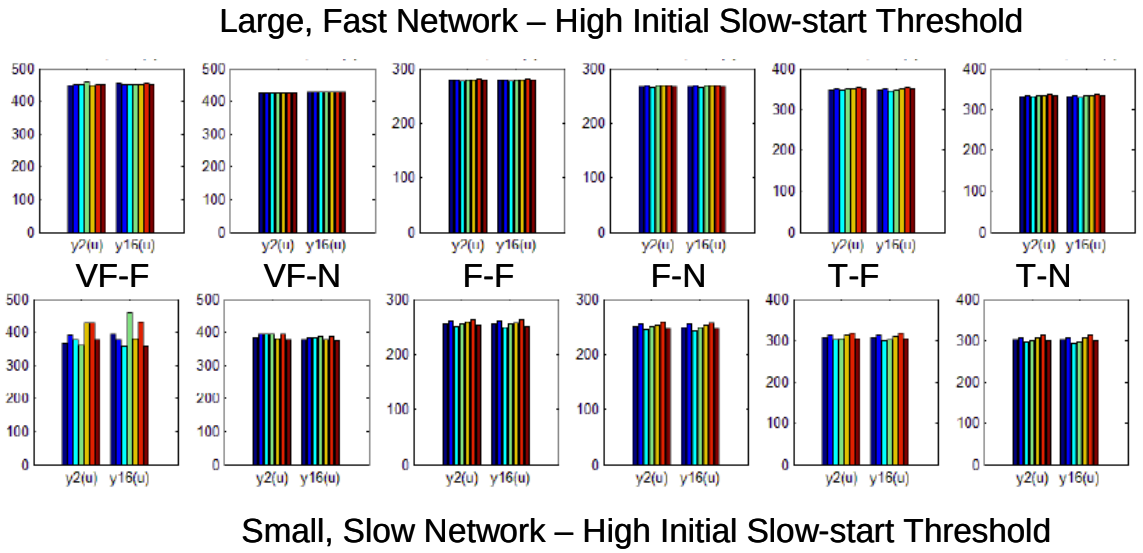


Figure 9-14. Average Goodputs (pps) on Web Objects under Combinations of Path Class and Interface Speed (Large Fast Network vs. Small Slow Network)

Given the similarity in goodput for flows with the same file size, regardless of whether using standard TCP or alternate congestion control procedures, we decided to see if factors other than file size influenced goodput on flows. To investigate, we conducted a principal components analysis (PCA) of the average goodput data across all flow groups. Fig. 9-15 plots the resulting information, which reveals four main groups: (1) a group where network speed is higher (x1 = 1), (2) a group where network speed is lower (x1 = -1), (3) a group where propagation delay is higher (x2 = 1) and (4) a group where propagation delay is lower (x2 = -1). Within each group, two subgroups appear: (1) a subgroup where file sizes are larger (x5 = 1) and (2) a subgroup where file sizes are

smaller (x5 = -1). Thus, PCA reveals that differences in flow goodput are influenced mainly by network speed, propagation delay and file size – not by congestion control algorithm.

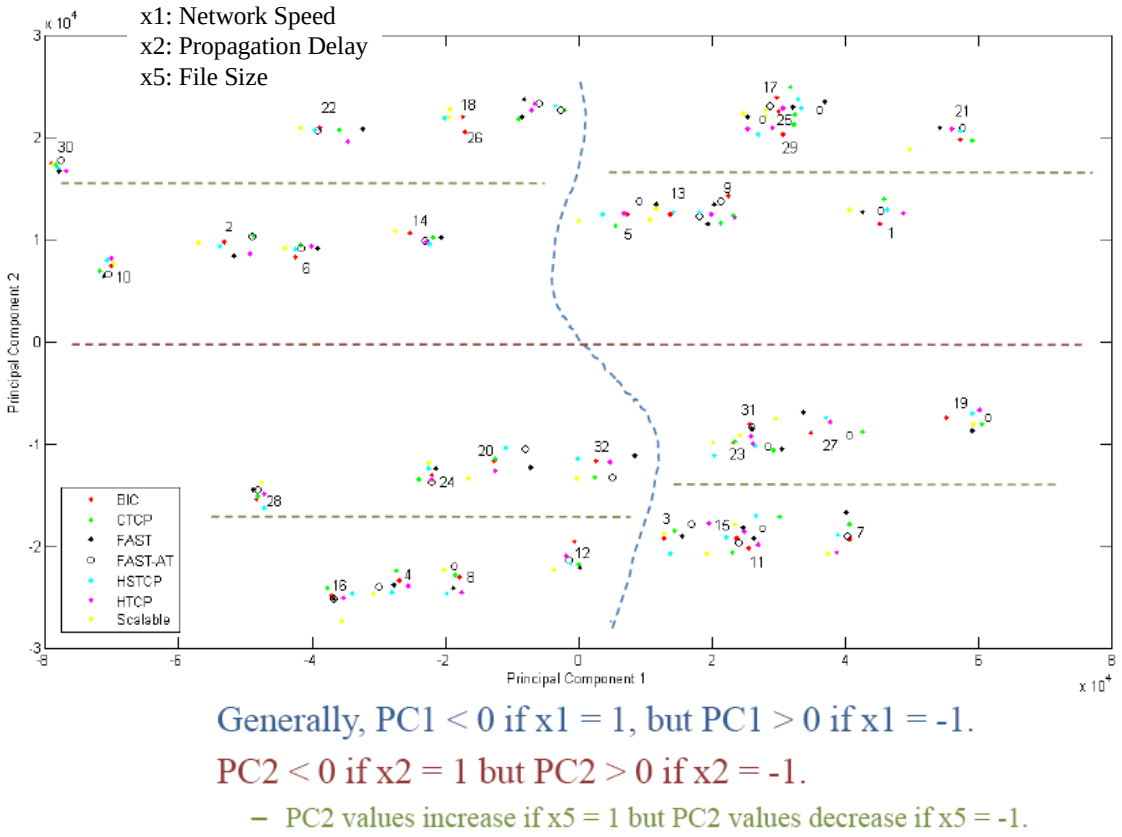


Figure 9-15. Principal Component 1 (x axis) vs. Principal Component 2 (y axis) from Average Goodput Data in a Large, Fast Network with High Initial Slow-Start Threshold – the blue dashed line separates (but not crisply) PC1 values for a network with higher (left) and lower (right) propagation delays, the red dashed line separates PC2 values with higher (top) and lower (bottom) network speeds, and the green dashed lines subdivide the two PC2 areas (one above and one below the red line) by file size: larger (above the green lines) and smaller (below the green lines)

In experiments reported in Chapter 8, we found that under conditions with higher congestion flows using several alternate congestion control algorithms (e.g., BIC, HSTCP and Scalable TCP) had significantly higher goodput than competing TCP flows. Given the generally lower overall congestion when simulating a larger, faster network, can such differences still be discerned? To investigate, we used scatter plots and per-condition bar graphs, as introduced in Sec. 8.3.2. Fig. 9-16 gives seven scatter plots, each showing TCP goodput (y axis) vs. goodput of an alternate (as labeled) congestion control algorithm for movies transferred on very fast paths with a fast interface speed. The scatter plots show no significant difference in goodput for TCP flows vs. flows using alternate congestion control algorithms. Fig. 9-17, which gives differences in goodput between TCP flows and alternate congestion control algorithms under each of 32 simulated conditions, also shows no significant differences. The lack of differences can be attributed to the fact that very

fast paths exhibit little congestion, which means that few losses occur and so one should expect little difference in flow goodputs regardless of congestion control algorithm.

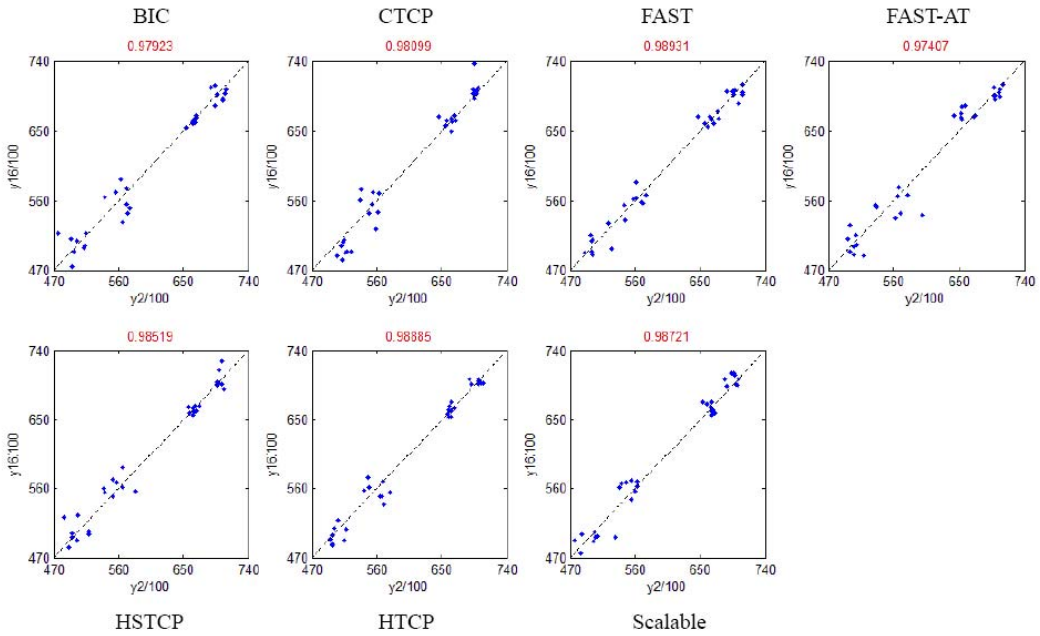


Figure 9-16. Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Movies on Very Fast Paths with Fast Interfaces in a Large, Fast Network with High Initial Slow-Start Threshold

When we examine path classes with higher likelihood of congestion, BIC, HSTCP and Scalable TCP flows have a goodput advantage over standard TCP flows on very large files – i.e., movies. For example, Fig. 9-18 shows related scatter plots that reveal the tendency of alternate congestion control algorithms to have better goodputs than TCP flows. As in Chapter 8, the effect is most pronounced for BIC, HSTCP and Scalable TCP. This occurs because large files have a tendency to accumulate more losses on more congested paths, which allows for the loss/recovery procedures of the alternate congestion control algorithms to be activated more often. As previously shown, BIC, HSTCP and Scalable TCP tend to resist lowering transmission rate on sporadic losses, so flows using those regimes achieve significantly higher goodputs vs. TCP flows, which reduce their transmission rate in half on each loss. Fig. 9-19 suggests that the advantage of the alternate congestion control algorithms over TCP tends to increase with increasing congestion, at least until congestion becomes so pervasive that all flows suffer significant reductions in goodput.

The advantage of alternate congestion control algorithms decreases with decreasing file size because there are fewer packets on each flow to incur losses. This effect can be seen in the scatter plots in Fig. 9-20 for service packs sent over fast paths with fast interfaces and in the accompanying bar graphs plotted in Fig. 9-21. Notice that Fig. 9-21 confirms that alternate congestion control algorithms can achieve better goodputs than TCP flows as congestion increases, as seen in conditions 26, 18, 27, 9, 15 and 17.

Table 9-12 gives a summary of goodput differences as percentages for each of the 24 flow groups measured. Differences under a smaller, slower network with a high initial slow-start threshold are reported (taken from Table 8-30) in three columns: (1) **AMONG ALTs** gives the range of percentage difference between flows using the alternate congestion control algorithms with the highest and lowest average goodput; (2) **AMONG TCPs** gives the range of percentage difference between TCP flows with the highest and lowest average goodput when competing with alternate congestion control algorithms; (3) **ALTs > TCPs** gives the percentage increase in average goodput for flows using alternate congestion control algorithms over competing TCP flows (note that when given in red, TCP flows achieved higher average goodput). A similar set of three columns reports goodput differences under a large, fast network with high initial slow-start threshold.

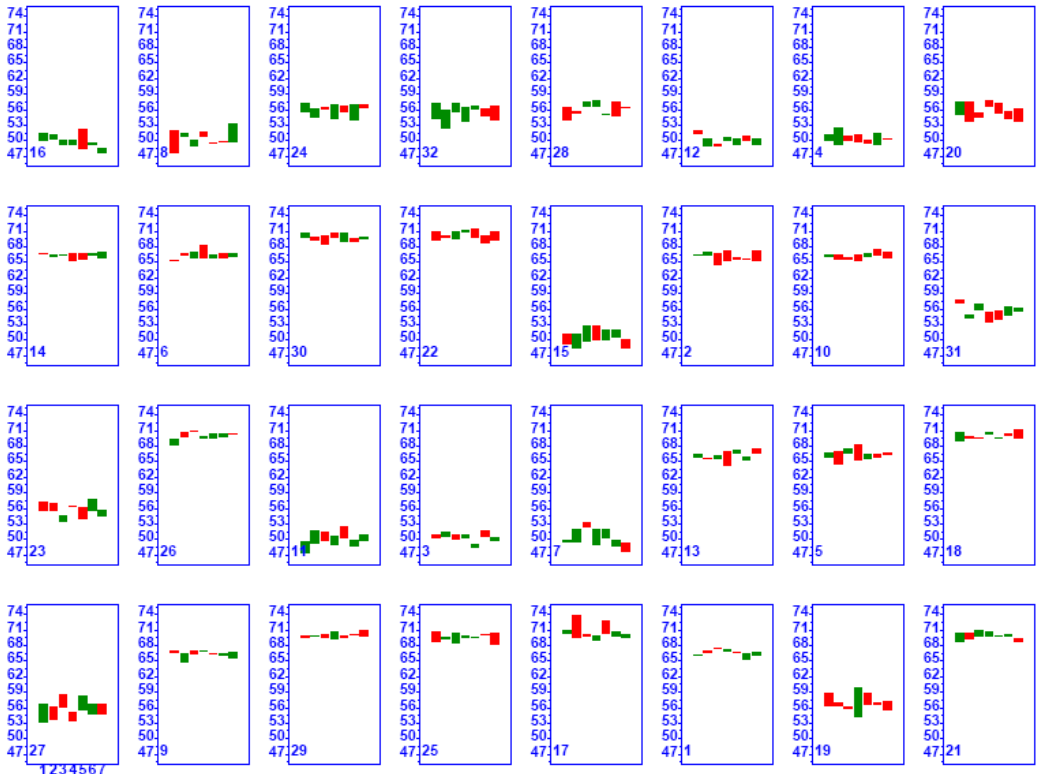


Figure 9-17. Bar Graphs (one for each simulated condition) plotting Goodput Differences (pps/1000) on TCP Flows vs. Non-TCP Flows for Movies Transferred on Very Fast Paths with Fast Interfaces in a Large, Fast Network with a High Initial Slow-Start Threshold (Each graph contains seven bars, one per congestion control algorithm, ordered left to right by algorithm identifier. Each bar plots the magnitude of the difference in average goodput for TCP flows – y16(u) – versus competing alternate flows – y2(u). If the bar is red, y16(u) is greater; if the bar is green, y2(u) is greater. The 32 bar graphs are sorted from least to most congestion by condition, as indicated in the lower left-hand corner of each plot.)

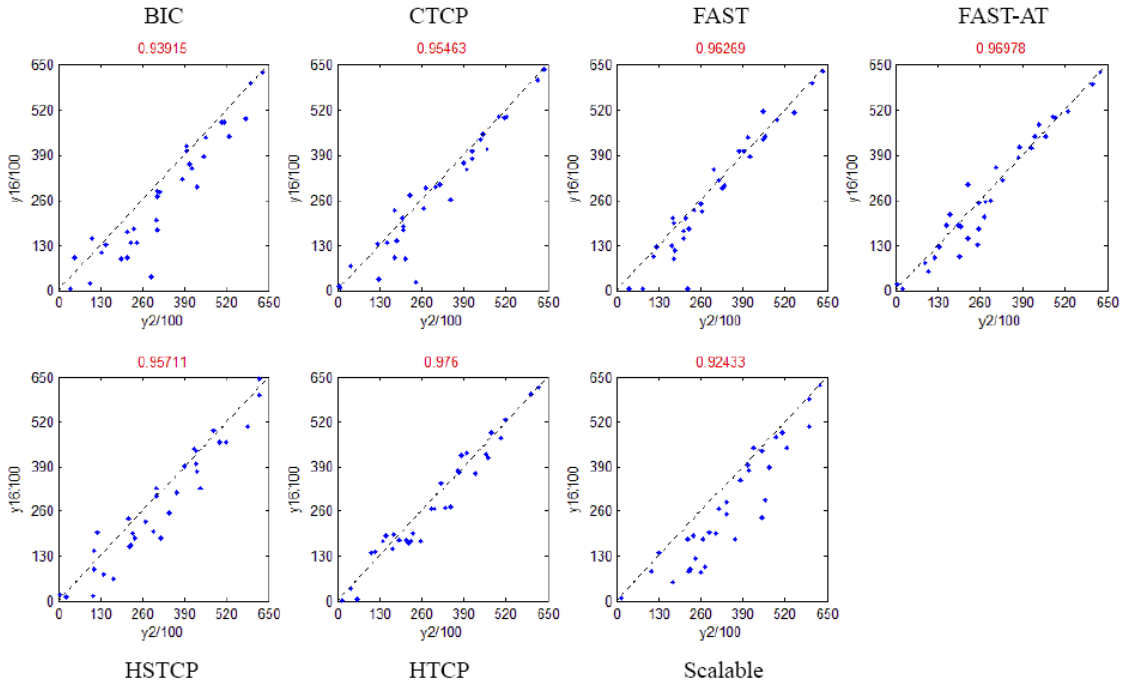


Figure 9-18. Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Movies on Fast Paths with Fast Interfaces in a Large, Fast Network with High Initial Slow-Start Threshold

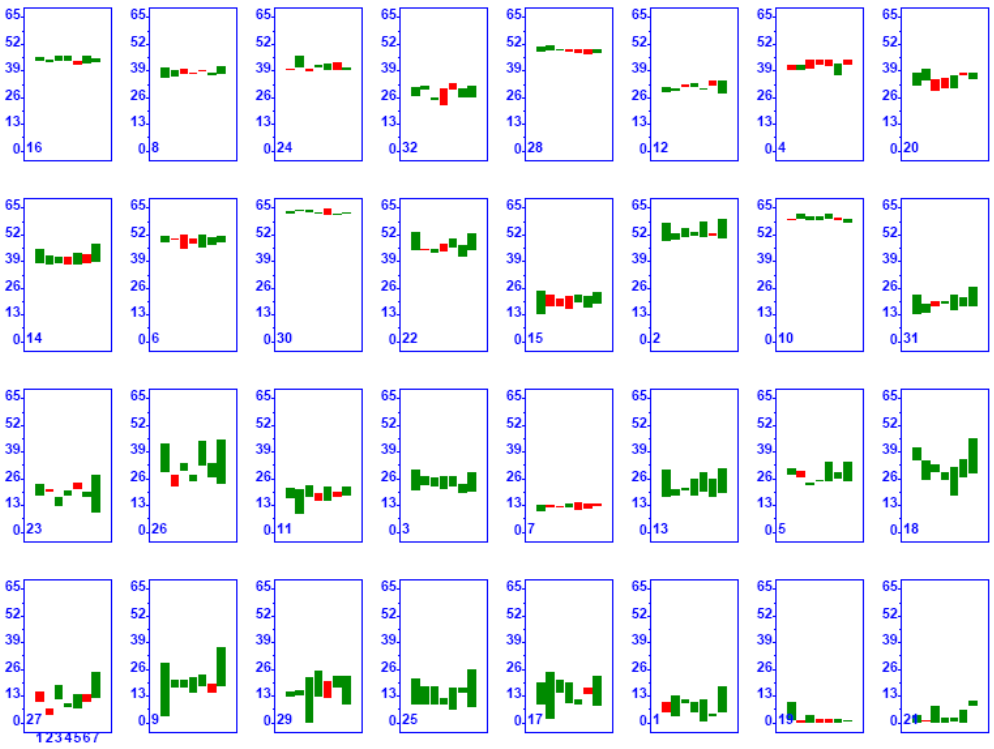


Figure 9-19. Bar Graphs (one for each simulated condition) plotting Goodput Differences (pps/1000) on TCP Flows vs. Non-TCP Flows for Movies Transferred on Fast Paths with Fast Interfaces in a Large, Fast Network with High Initial Slow-Start Threshold (green bars indicate flows using alternate algorithm have higher goodput and red bars indicate competing flows using TCP have higher goodput)

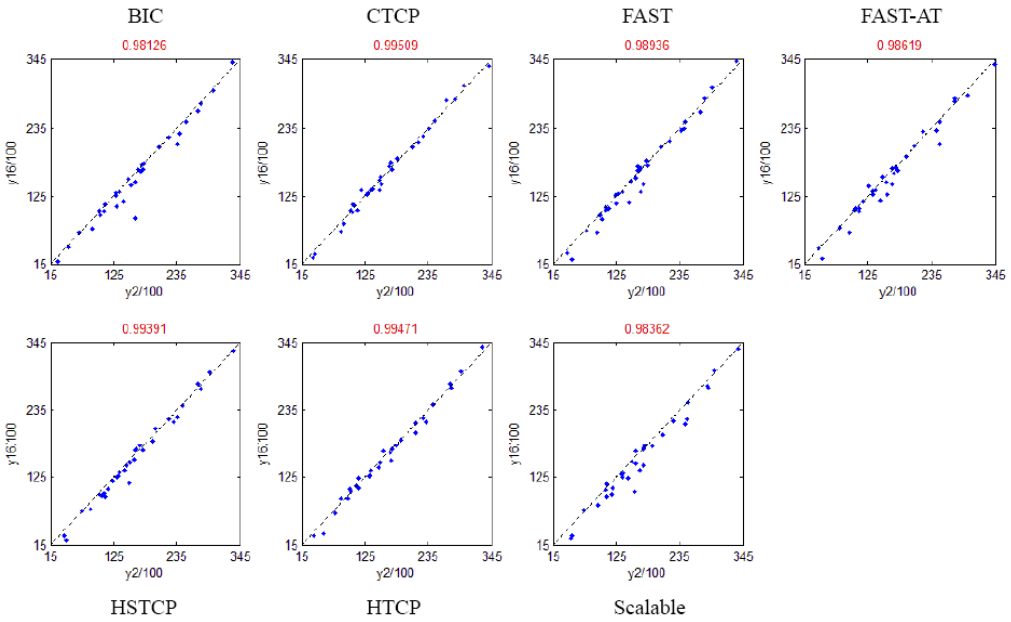


Figure 9-20. Goodput on TCP Flows (y axes give y16u/100 pps) vs. Non-TCP Flows (x axes give y2u/100 pps) for Service Packs on Fast Paths with Fast Interfaces in a Large, Fast Network with High Initial Slow-Start Threshold

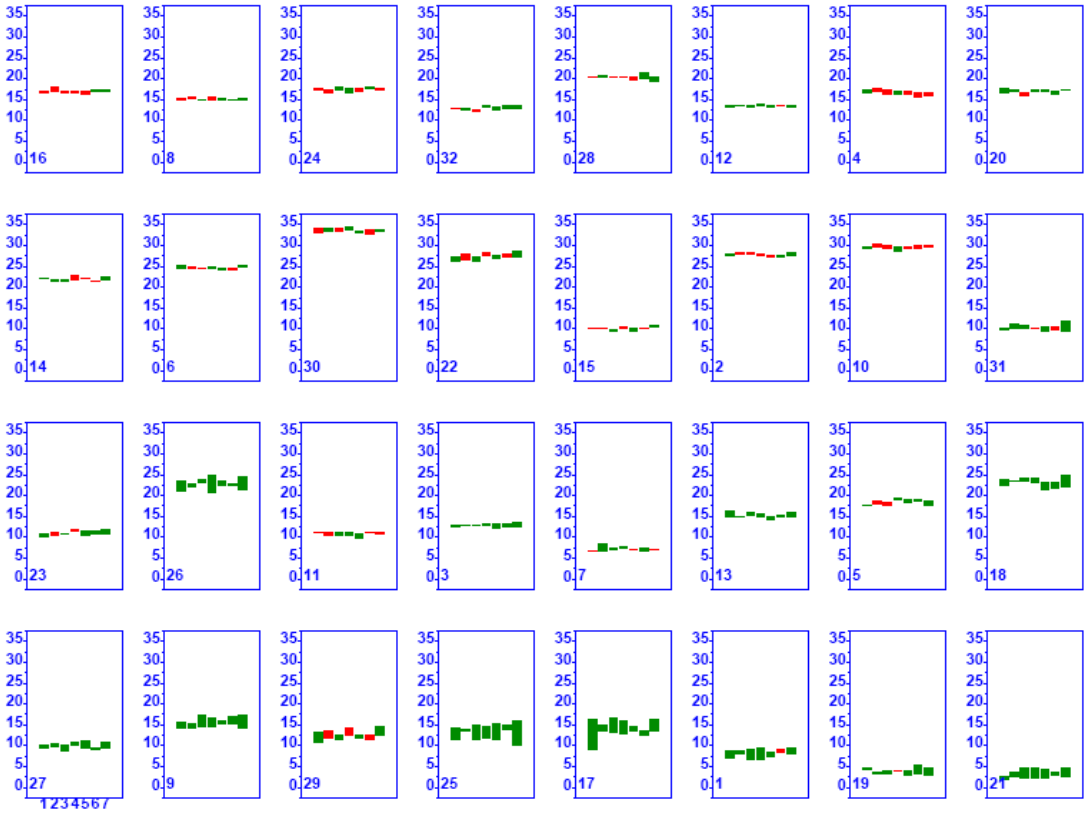


Figure 9-21. Bar Graphs (one for each simulated condition) plotting Goodput Differences (pps/1000) on TCP Flows vs. Non-TCP Flows for Service Packs Transferred on Fast Paths with Fast Interfaces in a Large, Fast Network with High Initial Slow-Start Threshold (green bars indicate flows using alternate algorithm have higher goodput and red bars indicate competing flows using TCP have higher goodput)

Examination of Table 9-12 reveals that goodput differences among alternate congestion control algorithms and among competing TCP flows narrowed as network size and speed increased. In addition, goodput improvements provided by alternate congestion control algorithms over TCP flows disappeared for most flow groups. Alternate congestion control algorithms provided improved goodputs (over TCP) only on flows where files were large (movies and service packs) and where congestion was significant (fast and typical path classes.)

Table 9-12. Range of Goodput Differences (%) for Flow Groups under High Initial Slow-Start Threshold for Small, Slow Network and for Large, Fast Network (Differences are shown: among Alternate Congestion Control Algorithms, among TCP Flows Competing with Alternate Algorithms and between Alternate Algorithms and TCP Flows)

			RANGE OF GOODPUT DIFFERENCES (%)					
			SMALL, SLOW NETWORK HIGH INITIAL SSTHRESH			LARGE, FAST NETWORK HIGH INITIAL SSTHRESH		
File	Path	Interface	AMONG ALTs	AMONG TCPs	ALTs > TCPs	AMONG ALTs	AMONG TCPs	ALTs > TCPs
M	VF	F	10	11	11	1	1	<<-1
		N	<1	8	3	0	0	<<-1
	F	F	35	16	35	20	4	12
		N	21	20	21	14	4	12
	T	F	30	11	30	23	5	15
		N	30	17	30	11	4	12
SP	VF	F	4	6	3	1	1	<-1
		N	<3	5	1	0	0	<-1
	F	F	12	8	15	4	4	4
		N	15	10	15	5	3	6
	T	F	20	6	20	6	2	5
		N	20	7	20	6	4	6
D	VF	F	5	6	<1	1	1	<-1
		N	<3	4	<2	1	1	<-1
	F	F	4	7	<2	2	2	<1
		N	5	7	<2	1	2	<1
	T	F	3	5	2	1	2	<1
		N	<4	7	2	1	2	<1
WO	VF	F	16	5	-1	2	1	<-1
		N	<5	5	<2	1	0	<-1
	F	F	5	4	<1	1	2	<-1
		N	4	4	<1	1	2	<-1
	T	F	5	6	<1	2	2	<<1
		N	5	5	<1	2	2	<<1

9.4.3 Relative User Experience

In this section, we set aside absolute differences in average goodput and consider instead relative differences. As discussed in Sec. 8.4.3, for each simulated condition, we ranked from high (7) to low (1) the average goodput – y2(u) – provided by the seven alternate congestion control algorithms and we also computed the average goodput across all seven algorithms. We took similar steps with respect to average goodput – y16(u) – among TCP flows competing with each of the alternate algorithms. Using this information, we generated seven pairs of rank matrices. One member of each pair relates to y2(u) and the other member to y16(u). (See Fig. 8-32 for a sample rank matrix.) Each matrix contains (32 conditions x 24 flow groups =) 768 cells, where each cell holds the rank (of average goodput among the seven competing algorithms) for the congestion control algorithm associated with the matrix. If the rank in a cell is rendered in green, then the goodput associated with the rank was above the average goodput for all algorithms. If red, then the goodput was below the relevant average. When a highest ranked (7) cell was farther from the average goodput than the lowest ranked (1) cell, then the cell is highlighted in green. In the reverse case, the lowest ranked cell is highlighted in red.

The columns in each matrix are divided into four vertical sections that each relate to a specific file size (movie, service pack, document and Web object). Each section contains three pairs of flow groups (labeled on the x axis) ordered by path class (very fast, fast and typical). Within each flow-group pair the ordering is by interface speed (fast and normal). The matrix rows are ordered by condition (labeled on the y axis) from least (top) to most (bottom) congested. We reproduce the matrices (Figs. 9-22 through 9-35) to show any patterns that occur. We computed the average rank for each congestion control algorithm for each file size. Similarly, we computed the average rank for TCP flows competing with each congestion control algorithm for each file size. We also determined the standard deviation in rank for each alternate congestion control algorithm, across all files sizes and considering both y2(u) and y16(u). We report these averages and standard deviations in a summary table (Table 9-13). We use the information from the summary table to generate a scatter plot (Fig. 9-36) of average rank (x axis) vs. standard deviation in rank (y axis), which reveals differences in relative user experience among the seven alternate congestion control algorithms.

Table 9-13 shows standard deviation in rank to fall and narrow significantly (0.23 to 0.73) compared with the smaller, slower network (Table 8-31), so ranks of all alternate congestion control algorithms became closer in the larger, faster network. This is congruent with other analyses of the average goodput data. The relative rank of Scalable TCP improved due to higher goodputs for movies, while differences narrowed for other file sizes. The relative rank of FAST-AT improved because the algorithm ranked very well among all file sizes except movies. The relative rank of HTCP and CTCP fell because fewer losses gave fewer opportunities to activate the TCP-friendly[2] loss/recovery procedures of the two algorithms.

[2] TCP friendliness implies that an alternate algorithm behaves similarly to TCP, e.g., reduces transmission rate in half (or nearly so) on a loss and then does not increase transmission rate very quickly. HTCP reduces transmission rate up to 50 % on a packet loss and then increases transmission rate only linearly for one second after a loss. CTCP reduces transmission rate 50 % on a packet loss and can increase transmission rate quickly, but only when the congestion window is above 41 packets and delay is not increasing on the path between a source and receiver.

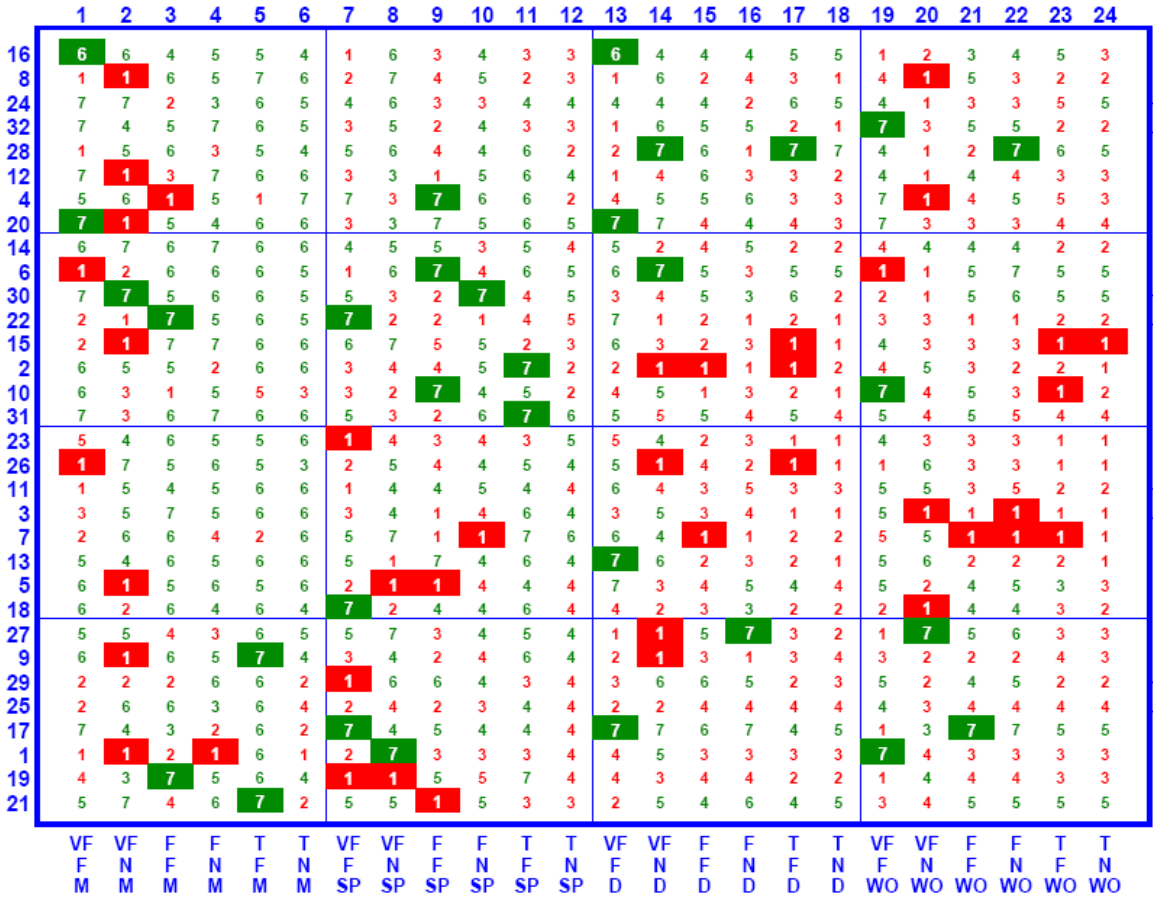


Figure 9-22. Goodput Rank Matrix – y2(u) – BIC (Large, Fast Network, High Initial Slow-Start) Rank (7 high) in each cell denotes ordering of y2(u) for each condition (y axis) and flow group (x axis) – conditions are sorted from least (16) to most (21) congested and flow groups are ordered by file size – movies (M), service packs (SP), documents (D) and Web objects (WO) – and by path class – very fast (VF), fast (F), and typical (T) – within each file size and by interface speed – fast (F) or normal (N) – within each path class.

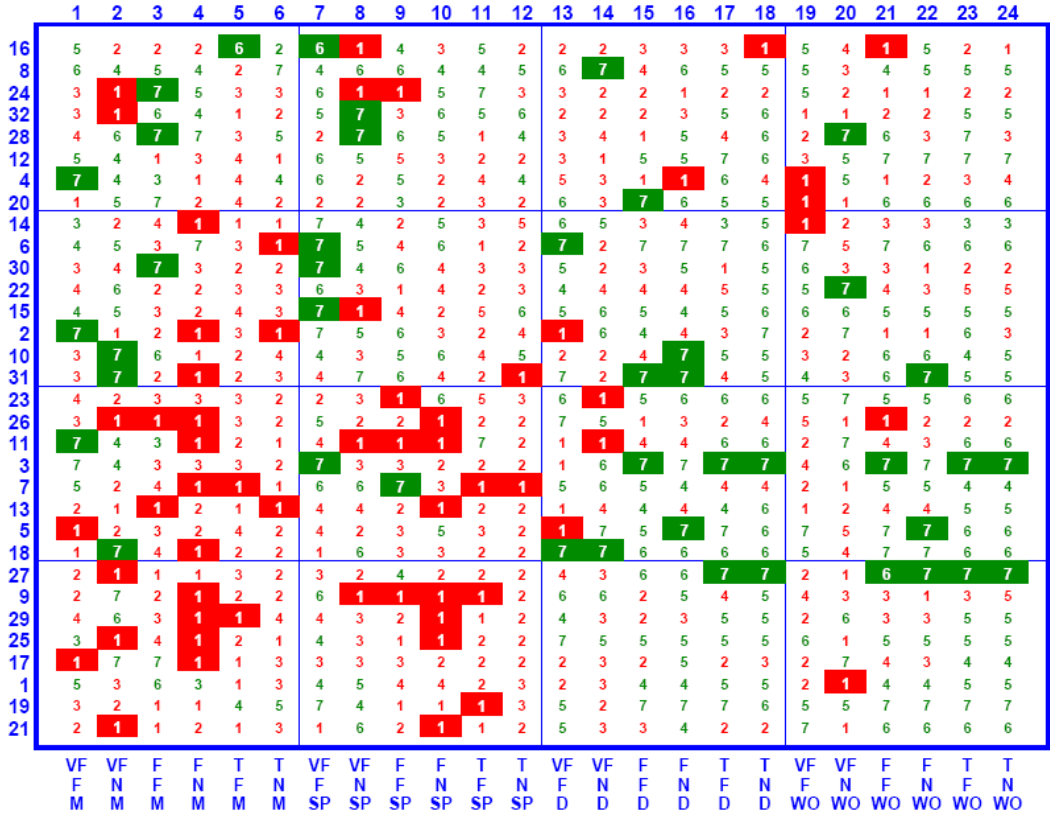


Figure 9-23. Goodput Rank Matrix – y2(u) – CTCP (Large, Fast Network, High Initial Slow-Start)

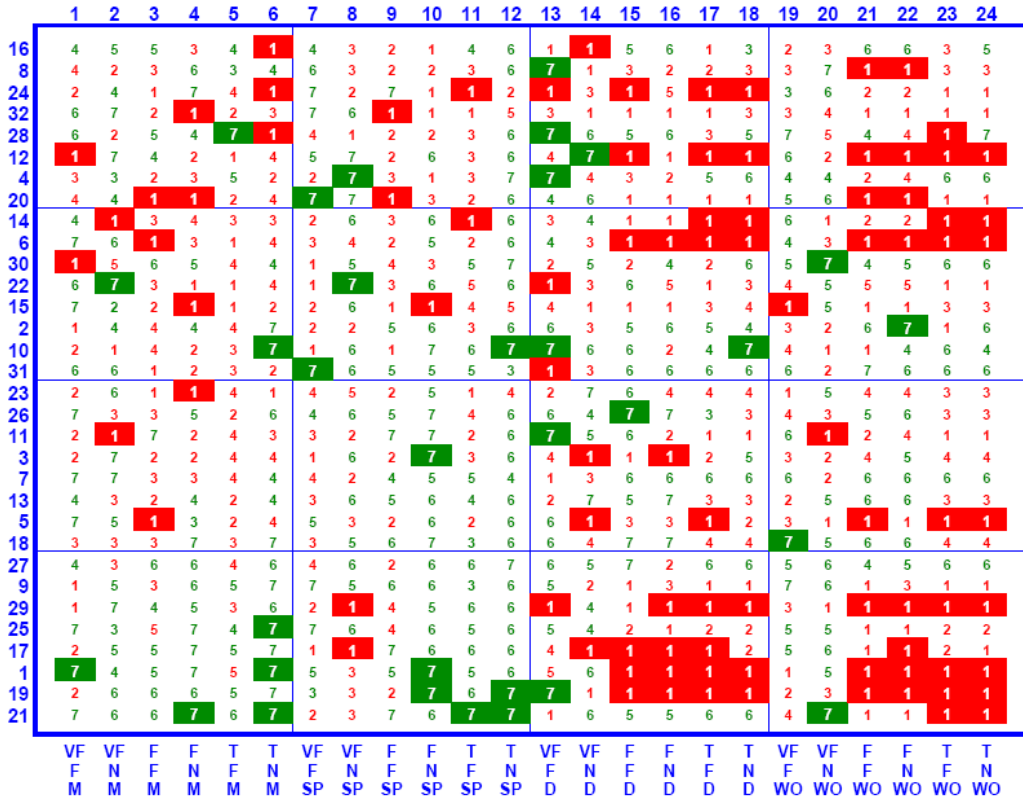


Figure 9-24. Goodput Rank Matrix – y2(u) – FAST (Large, Fast Network, High Initial Slow-Start)

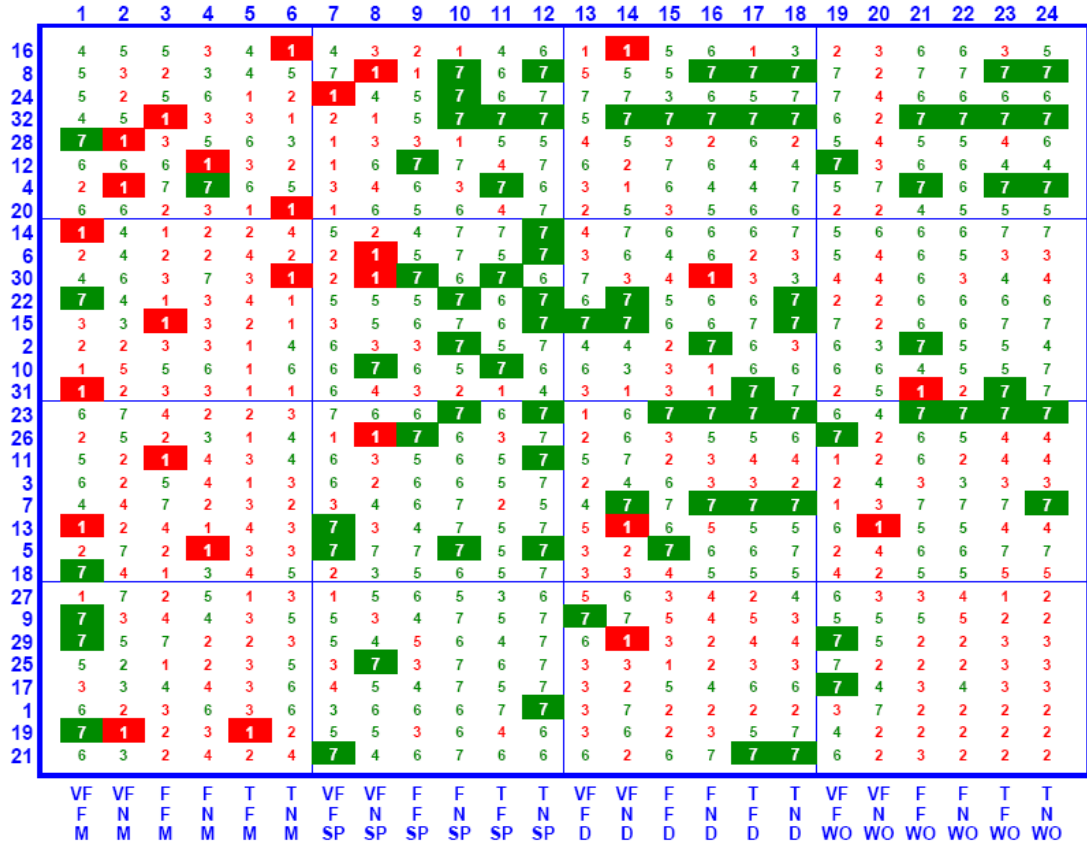


Figure 9-25. Goodput Rank Matrix – y2(u) – FAST-AT (Large, Fast Network, High Initial Slow-Start)

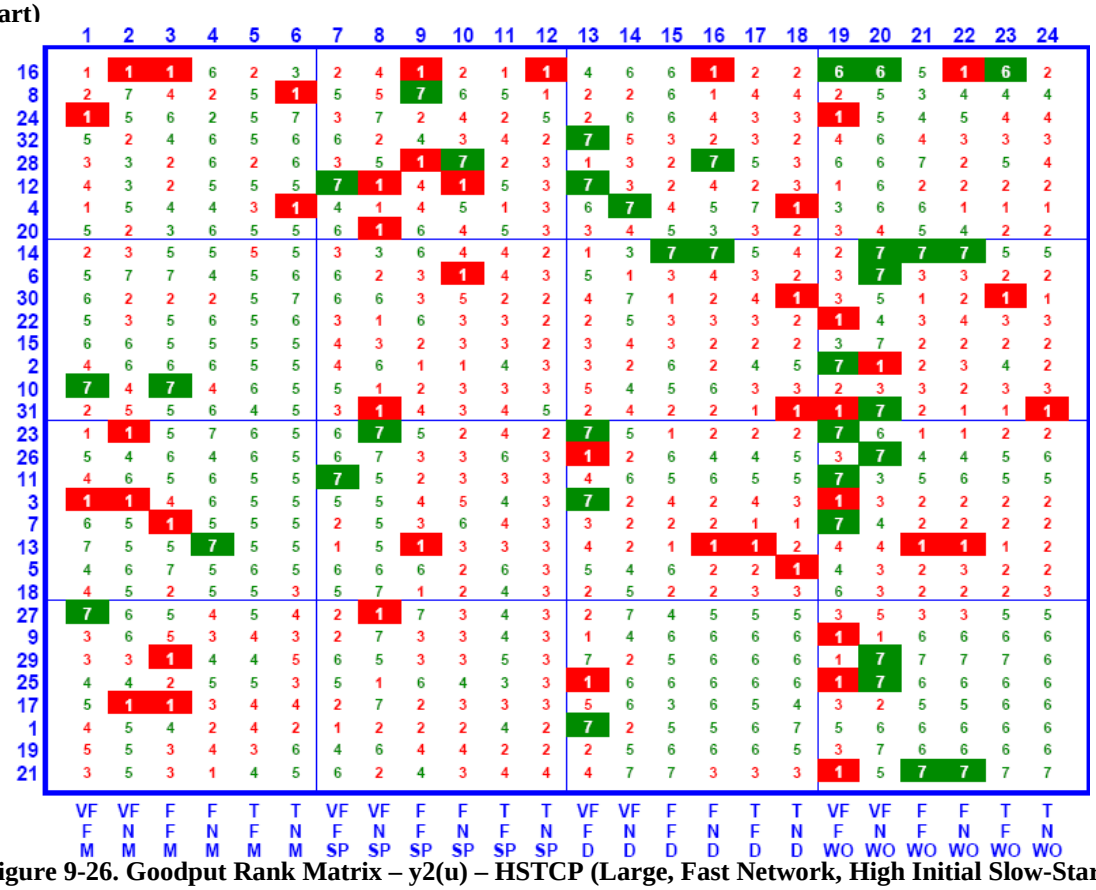


Figure 9-26. Goodput Rank Matrix – y2(u) – HSTCP (Large, Fast Network, High Initial Slow-Start)

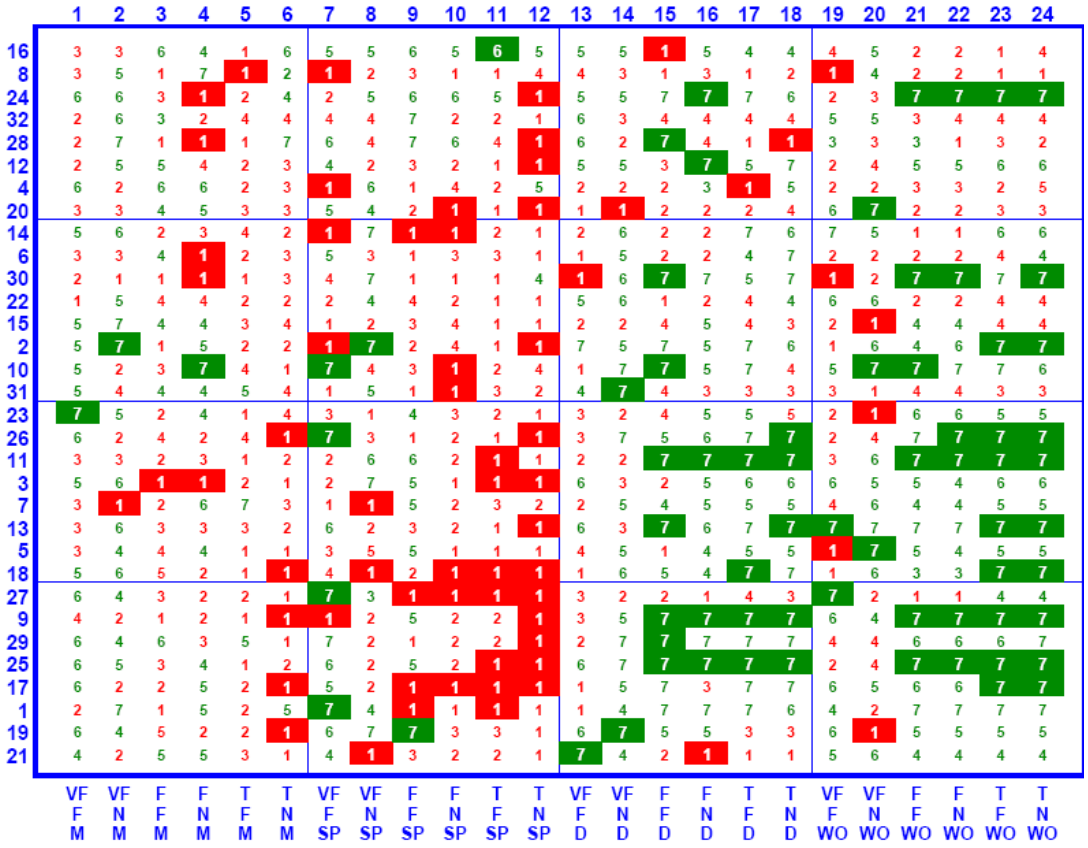


Figure 9-27. Goodput Rank Matrix – y2(u) – HTCP (Large, Fast Network, High Initial Slow-Start)

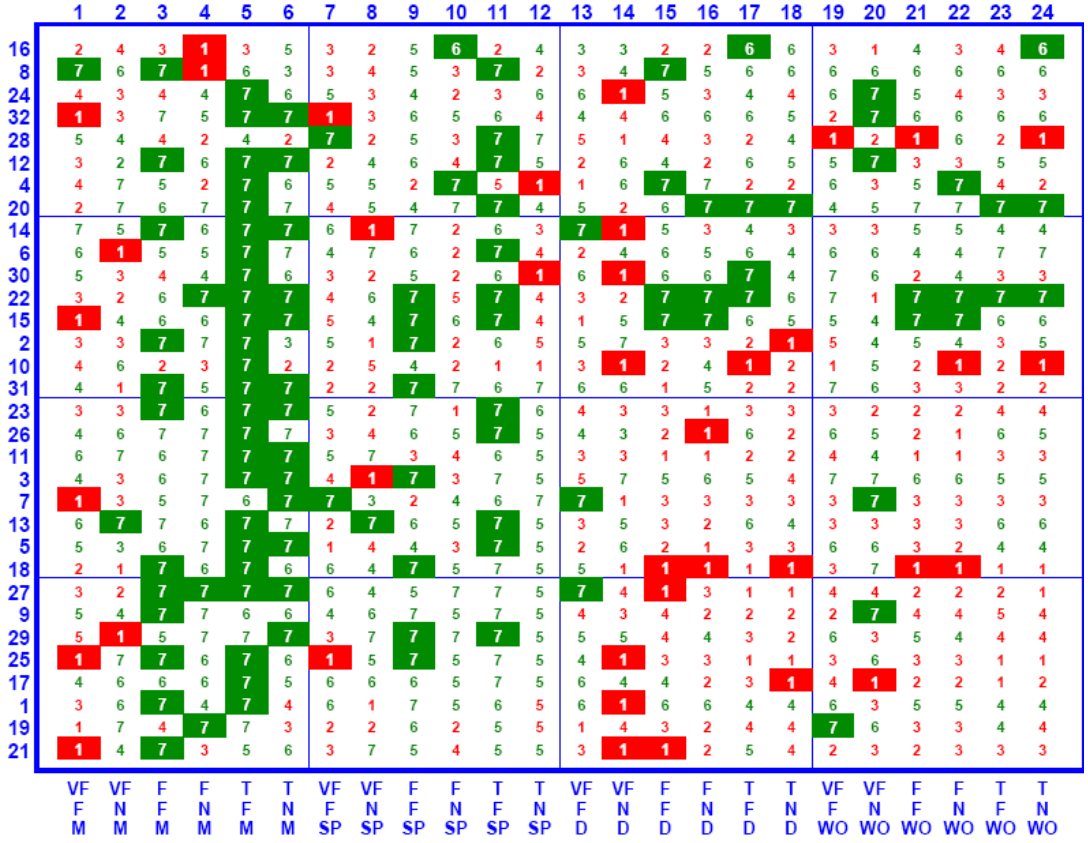


Figure 9-28. Goodput Rank Matrix – y2(u) – Scalable (Large, Fast Network, High Initial Slow-Start)

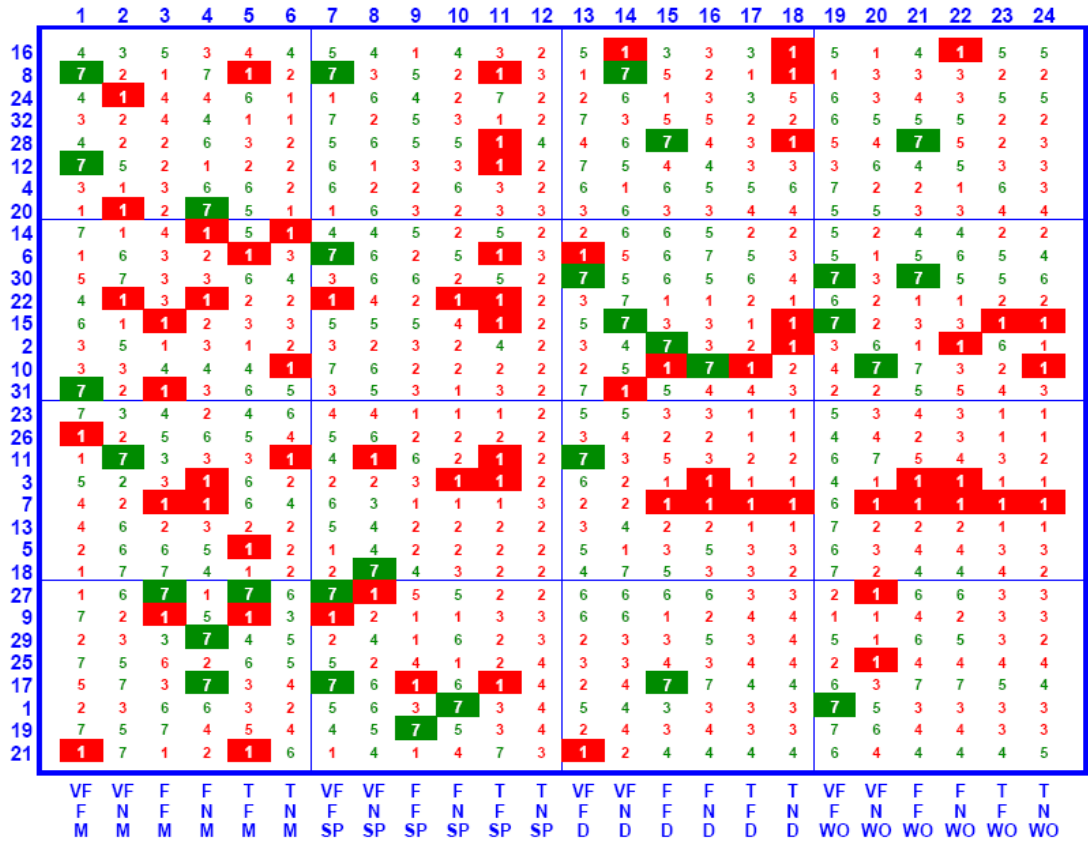


Figure 9-29. Goodput Rank Matrix – y16(u) – BIC (Large, Fast Network, High Initial Slow-Start)

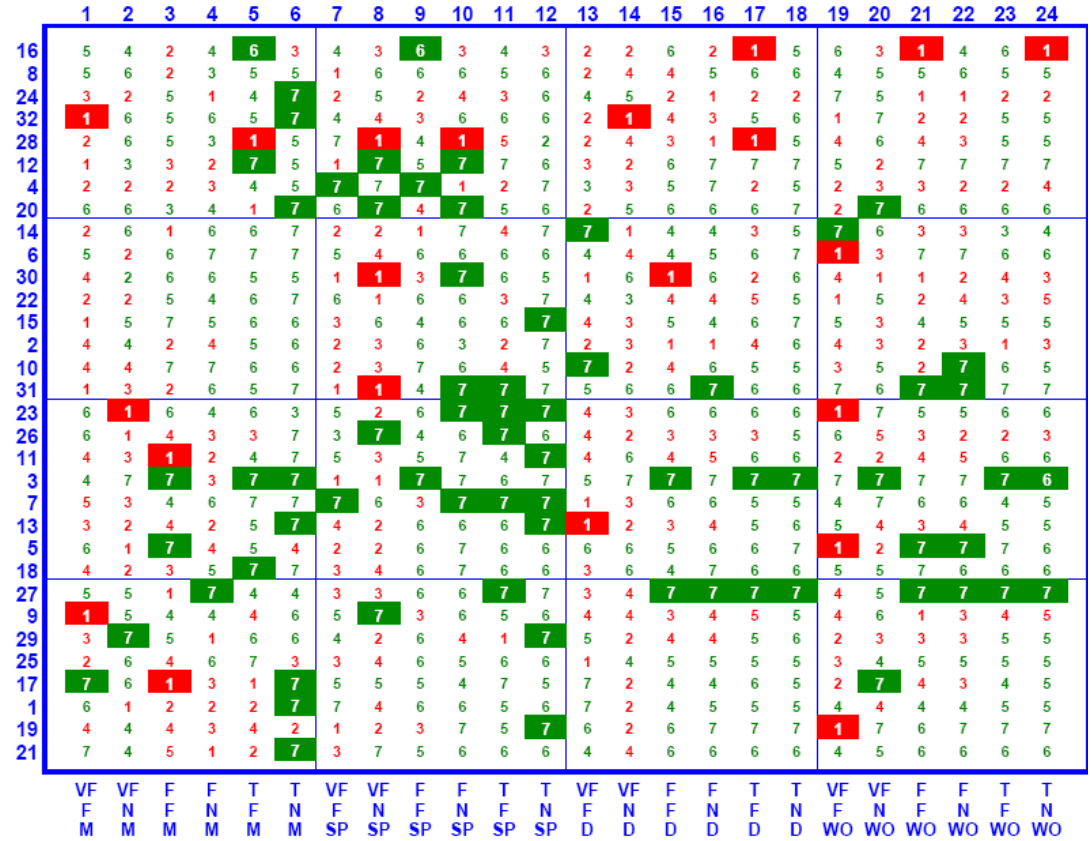


Figure 9-30. Goodput Rank Matrix – y16(u) – CTCP (Large, Fast Network, High Initial Slow-Start)

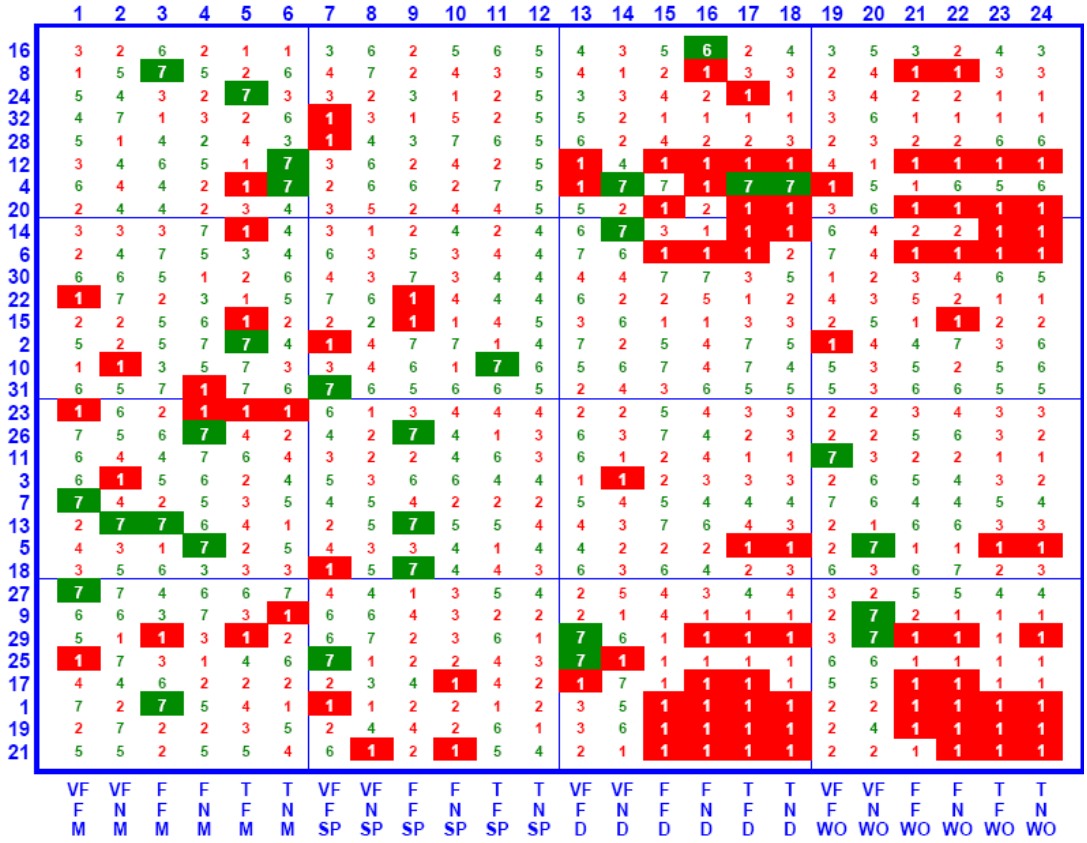


Figure 9-31. Goodput Rank Matrix – y16(u) – FAST (Large, Fast Network, High Initial Slow-Start)

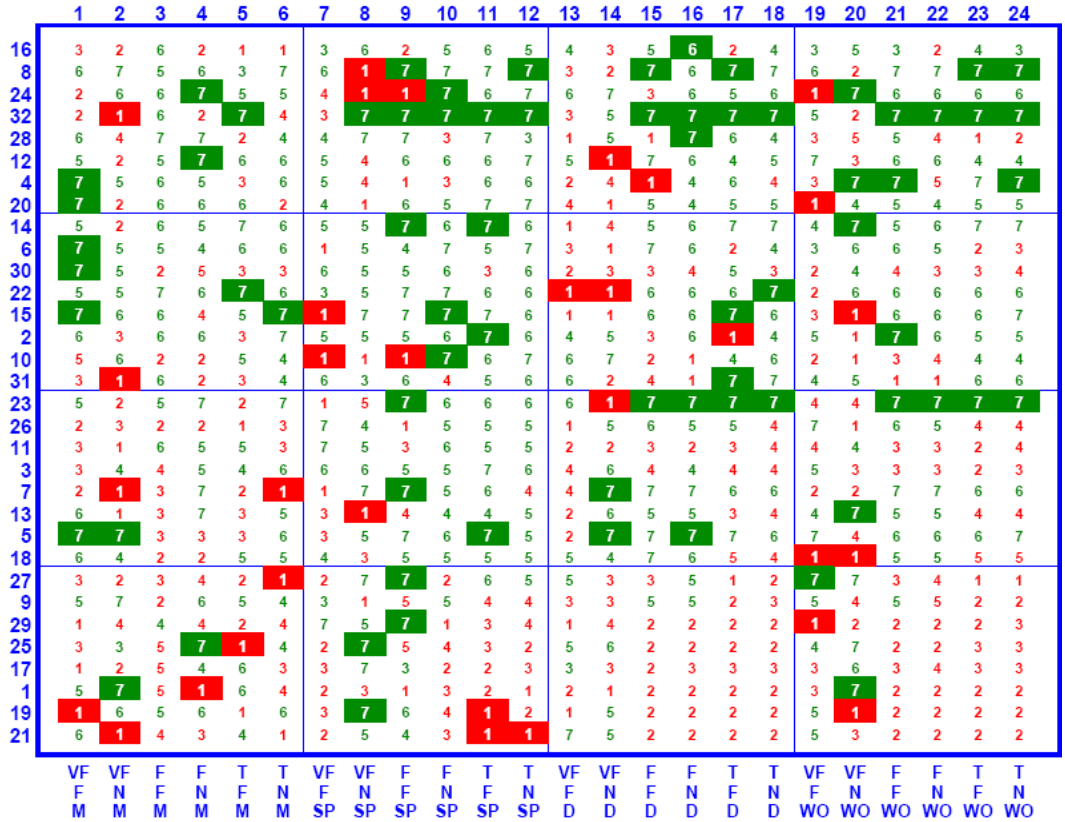


Figure 9-32. Goodput Rank Matrix – y16(u) – FAST-AT (Large, Fast Network, High Initial Slow-Start)

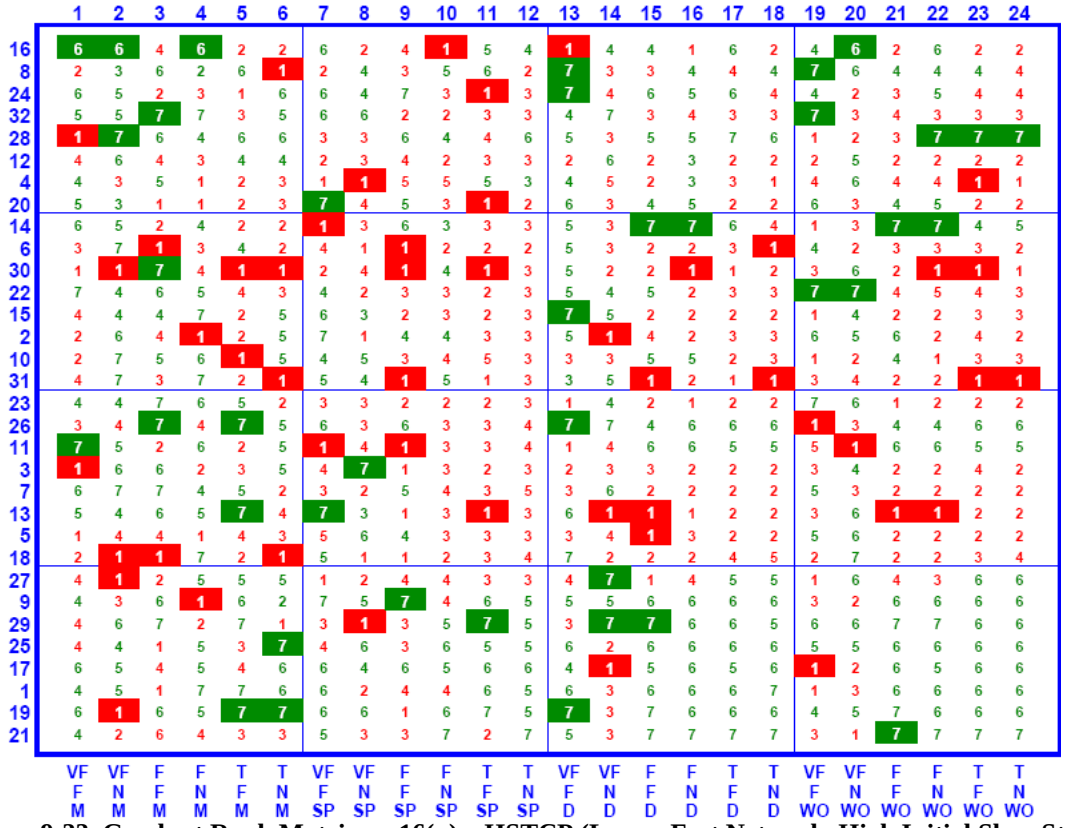


Figure 9-33. Goodput Rank Matrix – y16(u) – HSTCP (Large, Fast Network, High Initial Slow-Start)

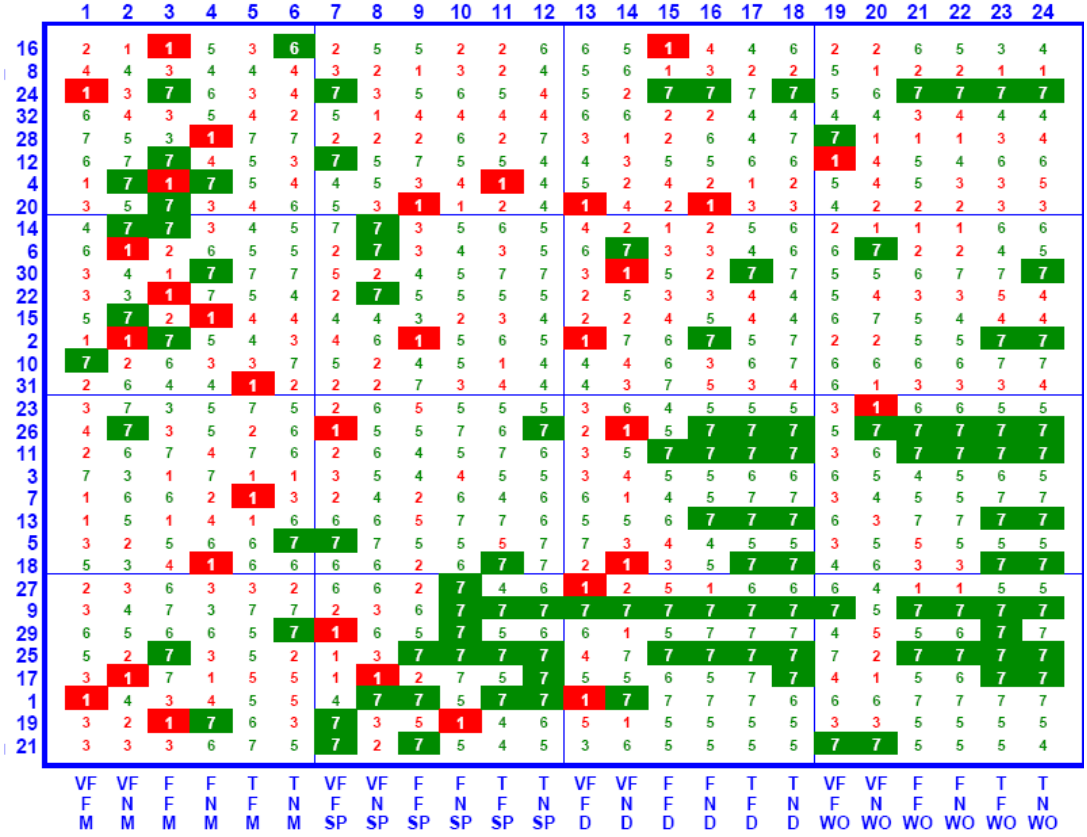


Figure 9-34. Goodput Rank Matrix – y16(u) – HTCP (Large, Fast Network, High Initial Slow-Start)

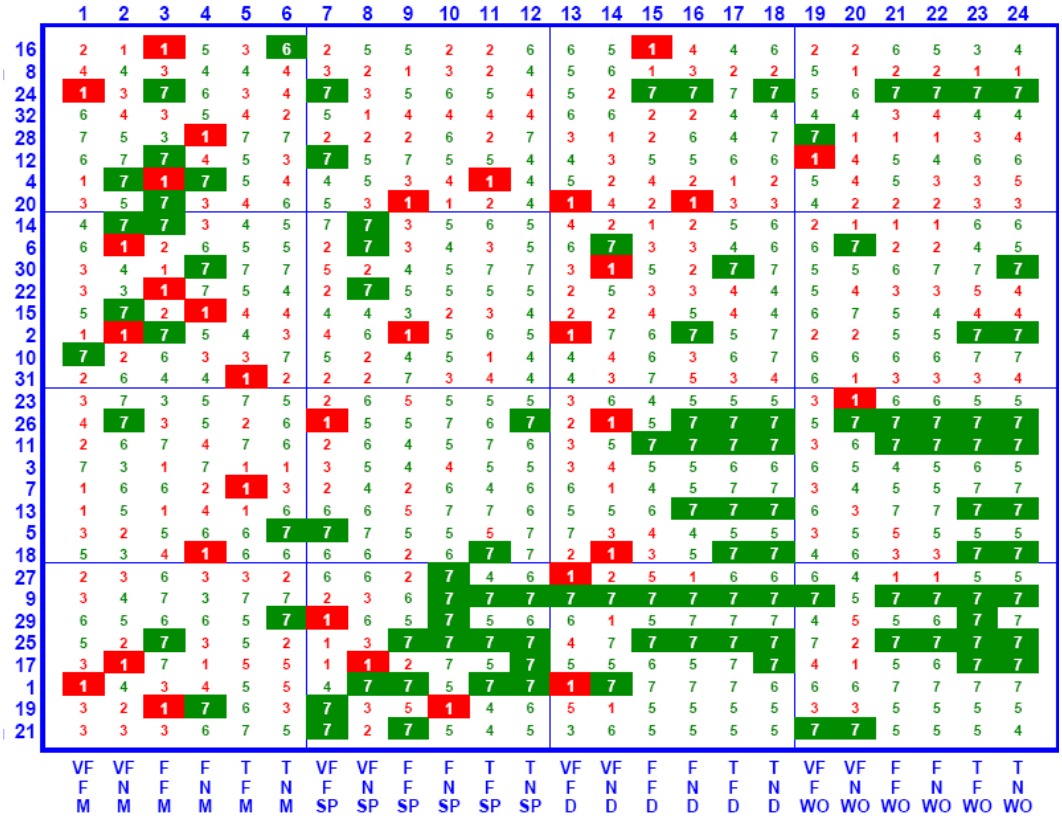


Figure 9-35. Goodput Rank Matrix – y16(u) – Scalable (Large, Fast Network, High Initial Slow-Start)

Table 9-13. Summary Average and Standard Deviation in Goodput Rankings for Flows using Alternate Congestion Control Algorithms and for Competing TCP Flows (Large, Fast Network, High Initial Slow-Start Threshold)

		BIC	CTCP	FAST	FAST-AT	HSTCP	HTCP	STCP
y2(u)	M	4.70	3.01	3.93	3.39	4.30	3.33	5.23
	SP	4.05	3.41	4.34	5.05	3.57	2.76	4.71
	D	3.55	4.39	3.34	4.61	3.79	4.50	3.69
	WO	3.37	4.32	3.24	4.52	3.81	4.56	4.08
	Avg.	3.92	3.78	3.71	4.39	3.87	3.79	4.43
y16(u)	M	3.51	4.34	3.93	4.28	4.06	4.19	3.56
	SP	3.20	4.88	3.69	4.71	3.65	4.55	3.24
	D	3.49	4.53	3.07	4.12	3.94	4.65	4.10
	WO	3.54	4.54	2.93	4.23	3.86	4.80	3.96
	Avg.	3.44	4.57	3.41	4.34	3.88	4.55	3.72
y2(u) & y16(u)	Avg.	3.68	4.18	3.56	4.36	3.87	4.17	4.07
	Std.	0.48	0.63	0.49	0.49	0.23	0.73	0.64

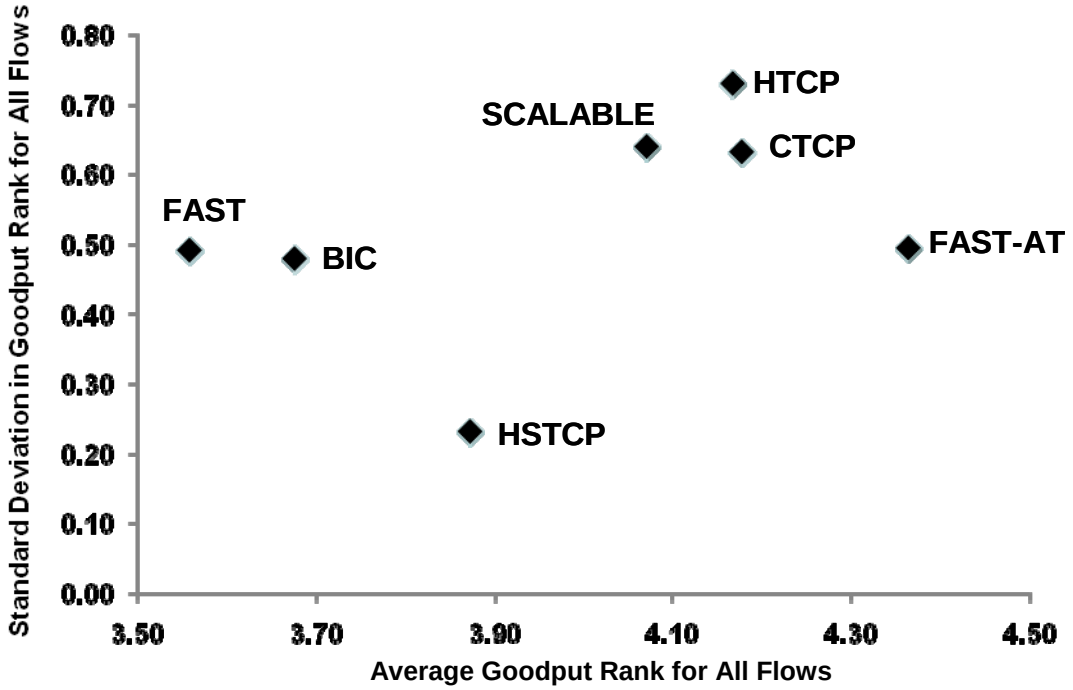


Figure 9-36. Average (x axis) vs. Standard Deviation (y axis) in Goodput Rank (Large, Fast Network, High Initial Slow-Start Threshold)

Looking at the rank matrices, summary table and scatter plot gives some impressions regarding relative goodput for flows operating under various congestion control algorithms as well as for competing TCP flows. Four of these impressions were seen and discussed before (in Sec. 8.4.3.1). First, CTCP, HTCP and FAST-AT[3] appear relatively friendly to TCP flows. Second, Scalable TCP ranks high in goodput for movies and for all file sizes under sporadic losses. Third, BIC, FAST, HSTCP and Scalable TCP are relatively unfriendly[4] to TCP flows. Fourth, HTCP ranks poorly with respect to large flows. Comparing relative ranks in a large, fast network against relative ranks in a smaller, slower network, revealed two additional impressions. First, differences in rank cover a lower range in the large, fast network (3.56 to 4.36) than was the case for a smaller, slower simulated network (3.16 to 4.63). Second, the standard deviation in ranks was much narrower in a large, fast network (0.23 to 0.73) than in a smaller, slower network (0.34 to 1.37).

Overall, then, assuming a high initial slow-start threshold, as a network becomes faster and less congested, differences in goodput offered by the alternate congestion control algorithms and competing TCP flows come closer together. Adopting a large initial slow-start threshold eliminates activation of enhanced window increase procedures

[3] FAST-AT reduces transmission rate 50 % on a packet loss and can increase rate quickly after that, but a falling transmission rate can cause FAST-AT to reduce the α parameter, which causes a slower increase in transmission rate when recovery occurs.

[4] TCP unfriendliness implies reducing transmission rate substantially less than 50 % following a packet loss and/or increasing transmission rate much more quickly than linearly when recovery occurs.

available in the alternate congestion control algorithms. When losses occur, differences in goodput can be discerned and attributed to loss/recovery characteristics of the various algorithms. As a network becomes less and less congested, alternate congestion control algorithms have fewer chances to invoke their enhanced loss/recovery procedures.

9.5 Findings

This experiment considered a range of files sizes (movies, service packs, documents and Web objects) being transferred across a largely uncongested network, where some (fast and typical) paths experienced more congestion than others (very fast paths) and where some flows could achieve a maximum rate of 80×10^3 pps, while others were constrained (by the interface speed of a sender or receiver) to at most 8×10^3 pps. Flows using TCP congestion control were mixed with flows using one of seven alternate congestion control algorithms. All flows adopted the same initial slow-start procedures to determine the maximum available transfer rate (i.e., all flows used a high initial slow-start threshold). In general, under these conditions (ignoring network speed and propagation delay), goodput experienced on individual flows is influenced by two main factors: (1) file size and (2) packet losses and related recovery procedures. The results of these experiments confirmed many of the findings discussed in Chapter 8.

9.5.1 Finding #1

Given a high initial slow-start threshold and the minimal congestion arising in a large, fast network, differences in average goodput narrowed in each flow group, whether using alternate or standard TCP congestion control procedures. That is, goodput differences shrank among alternate congestion control algorithms and between TCP flows and flows using alternate congestion control procedures. Assigning all flows a high initial-slow start threshold eliminated differences in increase procedures when determining the maximum available transfer rate. Increasing network speed and size reduced overall congestion by several orders of magnitude under most conditions. Lower congestion led to fewer losses, which reduced opportunities for alternate congestion control algorithms to activate enhanced loss/recovery procedures.

9.5.2 Finding #2

Under selected conditions, where file sizes were large (i.e., movies and service packs) and where congestion could appear (i.e., on fast and typical paths, which can experience sharing among more flows), differences in average goodputs could still be distinguished due to differences in loss/recovery procedures. Though the effects were somewhat muted because overall congestion was lower, the finding here is analogous to a similar finding in Chapter 8. Scalable TCP, BIC and HSTCP do not decrease their transmission rate as much as the other algorithms when a loss is detected. This means that already established flows continue to transmit at higher rates at the cost of inhibiting newer flows and also TCP flows, which cut their transmission rate in half on a loss. Thus, under congested conditions, these protocols provided higher goodput than TCP flows.

9.5.3 Finding #3

Overall, in this experiment, FAST-AT provided the best balance in relative goodput achieved on all flows. CTCP ranked second best overall, followed closely by HTCP.

FAST-AT ranked third most friendly (after CTCP and then HTCP) to TCP flows and ranked second best (after Scalable TCP) at providing goodput to flows using alternate congestion control procedures. Note that lower overall congestion narrowed significantly the differences in ranking among all the algorithms.

9.5.4 Finding #4

As seen in earlier experiments, this experiment showed that use of some alternate congestion control protocols altered selected macroscopic characteristics of the network. Here, as in Chapter 8, the characteristic changes were, in general, not statistically significant. We attribute this to two main factors: (1) overall congestion levels were kept much lower than in previous experiments and (2) FAST and FAST-AT, which have similar characteristics, where not separated in the analyses, which tended to reduce the statistical significance that might be attributed to either algorithm considered without the other. In general, the current experiments confirmed that FAST and FAST-AT tend to increase retransmission rate under higher congestion. Thus, more flows are pending in the connecting state and fewer flows complete per unit of time. In addition, Scalable TCP tends to increase buffer occupancy throughout the network. This can also lead to higher retransmission rates, to more flows pending in the connecting state and to fewer flows completing per unit time. At lower congestion levels, Scalable TCP performed worse on these metrics than FAST (and FAST-AT). At higher congestion levels, FAST (and FAST-AT) performed worse. Finally, we found again in this experiment that CTCP can exhibit a much higher average congestion window size than other congestion control algorithms. The increase appears most prominent under lower congestion levels.

9.6 Conclusions

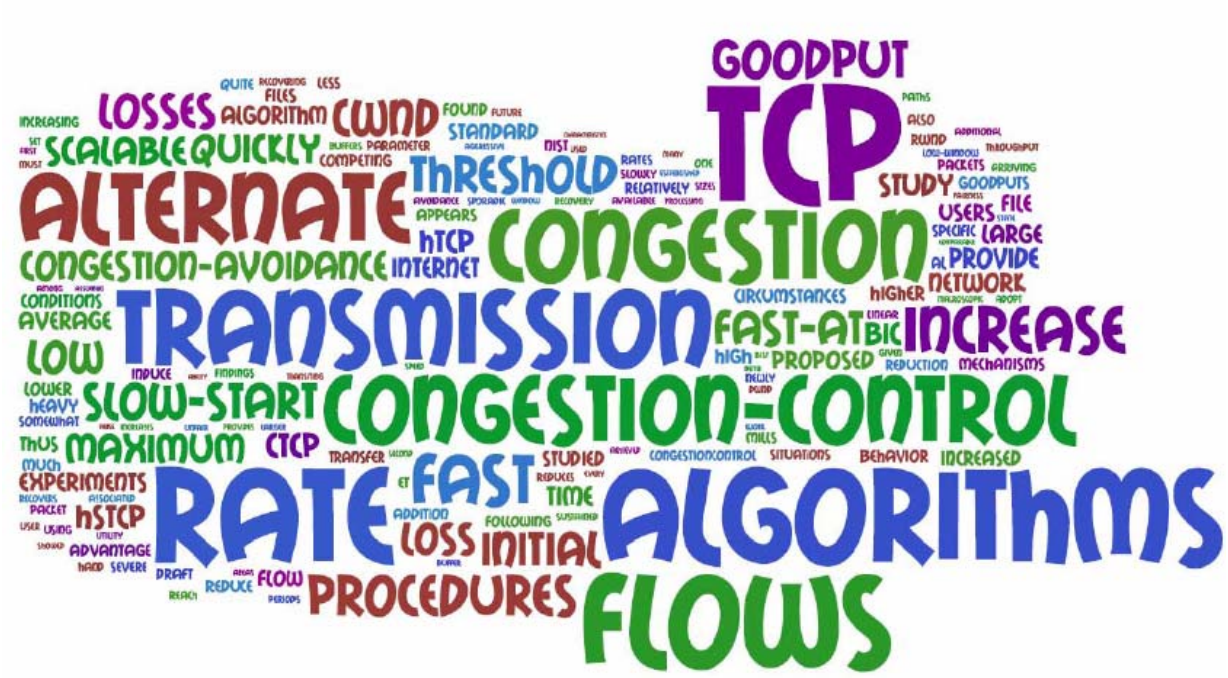
In this chapter, we described an experiment to investigate effects on macroscopic behavior and user experience when deploying various congestion control algorithms in a large, fast, simulated, heterogeneous network, i.e., a network that includes flows operating under standard TCP congestion control procedures together with flows operating under one of seven proposed alternate congestion control algorithms. In effect, we repeated, with a few changes, half the experiments from Chapter 8. Specifically, we repeated the experiments where all flows adopted a high initial slow-start threshold. We changed the network to increase router speeds and number of sources and receivers by an order of magnitude, which also changed buffer sizes. Increasing network speed and size required more than an order of magnitude increase in computational cost, which motivated us to repeat only half the experiments from Chapter 8.

We demonstrated that, under a larger, faster network (given a high initial slow-start threshold), reduced congestion levels narrowed differences in average goodput among flows using alternate congestion control algorithms and also between flows using alternate and standard TCP congestion control procedures. Lowered congestion meant fewer losses, which reduced the opportunities for alternate congestion control algorithms to activate enhanced loss/recovery procedures. We also confirmed some findings from the experiments described in Chapter 8. First, on a loss, Scalable TCP, BIC and HSTCP reduced transmission rate less than other algorithms, so these algorithms tended to be provide higher goodput than TCP flows on larger file sizes under congested conditions. Second, under conditions with higher congestion, FAST and FAST-AT exhibited higher

retransmission rates, more pending flow connections and fewer flows completing. Under conditions with lower congestion, Scalable TCP could also exhibit such undesirable network-wide properties. Third, CTCP, FAST-AT and HTCP showed better balance overall (than other alternate congestion control algorithms) with respect to relative average goodput for all flows, including both those using the alternate procedures and those using standard TCP procedures.

After completing five sets of simulation experiments (as described in Chapters 6 through 9), we accumulated sufficient information to draw some conclusions about the behaviors of the seven congestion control algorithms we studied. We also developed sufficient experience to evaluate the various methods we adopted. We turn to these topics next, where we conclude our study and identify future work.

Chapter 10 – Conclusions



10 Conclusions

Below, we provide conclusions in two general categories: conclusions and recommendations (Sec. 10.1) about the congestion control algorithms we studied and conclusions and recommendations (Sec. 10.2) about the methods we applied. Along with each set of conclusions and recommendations, we also provide suggestions for related future work.

10.1 Conclusions about Congestion Control Algorithms

The simulation and modeling studies reported here enabled us to draw a range of conclusions about the general utility and safety of seven proposed alternate congestion control algorithms for the Internet. We were also able to characterize each of the congestion control algorithms we studied. In the end, we developed some recommendations about whether it makes sense to deploy alternate congestion control algorithms at large scale on the general Internet. Finally, though our study is quite comprehensive, we recognize the need for future work to investigate some questions that we did not study. We address these topics, in turn, below.

10.1.1 Utility and Safety of Alternate Congestion Control Algorithms

Our simulation and modeling experiments showed that deploying alternate congestion control algorithms can provide improved user experience under specific circumstances. As discussed below, the nature of such circumstances bound the utility that alternate congestion control algorithms may provide. In addition, the experiments showed that some proposed algorithms can be deployed without driving large changes in macroscopic behavior throughout a network. On the other hand, other proposed algorithms altered behavior in undesirable directions under specific spatiotemporal situations. We address these topics in detail.

10.1.1.1 Increase Rate. One of the key questions for any data transport protocol is: How fast can the maximum available transfer rate be achieved on a network path? Assuming no congestion (i.e., no losses) protocols that can quickly attain the maximum rate will spend the largest portion of a file transfer at that rate. Each TCP flow begins without any knowledge of the maximum available transfer rate. For this reason, TCP specifies an initial slow-start process where the source transmits slowly but then, as feedback arrives from a receiver, quickly increases the transmission rate until reaching a specified (initial slow-start) threshold or encountering a loss. This initial slow-start process is not altered by any of the proposed alternate congestion control algorithms that we studied.

Assuming no (or low) congestion, the setting of the initial slow-start threshold can be quite important when comparing goodputs experienced by users on TCP flows with goodputs for users on flows operating under alternate congestion control algorithms.[1]

[1] Note that in real TCP flows receivers may convey a receiver window (*rwnd*) that can restrict goodput quite severely because sources pace transmission based on the minimum of the congestion window (*cwnd*) and *rwnd*.Tthe following may hold: *rwnd* < *cwnd*. In our studies, we assume an infinite *rwnd* in order to compare the effects of congestion control algorithms adjusting the *cwnd*. The goodput on many TCP flows in a real network might well be constrained by *rwnd*. In such cases, alternate congestion control algorithms would provide little advantage over TCP congestion control procedures.

When the initial slow-start threshold is set arbitrarily high, on average all flows achieve maximum transfer rate with the same quickness. Under such situations, the goodput seen on TCP flows and flows running alternate algorithms appears quite comparable. Flows carrying short files (e.g., Web objects and document downloads) tend to complete while in initial slow-start, which means that alternate congestion control procedures (restricted to the congestion avoidance phase of a flow) do not operate. Even flows conveying long files can operate for extended periods under initial slow-start because such flows do not enter congestion avoidance until encountering a loss.

When the initial slow-start threshold is set low (e.g., 64 Kbytes) all of the alternate congestion control algorithms that we studied increase transmission rate more quickly than the linear increase provided by the standard TCP congestion avoidance procedures. Thus, under low congestion, when the initial slow-start threshold is set low compared to the size of files transferred (and assuming the receiver window – *rwnd* – is not constraining transmission rate) users on TCP flows will see much lower goodput than users of alternate congestion control algorithms. The larger the file sizes being transferred the larger the goodput advantage of the alternate algorithms. The alternate congestion control algorithms provide different degrees of goodput improvement over TCP congestion avoidance procedures. As discussed below (Sec. 10.1.2), these goodput differences can be tied directly to the speed with which the alternate algorithms reach the maximum available transmission rate.

Under conditions of heavy congestion the setting of the initial slow-start threshold matters less because initial slow-start terminates upon the first packet loss and then a flow enters the congestion avoidance phase, which is where the alternate congestion control algorithms differ from TCP procedures. In such situations, the main difference in goodput experienced by users relates to the loss/recovery procedures defined by the alternate algorithms. We turn to this topic next.

10.1.1.2 Loss/Recovery Processing. Two key questions arise when a data transport protocol experiences a packet loss. (1) How much should the protocol reduce transmission rate upon a loss? (2) How quickly should the protocol increase transmission rate after the reduction? The standard TCP congestion avoidance procedures reduce transmission rate by one-half on each packet loss. Subsequently, TCP congestion avoidance procedures increase transmission rate linearly. The alternate congestion control algorithms we studied specify various procedures for transmission rate reduction and increase following a lost packet.

One group of algorithms (Scalable TCP, BIC[2] and HSTCP) reduce transmission rate less than TCP after a packet loss. As a result, these algorithms tend to retain a higher transmission rate and associated buffers than is the case for TCP flows. Smaller rate reduction can allow these algorithms to provide established flows with higher goodputs following packet losses. We found this effect to increase with increasing loss rate and also file size. In addition, these algorithms can be somewhat unfair (see Sec. 10.1.1.3) to algorithms (such as TCP) that exhibit a more reduced transmission rate following a loss,

[2] Note that on repeated losses occurring close in time, BIC can reduce *cwnd* substantially more than the standard TCP congestion avoidance procedures; thus, on paths with very severe congestion BIC can actually provide lower goodput than TCP and can also occupy fewer buffers.

as well as to flows that have not had sufficient time to attain a high transmission rate prior to a loss.

A second group of algorithms (CTCP, FAST and FAST-AT) reduce transmission rate in half following a loss. HTCP appears to be a hybrid, reducing transmission rate variably, between 20% and 50%, depending on conditions. The higher reduction occurs when transmission rate had been changing substantially in a round-trip time and the lower reduction occurs when transmission rate is less variable. To obtain higher goodput, these algorithms increase transmission rate more quickly than TCP flows following a rate reduction. As discussed below (Sec. 10.1.2), the rate of increase varies with the specific algorithm. Typically, HTCP and CTCP are less aggressive than FAST and FAST-AT when increasing transmission rate after a reduction. Though, FAST-AT will be less aggressive when sufficient congestion exists to force a reduction in the α parameter. An aggressive rate increase following a rate reduction can induce additional losses on a path. When such losses affect TCP flows, then linear recovery procedures lead to lower goodputs. Under severe congestion, CTCP and HTCP can provide better goodput than FAST and FAST-AT, which can underperform TCP.

In areas and at times of extreme congestion, most of the alternate algorithms we studied include procedures to adopt standard TCP congestion avoidance behavior. These procedures appear motivated by the theory that when congestion is sufficiently severe then existing TCP behavior provides the best approach to fairly share the limited available transmission rate. The most typical technique employed is to set a low-window threshold. When the congestion window (*cwnd*) is below the threshold then TCP congestion avoidance procedures are used. When *cwnd* is above the threshold then alternate congestion avoidance procedures are used. Specific values for the threshold vary among the alternate congestion control algorithms. The combination of different thresholds and different file sizes can lead to modest differences in user goodputs.

HTCP handles adaptation to TCP procedures somewhat differently than the other alternate algorithms we investigated. After a loss, HTCP adopts linear rate increase for a time. The time period is an HTCP parameter, set in these experiments to one second. We found that HTCP then adapts to TCP linear increase after every loss, regardless of file size or *cwnd* value. For larger files, which tend to have higher *cwnd* and to experience more losses during transmission, this approach tends to lower goodput significantly relative to other alternate algorithms, which do not adopt linear increase after every loss.

FAST and FAST-AT do not use standard TCP congestion avoidance procedures under any circumstances. In times and areas of heavy congestion, failure to adopt less aggressive rate increase can lead to oscillatory behavior and to an associated increase in loss rate. Increased losses lead to lower user goodputs. FAST-AT does somewhat better under heavy congestion because the α parameter can be lowered, causing less aggressive rate increases. Still, under many conditions, FAST-AT can exhibit a similar increased loss rate to FAST.

10.1.1.3 TCP Fairness. TCP fairness denotes the situation where competing flows transiting a shared path in the Internet will all receive an equal share of available goodput. Comparing alternate congestion control algorithms with respect to TCP fairness can be somewhat difficult because the alternate algorithms are designed to give better goodput than TCP for large file transfers on high bandwidth-delay paths. Thus, for

example, all of the alternate algorithms can increase transmission rate more quickly than TCP given a low initial slow-start threshold and large file sizes. Further, all alternate algorithms take steps to provide loss/recovery improvements over the standard TCP congestion avoidance procedures. On the other hand, most of the alternate algorithms take steps to adopt TCP congestion avoidance procedures when congestion is sufficiently high. Given these factors, one would expect all alternate congestion control algorithms to provide better goodput than TCP under optimal conditions. In addition, some of the alternate algorithms are assured of performing no worse than TCP under suboptimal conditions. The usual measures of TCP fairness do not apply in such circumstances because they would tend to measure how much of a goodput advantage a given alternate algorithm provides over TCP procedures. Instead, we measured *relative* TCP fairness by ranking the average goodput achieved by TCP flows when they competed with each alternate congestion control algorithm under the same conditions. We considered the average rank across four file sizes: Web objects, documents, software service packs and movies. In this way, we could elicit the relative TCP fairness of the alternate algorithms.

We found that CTCP and HTCP were most fair to TCP flows. We found FAST-AT third fairest to TCP flows under high initial slow-start threshold. Under low initial slow-start threshold, FAST-AT proved more unfair to TCP flows because of its quick increase in transmission rate after passing the initial slow-start threshold. Injecting more FAST-AT packets into the network induced more losses in TCP flows, which could not recover as quickly.

We found Scalable TCP, BIC and FAST to be most unfair to TCP flows. Established Scalable and BIC flows (large files) tended to maintain higher transmission rates after losses, while competing TCP flows cut transmission rates in half. By maintaining higher transmission rates and, thus, more buffer space, Scalable and BIC flows induced more losses in TCP flows. FAST could recover more quickly from losses than TCP flows and so FAST flows could occupy more buffers and induce more losses in TCP flows. In addition, like FAST-AT, FAST exhibited unfairness under low initial slow-start threshold because of its quick increase in transmission rate upon entering congestion avoidance.

HSTCP appeared moderately fair to TCP flows, especially under conditions of lower congestion and under a low initial slow-start threshold. HSTCP showed TCP unfairness, similar to Scalable TCP, under conditions of heavy congestion.

We believe that Scalable TCP, BIC and HSTCP could also be unfair to competing flows that are newly arriving. Given that some large flows operating under Scalable TCP, BIC and HSTCP have established relatively high transmission rates and associated large buffer states and given that newly arriving flows induce losses, the established flows will not reduce transmission rate very much and will maintain large buffer states. The newly arriving flows will be forced into congestion avoidance on the loss. Further, Scalable TCP and HSTCP do not increase transmission rate very fast early in a flow's life, so newly arriving flows of these types can face difficulty increasing transmission rate.

10.1.1.4 Utility Bounds. We showed that alternate congestion control protocols could provide increased utility (goodput) for users, but we also found that this increased utility would be maximized only under specific, bounded circumstances. First, the *rwnd* must not be constraining flow transmission rate. Second, a flow must be using a relatively low

initial slow-start threshold. Third, a flow must be transmitting a large file. Fourth, a flow's packets must be transiting a relatively uncongested path (i.e., experiencing only sporadic losses from congestion or corruption) or else users must be willing to accept marked unfairness (e.g., as seen with Scalable TCP) in trade for increased goodput. These bounds arise from some simple factors.

If a flow is restrained by receipt of a relatively small *rwnd*, then the ability of alternate congestion control regimes to increase to a high *cwnd* cannot be used to transmit faster on a flow. Assuming *rwnd* does not constrain flow goodput, flows can increase goodput in concert with *cwnd* by using slow start to discover the maximum transmission rate. Given a high initial slow-start threshold, then all flows can discover the maximum *cwnd* with the same quickness. In this case, TCP flows would reach maximum *cwnd* on average with the same pace as flows running alternate algorithms. Only when the initial slow-start threshold is low, forcing early entry into congestion avoidance, could flows using alternate algorithms reach maximum *cwnd* more quickly than TCP. If flows are transferring large files, then the ability to reach maximum transmission rate quickly provides a substantial goodput advantage, and the advantage increases with file size. Under small files a transfer could complete under initial slow-start and, thus, the advantage inherent in congestion avoidance increase procedures for the alternate algorithms would not be realized. When flows transit heavily congested paths in the network, then most of the alternate congestion control algorithms adopt standard TCP congestion avoidance procedures, which negate any goodput advantage over TCP flows. Though FAST and FAST-AT do not adopt standard TCP congestion avoidance procedures, we found that heavy congestion can cause the transmission rate to oscillate on FAST and FAST-AT flows, which leads to higher loss rates, more retransmissions and lower goodput.

We are unable to determine how likely a particular flow is to operate under the bounded circumstances required for alternate congestion control algorithms to provide improved goodput over standard TCP. Certainly it would be possible to engineer a network, or segments of a network, to provide specific users with high utility from alternate congestion control algorithms. On the other hand, we suspect a rather low probability for such circumstances to arise generally in a network. Thus, we conclude that alternate congestion control algorithms can provide improved user goodput, but most users seem unlikely to benefit very often.

10.1.1.5 Safety. Given that on occasion some users could benefit from the increased goodputs available from alternate congestion control algorithms, we need to consider whether widespread deployment of such algorithms could induce undesirable macroscopic characteristics into the network. In other words, are there significant costs that might offset the modest benefits associated with deploying alternate congestion control algorithms? We can answer this question only in part because we simulated networks where sources used either a single congestion control regime or where some sources used a selected alternate congestion control algorithm while other sources used standard TCP congestion control procedures. There could be additional cautionary findings that arise from a heterogeneous mixture of alternate congestion control algorithms. We postpone such investigations to future work.

In our experiments, we simulated a wide range of conditions and we considered numerous scenarios comparing network behavior under specific alternate congestion control algorithms, sometimes mixed with TCP procedures. For most algorithms under most conditions, we found little significant change in macroscopic network characteristics. One exception relates to FAST and FAST-AT. In spatiotemporal realms with high congestion, where there were insufficient buffers to support the flows transiting specific routers, FAST and FAST-AT exhibited oscillatory behavior where the flow *cwnd* increased and decreased rapidly with large amplitude. Under these conditions, the network showed increased loss and retransmission rates, a higher number of flows pending in the connecting state and a lower number of flows completed over time. Thus, FAST and FAST-AT should be deployed on a wide scale only with great care. There appears to be some possibility that FAST could cause significant degradation in network performance in selected areas and for selected users. We recommend the need for additional study of FAST and FAST-AT prior to widespread deployment and use on the Internet.

10.1.2 Characteristics of Individual Congestion Control Algorithms

Below, we provide a brief summary of the characteristics found from our experiments for each alternate congestion control algorithm. For each algorithm we consider four characteristics. The first characteristic, *implementation complexity*, assesses how much code might be required to implement an algorithm. The second characteristic, *activation trigger*, identifies the condition (usually a specific congestion window size) that causes a flow to switch between standard TCP congestion avoidance procedures and alternate procedures defined by an algorithm. The third characteristic, *goodput latency*, measures the time required for a flow to achieve maximum transmission rate on long-lived flows when operating under an algorithm's alternate congestion avoidance procedures. The fourth characteristic, *recovery latency*, measures the time required for a long-lived flow to recover maximum transmission rate after a period of congestion with sustained losses. Table 10-1 compares the seven alternate congestion control algorithms with respect to these four characteristics. We discuss the algorithms in alphabetical order, as shown in the table.

Table 10-1. Comparing Four Characteristics of Individual Alternate Congestion Control Algorithms

Algorithm	Implementation Complexity	Activation Trigger	Goodput Latency (avg)	Recovery Latency (avg)
BIC	high	14 packets	18.8 s	71.3 s
CTCP	moderate	41 packets	7.9 s	2.9 s
FAST	low	none	3.7 s	6.6 s
FAST-AT	moderate	none	3.7 s	26.0 s
HSTCP	low	31 packets	22.4 s	10.0 s
H-TCP	moderate	1 s w/o loss	16.6 s	10.0 s
Scalable TCP	low	16 packets	17.8 s	22.5 s

10.1.2.1 BIC. Clearly, among the seven algorithms we studied, BIC is the most complex to code and implement, requiring a potentially substantial amount of processing to adjust the *cwnd*. BIC uses standard TCP congestion avoidance procedures when *cwnd* is below a low-window threshold (14 packets, here). Under congestion with losses spaced sufficiently in time, BIC reduces *cwnd* less quickly than standard TCP, so BIC can achieve higher goodputs under sporadic losses by maintaining a high transmission rate and associated buffer state. This can be somewhat unfair to newly arriving flows. On the other hand, when congestion becomes severe, with losses spaced closely in time, BIC reduces *cwnd* much more quickly than TCP. Under such circumstances, BIC can take substantial time (average 71.3 s in our experiments) to recover maximum goodput after congestion eases. When considering the rate of increase in transmission speed under low congestion after reaching initial slow-start threshold, BIC averaged about 18.8 s to reach maximum transfer speed on long-lived flows. This rate of increase ranked fifth (of six) overall, and was competitive with HTCP, Scalable TCP and HSTCP.

10.1.2.2 CTCP. The algorithm for CTCP requires periodic processing to adjust an auxiliary delay window (*dwnd*), which increases the processing cost beyond that found in standard TCP congestion control. Under congestion, CTCP reduces transmission rate by one-half and then recovers relatively quickly. The advantage of CTCP recovery procedures appears most obvious after a period of severe congestion on a path. Under easing congestion, *dwnd* can increase quite quickly. Since CTCP augments the *cwnd* with the *dwnd*, transmission rate can also increase quickly – returning to maximum rate in an average 2.9 s in our experiments. In fact, in some situations, the rate of increase in *dwnd* appears unbounded. CTCP implementations should probably include a bound on maximum *dwnd*. Under periods of heavier congestion, increase in *dwnd* is constrained. In addition, the CTCP algorithm appears quite fair to competing CTCP flows as well as TCP flows. CTCP had the highest default low-window threshold (41 packets, here) among the algorithms we studied. Further, CTCP averaged about 7.9 s to reach maximum transfer speed on long-lived flows under low congestion and low initial slow-start threshold. This rate of increase ranked second overall behind only FAST and FAST-AT, which tied for first.

10.1.2.3 FAST. The algorithm for FAST requires periodic processing to adjust the target *cwnd*. While each adjustment demands little computation, the default periodicity (20 ms, here) can require multiple adjustments within a single round-trip time. FAST does not have a low-window threshold; thus, after initial slow-start, FAST flows never use standard TCP congestion avoidance procedures. Under congestion, FAST reduces transmission rate by one-half and then recovers very quickly. The advantage of FAST recovery speed appears under both sporadic losses and when congestion eases following a period of severe congestion on a path. Under easing congestion, FAST recovered maximum transmission rate in an average of 6.6 s in our experiments. On the other hand, for flows transiting congested areas, with insufficient buffer space for all flows, FAST exhibits oscillatory behavior that increases losses and, thus, retransmissions, which reduces user goodput. Under severe congestion, FAST causes an increase in flows pending in the connecting state because SYN packets are lost with increased probability. In addition, FAST can significantly reduce the number of flows completed over time in a

network. Among the algorithms we studied, FAST achieves maximum available transmission rate in the shortest time (3.7 s average) on long-lived flows under low congestion and low initial slow-start threshold. The ability of FAST to accelerate transmission rate led to superior goodputs (under low congestion and low initial slow-start threshold) for file sizes larger than Web objects, and the advantage of FAST increased with file size. The ability of FAST to quickly attain high transmission rates for large files tended to induce losses in competing flows. Since TCP flows could not recover quickly, FAST flows could attain much higher goodputs than competing TCP flows.

10.1.2.4 FAST-AT. The FAST-AT algorithm augments FAST with periodic procedures to monitor throughput and tune the α parameter used when adjusting the target *cwnd*. Without α tuning, FAST sets the α parameter to a fixed value. FAST-AT monitors throughput every round-trip time and tunes the α parameter periodically (every 200 s, here). As throughput improves past specified thresholds α is increased and as throughput declines past specified thresholds α is decreased. FAST-AT exhibits many of the same positive and negative properties as FAST. The main difference was that, under severe and sustained congestion, FAST-AT reduced the α parameter from a default setting of 200 to as low as 8. In such, circumstances FAST-AT recovers much more slowly than FAST. When throughput begins increasing, FAST-AT adjusts the α parameter only every 200 s and must make two adjustments (8 to 20 followed by 20 to 200) before reaching the maximum recovery rate. In our experiments, when recovering from sustained periods of heavy congestion, FAST-AT took longer (26 s average) to reach maximum transmission rate than all alternate algorithms except BIC. On the other hand, by recovering transmission rate more slowly under heavy congestion, FAST-AT proved more TCP friendly than FAST. This occurred because under such circumstances FAST-AT did not induce as many losses in competing TCP flows.

10.1.2.5 HSTCP. The HSTCP algorithm is relatively straightforward, updating the *cwnd* no more frequently than standard TCP. The HSTCP *cwnd* updates involve somewhat costly logarithmic and exponentiation operations. HSTCP uses standard TCP congestion avoidance procedures when the *cwnd* is below a low-window threshold (31 packets, here). HSTCP reduces *cwnd* less on a loss than standard TCP and provides more than linear increase in *cwnd* during congestion avoidance. Under both sporadic and heavy congestion, HSTCP retains a higher transmission rate (and associated buffers) than TCP. By maintaining more buffered packets, HSTCP can induce losses in competing flows. In such situations, newly arriving HSTCP flows can have difficulty increasing transmission rate, especially on paths with longer propagation delays. In addition, losses induced on competing TCP flows hurt goodput for TCP users because TCP recovers only linearly. When recovering from periods of sustained heavy congestion, HSTCP tied for third best (10 s average) in our experiments, but the short recovery time can be attributed mainly to the fact that, in comparable situations, HSTCP flows did not reduce transmission rate as much as most other congestion control algorithms. Under low congestion and low initial slow-start threshold, HSTCP achieved maximum transmission rate more slowly (22.4 s average) on long-lived flows than all other alternate congestion control algorithms we studied.

10.1.2.6 HTCP. The HTCP algorithm requires a periodic (250 ms, here) process to monitor flow throughput. HTCP uses standard TCP congestion avoidance procedures for a specified period (1 s, here) after a packet loss. Under congestion, HTCP behaves like standard TCP congestion avoidance. The heavier the congestion, the more time HTCP spends using TCP procedures. When recovering from periods of sustained heavy congestion, HTCP tied for third best (10 s average) in our experiments. Under sporadic losses, HTCP can spend too much time using TCP's linear increase. In our experiments, this trait led HTCP to provide lower goodput than other alternate congestion control algorithms on large files. On the other hand, by adopting standard TCP congestion avoidance procedures following packet loss, HTCP is quite TCP friendly. Under low congestion and low initial slow-start threshold, HTCP achieved maximum transmission rate somewhat slowly (16.6 s average), comparable to BIC, HSTCP and Scalable TCP, but significantly slower than CTCP, FAST and FAST-AT.

10.1.2.7 Scalable TCP. The Scalable TCP algorithm is a small modification of standard TCP congestion avoidance procedures. Scalable TCP increases *cwnd* by a constant on each acknowledgment and decreases *cwnd* by 12.5 % on each loss. In addition, Scalable adopts standard TCP congestion avoidance procedures when *cwnd* is below a low-window threshold (16 packets, here). Under congestion, established Scalable TCP flows do not reduce transmission rate very quickly. By maintaining more buffered packets, Scalable TCP can induce losses in competing flows. In such situations, newly arriving Scalable TCP flows can have difficulty increasing transmission rate, especially on paths with longer propagation delays. In addition, losses induced on competing TCP flows hurt goodput for TCP users because TCP recovers only linearly. When recovering from periods of sustained heavy congestion, Scalable performed fifth best (22.5 s average) in our experiments, but the recovery time can be attributed mainly to the fact that, in comparable situations, Scalable TCP flows did not reduce transmission rate as much as most other congestion control algorithms. Under low congestion and low initial slow-start threshold, Scalable TCP achieved maximum transmission rate somewhat slowly (17.8 s average). In fact, Scalable increased transmission rate very slowly for the first few seconds of long-lived file transfers, which means that Scalable provides a steep increase in transmission rate only for large files.

10.1.3 Recommendations

Under some circumstances, users can benefit from adopting alternate congestion control algorithms to transfer files on the Internet. For that reason, it makes sense to deploy such algorithms into computers attached to the Internet. Of course, the probability appears quite low that a specific user will see benefits on any particular file transfer. Among the alternate congestion control algorithms we studied, CTCP appears to provide the best balance of properties. Under low congestion, CTCP can increase transfer rate relatively quickly when operating in the congestion avoidance phase. Further, CTCP reduces transmission rate relatively quickly in the face of sustained congestion and recovers to the maximum transmission rate quite quickly when congestion eases. CTCP appears relatively friendly to flows using standard TCP congestion avoidance procedures. CTCP, along with most of the other alternate congestion control algorithms we studied, is unlikely to induce large shifts in the macroscopic behavior of the Internet. FAST and

FAST-AT have some appealing properties, especially with respect to achieving maximum transmission rate quickly on high-bandwidth, long-delay paths and recovering quickly from sporadic losses. Unfortunately, when transiting highly congested paths with insufficient buffers to support the flow volume, FAST and FAST-AT can enter an oscillatory regime that could significantly increase loss and retransmission rates. Flows transiting affected areas would take longer to connect and complete and would receive lower goodputs.

10.1.4 Future Work

We studied seven proposed replacement congestion control mechanisms for the Internet. Despite the comprehensive nature of our study, more work remains to be done in at least four directions. First, we limited our study to a bounded set of alternate congestion control algorithms for which we could find empirical data against which to validate our simulations. Researchers have proposed many congestion control algorithms that were not included in our study, so one direction for future work is to consider the behavior of additional algorithms. Of particular interest is CUBIC, which has replaced BIC as the congestion control algorithm enabled by default in Linux.

Second, we have not considered scenarios where multiple alternate congestion control algorithms are mixed together in the same network. Increasing the heterogeneity of algorithms might reveal additional insights about the advantages and disadvantage of the various algorithms, as well as uncover undesirable macroscopic behaviors resulting from such mixtures. Where undesirable behaviors do not appear, then such a study would increase confidence in the safety of deploying alternate congestion control regimes. Of course, conducting such a study would likely require substantial increase in demand for computation resources in order to simulate long enough network operation to accumulate sufficient samples to reveal statistically significant behavioral patterns.

Third, we have not validated our findings against live, controlled experiments configured in GENI or a similar test bed environment. Conducting such a validation would substantially increase confidence in the findings of our study. We intend to undertake such a validation as soon as we can gain access to sufficient resources to support our experiments. In the meantime, we also plan to consider how we might attempt to validate our findings using test environments of smaller scale. One way to approach this may be to make predictions about behaviors we should see replicated even at smaller scale than the network sizes and speeds we simulated.

Fourth, our study revealed various strengths and weaknesses in the congestion control algorithms we investigated. Future researchers could exploit our findings to propose algorithm improvements that compensate for identified weaknesses, while retaining strengths. Further, our general findings may also help other researchers to improve future designs for additional congestion control algorithms.

10.2 Conclusions about Methods

The simulation and modeling studies reported here also enabled us to evaluate each of the modeling and analysis methods we used. Below, we first discuss the use of discrete-event simulation as a technique to model systems at large scale. Subsequently, we evaluate the specific methods we applied to solve each of the five hard problems that we identified in

Sec. 2.4. We than provide some overall recommendations for those seeking to model and analyze large distributed systems. We close with suggestions for future work.

10.2.1 Discrete Event Simulation

Recall that our discrete-event simulation model, MesoNet, was constructed because an existing cellular automaton model proved unable to scale to simulate networks of the size and speed required for this study. The cellular automaton model simply demanded far too many computation resources. What about discrete-event simulation? We demonstrated that MesoNet could feasibly simulate a network operating for one hour at contemporary router speeds while transporting hundreds of thousands of simultaneous flows with a mix of 100 Mbps and 1 Gbps sources and receivers sending flows with sizes ranging from tens of kilobytes to gigabytes. Of course, running such a simulation for an average parameter combination required about 17 ½ days of processing time. In the best case, such a scenario required just over 8 days and in the worst case just over 30 days. The speed of individual processors seems unlikely to improve much in the future. Instead, computer systems will be outfitted with an increasing number of processors that can be used in parallel. Increasing parallelization is a nice match for orthogonal fractional factorial experiment designs (see Sec. 10.2.2.3 below), but each individual experiment run must still be completed within a time budget. We conclude that discrete-event simulation is unlikely to support network simulations much beyond the scale we used in our study. Even if one is willing to wait 60 or 90 days for a single simulation run to complete, the odds seem low that the underlying hardware, operating system, simulation environment and model could run so long without incurring some sort of failure. Researchers are investigating parallel simulation as a means to increase the scale of runs that can be executed, but temporal relationships among elements in network simulations will probably restrict the degree of speedup that can be achieved. We conclude that increasing the scale of a simulated network will likely require a different paradigm, such as fluid-flow or hybrid simulations. We discuss such models further in Appendices A and B.

10.2.2 Scale Reduction Techniques

We adopted five specific techniques to reduce the scale of parameter and response spaces in our experiments. Below, we evaluate each technique in turn.

10.2.2.1 Model Restriction and Parameter Clustering. Restricting model parameters to those germane to this specific study led to substantial reduction in intellectual effort associated with identifying and assigning values to both fixed and variable parameters. Further, reduction in the parameter space lowered the overall computational demand associated with individual experiment runs and with experiment campaigns. Clustering individual parameters into groups, each representing a key factor driving a simulated network, further reduced the intellectual effort needed to parameterize experiments and also to analyze responses and assess the influence of particular input factors. Of course, the reductions associated with restricting and clustering input parameters were insufficient alone to achieve computational tractability for the experiments in this study. Other reductions were required (see Sec. 10.2.2.2 and Sec. 10.2.2.3 below). Further, significant domain expertise was needed to identify reasonable parameter restrictions and

groupings. This may impede studies where such expertise is unavailable. In some cases, substantial simulation executions and data analyses were required to identify parameter reduction decisions that were inappropriate. For example, we required about one week of simulation to identify a gradual trend toward an increasing number of active flows. Discovering this trend led us to introduce connection establishment procedures into the simulation. This finding illuminated the important role that connection establishment procedures play in controlling network-wide congestion.

10.2.2.2 Two-Level Experiment Designs. The largest reduction in computation demand for simulations in this study arose from the simple act of limiting parameter settings to only two levels. Of course, taking this step incurred several drawbacks. First, the experiment designer must select specific values for each level. This requires significant domain expertise. Second, the results obtained for each experiment are robust only over the range defined by the selected level settings. Third, drawing conclusions from a two-level experiment design entails an assumption that a model behaves monotonically within the range defined by the selected settings. To mitigate these restrictions, the study adopted several experiments and varied level settings between experiments. While some may cringe at limiting parameter settings to two levels, we demonstrated in this study that significant insight can be gained even under such a severe restriction.

10.2.2.3 Orthogonal Fractional Factorial Experiment (OFF) Designs. OFF experiment designs enabled us to further reduce computational demand in cases where simulating all combinations of parameter settings proved too costly, even after limiting parameters to only two levels. In general, OFF designs allow an experiment designer to simultaneously vary parameter combinations in a balanced and orthogonal fashion to provide the maximum amount of information given a limit on the affordable number of experiment runs. Since each selected combination of parameters represents an independent simulation run, OFF experiment designs create a suite of simulations that can be executed in parallel, across all available processors, one simulation per processor. Recall, however, that each individual simulation run must still be computationally feasible (as discussed in Sec. 10.2.2.1 above). Another advantage to two-level OFF designs arises from an effective match with a ten-step graphical analysis technique developed at NIST (see Sec. 10.2.4.1 below). Pairing two-level OFF designs with the graphical and analytical techniques used in our study reveals substantial information about system behavior – within the restrictions of two-level designs (as discussed above in Sec. 10.2.2.2). Of course, OFF designs further reduce the potential parameter combinations examined in a particular study. In general, no study can cover all potential parameter combinations. The most typical approach adopted by network researchers entails fixing all parameter settings except one, which is varied across a range of levels. The results of this one-factor-at-a-time approach can produce nice x-y plots, but any resulting conclusions are valid only under a specific combination of fixed parameters. OFF designs provide a principled technique to vary multiple parameter settings simultaneously, which yields more information about overall behavior of a system. Further, OFF designs can identify model errors more readily than one-factor-at-a-time experiments because OFF designs probe a model under a larger variety of parameter combinations.

10.2.2.4 Correlation Analysis and Clustering. Correlation analysis proved an effective technique to reduce the response space that we needed to examine. In one of our sensitivity analyses (see Chapter 4), for example, we showed how 22 potential responses could be reduced to only seven. Assuming availability of a domain expert, correlation results may also aid in model validation. For example, a domain expert should be able to verify whether or not the correlations make sense. A domain expert should also be able to attribute surprising correlations to modeling error or to new insights. We found that a given set of correlation results apply only to the specific range of parameter combinations used to generate the related responses. For example, the correlations identified in Appendix C differ in some ways from the correlations identified in Chapter 4. Examination by a domain expert revealed that both sets of correlations are valid; differences arose from variations in the range of parameters simulated. We conclude that correlation analysis should be applied separately to each suite of experiments where level settings differ.

10.2.2.5 Principal Components Analysis (PCA). We used PCA to complement correlation analysis[3]. PCA aims to identify orthogonal variations, so-called principal components, in response data and to assign weights to indicate the degree to which responses influence each identified principal component. For the models simulated in this study, we found that most variation in response data could be accounted for by the first four principal components in a given analysis. This implies, for example, that we might be able to analyze four responses instead of the 22 used in our sensitivity analysis. Further, a domain expert could compare the findings from a PCA against the findings from a correlation analysis to determine if the two analyses were consistent. This consistency check helps to further validate a model. On the other hand, working with PCA results can be somewhat difficult for a few reasons. First, principal components are abstract linear weighted combinations of responses, so there is no specific domain interpretation behind a given component. An analyst or expert must invest considerable effort to develop a domain interpretation of even the top two or three principal components. In some cases (e.g., Chapter 4), a clear and reasonable interpretation can be achieved. In other cases (e.g., Appendix C), interpretation becomes more difficult. Second, principal components can take on both positive and negative values, which present an analyst with difficulty assigning meaning. In fact, conducting a main-effects analysis of principal components required us to refer to main-effects analyses of raw responses in order to develop an interpretation. Third, PCA sometimes identified components that proved coarser than similar response groupings developed with correlation analysis. When aiming to reduce the response state space, we conclude that PCA provides a reasonable complement to correlation analysis, but domain experts will often find correlation analysis more readily comprehensible than PCA.

10.2.3 Model Validation Techniques

As we discussed in Chapter 2, network researchers typically do not know whether their models are valid. For this reason, we took two steps to increase confidence in the validity of MesoNet. We evaluate each step in turn.

[3] We also used PCA to identify sources of variation in data related to several experiments throughout our study. We evaluate PCA in these applications below in Sec. 10.2.4.3.

10.2.3.1 Sensitivity Analysis. Sensitivity analysis provided a tractable means to investigate the response of MesoNet to various changes in model input parameters. Using sensitivity analyses we were able to find and fix errors in early model formulations and, ultimately, to develop confidence that the model was fit for its intended role in our study. Of course, to conduct such analyses we combined many of the individual methods evaluated here, including two-level OFF experiment design, correlation and principal-components analyses and ten-step graphical analysis. For this reason, our approach to sensitivity analysis inherited the strengths and weaknesses associated with the individual methods. In particular, the two-level design limited the range of our conclusions about model validity. To mitigate that, we conducted a supplementary sensitivity analysis (see Appendix C) that adopted different level settings for each input factor evaluated. We also used preliminary sensitivity analyses to identify factors that did not much influence model responses. We could then reduce the search space in subsequent analyses. We conclude that rigorous sensitivity analysis becomes feasible when using a reduced-scale simulation model and two-level OFF design, combined with judicious choice of factors to vary. Results from sensitivity analyses guided us in designing specific experiments associated with our study. We recommend that campaigns of simulation (or numerical) experiments use only models that have been examined for sensitivity to changes in input parameters.

10.2.3.2 Key Empirical Comparisons. To increase confidence that we had correctly modeled specific congestion control algorithms, we relied on key comparisons between simulation results and empirical results. Such comparisons were facilitated by the existence of published empirical results measured under controlled circumstances in a small ("dumbbell") topology. We were easily able to model the small topology in MesoNet and to simulate the same scenarios and parameter settings used in the empirical studies. Comparing our simulation results with empirical results enabled us to identify errors in modeling several congestion control algorithms. We were also able to correct our models and ensure that we obtained results consistent with empirical results. In this way, we gained confidence in our models of the various congestion control algorithms prior to increasing the scale of topology we simulated. As an added benefit, the empirical study identified default parameter settings adopted by congestion control algorithms. We were able to adopt those settings for our large simulations. For a given study, empirical results may be unavailable, either because the problem under study is not yet implemented or because no one has published empirical results. Where feasible, we recommend that a small experimental configuration be used to generate empirical measurements in order to ground a mathematical model in reality. Preferably, the empirical measurements should be made from implementations developed independently from the models. The empirical measurements should capture key aspects of system behavior on a small scale. When empirical measurements cannot be made available, important aspects of a model may go unconfirmed. From our experiences, the resulting model can contain significant errors that lead to invalid behaviors. We recommend that significant studies endeavor to compare key model aspects with empirical measurements taken at small scale. Any reasonable expense required to obtain empirical measurements will be repaid by enhancing confidence in models used to study large-scale systems.

10.2.4 Data Analysis Methods

Data analysis comprises the third axis of our method; modeling and experiment design comprise the other two axes. We applied and explored four data analysis methods during our study. We evaluate each method below.

10.2.4.1 Ten-Step Graphical Analysis. In support of exploratory analysis, NIST designed a ten-step graphical method (explained in Appendix D), where each step generates an individual plot designed to answer one specific question regarding a data set. We employed this ten-step method as a main part of our sensitivity analysis, where we applied all ten steps to each response. The analysis method is designed to match well with a two-level OFF experiment design. Clearly, for our application, the main-effects plot (D.3) proved most insightful – revealing changes in system responses resulting from changes in input factors. The interaction effects matrix (D.4) also helped to identify that MesoNet was driven primarily by main effects, rather than by two-factor interactions. Other plots proved useful for specific, limited purposes. For example, the ordered data plot (D.1) identified specific combinations of factor settings that produced significant effects on the responses. The multi-factor scatter plot (D.2) summarized how the distribution in responses changed with respect to changes in input factors. Several other plots provided redundant information, which served to confirm related results or to identify analysis errors. For example, the Youden plot (D.6) identified the most significant factors driving particular responses, which could also be ascertained from main effects plots, as well as a number of other plots. The |Effects| plot (D.7) and the cumulative residual standard deviation plot (D.9) helped to visualize whether a linear model could approximate a system's response to input factors. A derived contour plot (D.10) suggested how specific changes in the two main factors influencing a response might drive the response in particular directions. For our purposes, the box plot (D.5) did not provide significant new information. Overall, the ten-step graphical analysis proved quite useful in analyzing a model's sensitivity to changes in input parameter settings. We applied all ten steps to our initial sensitivity analysis. Subsequently, we used only main effects plots and interaction effects matrices, which provided the most important information for our supplementary sensitivity analysis. We recommend applying all ten steps of the graphical analysis during early stages of model development and investigation. The various plots reveal a range of confirming and complementary information that could prove quite insightful. In later stages of analysis, we recommend limiting selected plots to only those necessary to address specific questions of interest.

10.2.4.2 Cluster Analysis. We employed cluster analyses to reveal overall patterns of similarities and differences in multidimensional responses. We sought patterns in behavior among selected congestion control algorithms under conditions, composed of combinations of input parameters. Because parameter combinations varied greatly, we clustered algorithms only with respect to individual conditions. To identify clustering patterns, we needed to characterize differences among conditions. Such characterization required external analyses. Given dendrograms for each condition, along with characterizations of each condition, we were able to identify patterns where selected algorithms clustered together. These clustering patterns provided general relationships among algorithms and congestion. On the other hand, the patterns did not identify

specific relationships among responses that led to the patterns; for this we required more detailed analyses (see 10.2.4.3 below). Cluster analysis appears well suited to identify general patterns where such patterns exist. Further, using cluster analysis, we were able to identify close correspondence in the behaviors of two congestion control algorithms, which enabled us to make informed decisions about more detailed analyses. On the whole, cluster analysis can provide a concise overview of patterns in data sets, but one should not expect cluster analysis alone to provide complete insight about causality. At best, we found that cluster analysis could help to identify aspects of the data that should be given closer scrutiny.

10.2.4.3 Custom Multidimensional Visualizations. Using Dataplot, a statistical scripting language and supporting run-time environment developed at NIST, we could construct custom visualizations designed to explore specific relationships among multidimensional data sets. We developed several custom, multidimensional, visualizations for our study. The resultant representations provided substantial insight into overall system behavior. In fact, custom visualizations provided launching points for many causality analyses (see Sec. 10.2.5). Custom visualization can provide a domain expert with concise and precise information regarding the questions under study. Further, detailed custom visualizations can be subjected to custom summarizations that identify key patterns in experiment data. On the other hand, successful custom visualizations entail collaboration between an expert in statistical visualization and a domain expert, who must iterate over the design and construction of each visualization until a useful result emerges. We found, however, that a few (four or five) well-crafted multidimensional visualizations could be reused to analyze data from most experiments in our study. We recommend custom multidimensional visualizations as a key tool for analysis of data sets for complex systems. Of course, we were fortunate that one study participant was expert in the design and programming of statistical visualizations. Custom visualizations would be difficult to create and apply without access to the necessary expertise.

10.2.4.4 Exploratory Multidimensional Interactive Visualization. Early in our study, we collaborated with visualization experts, who constructed DiVisa, a general purpose system for interactive exploration of multidimensional data. DiVisa enables an analyst to view multiple, related, data simultaneously, while assigning custom visual attributes to represent various dimensions in the data. For example, visual attributes may include color, size and shape. Altogether, DiVisa allowed an analyst to assign up to eight different attributes to data. In using DiVisa, an analyst needs to remember how attributes are assigned. The resulting visualizations proved quite abstract and difficult to interpret. Late in development, and at the request of a domain expert, DiVisa was extended to support display of topological information associated with a given simulation. Since a topology is quite natural for a networking expert, DiVisa acquired increased utility for our study. In fact, given voluminous spatiotemporal information, such as queue sizes changing over time in every router in a network topology, DiVisa could replay the dynamic behavior of a MesoNet simulation, which enabled us to detect unwanted behaviors in various simulations and to adjust model parameter settings as necessary to achieve desired effects. Unlike custom multidimensional visualizations, interactive visualization systems require a domain expert to explore system data and to develop

revealing representations by assigning attributes to data dimensions. We found this quite difficult to do. Perhaps our experience would have been different had we collaborated with an expert in interactive multidimensional visualization. We do not recommend using abstract interactive systems for visualizing multidimensional data unless the resulting displays can be readily related to concepts comprehensible to a domain expert.

10.2.5 Causality Analysis Methods

We chose data analysis methods mainly based on an ability to reveal overall patterns in behavioral data derived from models of large systems. Once significant patterns were revealed, we typically needed to delve into more depth in order to determine root causes for the patterns. In this study, we adopted four main techniques

10.2.5.1 Principal Components Analysis (PCA). We were sometimes able to apply PCA to identify causality underlying variations in response data. For example, when searching for causes in goodput variation among many flow groups under generally low congestion, PCA was able to identify the main drivers as network speed, propagation delay and file size. PCA could also discern cases where congestion control algorithms did contribute to variation in goodput. In general, after using PCA to find the two main principal components, an analyst creates a scatter plot of component one vs. component two, where each point represents a particular parameter combination. Visual inspection may then be used to discriminate clusters, or groupings, of points. Using supplementary analyses, an analyst can characterize common factor settings among points in each grouping. Using this technique, factors underlying variation in the data can become quite explicit. Further, this level of analysis requires little domain expertise and may be accomplished based solely on the factors and settings in a two-level OFF experiment design.

10.2.5.2 Detailed Measurements. Causality exploration sometimes requires detailed spatiotemporal data related to a specific question under investigation (e.g., time series of changing queue sizes in individual routers in a topology). At other times, an analyst may need to peruse aggregated spatiotemporal data (e.g., time series of the distribution of flow states within the network) to determine if a system is behaving as expected. We chose to collect detailed spatiotemporal data as an integral part of our simulation model, MesoNet. In fact, for pattern analysis we generated summary data from the detailed measurements. We found several advantages to this approach. First, we could use the spatiotemporal behavior of our model to determine what range of data to summarize in order to avoid transient startup behavior. Second, we could subject our model to exploratory analyses (see Sec. 10.2.4.4 and Sec. 10.2.5.4). Third, should patterns from data analysis indicate need to further investigate detailed behavior, the data would already exist.[4] Fourth, should other researchers wish to investigate particular questions not addressed in this study, the data would be available for later use. Some drawbacks also arise from

[4] In practice, we made initial guesses about the detailed data we needed to collect. During our study, specific issues revealed the need to collect additional details, such as the temporal posture of the network with respect to the state (e.g., idle, connecting, active) and phase (e.g., initial slow-start, normal congestion avoidance, alternate congestion avoidance) of all flows. So, while one can arrange to collect substantial detailed data during model construction, the need might arise to add additional measurement data during a particular study.

collecting such detailed data. First, collecting extensive spatiotemporal data can require substantial memory within a running simulation. We mitigated this by permitting the simulation to periodically dump the measurement buffer to disk and then reuse the space for additional measurements. Of course, this increases the failure surface of the simulation. In practice, we found that making incremental measurements worked effectively, even when writing results to a file server located on a network.[5] Second, collecting extensive spatiotemporal data requires availability of sufficient disk storage. The experiments in this study required approximately 250 GBytes of storage, so this was easily accommodated. For studies investigating behavior in large distributed systems, we recommend collecting detailed spatiotemporal data regarding every conceivable aspect that might be of interest. Summary data can be created from the details, as required for specific analyses.

10.2.5.3 Scientific Method. Given a pattern of interest revealed by data analyses, we used the scientific method to search for causality. In general, we would posit a hypothesis to explain an observed pattern. We would then suggest detailed behavioral data that should exist to confirm our hypothesis. We would investigate the detailed data and either confirm or refute the hypothesis. If the hypothesis were refuted, then we would develop an alternate hypothesis and repeat. Eventually, we would construct a confirmed hypothesis for a given pattern and that would establish a causal link. This approach often proved time consuming, especially when a domain expert had insufficient insight to formulate a good, initial hypothesis. On the other hand, the rigorous nature of the entire modeling and analysis method we used built increasing insight into system behavior as the study progressed. For this reason, it became easier over time to generate good hypotheses. We were able to establish causality for every pattern of interest to us. Of course, our data is available for use by other researchers who might reach different conclusions than we have about particular causal links. While we would prefer to suggest a more direct process to establish causality, we had little recourse but to adopt the scientific method in cases where PCA could not provide sufficient insights.

10.2.5.4 Exploratory Analysis. While the scientific method provides an orderly approach to establish causality, we also sometimes adopted a more exploratory approach. In general, we would select some related aspects of system behavior and then analyze or interact with time varying data to discover trends. We used this technique, for example, to characterize temporal changes in the distribution of flow states and phases arising from various levels of congestion. We also sometimes used exploratory analysis to develop hypotheses about causes underlying patterns arising from analysis of summary data. For example, we used exploration of temporal variations in congestion window size on specific flows to create the hypothesis that frequent, high amplitude oscillations in the

[5] We did find it necessary to modify the simulation to detect network outages that prevented writing measurements and then to detect resumption of a network path so that the measurements could be written. During times of prolonged network outage the simulation halts while waiting to purge the measurement buffer. In some instances, when the file server crashed, the simulation could not write measurement results because the file handle was stale. Failure to recover from a stale file handle required a simulation to be restarted. Such instances were relatively rare. We were unable to use the SLX checkpoint and restart functions because the SLX product requires that a simulation be reloaded into the same memory addresses. We could not guarantee that this would be the case.

congestion window were responsible for increased loss rates exhibited by FAST and FAST-AT under high spatiotemporal congestion. Of course, to engage in exploratory analysis, one needs to have sufficient data collected under various well-understood conditions.

10.2.6 Experiment Selection Methods

Despite the powerful modeling and analysis techniques available to study behavior in large systems, significant spatiotemporal patterns can be missed completely if the selected experiments do not create the necessary conditions. We relied on three methods to help ensure good coverage of key behaviors. Domain expertise played a crucial role.

10.2.6.1 Factor Analysis. We exploited the sensitivity analysis of MesoNet to identify input factors exerting the largest influence on model response (as described in Sec. 4.6.3.2). This enabled us to concentrate our experiment campaign initially on varying those factors most likely to drive model behavior. We were able to keep other factors fixed. This shows how findings from sensitivity analyses can help to craft effective experiment designs.

10.2.6.2 Domain Expertise. In designing our initial experiment (described in Chapter 6), we relied mainly on domain expertise. A domain expert conceived a temporal scenario that started with typical Web browsing traffic, added heavy congestion in a spatiotemporally localized form and then removed the heavy congestion to allow for resumption of normal Web browsing. A domain expert also introduced three long-lived flows that could provide a detailed view of congestion control behavior. This basic scenario proved well-suited to investigate many operational aspects of congestion control algorithms. Insufficient domain expertise could create a significant impediment to designing insightful experiment scenarios.

10.2.6.3 Incremental Design. We used incremental design to help construct effective and efficient experiment campaigns. In incremental design, results of preceding experiments are used to select parameters and scenarios for subsequent experiments. For example, our first experiment showed that using a large initial slow-start threshold reduced differences among most congestion control algorithms. The initial experiment also identified some distinctive behaviors arising from FAST. Given these factors, we were able to craft our second experiment (see Chapter 7) to examine any changes that resulted from using a low initial slow-start threshold and from including a version of FAST with α tuning. At the same time we were able to determine whether reducing the size and speed of a simulated network would reveal new information. We made these changes while retaining the fundamental scenario from the initial experiment. We used the findings from the second experiment to design a subsequent pair of experiments (discussed in Chapter 8) that examined the influence of initial slow-start threshold in a network supporting standard TCP flows mixed together with flows using an alternate congestion control algorithm. At the same time, domain expertise injected much lower overall congestion and a richer variety of traffic sources into the experiments. Based on findings from these experiments, we decided to design an experiment (reported in Chapter 9) that focused on loss/recovery procedures within the congestion control algorithms, while at the same time increasing

the size and speed of the simulated network. We recommend interweaving domain expertise with sensitivity analysis and incremental design to construct the most effective experiment campaign to fit within the allotted time.

10.2.7 Recommendations

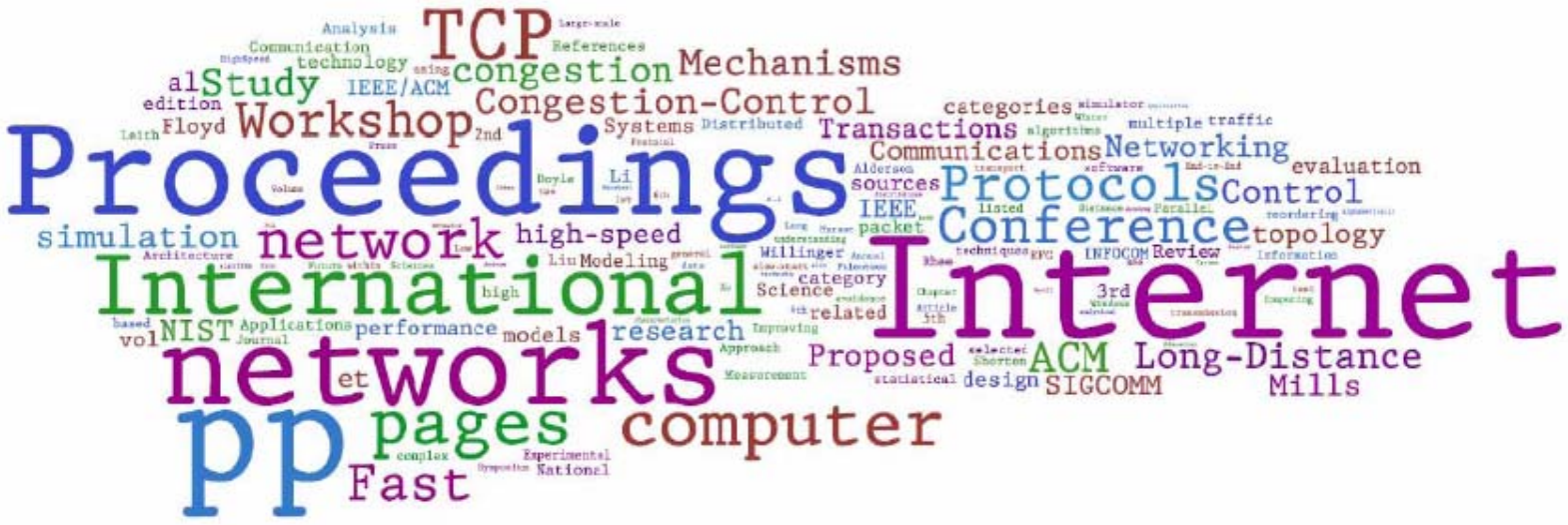
Investigating the behavior of large distributed systems benefits from rigorous application of a coherent set of experiment design and analysis methods. We recommend that such investigations adopt two-level OFF experiment designs, which can facilitate a wide range of compatible analysis methods. Two-level designs allow a system to be investigated under a diverse set of conditions for a reasonable cost. Once overall behavior of a model is understood, later experiments can focus on fewer factors with more levels, as needed to investigate specific aspects of behavior. The quality of these more focused experiments will be improved when placed within the context of a well-understood model. We also recommend that system models (whether numerical or simulation) be subjected to sensitivity analyses in order to understand response to changing input parameters. We advocate the use of incremental design when constructing an experiment campaign. We suggest that discrete-event simulations can capture more detailed aspects of system behavior than would typically be feasible with more abstract models. We found that various scale-reduction methods can lead to tractable simulations for systems of significant size, but there appears to be a limit. We recommend validating key model behaviors against empirical measurements, where feasible. We also identified several useful data analysis methods that can reveal overall behavior in large systems. Some methods, such as cluster analysis, reveal the presence of behavioral patterns without providing much insight into underlying causes. Other methods, such as the NIST-developed ten-step graphical analysis, give better insights. We found that custom multidimensional visualizations can be quite informative, but creating such visualizations requires significant iteration between a domain expert and an expert in statistical visualization. Causality analysis remains largely beyond the grasp of automated analysis methods. Investigating causality required iterative application of the scientific method: a domain expert developed a hypothesis regarding a macroscopic pattern of behavior and then used evidence from detailed spatiotemporal data to confirm or refute the hypothesis. For this reason, we recommend capturing data in as much spatiotemporal detail as a model will permit. Finally, we found that effective use of software for interactively exploring multidimensional data requires visualizations that relate to concrete concepts within the domain under investigation.

10.2.8 Future Work

We suggest three areas for future work on modeling and analysis methods for large distributed systems. First, we recommend investigating methods that enable abstract models to yield improved accuracy. For example, some researchers have developed a hybrid model combining continuous-time logic with discrete events to achieve efficient simulation of system behaviors (see Appendix B). Similarly, we are working to improve the accuracy of fluid-flow models of congestion control algorithms (see Appendix A). Such hybrid or fluid-flow models could be augmented with features necessary to support the experiments adopted in the current study and then the experiments could be repeated. Perhaps one of these abstract models could reveal the same findings at reduced cost.

Second, we suggest investigating approaches to automate design of custom visualizations for multidimensional data. Currently, successful application of custom visualization (such as the detailed response analyses used throughout Chapters 6 through 9) requires substantial collaboration between a domain expert and an expert in statistical visualization. Perhaps the knowledge of a statistical visualization expert can be packaged in an automated form that enables a domain expert to create effective visualizations? Third, we encourage research into automated support for causality analysis. In this study, establishing causality required iteration of the scientific method by a domain expert. This approach was error prone, time consuming and difficult.

Chapter 11 – Bibliography



11 Bibliography

Below we list numbered, bibliographic sources used in performing this study. All listed sources are referenced by number within the report. Sources may be referenced multiple times. Sources listed below are grouped into topical categories. Within each category, we list sources alphabetically by the first author's last name. Multiple sources in the same category by the same author are listed in increasing order of the year of publication. Sources without listed authors are placed at the end of relevant categories and ordered alphabetically by first word in the title.

We order categories, based on their relation to the study, from more general to more specific. We begin with two categories (11.1 and 11.2) related to fundamental design of the Internet and operation of standard TCP. Three subsequent categories (11.3, 11.4 and 11. 5) reflect current understanding about selected aspects of Internet topology, traffic and technology. A sixth category (11.6) identifies selected research regarding data transport in high-speed, high-delay networks. A seventh category (11.7) identifies some proposed replacement congestion control algorithms motivated by evolution of the Internet toward high transmission speeds. An eighth category (11.8) includes selected references to research regarding evaluation of Internet congestion control mechanisms. A ninth category (11.9) encompasses various Internet simulation models and related research. Two additional categories (11.10 and 11.11) cover software and statistical techniques used in performing this study. The final three categories (11.12, 11.13 and 11.14) include general references and references related to empirical Internet test beds and to analytical models for studying networks.

11.1 Fundamentals of Internet Design

[1] Chiu, D.-M. and Jain, R. (1989) "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", *Computer Networks and ISDN Systems*, 17:1, pp. 1-14.

[2] Floyd S. and Fall K. (1999) "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, 7: 4, pp. 458-472.

[3] Saltzer, J., Reed, D., Clark, D. (1984) "End-to-End Arguments in System Design", *ACM Transactions on Computer Systems,* 2:4, pp. 277-288.

11.2 Standard Transmission Control Protocol (TCP)

[4] Cho, S. and Bettati, R. (2006) "Improving quality of service of TCP flows in strictly prioritized network", *Proceedings of the 2nd IASTED International Conference on Advances in Computer Science and Technology*, pp. 239-244.

[5] Comer, D. E. (2001) Computer Networks and Internets with Internet Applications, 3rd edition, Prentice-Hall, 683 pages.

[6] Fall, K. (1999) "TCP Congestion Control; Fast Retransmit; Round-Trip Estimation & Time-out; Silly Window Syndrome", University of California at Berkeley, EECS 122, Lecture 22, April 13, 1999.

[7] Floyd, S. (2004) Limited Slow-Start for TCP with Large Congestion Windows, RFC 3742, The Internet Society, 7 pages.

[8] Peterson, L. L. and Davie, B. S. (2007) Computer Networks: A Systems Approach, 4th edition, Elsevier, 806 pages.

[9] Stevens, W. R. (1994) TCP/IP Illustrated, Volume 1: the Protocols, 1st edition, Addison-Wesley Professional, 600 pages.
[10] Stevens W. (1997) TCP Slow Start, Congestion Avoidance, Fast Retransmit, RFC2001, 6 pages.
[11] Microsoft® Help and Support. (2006) "TCP/IP and NBT configuration parameters for Windows XP", Article ID 314053, Revision 3.2, http://support.microsoft.com/kb/314053.

11.3 Internet Topology

[12] Albert, R. and Barabasi, A.-L. (2002) "Statistical mechanics of complex networks", *Reviews of Modern Physics*, 74:1, pp. 47-97.
[13] Alderson, D., Li, L., Willinger, W. and Doyle J.C. (2005) "Understanding Internet Topology: Principles, Models, and Validation", *IEEE/ACM Transactions on Networking*, 13:6, pp. 1205-1218.
[14] Alderson D., Willinger W., Li L. and Doyle J. (2005) "An Optimization-Based Approach to Modeling Internet Topology", Chapter 6 in Telecommunications Planning: Innovations in Pricing, Network Design and Management. S. Raghavan and G. Anandlingham, eds. Springer, 388 pages.
[15] Barabasi, A.-L. and Bonabeau, E. (2003) "Scale-Free Networks", *Scientific American*, 288:5, pp. 50-59.
[16] Barrat, A., Alvarex-Hamelin, I., Dall'Asta, L. Vazquez, A., and Vespignani, A. (2006) "Sampling of Networks with Traceroute-Like Probes", *Complexus*, 2006:3, pp. 83-96.
[17] Burch, H. and Cheswick, F. (1999) "Mapping the Internet", *Computer*, 32:4, pp. 97-98.
[18] Carmi, S., Havlin, S., Kirkpatrick, S., Shavitt, Y. and Shir, E. (2007) "A model of Internet topology using *k*-shell decomposition", *Proceedings of the National Academy of Sciences*, 104:27, pp. 11150-11154.
[19] Chang, H., Jamin, S. and Willinger, W. (2006) "To Peer or Not to Peer: Modeling the Evolution of the Internet's AS-Level Topology", *Proceedings of the 25th IEEE International Conference on Computer Communications* (INFOCOM 2006), pp. 1-12.
[20] Chen B., Xu, Y., Hu J., and Zhang, L. (2008) "Modeling and Analysis Traffic Flows of Peer-to-Peer Application", *Proceedings of the 3rd International Conference on Innovative Computing Information and Control*, pp. 383-386.
[21] Doyle, J.C., Alderson D. L., Li, L., Low, S. Roughan, M., Shalunov, S. Tanaka, R and Willinger, W. (2005) "The 'robust yet fragile' nature of the Internet", *Proceedings of the National Academy of Sciences*, 102:41, pp. 14497-14502.
[22] Faloutsos, F., Faloutsos, P and Faloutsos, C. (1999) "On Power-Law Relationships of the Internet Topology", *Proceedings of SIGCOMM'99*, ACM, pp. 251-262.
[23] Finsterwalder, R. (2000) "A Generic Client/Server Architecture for Distributed Web-Based Simulation Experimentation", *Proceedings of IEEE International Symposium on Computer-Aided Control System Design*, pp. 185-189.
[24] Govindan, R. and Tangmunarunkit, H. (2000) "Heuristics for Internet map discovery", *Proceedings of the 19th IEEE International Conference on Computer Communications* (INFOCOM 2000), vol. 3, pp. 1371-1380.

[25] Kratz, M., Ackerman, M., Hanss, T. and Corbato, S. (2001) "NGI and Internet2: Accelerating the Creation of Tomorrow's Internet", in MEDINFO 2001, IOS Press, Vol. 84, pp. 28-32.
[26] Li, L., Alderson, D., Willinger, W. and Doyle, J. (2004) "A First-Principles Approach to Understanding the Internet's Router-level Topology", *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM, pp. 3-14.
[27] Magoni, D. and Pansiot J.-J. (2002) "Internet topology modeler based on map sampling", *Proceedings of the 7th International Symposium on Computers and Communications* (ISCC 2002), pp. 1021-1027.
[28] Mahadevan, P., Hubble, C., Krioukov, D., Huffaker, B., and Vahdat, A. (2007) "Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies", *ACM SIGCOMM Computer Communications Review*, 37:4, pp. 325-336.
[29] Newman, M. E. J. (2003) "The Structure and Function of Complex Networks", *SIAM Review*, 45:5, pp. 167-256.
[30] Paxson, V. (1996) "End-to-End Routing Behavior in the Internet", *IEEE/ACM Transactions on Networking*, 4:5, pp. 601-615.
[31] Rabbat, M., Nowak, R. and Coates, M. (2004) "Multiple source, multiple destination network tomography", *Proceedings of the 23th IEEE International Conference on Computer Communications* (INFOCOM 2004), vol. 3, pp. 1628-1639.
[32] Strogatz, S. H. (2001) "Exploring complex networks", *NATURE*, 410:8, pp. 268-276.

11.4 Internet Traffic Characteristics

[33] Crovella M., Taqqu M., and Bestavros. A. (1998) "Heavy-tailed probability distributions in the world wide web", Chapter 1 in A Practical Guide to Heavy Tails, Chapman & Hall, pp. 3-26.
[34] Downey, A. (2001) "Evidence for long-tailed distributions in the Internet", *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, pp. 229-241.
[35] Figureiredo D. R., Liu B., Misra V., and Towsley D. (2002) "On the autocorrelation structure of TCP traffic", *Computer Networks: The International J. Computer Telecommunication Networking* 40 (3), pp. 339-361.
[36] Veres A., Kenesi Zs., Moln Ar S., and Vattay G. (2000) "On the Propagation of Long-Range Dependence in the Internet", *Proceedings of ACM SIGCOMM*.

11.5 Internet Technology

[37] Appenzeller G., Keslassy, I. and McKeown, N. (2004) "Sizing Router Buffers", *Proceedings of ACM SIGCOMM*.
[38] Bellardo, J. and Savage, S. (2002) "Measuring packet reordering", *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, pp. 97-105.
[39] Bennett, J., Partridge, C., and Shectman, N. (1999) "Packet Reordering in Not Pathological Network Behavior", *IEEE/ACM Transactions on Networking*, 7:6, pp. 789-798.
[40] Bush R. and Meyer D. (2003) Some internet architectural guidelines and philosophy, RFC 3439.

[41] Engbersen, A. P. J. (2003) "Prizma switch technology", *IBM Journal of Research and Development*, 47:2/3, 195-210.
[42] Kandula, S., Katabi, D., Sinha, S. and Berger A. (2007) "Dynamic Load Balancing Without Packet Reordering", ACM SIGCOMM Computer Communication Review, 37:2, pp. 53-62.
[43] Liu, H. (2002) "A trace driven study of packet level parallelism", *Proceedings of the IEEE International Conference on Communications* (ICC 2002), vol. 4, pp. 2191-2195.
[44] Villamizar C. and Song C. (1994) High performance tcp in ansnet. *ACM Computer Communications Review*, 24(5):45-60.
[45] Wang, Y., Lu, G., and Li, X. (2004) "A Study of Internet Packet Reordering", in Information Networking, LNCS 3090, pp. 350-359.

11.6 Research on Data Transport in High Speed, Long Delay Networks

[46] Bhandarkar, S., Jain, S. and Reddy, A. (2005) "Improving TCP Performance in High Bandwidth High RTT Links Using Layered Congestion Control", *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks*.
[47] Bresnahan, J., Link, M., Khanna, G., Imani, Z., Kettimuthu, R. and Foster, I. (2007) "Globus GridFTP: what's new in 2007", *Proceedings of the 1st International Conference on Networks for grid applications*, article 19.
[48] Gu, Y. and Grossmon, R. (2004) "UDT: An Application Level Transport Protocol for Grid Computing", *Proceedings of the 2nd International Workshop on Protocols for Fast Long-Distance Networks*.
[49] Guojun, J. (2004) "Packet drop avoidance for high-speed network transmission protocol", *Proceedings of the 2nd International Conference on Information Technology: Research and Education*, pp. 1-5.
[50] Leith, D. and Shorten, R. (2008) "Next Generation TCP: Open Questions", *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks*.
[51] Marcondes, C., Persson, A., Sanadidi, M., Gerla, M., Shimonishi, H., Hama, T. and Murase, T. (2006) "Inline Path Characteristic Estimation to Improve TCP Performance in High Bandwidth-Delay Networks", *Proceedings of the 4th International Workshop on Protocols for Fast Long-Distance Networks*.

11.7 Proposed Replacement Congestion Control Mechanisms for the Internet

[52] Floyd, S. (2003) HighSpeed TCP for Large Congestion Windows, RFC 3649, The Internet Society, 34 pages.
[53] Kelly, T. (2003) "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 83-91.
[54] Leith, D. and Shorten, R. (2004) "H-TCP: TCP for High-speed and Long-distance Networks", *Proceedings of the 2nd International Workshop on Protocols for Fast Long-Distance Networks*, 16 pages.
[55] Liu, S., Basar, T. and Srikant, R. (2006) "TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks", *Proceedings of the 1st*

International Conference on Performance Evaluation Methodologies and Tools, ACM, Vol. 180, article 55, 13 pages.
[56] Rhee, I. and Xu, L. (2005) "CUBIC: A New TCP-Friendly High-Speed TCP Variant", *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks*, 6 pages.
[57] Shimonish, H., Hama, T. and Murase, T. (2006) "TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm", *Proceedings of the 4th International Workshop on Protocols for Fast Long-Distance Networks*, 5 pages.
[58] Tan, K., Song, J., Zhang, Q. and Sridharan, M. (2006) "A Compound TCP Approach for High-Speed and Long Distance Networks", *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2006), pp. 1-12.
[59] Vacirca, F., Baiocchi A. and Castellani, A. (2007) "YeAH-TCP: Yet Another Highspeed TCP", *Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks*, pp. 37-42.
[60] Wei, D. X., Jin, C., Low, S. H. and Hegde, S. (2006) "FAST TCP: Motivation, Architecture, Algorithms, Performance", *IEEE/ACM Transactions on Networking*, 14:6, pp. 1246-1259.
[61] Xu, L., Harfoush, K. and Rhee, I. (2004) "Binary Increase Congestion Control for Fast, Long Distance Networks", *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2004), vol. 4, pp. 2514- 2524.

11.8 Evaluating Internet Congestion Control Mechanisms

[62] Andrew, L., Marcondes, C., Floyd, S., Dunn, L., Gullier, R., Gang, W., Eggert, L. Ha, S., and Rhee, I. (2008) "Towards a Common TCP Evaluation Suite", *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks.*
[63] Floyd, S. (2008) Metrics for the Evaluation of Congestion Control Mechanisms, RFC 5166.
[64] Jackson, T. and Smith, P. (2008) "Building a Network Simulation Model of the TeraGrid Network", *Proceedings of TeraGrid'08.*
[65] Lee, G., Lachlan, A., Tang, A. and Low, S. (2007) "WAN-in-Lab: Motivation, Deployment and Experiments", *Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks.*
[66] Leith, D., Andrew, L., Quetchenbach, T., Shorten, R., Lavi, K. (2008) "Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms", *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks.*
[67] Li, Y.-T., Leith, D. and Shorten, R.N. (2007) "Experimental Evaluation of TCP Protocols for High-Speed Networks", *IEEE/ACM Transactions on Networking*, 15:5, pp. 1109-1122.
[68] Shimonishisi, H., Sanadidi, M. and Murase, T. (2007) "Assessing Interactions among Legacy and High-Speed TCP Protocols", *Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks.*

11.9 Internet Simulation and Models

[69] Cowie, J., Liu, H. Liu, J., Nicol, D. and Ogielski, A. (1999) "Towards Realistic Million-Node Internet Simulations", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 28- July 1, 1999, Las Vegas, Nevada.

[70] Floyd, S. and Kohler, E. (2003) "Internet Research Needs Better Models", *ACM SIGCOMM Computer Communication Review*, 33:1, pp. 29-34.

[71] Lee, J., Bohacek, S., Hespanha, J. and Obraczka, K. (2007) "Modeling Communication Networks with Hybrid Systems", *IEEE/ACM Transactions on Networking*, 15:3, pp. 630-643.

[72] Paxson, V. and Floyd, S. (1997) "Why We Don't Know How To Simulation The Internet", *Proceedings of the 29th Winter Simulation Conference*, pp. 1027-1044.

[73] Yi, Y. and Shakkottai, S. (2007) "FluNet: A hybrid internet simulator for fast queue regimes", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51:18, Elsevier North-Holland, pp. 4919-4937.

[74] Yuan, J. and Mills, K. (2006) "Simulating Timescale Dynamics of Network Traffic Using Homogeneous Modeling", *The NIST Journal of Research*, Volume 111, No. 3, May-June 2006, pp. 227-242.

[75] Yuan, J. and Mills, K. (2005) "Monitoring the Macroscopic Effect of DDoS Flooding Attacks", *IEEE Transactions on Dependable and Secure Computing*, Volume 2, No. 4, October-December 2005, pp. 324-335.

[76] Zeng. X., Bagrodia, R. and Gerla, M. (1998) "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks", *Proceedings of the 12th Workshop on Parallel and Distributed Simulations* (PADS '98), pp. 154-161. http://pcl.cs.ucla.edu/projects/glomosim/.

[77] Georgia Tech Network Simulator (GTNetS) http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[78] J-Sim http://www.j-sim.org/.

[79] Network Simulator – ns-2 http://www.isi.edu/nsnam/ns/.

[80] ns-3 Network Simulator http://www.nsnam.org/.

[81] OMNeT++ http://www.omnetpp.org/.

[82] Parallel Distributed NS (PDNS) http://www.cc.gatech.edu/computing/compass/pdns/index.html.

[83] Scalable Simulation Framework Network Models (SSFNet) http://www.ssfnet.org/homePage.html.

11.10 Supporting Software Tools

[84] Henriksen, J.O. (1996) "An introduction to SLX™", *Proceedings of the 28th Winter Simulation Conference*, pp.468-475.

[85] Henriksen, J.O. (2000) "SLX: the X is for extensibility", *Proceedings of the 32nd Winter Simulation Conference*, pp.183-190.

[86] Houard, C. and Hagedorn, J. (2008) DiVisa: Multi-Dimensional Visualization, Publicly Available Software from NIST. (see http://math.nist.gov/mcsd/savg/software/divisa/)
[87] The MathWorks. (2008) MATLAB® Statistics Toolbox™ 7 Users' Guide, 1749 pages.

11.11 Supporting Statistical Techniques

[88] Antony, J. and Capon, N. (1998) "Teaching Experimental Design Techniques to Industrial Engineers", *International Journal of Engineering Education*, 14:5, pp. 335-343.
[89] Box, G., Hunter, J. and Hunter, W. (2005) Statistics for Experimenters, 2nd edition, Wiley, 639 pages.
[90] Cohen, P., Cohen, J., West, S., and Aiken, L. (2002) Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, 3rd edition, Lawrence Erlbaum, 736 pages.
[91] Croarkin, C. Tobias, P., Filliben, J., Hembree, B. Guthurie W., Trutna, L., and Prins, J. (2006) "Grubbs' Test for Outliners", NIST/SEMATECH e-Handbook of Statistical Methods, Chapter 1, Explore, paragraph 1.3.5.17. (see also http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm)
[92] Filliben, J. (1984) DATAPLOT – Introduction and Overview, NBS Special Publication 667, 131 pages. (See also http://www.itl.nist.gov/div898/software/dataplot/homepage.htm for current software versions and related documentation).
[93] Fodor, I.K. (2002) A Survey of Dimension Reduction Techniques. Lawrence Livermore National Laboratory Technical Report no. UCRL-ID-148494.
[94] Law, A. and Kelton, W. D. (2000) Simulation Modeling and Analysis, 3rd edition, McGraw Hill,
[95] Mawdsley, J., Su, Y.J., Faber, K., and Bernecki, T. (2001) "Optimization of small-particle plasma-sprayed alumina coatings using designed experiments", *Materials Science and Engineering A*, 308:1, pp. 189-199.
[96] Ramsay, J. and Sliverman, B. (1997) Functional Data Analysis, Springer-Verlag, 310 pages.

11.12 Empirical Internet Test Beds

[97] Arora, P., Wang, Y. and Rhee, I. (2009) "Netset: Automating Network Performance Evaluation", *Proceedings of the 7th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports*, pages 25-30, Tokyo, Japan, May 2009.
[98] Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K. and Lepreau, J. (2008) "Large-scale Virtualization in the Emulab Network Testbed", *Proceedings of the 2008 USENIX Annual Technical Conference*, pages 113–128, Boston, MA, June 2008
[99] White, B., Lepreau, J. Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C. and Joglekar, A. (2002) "An Integrated Experimental Environment for Distributed Systems and Networks", *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 255-270, Boston, MA, December 2002.

[100] GENI – Exploring Networks of the Future
http://www.geni.net/
[101] PlanetLab – An open platform for developing, deploying, and accessing planetary-scale services
http://www.planet-lab.org/support

11.13 General Related References

[102] Bieberstein N., et al. (2005) Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap, IBM Press, 272 pages.
[103] Gardner G., Baumstarck J., Sego P. (2005) Industry Best Practices in Achieving Service Oriented Architecture (SOA), Association for Enterprise Integration, April 22, 31 pages.
[104] National Research Council, Committee on Network Science for Future Army Applications (2005) Network Science, the National Academies Press, 2005.
[105] National Institute for Standards and Technology. (2006) Measurement Science for Complex Information Systems, Innovations in Measurement Science project.
http://www.nist.gov/itl/antd/emergent_behavior.cfm

11.14 Analytical Internet Models

[106] Baccelli, F. and Hong, D. (2005) "Interaction of TCP flows as billiards", *IEEE/ACM Transactions on Networking*, 13(4):841–853.
[107] Baccelli, F., McDonald, D. and Reynier, J. (2002) "A mean-field model for multiple TCP connections through a buffer implementing RED", *Performance Evaluation*, 49(1/4):77–97.
[108] Baccelli, F. and Hong, D. (2003) "Flow level simulation of large ip networks", *Proceedings of INFOCOM*, volume 3, pp. 1911– 1921.
[109] Blanc, A., Avrachenkov, K. and Collange, D. (2009) "Comparing some high speed TCP versions under bernoulli losses, *Proceedings of the International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNet 2009)*, pp. 59-64.
[110] Dumas, V., Guillemin, F. and Robert, P. (2002) "A markovian analysis of additive-increase multiplicative-decrease (AIMD) algorithms", *Advances in Applied Probability*, pp. 85–111.
[111] Fredj, S. B., Bonald, T., Proutiere, A., Régnié, G. and Roberts, J. W. (2001) "Statistical bandwidth sharing: a study of congestion at flow level", *SIGCOMM Comput. Commun. Rev.*, 31(4):111–122.
[112] Genin, D. and Marbukh, V. (2009) "Bursty fluid approximation of TCP for modeling internet congestion at the flow level", *Proceedings of the 47th Annual Allerton Conference on Communication, Control and Computing*.
[113] Geurts, P., El Khayat, I. and Leduc, G. (2006) "On the accuracy of analytical models of TCP throughput", volume 3976 Springer.
[114] Goel, P. and Gautam, N. (2006) "On using fluid flow models for performance analysis of computer networks", *Proceedings of the 8th INFORMS Telecommunications Conference*.
[115] Jiang, H. and Dovrolis, C. (2004) "The origin of TCP traffic burstiness in some time scales", *Proceedings of IEEE INFOCOM*.

[116] Maulik, K. and Zwart, B. (2009) “An extenion of the square root law of TCP”, *Annals of Operations Research*, 170(1):217–232.
[117] Padhye, J., Towsley, D., Firoiu, V. and Kurose, J. (2000) “Modeling TCP Reno performance: a simple model and its empirical validation”, *IEEE/ACM Transactions on Networking*, 8(2):133–145.
[118] Paxson, V. and Floyd, S. (1995) “Wide-area traffic: The failure of poisson modeling”, *IEEE/ACM Transactions on Networking*, 3:226–244.
[119] Presti, F., Liu, Y., Misra, V., Towsley, D. and Gu, Y. (2003) “Fluid models and solutions for large-scale IP networks”, *SIGMETRICS*.
[120] Raina, G. and Wischik, D. (2005) “Buffer sizes for large multiplexers: TCP queuing theory and instability analysis”, *Next Generation Internet Networks*.
[121] Sondhi, M.M., Anick, D., Mitra, D. (1980) “Stochastic theory of a data handling system with multiple sources”, *Proceedings of the International Conference on Communications*.
[122] Srikant, R. (2001) Mathematics of Internet congestion control. Wiley.
[123] Srikant, R. and Shakkottai, S. (2002) “How good are deterministic fluid models of Internet congestion control”, *Proceedings of INFOCOMM*.
[124] Srikant, R., Ying, L. and Dullerud, G. (2006) “Global stability of Internet congestion controllers with heterogeneous delays”, *IEEE/ACM Transactions on Networking*, 14(3):579–591.
[125] Tan, D., Kelly, F. and Maulloo, A. (1998) “Rate control for communication networks: shadow prices, proportional fairness and stability”, *J. Oper. Res. Soc.*, 49:237–252.
[126] Tan, D. and Johari, R. (2001) “End-to-end congestion control for the Internet: delays and stability”, *IEEE/ACM Transactions on Networking*, 9(6):818–832.
[127] Towsley, D., Misra, V. and Gong, W. (2000) “Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED”, *Proceedings of SIGCOMM*.

Appendix A – Understanding, Improving and Applying Fluid-Flow Models

Appendix A Understanding, Improving and Applying Fluid-Flow Models

In order to insure efficient and stable operation of the Internet it is important to be able to estimate network performance characteristics under TCP traffic, which today constitutes the bulk of Internet data freight. The main obstacle to achieving this goal is posed by the dynamic nature of TCP congestion control: very complex collective behavior arises as a result of interactions between congestion control algorithms of concurrent flows. A good analogy that we will return to below is that of gas or fluid dynamics in which relatively simple interactions between molecules comprising the substance lead to familiar but complex bulk properties such as viscosity, temperature, and pressure, which are not easily understood or computed from the microscopic, molecular description.

As in fluid dynamics there are two possible approaches to gauging the properties of an aggregate network — by simulating the microscopic dynamics of transmission and transit of individual packets or by studying heuristically derived high-level approximations describing the packet traffic as a kind of a continuous substance flowing along the network links. The advantage of the former approach (adopted in the main body of this study) is that it yields very detailed information that is easily compared against traces collected on experimental test beds, for example, for verification. On the other hand, simulating a network of a realistic size over a large number of network parameter combinations may prove computationally infeasible. The fluid approximation models by comparison have very modest resource demands, although they are also less detailed. Fluid approximation models have another advantage over simulations in that they can sometimes give precise mathematical relationships between performance and network parameters, which can then be used as a guide in design of future networks and protocols as well as to improve the performance of current systems.

We begin in Sec. A.1 by introducing fluid-flow approximation models for TCP Reno flows and then we discuss the utility and limitations of such models. In Sec. A.2, we use fluid-flow approximation to develop response functions for TCP Reno, as well as CUBIC [56] and Compound [58] TCP and then compare the estimated equilibrium throughput of these alternatives. We close in Sec. A.3, where we outline future work related to fluid-flow approximation of Internet congestion control algorithms.

A.1 Fluid-flow Approximation Models

How would the Internet appear when visualizing packets as points moving along links and through routers? With hundreds of thousands of packets crossing a typical router every second they would appear to be an uninterrupted blur of motion, as if a fluid were flowing through pipes rather than a series of discrete packets flowing along links. This is the basic idea of fluid approximation: if the number of packets in the network is very large and they are moving very fast then the packet traffic will be well approximated by an abstract continuous stream.

Although each individual TCP flow behaves deterministically, aggregate dynamics, for a network of any significant size, will appear as nearly random. A good physical analogy is molecular dynamics in a volume of gas: while each molecule obeys simple Newtonian laws of motion, their collective behavior is essentially stochastic.

Describing the paths of individual molecules is a hopeless and ultimately useless endeavor; on the other hand, bulk properties such as pressure and temperature are neatly connected by the ideal gas equations. Similarly, one may hope that for a large collection of TCP flows the aggregate throughput may be related to the round-trip delay, capacity and buffer size in a simple way. Thus, the ultimate goal of fluid-flow approximation models is to develop a kind of TCP network thermodynamics.

We introduce fluid approximation by briefly describing the congestion window and throughput dynamics in the case of an isolated flow. To keep the presentation as simple and clear as possible we will discuss the most basic version of TCP Reno preserving only the most salient features of the protocol. That is, by TCP we will henceforth mean TCP Reno without selective acknowledgment, fast retransmit and fast recovery. Mathematical models incorporating these advanced features of TCP have been studied elsewhere [127]. For an isolated flow there can only be one bottleneck router along its path and so the rest of the path contributes only in terms of propagation delay. We assume further that the bottleneck router is positioned immediately in front of the source on the outgoing link. The general case is not substantially different. Let the router capacity at the bottleneck be denoted by C packets per second (pps), let its buffer size be B packets and the round-trip propagation delay T sec. If the variation in the round-trip time due to queuing at the router is negligible, the size of the congestion window in the congestion avoidance phase is approximated by the differential equation

$$\frac{dW}{dt} = \frac{1}{T} - \frac{1}{2}W(t)P(t), \tag{1}$$

where $W(t)$ interpolates the discrete congestion window size and $P(t)$ is a sum of delta functions $P(t) = \sum \delta(t - t_j)$, with t_j corresponding to moments of congestion window reduction due to detection of packet losses. In the case of an isolated connection the sequence t_j is periodic and can be computed explicitly. Note, also, that if the buffer is large T must be replaced with the equilibrium round-trip time that includes the equilibrium queuing delay.

If $B \approx CT$ and the variation in queuing delay is significant, then queue length has to be explicitly included in the model. Equation (1) then becomes

$$\frac{dW}{dt}(t) = \frac{1}{T + Q(t-T)/X} - \frac{1}{2}W(t)P(t) \tag{2}$$

$$\frac{dQ}{dt}(t) = \left(\frac{W(t)}{T + Q(t-T)/X} - X\right)\chi_{[0,B]}(Q(t)), \tag{3}$$

where $Q(t)$ is the buffer queue length at the router and $\chi_{[0,B]}(x)$ is the characteristic function of $[0,B]$.

While equations (1) and (2) describe the evolution of the congestion window, the quantity of real interest is usually the transmission rate. It is not hard to see that the rate at which the TCP sliding window[1] advances, and hence the transmission rate, is equal to $W(t)$ divided by the total round-trip delay $T+Q(t)/C$. So long as queuing delay is small relative to the total round-trip propagation time, the transmission rate, $X(t)$, can be approximated by $W(t)/T$. Dividing (1) through by T we have

$$\frac{dX}{dt} = \frac{1}{T^2} - \frac{1}{2}\sum X(t)P(t). \qquad (4)$$

Note that, as expected, transmission rate attains its maximum value C at $W=CT$ and stays at this value even as W continues to increase. This means that when the buffer size is comparable with the bandwidth-delay product $X(t)$ does not carry enough information to reconstruct the system state completely because it is impossible to compute $X(t)$ following a congestion window reduction given only the value of $X(t)$ before the reduction. In this case, throughput has to be computed by solving (2) first.

When considering the case of multiple flows, the complexity of the problem increases in two ways. First, as the number of flows increases, the simple periodic congestion window dynamics described above breaks down because packet loss now depends on the collective behavior of all flows. Consequently, the evolution of congestion windows becomes more and more complex and chaotic. The difficulty of the problem also increases with increasing complexity of the network structure. As the web of interactions among flows becomes increasingly complicated so does the global dynamics of the network. This topological aspect of the problem has so far received comparatively little mathematical treatment, mainly because describing a large number of flows in a simple topology is already a formidable challenge. Most mathematical models in current literature treat very simple network topologies. The one most often considered is that of a single bottleneck link shared by a large number of identical flows. While this simple topology may not exhibit the full range of dynamics that may exist in more complicated networks, it is, nevertheless, an important special case both practically and theoretically. We turn to this case next.

A.1.1 Modeling Many Flows on One Link

We briefly outline the derivation of the fluid approximation for the one-link-many-flows case. Let the number of flows N be large and let capacity and buffer size of the router scale with N as NC and N^aB, $0 \le a \le 1$, respectively. Then the number of packets passing through the shared link per unit time will be large, satisfying the intuitive condition necessary for the fluid approximation to hold. While the system as a whole will remain deterministic as N grows, the packet loss process will be increasingly well approximated by a stochastic one. Thus it makes sense to model evolution of congestion windows with

[1] The sliding window is the interval of packet numbers corresponding to already sent but not yet acknowledged packets. The size of the sliding window is bounded by the size of the congestion window; its right edge advances when a new packet is sent and its left when a previously sent packet is acknowledged [9].

a corresponding collection of random processes. Let $W^N(t)$ be the random process describing aggregate congestion window size when the number of concurrent flows is N

$$W^N(t) = \sum_{i=1}^{N} W_i^N(t), \tag{5}$$

where $W_i^N(t)$ describes the congestion window size of flow i at time t. Since the flows are identical, it is reasonable to assume that the $W_i^N(t)$ are identical random processes. We will also assume that the $W_i^N(t)$ become independent as N tends to infinity. Applying the law of large numbers we have that $1/N\, W^N(t)$ converges to some deterministic process $w(t)$ as N goes to infinity. The deterministic process $w(t)$ is the *fluid approximation* of $W^N(t)$.

Considering the simpler case of small buffers $a<1$ first [120], we have from (1) and (5)

$$\frac{dW^N}{dt}(t) = \frac{N}{T} - \frac{1}{2}\sum_{i=1}^{N} W_i^N(t) P_i^N(t). \tag{6}$$

Note that $P_i^N(t)$ are now coupled random variables. Dividing through by N and letting N go to infinity we obtain the governing equation for $w(t)$

$$\frac{dw}{dt}(t) = \frac{1}{T} - \frac{1}{2} w(t)\, p(t), \tag{7}$$

where $p(t) = \lim_{N\to\infty} 1/N \sum_{i=1}^{N} P_i(t)$ (assuming the limit exists) is the aggregate loss density function. If the variation in round-trip time is small, $p(t)$ can be assumed to depend only on $w(t–T)$ (ignoring the rest of the network parameters for the moment). Furthermore, the number of packets lost per unit time can be approximated as $p(w(t–T))w(t–T)/T$, where $p(w)$ now stands for the probability that an arriving packet will be dropped due to buffer overflow when the aggregate congestion window size is w. Note that because of the round-trip delay the source detects packet loss only after a round-trip time T so that p depends on the transmission rate T seconds in the past. Formula (7) can be approximated as

$$\frac{dw}{dt}(t) = \frac{1}{T} - \frac{1}{2}\frac{w(t)\, p(w(t-T))w(t-T)}{T}. \tag{8}$$

If the buffer is large (a=1) then $p(w)$ will depend not only on $w(t–T)$ but also on buffer content $q(t–T)$. Using equations (2) and (5) gives

$$\frac{dw}{dt}(t) = \frac{1}{T + q(t-T)/C} - \frac{1}{2}\frac{w(t)\,p(w(t-T), q(t-T))w(t-T)}{T + q(t)/C}$$

$$\frac{dq}{dt}(t) = \left(\frac{w(t)}{T + q(t-T)/C} - C\right)\chi_{[0,B]}(q(t)), \qquad (9)$$

where we assume that the per flow buffer content $q(t) = \lim_{N\to\infty} Q^N(t)/N$ converges to a continuous deterministic variable.

A.1.2 Utility of Fluid-flow Approximation Models

There are two main types of results that can be obtained from models such as equations (8) and (9). First, one can deduce existence and uniqueness of the equilibrium, and its dependence on network parameters. Secondly, one can analyze stability properties of the equilibrium solution and the dependence of the equilibrium solution on the network parameters. Both types of results have clear practical applications, the first for optimal resource utilization and the second for stable network design. We will give a brief survey of both types of results.

By setting the right side of (8) and (9) to 0 we can obtain the expression for the equilibrium mean congestion window size w^*

$$w^* = \sqrt{\frac{2}{p^*}}, \qquad (10)$$

where $p^*=p(w^*)$ or $p(w^*,q^*)$ respectively. This shows that an equilibrium exists, since $p^*\neq 0$ is satisfied. Making the natural assumption that $p(w)$ is monotonically increasing in w, equation (10) also shows that the equilibrium is unique [122]. Formula (10) is close to experimental measurements [117] and also agrees with first principles derivations [117]. Unfortunately, the dependence of w^* on T and B is hidden inside the unknown function $p(w)$, which limits the usefulness of (10) for making a priori throughput estimates. Simple forms for $p(w)$ such as $\left(\frac{w}{CT}\right)^B$ corresponding to buffer overflow probability in an M/M/1/B queuing system have been found to be far from accurate [112-113]. Recently, an alternative packet loss model based on the Anick-Mitra-Sondhi on-off fluid queuing model has been proposed [112, 114] and found to be substantially more accurate than the M/M/1/B model at reproducing dependence of packet loss on w and the network parameters.

Even without knowing the exact expression for $p(w)$, however, sufficient conditions for linear stability of equilibrium (10) can be deduced in terms of p^* and $p'^*=p'(w^*)$. Using standard methods of control theory it has been shown [126] that equilibrium (10) of equation (8) will be stable in linear approximation provided that

$$w^*(2p^* + w^* p'^*) < \frac{p}{2}. \quad (11)$$

Inequality (11) is a necessary but not sufficient condition for stability, i.e., if it is violated the equilibrium will definitely be unstable but satisfying (11) does not guarantee (non-linear) stability. Based on (11) one can conclude, for example, that p'^* must decrease with increasing bandwidth-delay product in order for the equilibrium not to lose stability, since larger bandwidth-delay product corresponds to larger equilibrium congestion window size w^*. With more advanced methods it is also possible to derive conditions guaranteeing global stability of equilibrium (10), although, the resulting inequalities [124] are considerably more complicated and less informative than (11).

Fluid approximation models have also been used in a global optimization framework for the Internet, originally developed by F.P. Kelly, et al. [125] to show that per-flow TCP congestion control can be viewed as optimizing a certain global utility function. That is, TCP congestion control can be seen as a decentralized iterative algorithm for solving a network wide optimization problem. This point of view revolutionized understanding of the effect of TCP congestion control on global network dynamics. In particular, it paved the way toward a top-down protocol design, where starting with a desirable global network state first, the end-to-end congestion control can be tailored to achieve this global state.

Finally, fluid approximation models have been used to create fast simulators for large networks [106, 119] by leveraging fast numerical methods for solving systems of differential equations. While not as accurate or detailed as packet level simulators, the results from the first implementations are encouraging.

A.1.3 Limitations of Fluid-flow Approximation Models

In spite of great theoretical value, and even some practical applications, the fluid approximation framework falls short of being truly useful to practitioners of network and protocol design mainly due to lack of accuracy [112-113]. The main obstruction to the accuracy of fluid approximation models is lack of an accurate packet loss process model, which determines the equilibrium as well as dynamic behavior of the network. At the outset, the packet loss process was assumed to be well approximated by the loss process in an M/M/1/B queuing system. However, there is experimental evidence against the Poisson packet arrival hypothesis [118]. Based on packet traces collected from the Internet it was shown that the packet arrival process has rather different statistical properties from a Poisson process. In particular, it was observed that it is much burstier and is, moreover, bursty on all time scales. Due to the difficulty of mathematical analysis of queuing systems fed by such self-similar traffic, relatively little headway has been made toward obtaining a closed form expression for packet loss usable in the fluid approximation framework. In fact, it is still common in current publications [112, 120, 122, 123] to find computations based on the M/M/1/B queuing system.

Elsewhere [112] we proposed and tested a new expression for packet loss based on a queuing model of Anick, Mitra and Sondhi (AMS) [121]. Briefly, the model consists of a single fixed rate server fed by a superposition of fluid, fixed-rate, on-off sources with exponentially distributed “on" and “off" periods. This model is essentially a packet level fluid approximation. Observations [112, 114-115] of *ns2* traces suggest that TCP sources tend to concentrate packets in bursts (corresponding to a single congestion window)

rather than transmitting packets at a uniform rate on the time scale of a round-trip time. Thus setting the mean duration of the "on" periods to the congestion window size allows us to simulate burstiness arising from the non-uniformity in the transmission rate at the round-trip time scale.

The resulting mathematical model turns out to have a closed form solution in terms of the basic system parameters such as the number of sources, the server and source rates and the mean duration of the “on" and “off" periods. While this model is certainly only an approximation, since, for example, the window size distribution is expected to be non-exponential [107, 110], numerically it produces much better results than the commonly used M/M/1/B model [112]. Moreover there are some indications that the AMS packet loss model may be applicable to any unpaced TCP variant and not just TCP Reno, for which it was developed. Assuming this is so we can then use the fluid approximation framework to easily compare alternative congestion control algorithms in a variety of different network set ups with varying link bandwidths, buffer sizes and propagation delays. Next, we show how this can be accomplished.

A.2 Applying Fluid-flow Approximation Models to Compare Alternate Congestion Control Algorithms

In what follows, we briefly illustrate how the fluid approximation framework can be used to compare throughput performance of different TCP variants in a simple network. Specifically we consider an extension of the dumbbell topology — a network with a single link shared by a large number of continuously transmitting TCP flows with similar round-trip times (RTTs). We concentrate our attention on the standard TCP Reno and two other TCP variants — CUBIC [56] and Compound [58] TCP — which are currently increasingly deployed in the Internet due to their inclusion in Linux and Windows® Vista and Server operating systems, respectively [109]. In the following we will be interested only in the equilibrium throughput and so we will ignore the transient convergence dynamics described by the fluid approximation differential equations model and concentrate on the equilibrium solution. As previously explained, because congestion control mechanisms regulate transmission speed by opening and closing the congestion window, it is this window rather than throughput that is typically the main variable in mathematical models of TCP. The throughput is roughly proportional to the congestion window size divided by the round-trip time (including propagation and queuing delays).

The equilibrium mean congestion window size is described by a system of equations of the form

$$w^* = w(p^*) \tag{12}$$

$$p^* = p(w^*, C, B, T, N)$$

where w^* and p^* are the equilibrium congestion window size and packet loss probability, respectively. In special cases $w(p)$ may also depend on other network parameters such as the round-trip time (RTT).

Generally speaking one might expect that the second equation in (12), describing the dependence of packet loss on network parameters and equilibrium congestion window size, is roughly the same for all congestion control algorithms that use ACK self-

clocking, i.e., insert new packets into the network only in response to acknowledgments from the sink. The reason for this is that the sliding window algorithm in combination with ACK self-clocking largely determines the statistics of the aggregate packet arrival process at the bottleneck router, which, in turn, determines the statistics of packet loss. The packet loss statistics will vary with the TCP congestion control algorithm to the degree to which the equilibrium congestion window size distribution varies with congestion control algorithm. Unfortunately, due to the complexity of the problem relatively little is known about the properties of this distribution. For TCP Reno the stationary congestion window distribution has been computed under the assumption of Poisson losses [107, 110] and for more general additive-increase multiplicative-decrease algorithms [116]. On the other hand, the packet loss probability function under the assumption of exponential congestion window size distribution has also been derived [112].

We first, show how TCP variants can be qualitatively compared without knowing the form of the packet loss probability function provided it can be assumed to be approximately independent of the specifics of the congestion control algorithm. Indeed, if the second equation in (12) is independent of the specific algorithm, then the relative position of equilibria of TCP variants is determined by the first equation, which strongly depends on the particular congestion control algorithm used. The function $w(p)$ is often referred to in literature as the response function of the congestion control algorithm. As one would intuitively expect it is a decreasing function of p — the less frequent the losses the larger the equilibrium congestion window. Since the congestion window growth functions used in congestion control typically exhibit polynomial growth due to stability requirements, the response function itself is also typically polynomial in p, i.e., $w(p)=cp^{-\alpha}$ for some c, $\alpha>0$. Since the packet loss instances form a random process the form of $w(p)$ depends not only on the average loss probability but also on the statistics of the loss process. For simplicity, it is usually assumed that each packet is lost with probability p independent of previous losses, i.e., packet loss is a Bernoulli process. Suppose two TCP variants TCP_0 and TCP_1 have corresponding response functions $w_0(p)$ and $w_1(p)$. We will say that $w_0(p)$ *dominates* $w_1(p)$ if $w_0(p) > w_1(p)_1$ for $p \in [0,1]$. If TCP throughput is approximated by $(1-p^*)w^*/RTT$, then if $w_0(p)$ dominates $w_1(p)$ (as in Figure A-1) for $p \in [0,1]$ and p^* not too close to 1, TCP_0 will have strictly higher throughput than TCP_1.

In practice[2], p^* is usually less than 0.10. One must however keep in mind that this comparison in itself is an oversimplification in that it omits certain details such as bandwidth lost due to retransmissions, which may in practice lead to significantly lower overall throughput. For example, if we assume that the packet loss probability is equal to the blocking probability in an M/M/1/B queue (an admittedly optimistic scenario) it is not hard to show that the throughput increases with increasing w^* even when p^* is near 1, which translates into the best congestion control algorithm being no congestion control at all! That is, the faster the sources push packets into the network the higher the predicted throughput. Yet it is equally easy to see that this is a recipe for a congestion collapse,

[2] For example, the global Internet packet loss rate for 24 hours starting at 12:55 PM on April 22, 2010 averaged just below 7 %, as measured by the Internet Traffic Report. For the preceding month, the measured average loss rate did not reach 10 %. http://www.internettrafficreport.com/

since with packet loss near 1 the network will quickly fill with retransmitted copies of lost packets driving overall throughput to zero.

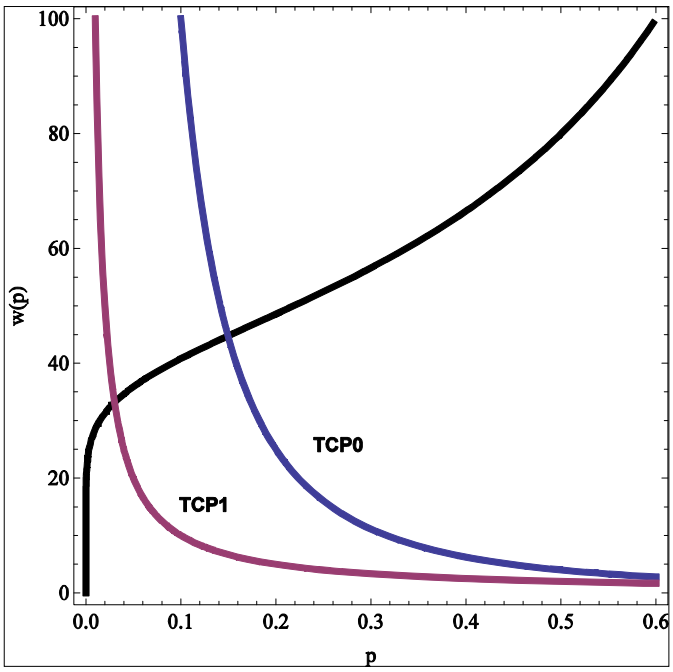


Figure A-1. Response curves of two hypothetical TCP variants TCP0 (blue) and TCP1 (red) and the graph of a hypothetical packet loss function (black).

If a dominance relationship between response functions cannot be established, then information about the packet loss function $p(w,\ldots)$ is necessary to compare congestion control algorithms. We will base our quantitative comparisons of TCP variants below on the packet loss model [112] discussed in Sec. A.1.3 above, since it was shown to produce more accurate results than models based on M/M/1/B queuing systems.

A.2.1 Computing Response Functions

We begin by computing the congestion response functions of TCP Reno, TCP CUBIC and Compound TCP. The response function of TCP Reno is well known and has been extensively experimentally verified [117]. We present a brief outline of the derivation as a simple illustration, since the computations become more involved for the other two variants.

A.2.1.1 TCP Reno. Suppose the per packet loss probability is p, then the average number of packets transmitted before a loss occurs is $N = (1–p)/p$. Suppose the equilibrium congestion window size just after a packet loss is w_0. Let a *round* be the number of packets equal to the current congestion window size and suppose for simplicity that between consecutive losses the number of rounds delivered is always an integer. Then the number of rounds k between consecutive losses is related to N by the equation

$$N=\sum_{i=0}^{k} w_0+k=(k+1)w_0+\frac{1}{2}k(k+1).$$

Solving for k gives

$$k=\frac{1}{2}(-2w_0-1+\sqrt{(2w_0-1)^2+8N}).$$

Thus at the end of a loss free period, just before packet loss occurs, the expected congestion window size will be

$$w_0+k=w_0+\frac{1}{2}(-2w_0-1+\sqrt{(2w_0-1)^2+8N}).$$

Since we assume the connection to be in equilibrium w_0 should be constant on average so

$$w_0=\frac{1}{2}\left(w_0+\frac{1}{2}(-2w_0-1+\sqrt{(2w_0-1)^2+8N})\right).$$

Solving the above equation for w_0 and discarding constant terms small in comparison with N we get

$$w_0=\sqrt{\frac{2/3}{N}}=\sqrt{\frac{2/3(1-p)}{p}}$$

Finally, to obtain the equilibrium mean congestion window size we multiply w_0 by 3/2

$$w(p)=\sqrt{\frac{3/2(1-p)}{p}}.$$

Since in practice p is usually close to 0 an approximation $w(p)=\sqrt{(3/2)/p}$ is often used.

A.2.1.2 Cubic TCP. CUBIC TCP differs from TCP Reno in several important respects, two of which make computation of the response function considerably more difficult as compared to the Reno computations outlined above. First, CUBIC's congestion window growth function depends on time rather than the number of acknowledged packets. Second, the congestion window growth function depends not only on the congestion window size before the packet loss (as in Reno) but also on how this value compares with the congestion window size at the end of the previous loss free period. Thus congestion window size alone no longer determines the full state of the algorithm and an additional parameter — last maximum achieved — must also be tracked to have a full state. Finally, nonlinearity of the congestion window growth function, the very feature that is supposed

to improve performance, makes computation of expectations hard. In view of these difficulties we are forced to content ourselves with the few rough approximations that are computable.

Let us begin by considering a very simple case when the packet losses are periodic in time. In this case the congestion window growth function can be shown to converge to the convex part of the cubic root, which drastically simplifies computations. Let the time between losses be τ and the congestion window just before a loss be w_0 then the congestion window at the end of the loss free period will be

$$c(\tau - k)^3 + w_0$$

where $k = \sqrt[3]{w\beta/c}$ and β and c are constants [56]. Since in equilibrium the w_0 is constant we have

$$c(\tau - \kappa)^3 + w_0 = w_0.$$

Substituting in for k and solving for w_0 we get

$$w_0 = \frac{c\tau^3}{\beta}.$$

Thus in equilibrium $k=\tau$. We can now compute the equilibrium mean congestion window size as a function of τ

$$w(t) = \frac{1}{\tau}\int_0^t c(\tau - t)^3 + w_0 d = c\tau^4\left(\frac{1}{\beta} - \frac{1}{4}\right) \qquad (13)$$

To convert this into a function of p we compute the number of packets sent during a loss free period, which is approximately

$$N = \frac{1}{T}\int_0^\tau c(t-k)^3 + w_0 dt = \frac{c\tau^4}{\mathrm{T}}\left(\frac{1}{\beta} - \frac{1}{4}\right)$$

where T is the round-trip time, which we assume to be approximately constant. Solving for τ we get

$$t = \sqrt[4]{\frac{NT}{c}\frac{4\beta}{4-\beta}}.$$

Substituting for τ in (13) we get

$$w(p) = \left(\frac{c(4-b)(NT)^3}{4\beta} \right)^{1/4}$$

Finally, assuming only one packet is lost in each congestion event so that p=1/N we get

$$w(p) = \left(\frac{c(4-\beta)}{4\beta} \left(\frac{T}{p} \right)^3 \right)^{1/4}$$

which for the default settings of β=.2 and c=.4 gives

$$w(p) \approx 1.17 \left(\frac{T}{p} \right)^{3/4} . \quad (14)$$

This is the formula derived by the designers of CUBIC [56].

In practice, however, packet losses are never periodic even in the case when there is only one connection on the link. Therefore, we consider a more realistic case of a Poisson loss process with rate λ. That is, we assume that a loss event occurs on average every 1/λ seconds. Two possibilities have to be considered because the congestion window at the end of a loss free period can now fall below as well as above the last maximum. We assume that w_0 and k are stationary independent random variables. This is still not enough to make explicit computations possible, so we further replace w_0 and k by deterministic variables equal to the mean values of the respective random variables. With these, admittedly very crude, simplifying assumptions we can write down a pair of fixed point equations for w_0 and k

$$w_0 = (1-\frac{\beta}{2}) \int_0^k \lambda e^{-\lambda t} (c(t-k)^3 + w_0) dt + \int_k^\infty \lambda e^{-\lambda t} (c(t-k)^3 + w_0) dt$$

$$k = \int_0^k \lambda e^{-\lambda t} \sqrt[3]{\frac{\beta}{2c}(c(t-k)^3 + w_0)} dt + \int_k^\infty \lambda e^{-\lambda t} \sqrt[3]{\frac{\beta}{2c}(c(t-k)^3 + w_0)} dt .$$

The first equation can be solved numerically in terms of k. Substituting the resulting equation into the equation for k and solving numerically over a range of λ indicates, as one might expect, that the equilibrium value of k is very nearly proportional to 1/λ with coefficient of about 1.3. This gives

$$w_0 = \frac{c(3+0.8\beta)}{\beta\lambda^3} .$$

for equilibrium congestion window just before a loss and

$$w(\lambda) = \frac{.3b + 3.8}{\beta} \frac{c}{\lambda^3}$$

for mean congestion window size. Expressing the above in terms of losses per number of packets sent and substituting the default values for β and *c* we get

$$w(p) \approx 1.67 \left(\frac{T}{p} \right)^{3/4}, \quad (15)$$

which is remarkably close to the simple formula (14).

A.2.1.3 Compound TCP. Compound TCP (CTCP) attempts to use delay measurements to estimate the number of buffered packets and alleviate congestion. CTCP's congestion window is decomposed into two components: the standard Reno congestion window, which increases by one over the window size for every acknowledgment received, and the delay based component which grows polynomially but only as long as the number of buffered packets, as measured by the increase in the round-trip delay, is below a certain threshold γ, which is itself dynamically adjusted to match available buffer space. Because the queue length measurements are performed for every returning acknowledgment and the congestion window growth rate modified accordingly, the window growth function is tightly coupled to the queue length. Thus, in general, analysis of CTCP must include an explicit model of queue lengths along the connection's path. Unfortunately, modeling queuing dynamics is in itself a complex and largely unsolved problem and so we are again forced to make crude simplifying assumptions. Specifically, we will assume that statistical fluctuations dominate dynamics so that the smoothed round-trip delay remains roughly constant once the network reaches equilibrium. Under this assumption analysis of the congestion window response function simplifies because γ does not change over time. Moreover, dynamic γ tuning insures that on average congestion window growth is polynomial right up to the moment of loss. Proceeding as before we thus have

$$\left(\frac{(1-k)\alpha}{T} \tau + w_0^{1-k} \right)^{\frac{1}{1-k}}$$

for the congestion window size at the end of a loss free period of duration τ given that the starting congestion window size is w_0 [58]. Since the window is reduced by 1−β upon detection of packet loss the equilibrium w_0 is determined by the equation

$$w_0 = (1-\beta) \left(\frac{(1-k)\alpha}{T} \tau + w_0^{1-k} \right)^{\frac{1}{1-k}}.$$

Computing the equilibrium mean congestion window size as a function of τ we get

$$w(\tau) = \frac{1}{\tau}\int_0^{\tau} \left(\frac{(1-k)\alpha}{T}\tau + w_0^{1-k}\right)^{\frac{1}{1-k}} dt$$

$$= \frac{T}{\tau}\frac{\left(w_0^{1-k} + \frac{(1-k)\alpha\tau}{T}\right)^{\frac{2-k}{1-k}} - w_0^{2-k}}{(2-k)\alpha}$$

Substituting for w_0 we have

$$w(\tau) = \frac{T\left(\frac{(1-k)\alpha\tau}{T}\right)^{\frac{2-k}{1-k}}\left((1+\gamma)^{\frac{2-k}{1-k}} - \gamma^{\frac{2-k}{1-k}}\right)}{(2-k)\alpha\tau} \quad (16)$$

$$\gamma = \frac{(1-\beta)^{1-k}}{(1-\beta)^{1-k}-1}$$

The number of packets transmitted in a time interval τ is given by

$$N = \frac{1}{T}\int_0^{\tau} \left(\frac{(1-k)\alpha}{T}t + w_0^{1-k}\right)^{\frac{1}{1-k}} dt$$

$$= \frac{\left(w_0^{1-k} + \frac{(1-k)\alpha\tau}{T}\right)^{\frac{2-k}{1-k}} - w_0^{2-k}}{(2-k)\alpha}$$

$$= \frac{\left(\frac{(1-k)\alpha\tau}{T}\right)^{\frac{2-k}{1-k}}\left((1+\gamma)^{\frac{2-k}{1-k}} - \gamma^{\frac{2-k}{1-k}}\right)}{(2-k)\alpha},$$

which gives

$$\tau = \frac{T}{(1-k)\alpha}\left(\frac{(2-k)\alpha N}{\left(1+\gamma^{\frac{2-k}{1-k}}\right) - \gamma^{\frac{2-k}{1-k}}}\right)^{\frac{2-k}{1-k}}$$

Finally, substituting τ into (16) and assuming $p=1/N$ as before, we obtain

$$w(p) = \frac{1-k}{2-k}\left((1+\gamma)^{\frac{2-k}{1-k}} - \gamma^{\frac{2-k}{1-k}} \right)^{\frac{1-k}{2-k}} \left(\frac{(2-k)\alpha}{p} \right)^{\frac{1}{2-k}}$$

for the response function of CTCP. For the default values of α=1/8, β=1/2 and k=3/4 [58] this is

$$w(p) \approx 0.25 \frac{1}{p^{4/5}}. \quad (17)$$

A.2.2 Comparing Congestion Control Algorithms

We begin performance comparison with the qualitative method described in the A.2 (see Fig. A-1). Since all computed response functions are approximations we must allow for errors in the resulting models. At present, however, there is no theoretical framework for computing fluid model error bounds and we are forced to make a somewhat arbitrary, but we hope conservative, assumption that the model equilibrium congestion window size is within 50% of the average window size that would be observed in a similar physical network.

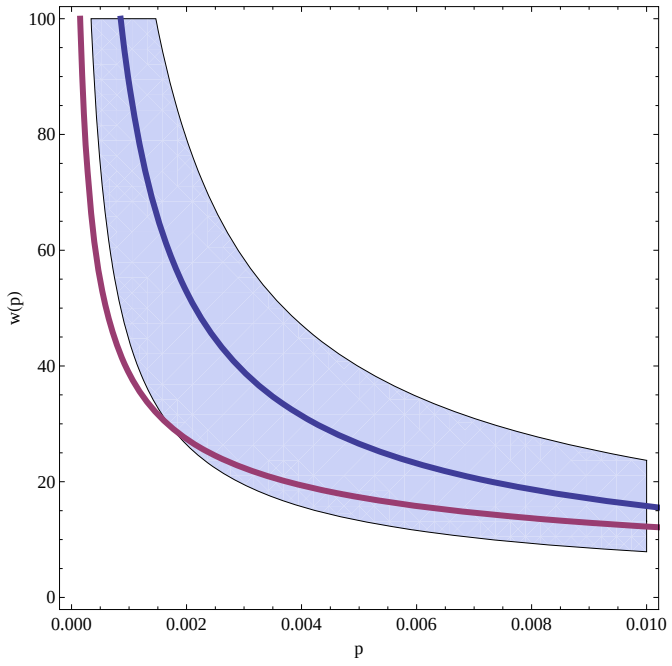


Figure A-2. Response curve for CUBIC (blue) with a ±50% error region (blue) vs. TCP Reno (red).

Fig. A-2 shows congestion window size as a function of packet loss for CUBIC and TCP Reno. For TCP Reno we do not plot the error region because the TCP Reno response function has been shown to be reasonably accurate [117]. As can be seen from the diagram, for the same probability of packet loss, CUBIC is likely to have a larger

congestion window, and so a higher throughput, for equilibrium packet loss rates up to about 1 %. We did not compute the response curve for higher loss rates because both CUBIC and TCP are likely to have limited throughput as loss rates become substantially higher. The response function plot for CTCP (Fig. A-3) shows that it will likely have a higher throughput than TCP Reno if equilibrium packet loss is below about .3 %. The plot also suggests that for equilibrium packet loss rates above about .5 % TCP Reno may actually outperform CTCP.

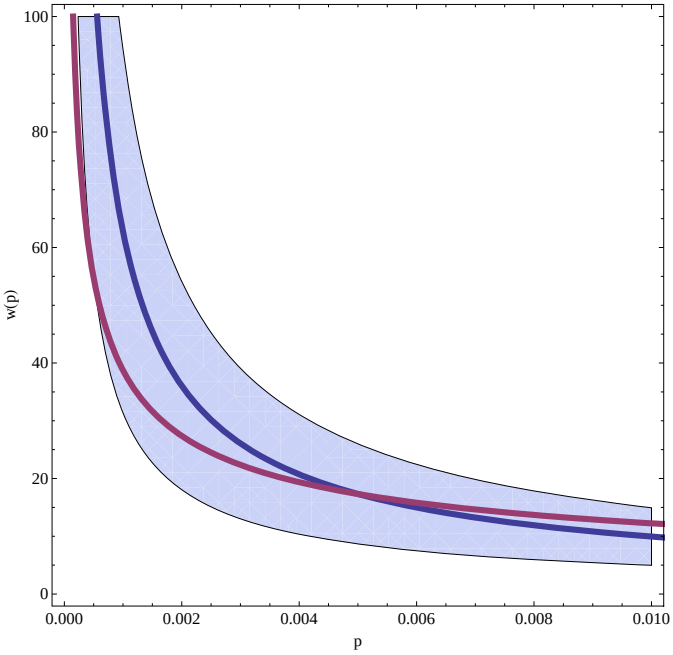


Figure A-3. Response curve for CTCP (blue) with a ±50% error region (blue) vs. TCP Reno (red)

Comparing response functions of CTCP and CUBIC (Fig. A-4) we observe that CUBIC is likely to have higher throughput than CTCP for the same equilibrium packet loss rate. However, because the error regions overlap considerably it is hard to say conclusively which of the two algorithms is likely to achieve higher throughput in practice.

We can also obtain some quantitative measures of the algorithms' performance by using a specific packet loss model, such as the one we introduced [112], to compute equilibrium throughput over a range of network parameters. Specifically we take

$$p(w) = \frac{Nw}{CT} e^{-1.5\left(\frac{C}{Nw} - \frac{1}{T}\right)\frac{BN}{C}}, \qquad (18)$$

where N is the number of concurrent flows, T is round-trip propagation delay, C is router capacity and B is buffer size. Tables A-1 through A-3 show average throughput for 1000 continuously transmitting flows over a 1 Gbps link for a range of propagation delays and buffer sizes. These quantitative results, unsurprisingly, largely agree with the above

qualitative analysis, but they also suggest some unanticipated conclusions. First, for round trip propagation times below 150 ms the alternative TCP variants do no better, and sometimes worse, than TCP Reno. This is presumably because the equilibrium packet loss rate for these scenarios is relatively high. Of course, CUBIC and CTCP were by design optimized for links with high capacity and long propagation delay (also called “long-fat pipes") and so their under-performance on links with relatively low bandwidth-delay product may be a chosen and accepted trade-off. Note, also, that this justifies the Reno-to-alternative mode switch present in most of the new TCP variants, including CUBIC and CTCP.

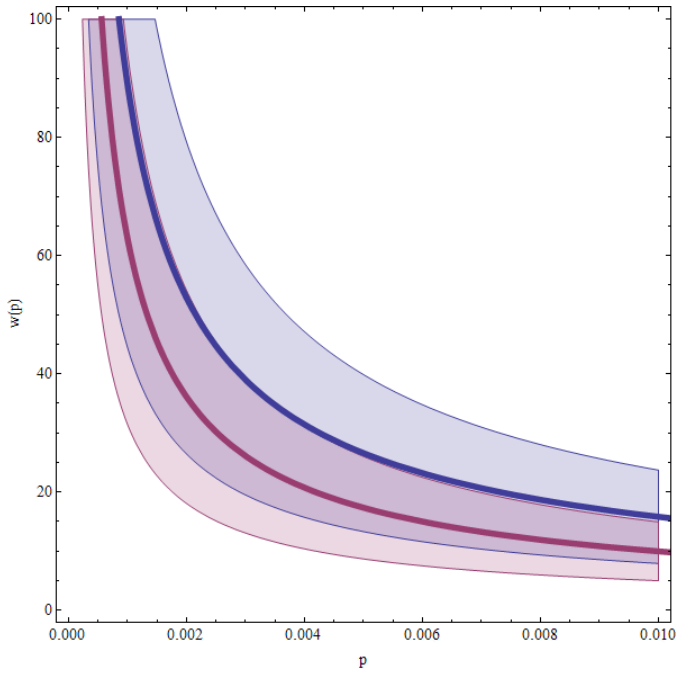


Figure A-4. Response curves for CUBIC (thick blue) and CTCP (thick red) with corresponding error regions

A second unexpected observation is that CTCP does not perform significantly better than TCP Reno even when round-trip delay becomes large, at least for the network parameters considered. Examining the graph of the CTCP response function (Fig. A-3), we see that this is most likely the result of a relatively high equilibrium packet loss rate, in the range of .3 % to .5 %, where the difference between response functions for CTCP and TCP Reno is small. On the other hand, in agreement with the qualitative analysis, CUBIC comes out ahead with an estimated improvement in throughput in the range of 10 % to 15 % over TCP Reno for networks with round-trip propagation delays longer than 150 ms.

While far from being exact or scalable to a network as large and complicated as the Internet, the mathematical models and methods presented here provide a cheap and fast way for evaluating alternative TCP congestion control algorithms even before any code is written. The value of these techniques is even greater when they are used as

design guideposts in the earliest stages of the development process of new TCP congestion algorithms.

Table A-1. Estimated throughput (p/ms) for CUBIC for 1000 concurrent flows on a link with a 122 p/ms capacity (for 1 KB packets)

CUBIC							
		B (pkts)					
		50	**100**	**150**	**200**	**250**	**300**
T (ms)	**50**	108	114	116	118	118	118
	100	98	107	111	113	114	115
	150	90	101	106	109	111	112
	200	84	96	102	105	108	110
	250	79	91	98	102	105	107
	300	76	88	94	99	102	105

Table A-2. Estimated throughput (p/ms) for CTCP for 1000 concurrent flows on a link with a 122 p/ms capacity (for 1KB packets)

CTCP							
		B (pkts)					
		50	**100**	**150**	**200**	**250**	**300**
T (ms)	**50**	112	115	117	117	118	118
	100	98	107	111	113	115	116
	150	87	98	104	108	110	112
	200	78	91	98	102	105	107
	250	70	84	92	97	101	103
	300	65	79	87	92	96	99

Table A-3. Estimated throughput (p/ms) for TCP Reno for 1000 concurrent flows on a link with a 122 p/ms capacity (for 1 KB packets)

TCP Reno							
		B (pkts)					
		50	**100**	**150**	**200**	**250**	**300**
T (ms)	**50**	115	116	116	117	117	118
	100	102	109	112	114	115	116
	150	89	99	105	108	110	112
	200	78	91	97	101	104	107
	250	70	83	90	95	99	102
	300	64	77	85	90	94	97

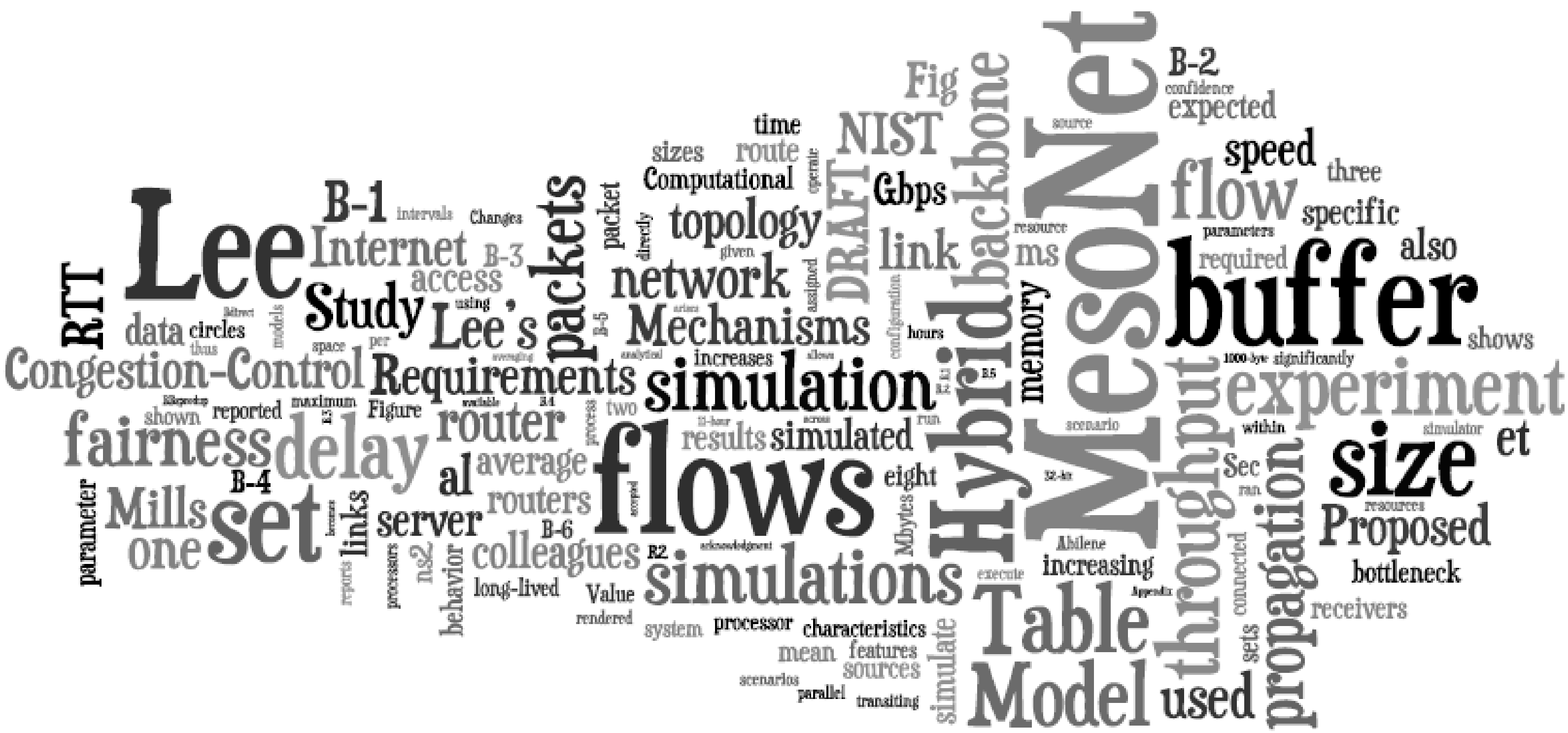
A.3 Future Work

Given that accurate modeling of packet loss is the key to accurate fluid approximation models, an important direction for future research is in the improvement and refinement of queuing models for TCP traffic. While the new packet loss model described in Sec. A.1.3 performs better than the commonly used, but highly inaccurate, M/M/1/B model, there is still room for improvement. In particular, the packet loss model in Sec. A.1.3 includes a finite buffer correction factor that is a rather crude patch in lieu of a solution for the finite buffer system. The model given in Sec. A.1.3 can also be improved by considering sources with non-exponentially distributed on-off periods since there is reason to expect that congestion window sizes have a non-exponential distribution.

Utility of the fluid approximation framework would also be greatly improved if response functions of the various new alternative congestion control algorithms could be computed more precisely. An important related question is: how do the specifics of the congestion control algorithm affect the congestion window size distribution? Answering this question would determine the sensitivity of the packet loss model to the TCP variant and hence the robustness of the comparison above.

The question of how network topology affects the equilibrium and stability of TCP traffic is another important direction for future work. Recently fluid approximation models have begun to be used for numerical simulations of large networks [108]. The low resource demands and high speed of these simulators permit, for the first time, an extensive exploration of the space of network topologies under a variety of simulated network conditions.

Appendix B – Computational Requirements: MesoNet vs. Hybrid Model



Appendix B Computational Requirements: MesoNet vs. Hybrid Model

Junsoo Lee and colleagues [71] observe that packet-level network simulations, such as *ns2* [79], require substantial computational resources for large-scale simulations and also entail so many parameters that it becomes difficult to understand the influence of specific factors on overall system performance. Lee also points out that aggregate fluid-flow models [e.g., 73] address these shortcomings but can only capture steady-state behaviors averaged over long time intervals. Lee describes a hybrid modeling framework that continuously approximates discrete variables by averaging over short intervals of time. Constraining the averaging interval allows generation of significant events, such as packet drops and related adjustments in congestion windows. Like MesoNet, Lee's hybrid framework aims to simulate a manageable parameter space and thereby illuminate the influence of specific factors on system behavior, while reducing computational requirements.

In this appendix, we use MesoNet to replicate a simulation experiment reported by Lee and colleagues [71]. The specific experiment conducted by Lee uses a hybrid model to simulate an 11-hour scenario involving 30 long-lived flows transmitting data across a subset of the Abilene topology. Lee reports that this scenario was infeasible using *ns2* because his available computer had only 512 Mbytes of memory, which proved insufficient. Replicating this experiment with MesoNet serves three purposes: (1) to illustrate that MesoNet can simulate a scenario found to be infeasible with a commonly used network simulator, (2) to show that MesoNet produces behavior similar to Lee's hybrid simulator (which was validated against predictions from a widely accepted analytical model) and (3) to compare computational requirements of MesoNet against reported computational requirements for Lee's hybrid model. In the process of achieving these objectives, we raise confidence in MesoNet and we demonstrate that hybrid network models hold promise as replacements for discrete-event network simulations.

We begin in Sec. B.1 by describing our experiment design. Where applicable, we identify and justify specific differences in the MesoNet experiment setup and the configuration used by Lee. In Sec. B.2, we outline how we executed the simulations and how we collected the required data. Next, in Sec. B.3, we present results regarding flow behavior. In Sec. B.4, we compare our findings with those reported by Lee. We conclude in Sec. B.5.

B.1 Experiment Design

The fundamental purpose of the experiment designed by Lee and colleagues [71] was to investigate the effect of buffer size on relative fairness among long-lived TCP flows that transit network routes with differing propagation delays and a shared bottleneck link. The expected result is that smaller buffer sizes allow propagation delay to be the dominant component of round-trip time (RTT), which implies that flows transiting longer paths should receive lower throughputs than flows transiting shorter paths. As buffer size increases, queuing delay becomes the dominant component of RTT, which implies that the throughput of all flows will come closer together. This expectation arises from a widely accepted analytical formula to predict TCP throughput, which generally

underestimates the fairness ratio, as confirmed by *ns2* simulations with small network topologies. Lee and colleagues show that their hybrid model yields the expected behavior in a large network based on the original Abilene topology. We aim to show that MesoNet also exhibits the expected behavior in the same topology used by Lee. This will increase our confidence in MesoNet. We will also be able to compare resource requirements of MesoNet against reported requirements for the hybrid model.

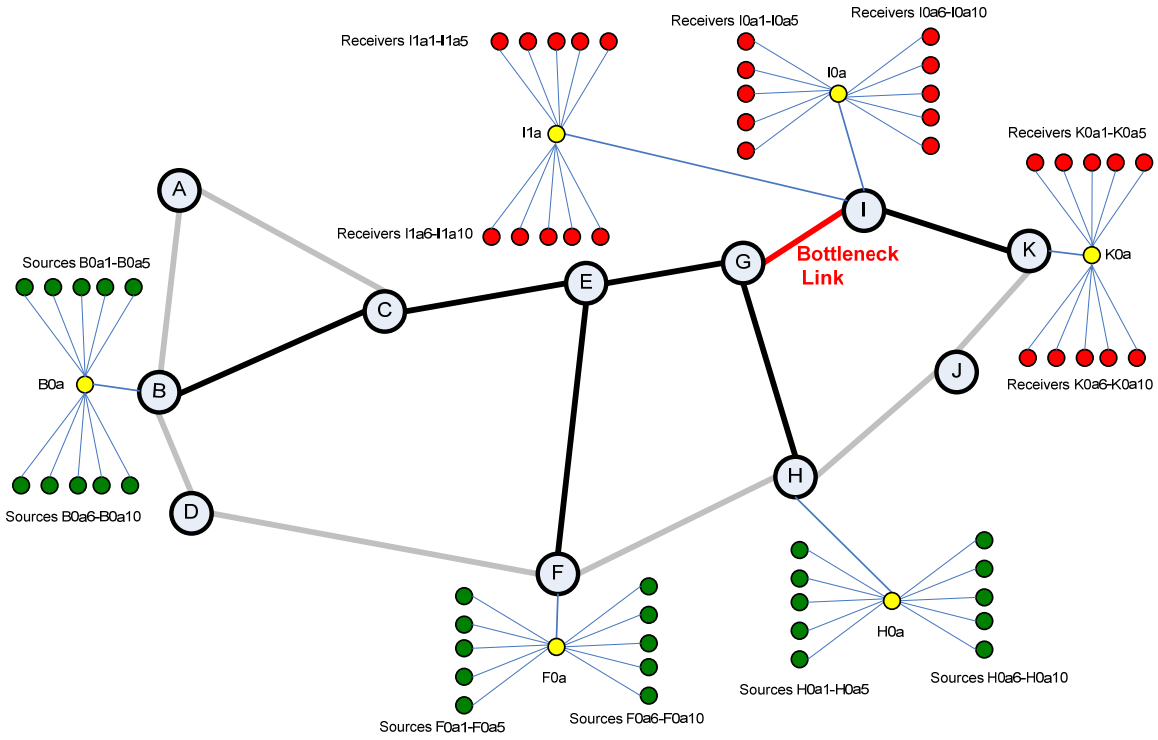


Figure B-1. Experiment Topology

Fig. B-1 shows the network topology we simulated. The backbone is derived from the original Abilene topology, as given by Lee [71]. The backbone consists of 11 routers (grey circles designated A-K in our topology) that each serve a different location within the United States. The backbone routers are connected by 14 bidirectional links. We assigned a propagation delay to each link, as specified in Table B-1. We used the same propagation delay for each direction on a given link (e.g., links A→B and B→A both have 17 ms propagation delays). We adopted the propagation delays used by Lee, except that we rounded to the nearest millisecond.

The seven grey links in Fig. B-1 are not used in this experiment because Lee focused on three sets of flows, where each set transits a different route and where the routes share a bottleneck link (G-I), rendered in red in Fig. B-1 (remaining links used by flows are shown in black). Flow sources are rendered as green circles in Fig. B-1 and flow receivers are rendered as red circles. As required by MesoNet, each source and receiver must be connected to an access router (yellow circles in Fig. B-1), and for this experiment each access router is connected directly to a backbone router (i.e., there are not Point of Presence routers in the topology). This differs from Lee's configuration, where sources and receivers connected directly to backbone routers.

Table B-1. One-Way Propagation Delay on Each Link in the Simulated Topology

source	destination	prop. delay (ms)
A	B	17
A	C	26
B	C	25
B	D	8
C	E	11
D	F	32
E	F	16
E	G	9
F	H	20
G	H	11
G	I	4
H	J	16
I	K	20
J	K	4

Table B-2 reports relevant characteristics for each set of simulated flows. The first set of flows has 10 sources under access router H0a. Each source transmits to one of 10 receivers located under access router I0a. In MesoNet, packets transiting access routers experience queuing delay but no propagation delay; a packet experiences propagation delay only when crossing backbone links. MesoNet sends data packets for these 10 flows over backbone route (H-G-I) and returns acknowledgments[1] over the reverse route (I-G-H); thus, the round-trip propagation delay between a data packet and its acknowledgment is 30 ms (twice the 15 ms propagation delay on the route). Similar information is provided for two additional sets of 10 flows. As the backbone route increases from two to three to five hops with each set of flows, relative propagation delay approximately doubles. Table B-2 highlights the bottleneck link shared by all flows.

Lee's experiment simulates backbone links operating at 10 Gbps. While Lee does not report the speed of simulated sources and receivers, we assume their speed is sufficient to achieve more than 10 Gbps when 30 flows are aggregated across the bottleneck link. Lee allows each flow to start at a random time, uniformly distributed

[1] Note that Lee's hybrid model does not specifically simulate acknowledgments. This represents another difference with MesoNet. Also, in MesoNet, packets have no specific size, so each acknowledgment consumes one packet of buffer space, which is also the buffer space consumed by each data packet.

over one second, and then the flows continue transmitting (as congestion permits) for just over 11 hours. Lee repeats this simulation six times, while increasing buffer sizes in increments of 25 x 10^3 (1000-byte) packets from 25 x 10^3 to 150 x 10^3.

Table B-2. Characteristics of Three Flow Sets Simulated in the Experiment

sets	# of flows	prop. delay	src/dest	route (symmetric)
set one	10	15 ms	H0a/I0a	H-G-I
set two	10	29 ms	F0a/I1a	F-E-G-I
set three	10	69 ms	B0a/K0a	B-C-E-G-I-K

Table B-3. MesoNet Parameter Settings for the Experiment

Parameter	Value
M	60 x 10^3
MI	660
R1	1250 p/ms
BBspeedup	1
R2	1
R3	1
Bdirect	1
QszAlg	Directly Set
Hfast	80
Flow Start	uniform (0..1s)

To match Lee's conditions, we assigned MesoNet parameters as specified in Table B-3. MesoNet assigns a speed to each router in the topology. Parameter **R1** specifies that backbone routers process 1250 packets/millisecond. Setting related parameters (**BBspeedup, R2, R3 and Bdirect**) to one ensures that all routers operate at the same speed. Assuming 1000-byte packets, each of the backbone and access routers then operate at 10 Gbps (1250 packets/milliseconds x 1000 milliseconds/second x 1000 bytes/packet x 8 bits/byte). We assigned sources and receivers to operate at (**Hfast** =) 80 packets/millisecond, which equates to a maximum of 640 Mbps (80 packets/milliseconds

x 1000 milliseconds/second x 1000 bytes/packet x 8 bits/byte). When 30 flows cross the bottleneck, the potential demand of 19.2 Gbps (640 Mbps/flow x 30 flows) exceeds the available link capacity. We measured system state every (**M** =) 60 x 10^3 milliseconds (i.e., once a minute) and we run the simulation for (**MI** =) 660 measurement intervals (i.e., for 660/60 = 11 hours). We set the buffer size in each router directly to the appropriate value for each repetition: we vary buffers from 25 x 10^3 to 200 x 10^3 packets[2] in 25 x 10^3 packet increments. Table B-4 gives the domain view of the parameter settings shown in Table B-3.

Table B-4. Domain View of the Simulated Network Characteristics

Characteristic	Value(s)
Measurement Interval Size	60 seconds
Simulation Duration	11 hours/run
Backbone Router Speed	10 Gbps
Access Router Speed	10 Gbps
Router Buffer Sizes	25 x 10^3 – 200 x 10^3 packets
Maximum Host Speed	640 Mbps
Max. Link Demand on G-I	19.2 Gbps

For each simulation run, we make the same measurements taken by Lee. Specifically, we measure throughput fairness ($\mathbf{FR_{i,j}}$) and RTT fairness ($\mathbf{RR_{i,j}}$). In equations (1) and (2), *i* and *j* (*i* not equal to *j*) each denote a specific set of flows. Thus, we average either the throughput (1) or RTT (2) for each set and then take the ratio of each pair of sets, where the denominator is chosen from the set expected to have the lower value in a given pair.

$$\mathbf{FR_{i,j}} \equiv \frac{\mathbf{mean}\left(\mathbf{Throughput_i}\right)}{\mathbf{mean}\left(\mathbf{Throughput_j}\right)} \quad (1)$$

$$\mathbf{RR_{i,j}} \equiv \frac{\mathbf{mean}\left(\mathbf{SRTT_i}\right)}{\mathbf{mean}\left(\mathbf{SRTT_j}\right)} \quad (2)$$

[2] While Lee simulated only six buffer sizes, we simulate eight buffer sizes because we had access to a server with eight processors. We ran the eight simulations in parallel on the server.

B.2 Experiment Execution and Data Collection

We ran eight, parallel instances (one per buffer size) of the MesoNet simulator, where each instance ran within one 32-bit SLX process on one processor within a computation server, configured as shown in Table B-5. Table B-6 reports the computation and memory resources required for each simulation.

Table B-5. Configuration of Compute Server for Simulations

Property	Characteristics
Operating System	Microsoft Windows Server 2003 R2 x64 Edition SP2
Server	Dell Server PE6950
Server Memory	32 Gbytes
Processor Chip	Four Dual-Core AMD Opteron Processors 8222SE
Processor Speed	3 GHz
Total Processors	(4 x 2 =) 8
Simulation Environment	SLX 32-bit Version Release 2.3 (PR229)

Table B-6. Resource Requirements for Simulations

Buffer Size (packets)	Processor Hours	Memory (Mbytes)
25000	80.33	34
50000	80.41	41
75000	99.23	47
100000	123.21	55
125000	102.19	60
150000	110.79	67
175000	129.08	73
200000	122.39	80

Every minute, we measured the instantaneous (60-second) average throughput and smoothed RTT seen on each of the 30 long-lived flows. This enabled us to collect 660 samples per metric per flow over an 11-hour simulation. We then averaged (660 x 10 =) 6600 samples to generate a mean throughput for each set of 10 flows. We similarly obtained an average RTT for each set of flows. We used these averages to form the fairness ratios defined in equations (1) and (2).

B.3 Results

Fig. B-2 plots the changing RTT fairness ratios as buffer size increases. Fig. B-3 shows variation in throughput fairness. These two plots exhibit the expected convergence in fairness as buffer size increases. The curves for throughput fairness bump up slightly, as buffer size moves from 100 x 10^3 to 125 x 10^3 packets, before continuing the downward trend. This bump arises from a dip in average throughput for flow set number 3, coupled with a slight increase in average throughput for flow set number 2, as shown in Fig. B-4. We attribute these fluctuations to randomness arising from using a single repetition of the simulation to generate each set of data points.

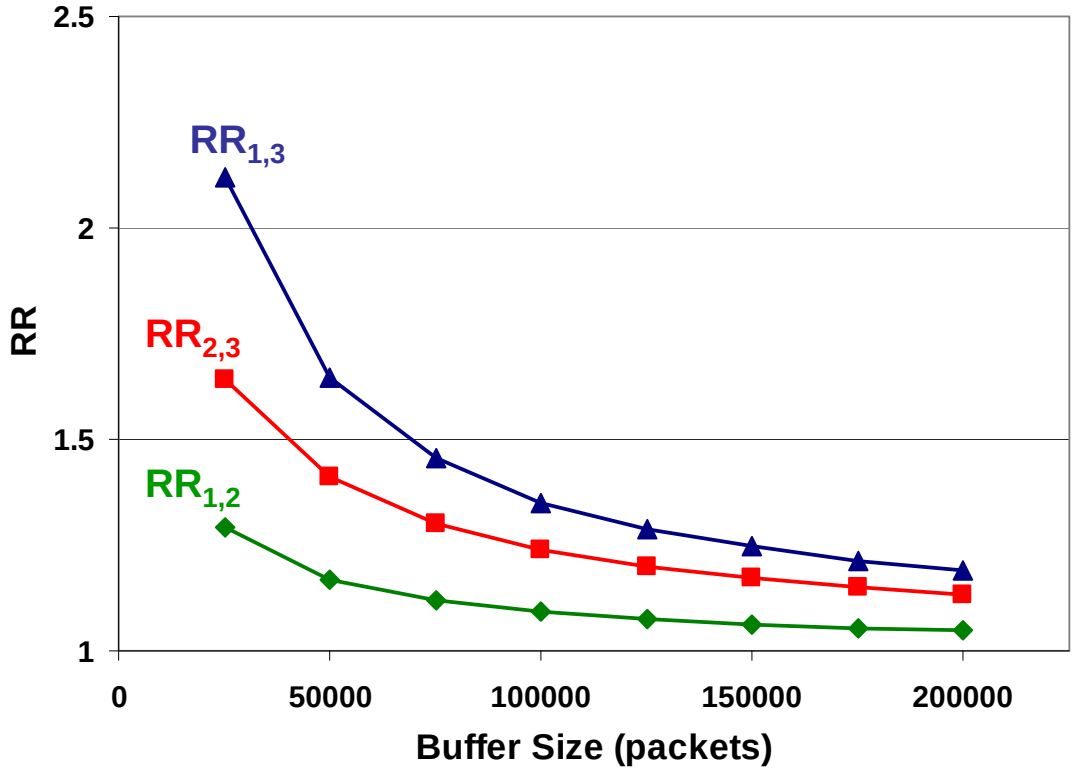


Figure B-2. Changes in RTT Fairness with Increasing Buffer Size

B.4 Discussion

As expected, mirroring the results of Lee and colleagues, RTT and throughput fairness converge with increasing buffer size. These results enhance our confidence in MesoNet. Further, Table B-6 shows that we can execute the required MesoNet simulations in under 100 Mbytes of memory, whereas Lee and colleagues found that they could not execute these simulations using *ns2* in a machine with 512 Mbytes of memory. On the other hand, running these MesoNet simulations took just under 5 ½ days, the time required by the maximum simulation run (buffer size of 175 x 10^3 packets). From reading the information provided by Lee and colleagues, we would expect the hybrid model, running all eight simulations in parallel, to complete in less than one day. This comparison of processing

requirements shows that hybrid models have the potential to significantly accelerate simulation in scenarios such as the one here.

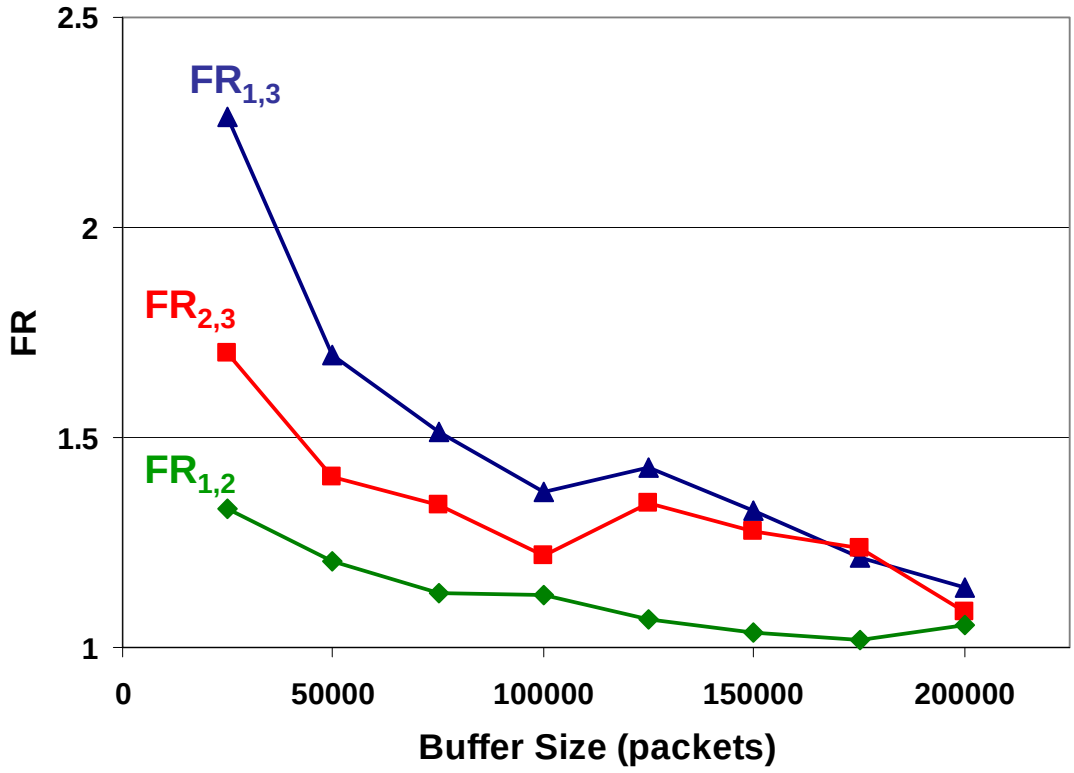


Figure B-3. Changes in Throughput Fairness with Increasing Buffer Size

When considering the use of a hybrid model for other scenarios, such as those described throughout this report, we note that Lee's model would need to be extended to include many features not currently present. Such features include: multiple routing tiers and router classes, arriving and departing flows, variety in flow types, many more measurements, connection establishment procedures, and support for arbitrary topologies. In principle, we expect that such features could be incorporated into a hybrid model. Further, we suspect that such a hybrid model would execute more swiftly that our MesoNet simulation. Confirming these hypotheses requires future work.

B.5 Conclusions

In this section, we used MesoNet to repeat an experiment conducted by Lee and colleagues. We compared the results obtained by Lee with MesoNet results, finding general agreement. We also demonstrated that MesoNet requires significantly fewer memory resources than *ns2*. Further, we showed the Lee's hybrid model could likely simulate scenarios involving long-lived flows at a rate more than five times faster than MesoNet, which relies on discrete-event simulation. Further work remains to extend Lee's hybrid model with features needed to conduct the full suite of experiments used in

the remainder of our study. We believe hybrid modeling holds the promise of significantly reducing resource requirements for network simulations.

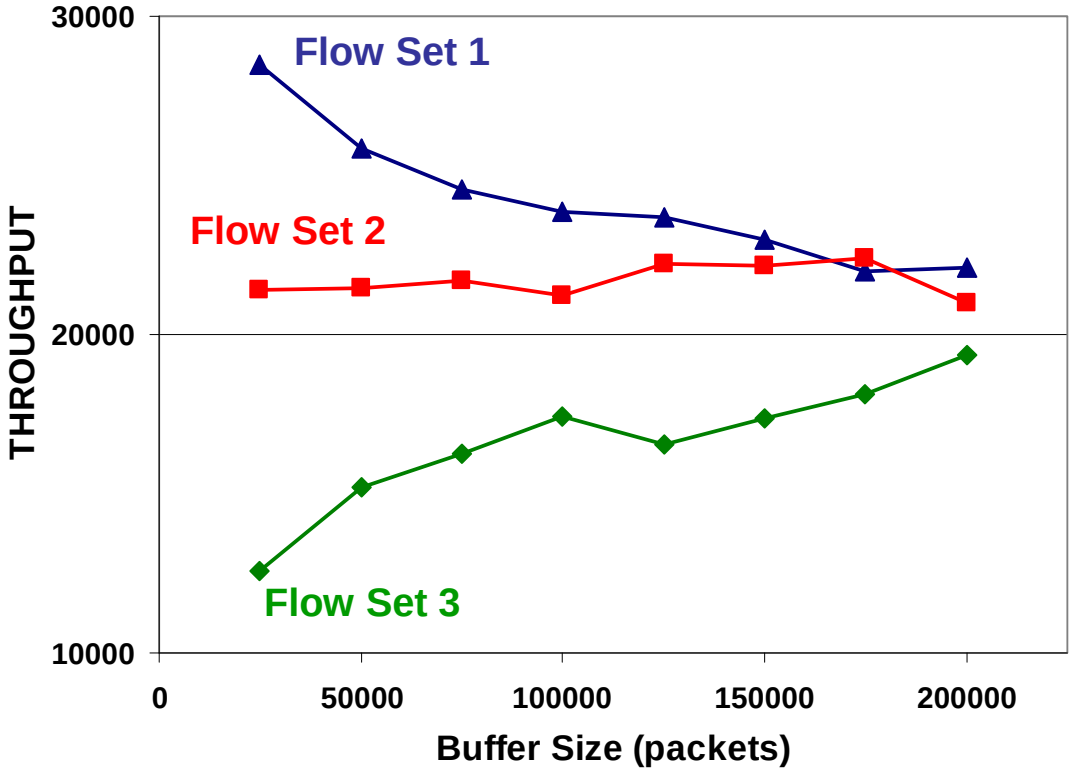
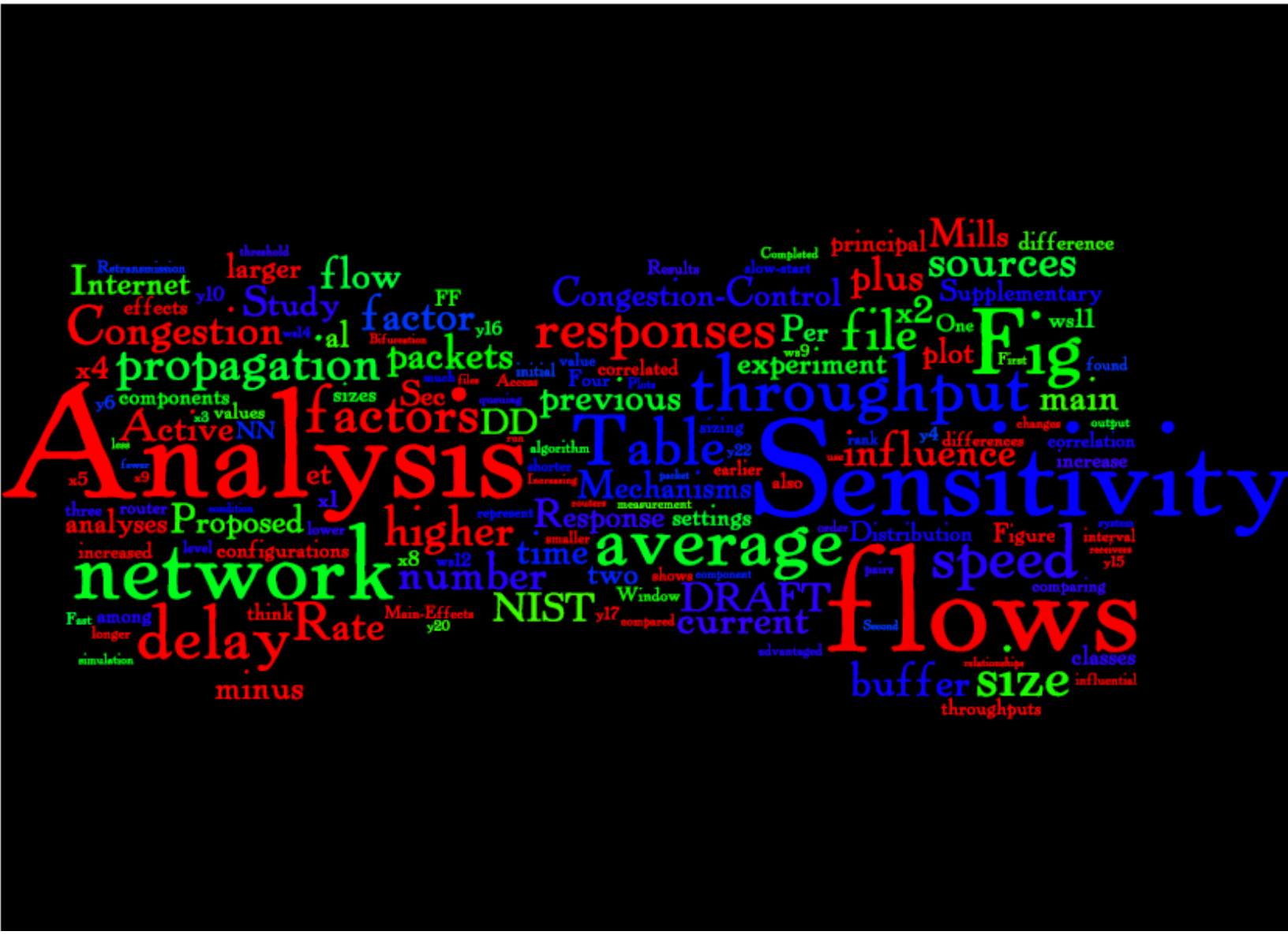


Figure B-4. Changes in Average Throughput with Increasing Buffer Size

Appendix C – Supplementary Sensitivity Analysis Results



Appendix C Supplementary Sensitivity Analysis Results

As pointed out in Chapter 2 and Chapter 4, two-level experiment designs exhibit some limitations, arising from the small number of values assigned to each parameter. First, conclusions drawn from such experiments are valid only for the combinations of levels investigated. Second, the system under investigation is assumed to exhibit monotonic behavior between the lower and higher values assigned to each factor. We addressed these shortcomings in part by running several experiments to explore system behavior with different pairs of values for selected factors and by subjecting the system to various scenarios. In this section, we provide a supplementary sensitivity analysis, repeating the experiment described in Chapter 4, but selecting alternate values for the two levels used for each of the 11 input factors. We expect this second sensitivity analysis to increase our confidence in MesoNet by confirming relationships consistent with earlier analyses, by identifying any new relationships not revealed previously and by allowing us to explain variances with previously established relationships. Increasing confidence in the validity of MesoNet should also increase confidence in our other experiments comparing behavior among various congestion control algorithms proposed for use on the Internet.

We begin in Sec. C.1 by describing our experiment design, which follows the same general approach explained in Chapter 4. In Sec. C.2, we outline how we executed the simulations to collect the required data. Next, in Sec. C.3, we present results regarding MesoNet sensitivity with respect to the two selected levels for each of 11 factors. In Sec. C.4, we compare and contrast our findings with those reported for the earlier sensitivity analysis (described in Chapter 4). We conclude in Sec. C.5.

C.1 Experiment Design

We adopt a 2^{11-5} orthogonal fractional factorial design, encoded with the same template shown previously in Fig. 4-1, and we use the 11 factors identified in Table 4-10, but here we select different values for the two levels assigned to each factor. Table C-1 identifies values we chose for the two levels of each factor. The reader may compare this with Table 4-11 to identify similarities and differences in factor settings between the current and earlier sensitivity analyses. Table C-2 defines values for selected fixed parameters. The three parameters highlighted in red have changed[1] from the previous sensitivity analysis: the network now contains more sources (**baseSources** is 10^3 instead of 100) and faster sources (**Hbase** is 8 p/ms instead of 1 and **Hfast** is 80 p/ms instead of 8). We deployed our sources over the same topology (recall Fig. 3-1), possessing defined link propagation delays (see Table 3-1) and leading to specific minimum round-trip times on designated routes (recall Table 3-2).

Three factors parameterize network properties, including propagation delay (x1), speed (x2) and buffer sizing (x3). Here, most markedly, we increase the network speed by an order of magnitude over the previous sensitivity analysis. The current experiment simulates network backbone speeds approximating up to 192 Gbps, while the previous experiment topped out at 9.6 Gpbs. Further, we increase the difference in speed to eightfold between the minus and plus levels, whereas the previous experiment evinced

[1] Unless otherwise noted, we assigned fixed parameters the same values used in Chapter 4.

only a doubling. The speed increase occurs for all router classes, as shown in Table C-3, because the speed of the backbone routers determines the relative speed of other routers. The increase in network speed justifies our choice to increase the number and speed of simulated sources and receivers in order to match the additional network capacity. We also extend, over the previous sensitivity analysis, the difference in propagation delays considered, increasing by a factor of two (plus level) or reducing in half (minus level) the base delays encoded within the topology. While we use the same two buffer-sizing algorithms adopted previously, we reduce buffer size computed from the RTTxC algorithm (plus level) by ½ and we increase buffer size computed from the RTTxC/sqrt(*n*) (minus level) by a factor of 2. The net effect of this modest change in buffer-sizing algorithm is overwhelmed by the increases in network speed and changes in propagation delay, both of which influence buffer size. In the previous sensitivity analysis, backbone buffer size averaged below 10^3 packets in 32 configurations and averaged above 16×10^3 packets in the other 32 configurations, reaching a maximum of just over 65×10^3 packets. In the current experiment (see Table C-4) backbone buffer size averaged below 10^3 packets in only 8 configurations, while exceeding 65×10^3 packets in 24 configurations. In general, the current experiment provides increased buffers under most configurations because the network speed has increased substantially.

Table C-1. Two-Level Settings for Each of 11 Factors in Sensitivity Analysis

	Factor	Factor Name	Plus Level	Minus Level
Network Factors	x1	Propagation Delay	2.5 times base delay	0.5 times base delay
	x2	Network Speed	16×10^3 p/ms	2×10^3 p/ms
	x3	Buffer Sizing	RTTxCx(Qfactor = 0.5)	RTTxC/sqrt(*n*)x(Qfactor = 2)
User Factors	x4	File Size	200 packets	25 packets
	x5	Think Time	10 seconds	1.25 seconds
	x6	Large File Probability	0.04	0.005
Source & Receiver Factors	x7	Fast Host Probability	0.80	0.10
	x8	Number of Sources	3 times base sources	1 times base sources
	x9	Source Distribution	0.1/0.3/0.6	0.3/0.1/0.6
	x10	Receiver Distribution	0.1/0.3/0.6	0.3/0.1/0.6
Protocol Factors	x11	Initial Slow-Start Threshold	1.07×10^9 packets	20 packets

User behavior is defined by three factors: average file size (x4), average think time (x5) and probability of transferring a (**Fx** = 10 times) larger file (x6). We increased the spread among the plus and minus values for each factor, when compared with the values chosen in Chapter 4. These choices implement a general strategy to increase the distance between the plus and minus settings for each factor in order to determine if such increases reveal strengthened relationships between factors and responses. For example, for the sole protocol factor, initial slow-start threshold (x11), we lower the minus level from 43 packets in the previous sensitivity analysis to 20, while keeping the plus value at the same arbitrarily high value used in Chapter 4.

Table C-2. Selected Fixed Parameters

Parameter	Name	Fixed Value
BBspeedup	Backbone Router Speed Multiplier	1
R2	POP Router Speed Divisor	4
R3	Access Router Speed Divisor	10
Bfast	Fast Access Router Speed Multiplier	2
Bdirect	Directly Connected Access Router Speed Multiplier	10
Hbase	Speed of Normal Network Interfaces	8 p/ms
Hfast	Speed of Fast Network Interfaces	80 p/ms
baseSources	Base Number of Sources Per Access Router	10^3
Fx	Large File Size Multiplier	10
alpha	File Size Shape Parameter	1.5

Table C-3. Router Speeds (p/ms) by Router Class for Each Level of Network Speed (x2)

Router Type	Plus	Minus
Backbone	16×10^3	2×10^3
POP	4×10^3	500
Typical Access	400	50
Fast Access	800	100
Directly Connected Access	4×10^3	500

Table C-4. Average Buffer Size (in packets) by Router Class for Specific Combinations of Propagation Delay (x1), Network Speed (x2) and Buffer-Sizing Algorithm (x3)

x1	x2	x3	Backbone Router Buffers (avg.)	POP Router Buffers (avg.)	Access Router Buffers (avg.)
-	-	-	368	140	48
+	-	-	3.36×10^3	1.306×10^3	435
-	+	-	4.453×10^3	1.789×10^3	600
+	+	-	12.335×10^3	5.308×10^3	1.751×10^3
-	-	+	20.800×10^3	5.200×10^3	827
+	-	+	102.182×10^3	25.545×10^3	4.062×10^3
-	+	+	166.400×10^3	41.600×10^3	6.614×10^3
+	+	+	817.455×10^3	204.363×10^3	32.492×10^3

The remaining factors determine the speed, number and distribution of the sources and receivers deployed in the topology. In this experiment, fast network interfaces for sources and receivers operate at a maximum of 80 p/ms (960 Mbps), while we parameterized slower network interfaces to operate at a maximum of 8 p/ms (96 Mbps). The probability that a given source or receiver has a fast network interface is determined by the fast host probability (x7), which we set to either .8 (plus) or .1 (minus), a .7 difference compared with the .2 difference used in the earlier sensitivity analysis. The previous experiment determined that this factor had little influence on system responses so we decided to increase the difference in probabilities in order to probe the invariance of this finding.

A combination of three factors, number of sources (x8) and distribution of sources (x9) and receivers (x10), determine the probability that flows go between specific combinations of access router classes: directly connected to directly connected (DD), directly connected to fast (DF), directly connected to normal (DN), fast to fast (FF), fast to normal (FN) and normal to normal (NN). We call these flow classes. Table C-5 shows the influence of these factors on the number and distribution of sources in the topology, while Table C-6 gives similar information regarding the number and distribution of receivers.

Table C-5. Relation between Factors and Number and Distribution of Sources

x8	x9	x10	Total Sources	% under D Routers	% under F Routers	% under N Routers
1	+	+	67.500×10^3	16	37.33	46.67
3	+	+	202.395×10^3	16	37.35	46.64
1	-	-	113.700×10^3	9.5	7.4	83.11
3	-	-	341.072×10^3	9.5	7.4	83.11
1	+	-	67.500×10^3	16	37.33	46.67
3	+	-	202.396×10^3	16	37.33	46.67
1	-	+	113.700×10^3	9.5	7.4	83.11
3	-	+	341.072×10^3	9.5	7.4	83.11

Table C-7 reports the influence of x8, x9 and x10 on the probability of various flow classes. Four combinations of parameters, the rows highlighted in purple, represent traffic patterns consistent with Web browsing augmented with some peer-to-peer (P2P) exchanges. Two parameter combinations, the rows highlighted in rose, represent traffic patterns with a slightly increased proportion of Web browsing compared to P2P exchanges. The remaining (white) rows show traffic patterns shifted substantially toward P2P traffic. The probabilities in Table C-7 represent a shift toward more Web-based traffic patterns when compared with the previous sensitivity analysis (see Table. 4-15), which had an even balance of configurations with Web and P2P traffic patterns. In addition, the P2P configurations in the current experiment represent a somewhat more

pronounced probability of DN, FN and NN flows. Further, the current experiment increases the proportion of DD flows for all configurations. Finally, while increasing the number of sources and receivers by about an order of magnitude over the previous sensitivity analysis, the current experiment expands the difference in number of sources and receivers among the configurations.

Table C-6. Relation between Factors and Number and Distribution of Receivers

x8	x9	x10	Total Receivers	% under D Routers	% under F Routers	% under N Routers
1	+	+	270.000×10^3	16	37.33	46.67
3	+	+	809.856×10^3	16	37.34	46.66
1	-	-	454.800×10^3	9.5	7.4	83.11
3	-	-	136.437×10^4	9.5	7.4	83.11
1	+	-	454.800×10^3	9.5	7.4	83.11
3	+	-	136.437×10^4	9.5	7.4	83.11
1	-	+	270.000×10^3	16	37.34	46.67
3	-	+	809.856×10^3	16	37.34	46.67

Table C-7. Relation between Factors and Distribution of Flow Classes

x8	x9	x10	% DD Flows	% DF Flows	% DN Flows	% FF Flows	% FN Flows	% NN Flows
1	+	+	2.56	11.95	14.93	13.94	34.84	21.78
3	+	+	2.56	11.95	14.93	13.94	34.84	21.76
1	-	-	0.9	1.4	15.79	0.55	12.28	69.08
3	-	-	0.9	1.4	15.79	0.55	12.28	69.08
1	+	-	1.52	4.73	17.73	2.76	34.48	38.79
3	+	-	1.52	4.73	17.73	2.76	34.48	38.79
1	-	+	1.52	4.73	17.73	2.76	34.48	38.79
3	-	+	1.52	4.73	17.73	2.76	34.48	38.79

To summarize the differences from the earlier sensitivity analysis: we increased network speed and size by an order of magnitude, we stretched the range of parameter values covered by the plus and minus settings of each factor, and we shifted the traffic patterns slightly to generate more DD flows and to give a higher prominence to Web browsing activity over P2P exchanges. These changes provided a very different set of configurations under which we could evaluate the relationship between model input

parameters and responses. We made no changes in fixed parameters controlling the simulation duration or the length of measurement intervals.

To permit ready comparison between results from both the earlier and current sensitivity analyses, we elected to measure the same responses in both experiments. One set of responses (repeated here as Table C-8) measured macroscopic behavior of the entire network and a second set of responses (see Table C-9) measured average instantaneous throughput on each of the six flow classes.

Table C-8. Responses Characterizing Macroscopic Network Behavior

Response	Definition
y1	Active Flows – flows attempting to transfer data
y2	Proportion of potential flows that were active: Active Flows/All Sources
y3	Data packets entering the network per measurement interval
y4	Data packets leaving the network per measurement interval
y5	Loss Rate: y4/(y3+y4)
y6	Flows Completed per measurement interval
y7	Flow-Completion Rate: y6/(y6+y1)
y8	Connection Failures per measurement interval
y9	Connection-Failure Rate: y8/(y8+y1)
y10	Retransmission Rate
y11	Congestion Window per Flow
y12	Window Increases per Flow per measurement interval
y13	Negative Acknowledgments per Flow per measurement interval
y14	Timeouts per Flow per measurement interval
y15	Smoothed Round-Trip Time
y16	Relative queuing delay: y15/(x1x41)

Table C-9. Responses Characterizing Average Instantaneous Throughput by Flow Class

Response	Definition
y17	Average Throughput for Active DD Flows
y18	Average Throughput for Active DF Flows
y19	Average Throughput for Active DN Flows
y20	Average Throughput for Active FF Flows
y21	Average Throughput for Active FN Flows
y22	Average Throughput for Active NN Flows

C.2 Experiment Execution and Data Collection

The experiment plan required 64 simulation runs, each simulating a different combination of factor settings, as constructed by mapping values from Table C-1 into the template shown in Fig. 4-1. We had 48 physical processors on which we could run our experiments, so we conducted simulations in parallel. We were sharing these processors with other projects, so we could not always use all of the available processors. Below, we give a brief discussion of the resource requirements for the simulations.

Table C-10 reports the characteristics of the 48 processors[2] available for our sensitivity analysis. Since MesoNet is implemented in SLX, each of the processors had access to an SLX simulation environment. SLX comes in two varieties: one configured to

[2] These 48 processors amounted to 6 servers that each had 8 processor cores.

run in a 32-bit address space and one configured to run in a 64-bit address space. We chose to run all our simulations using the 32-bit version of SLX because our simulations could fit easily within a 32-bit address space and 32-bit simulation runs faster than 64-bit simulation.

Table C-10. Configuration of Compute Servers for Simulations

Node	Processor Count	Speed (GHz)	Processor Type	Memory (GB)	Operating System
ws9	8	2.6	Dual-Core AMD Opteron 8218	32	Windows Server 2003 R2 x64 Edition SP2
ws10	8	2.6	Dual-Core AMD Opteron 8218	32	Windows Server 2003 R2 x64 Edition SP2
ws11	8	3.0	Dual-Core AMD Opteron 8222SE	32	Windows Server 2003 R2 x64 Edition SP2
ws12	8	3.0	Dual-Core AMD Opteron 8222SE	32	Windows Server 2003 R2 x64 Edition SP2
ws13	8	3.0	Dual-Core AMD Opteron 8222SE	32	Windows Server 2003 R2 x64 Edition SP2
ws14	8	3.0	Dual-Core AMD Opteron 8222SE	32	Windows Server 2003 R2 x64 Edition SP2

We executed the simulations continuously over about three days, starting as many simulations as available processors and then initiating a new simulation when one finished. Table C-11, organized into four three-column, color-coded groups of 16 simulations, reports the number of processor hours required by each simulation on a specified compute-server node. The average processor time for a simulation was 46.4 hours, about 5.5 times more than the average processor time required for the earlier sensitivity analysis, which simulated a slower and smaller network. The average memory used for a simulation was 1.1 Gbytes, nearly a tenfold increase over the earlier sensitivity analysis. We collected and summarized data using the same techniques adopted for the earlier sensitivity analysis. Refer to Sec. 4.3.2 for the details.

Table C-11. Execution Time (Processor Hours) Required for Each Simulation Run

Run	Node	Time	Run	Node	Time	Run	Node	Time	Run	Node	Time
1	ws9	36.6	17	ws11	11.4	33	ws14	36.4	49	ws11	7.8
2	ws9	14.8	18	ws11	1.3	34	ws14	23.4	50	ws11	3.7
3	ws9	14.7	19	ws11	1.3	35	ws14	29.8	51	ws11	3.2
4	ws9	70.3	20	ws11	8.0	36	ws14	46.8	52	ws12	7.4
5	ws9	27.7	21	ws11	2.6	37	ws14	17.8	53	ws12	1.9
6	ws9	32.2	22	ws11	6.8	38	ws14	34.6	54	ws9	14.9
7	ws9	56.8	23	ws11	4.9	39	ws14	96.0	55	ws9	15.8
8	ws9	19.8	24	ws11	2.4	40	ws11	14.2	56	ws9	1.8
9	ws10	31.0	25	ws12	22.9	41	ws11	30.7	57	ws14	19.5
10	ws10	47.1	26	ws12	30.3	42	ws11	42.9	58	ws14	36.3
11	ws10	233.7	27	ws12	48.7	43	ws11	218.9	59	ws11	100.0
12	ws10	156.8	28	ws12	20.7	44	ws13	78.4	60	ws11	15.6
13	ws10	42.5	29	ws12	35.8	45	ws14	36.5	61	ws9	43.3
14	ws10	32.7	30	ws12	11.0	46	ws13	31.4	62	ws12	27.7
15	ws10	97.0	31	ws12	10.7	47	ws11	172.8	63	ws12	23.1
16	ws10	238.6	32	ws12	70.2	48	ws11	239.0	64	ws12	55.1

C.3 Results

Below, we report results from subjecting (22 x 64 =) 1408 responses to three treatments: correlation analysis, principal components analysis and main effects analysis. We also employed some exploratory analyses, as used in Chapter 4. We discuss each analysis in turn.

C.3.1 Correlation Analysis

Given 64 average values (one per run) for 22 responses, we conducted a correlation analysis to investigate the degree to which pairs of responses are linearly correlated. We used the same techniques applied in the earlier sensitivity analysis (Sec. 4.4). We began by generating a scatter plot and computing the correlation for each pair of responses, as plotted together in Fig. C-1, which should be interpreted as explained earlier in Sec. 4.1.3.3. Of particular interest, correlations with magnitudes of .8 and above are colored red, magnitudes between .3 and .79 are blue and magnitudes below .3 are green.

Figure C-1. Combined Matrix of Scatter Plots and Correlation Values for 22 Responses

Scanning Fig. C-1 reveals some mutual correlations, for example among responses y17 through y22, which represent throughput for various flow classes. The figure also reveals some strongly correlated pairs: y1 and y2 (active flows and proportion of sources that are active), y3 and y4 (packets input and output), y5 and y10 (loss rate and retransmission rate), y8 and y9 (connection failures and connection-failure rate) and y13

and y14 (negative acknowledgments and timeouts). A few responses (e.g., y6 and y16) appear largely uncorrelated with other responses. Comparing Fig. C-1 with Fig. 4-13 from the earlier sensitivity analysis, one finds fewer strong correlations overall in the current experiment. Also of note, unlike the previous experiment, which correlated throughput for flow classes into three groupings, the current experiment shows strong positive correlation in throughput among all flow classes.

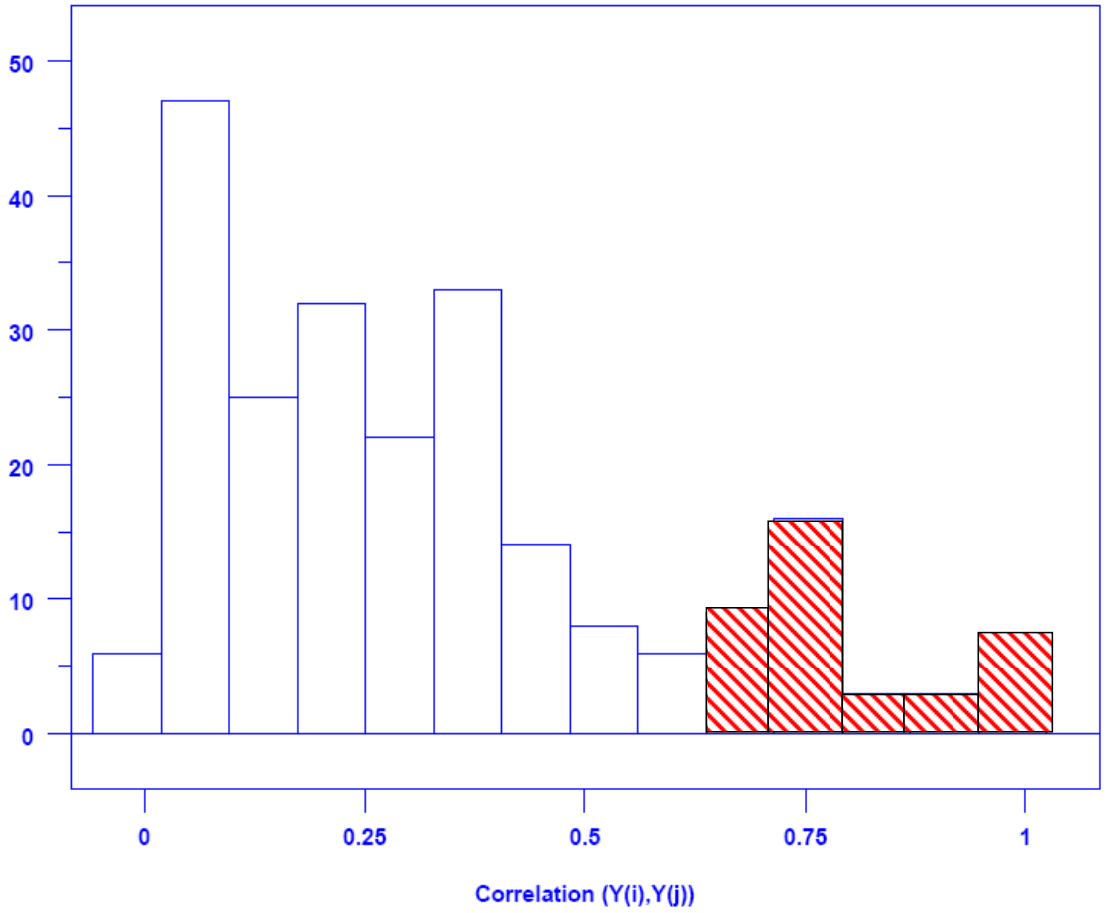


Figure C-2. Frequency Distribution of the Absolute Value of Correlations for All Pairs of Responses

Fig. C-2 shows the frequency distribution of the absolute value of correlations for all pairs of responses. The five, highlighted bins represent correlations with magnitude greater than 0.65, which we chose (consistent with our choice in Chapter 4) as the cutoff for correlations to be considered significant. Comparing Fig. C-2 with Fig. 4-14 from the earlier analysis indicates that the current analysis has 36 pairs that are significantly correlated, while the earlier analysis found 42 such pairs. We show (Fig. C-3) the 36 significantly correlated pairs as an index-index plot, ordering the indices on both axes as ordered in Fig. 4-14 in order to facilitate comparison. Comparing Fig. C-3 with Fig. 4-14 confirms that throughput among flow classes, previously organized into three groups ([y17], [y18, y20] and [y19, y21, y22]) are now mutually correlated. Other changes can also be noted. For example, y15 (round-trip time) and y16 (relative queuing delay) are no longer correlated. Further, y8 (connection failures), y9 (connection-failure rate) and y14 (timeouts) remain correlated but are no longer correlated with y5 (loss rate) and y10

(retransmission rate). In the current experiment, y5 (loss rate) and y10 (retransmission rate) are correlated with the y1 and y2 (active flows and proportion of sources active); the number of active flows was not a factor in the previous sensitivity analysis. This difference likely arises because the current experiment uses a faster network, requiring more active flows to generate load. In the current sensitivity analysis, congestion seems driven by the number of active flows. Throughput among all flow groups now seems correlated and thus likely driven by some common factors, but note that congestion window size (y11) is not correlated with throughput on DD flows (y17). In fact, the y11-y17 correlation is 0.60, which falls just below our cutoff (0.65). Finally, round-trip time (y15) and queuing delay (y16) are now uncorrelated. These correlation changes are considered in the discussion (Sec. C.4) after assessing the main factors that influence model responses.

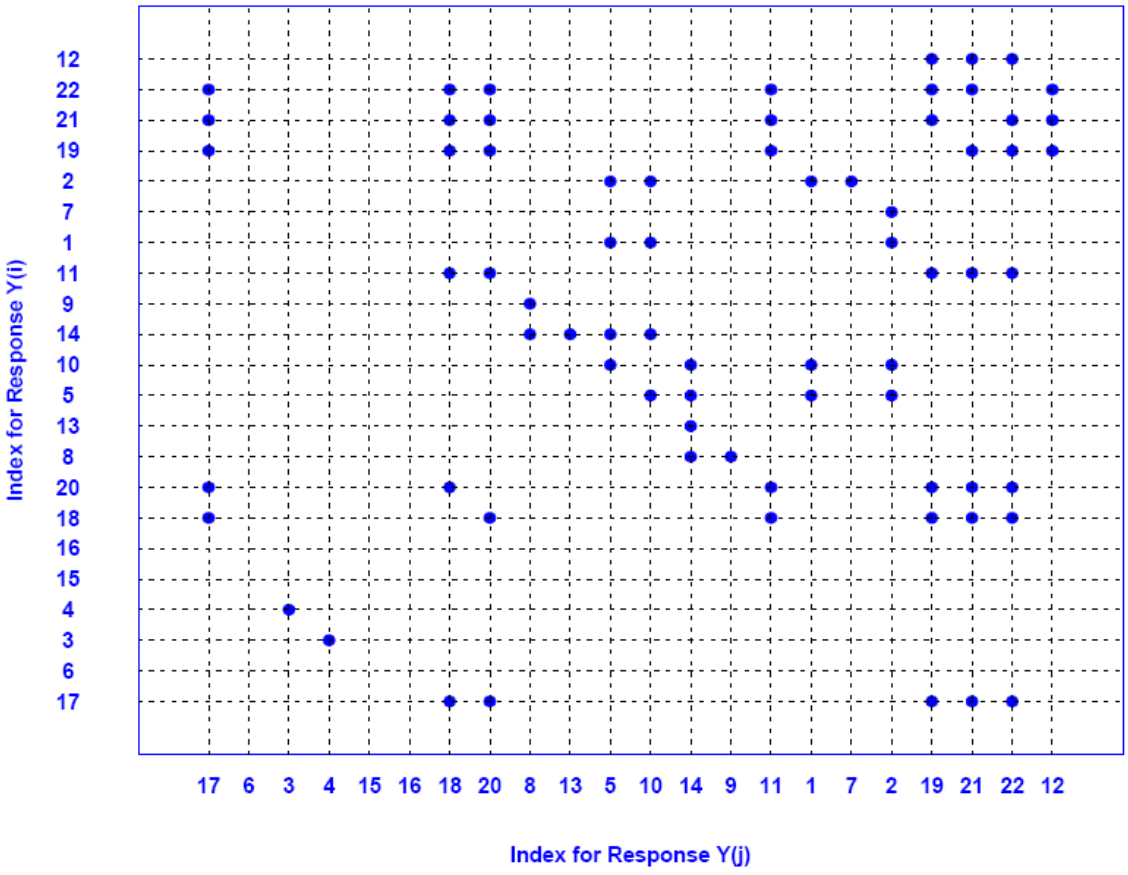


Figure C-3. Index-Index Plot for Correlation Pairs where |Correlation (Y_i, Y_j)| > 0.65

C.3.2 Principal Components Analysis

Given the changes noted in the correlation analysis, we should expect to see some changes in the principal components analysis (PCA) as well. Fig. C-4 shows the PCA for all 22 responses generated in the current experiment. The first four principal components account for 95 % of the response variance and thus we select these components for

further examination, as given in Fig. C-5, where we plot the relevant weight vectors. Comparing Fig. C-5 with Fig. 4-17 from the previous sensitivity analysis illustrates that the top four principal components have changed configurations. In fact, the principal components from the current experiment appear more difficult to interpret than those from the previous experiment. To provide additional information, we introduce Fig. C-6, containing four main effects plots, one per principal component, analogous to Figs. 4-26 through 4-29. Fig. C-6 shows that many of the same factors influence the top four principal components, suggesting we will be unable to find a clear and satisfying interpretation. We will do the best we can.

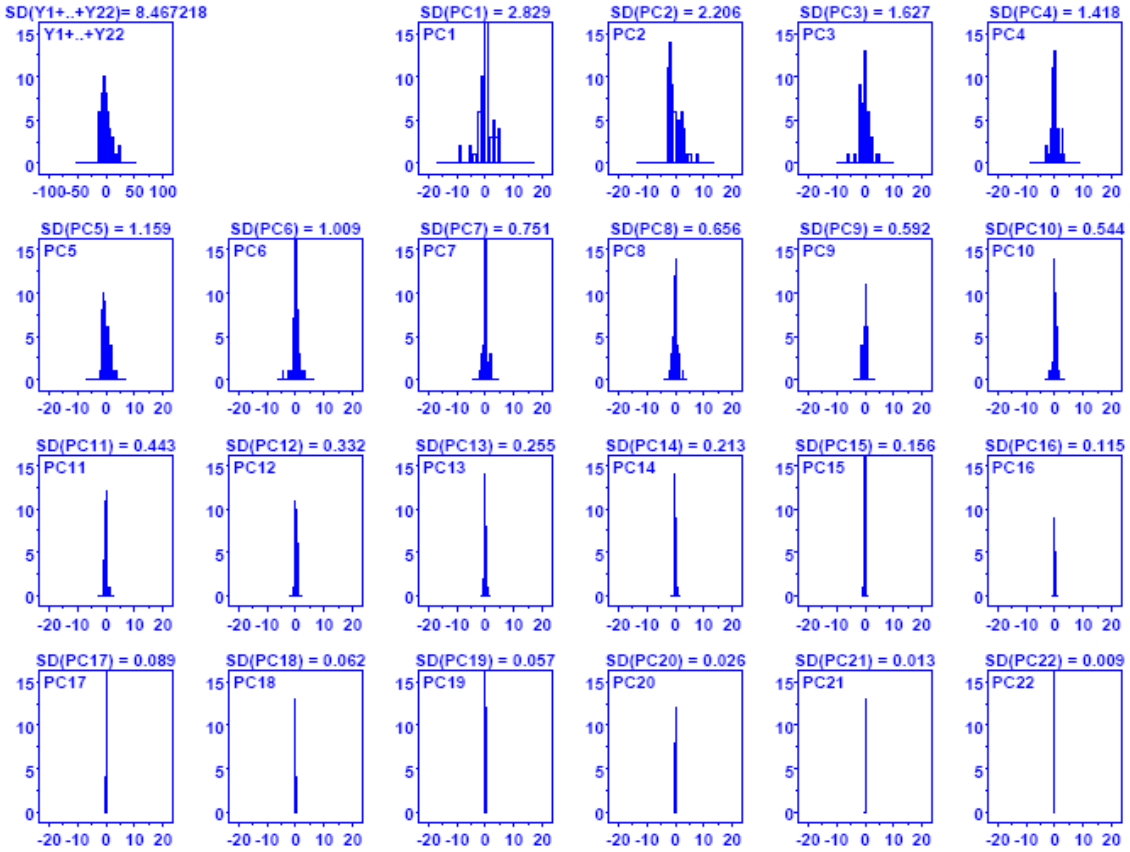


Figure C-4. Histograms for 22 Principal Components (x axis of each sub-plot identifies bins of normalized component values ranging from -20 to +20 and y axis the count of values within each bin). Above each sub-plot is the standard deviation in the data accounted for by the Principal Component. The first sub-plot gives the distribution of the normalized responses.

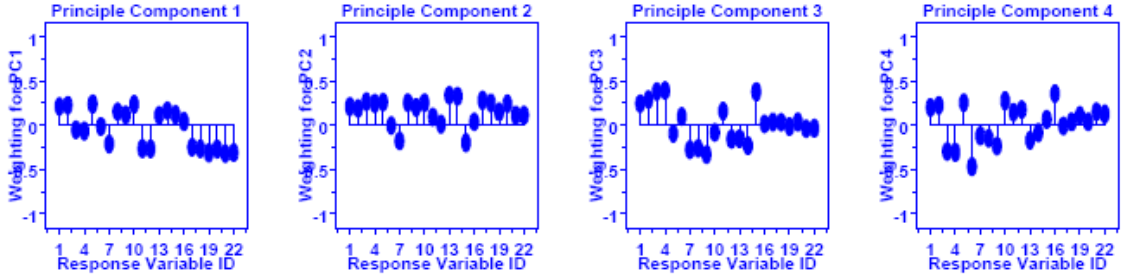


Figure C-5. Weight Vectors for the First Four Principal Components

The first principal component groups 12 responses in four main categories: active flows (y1 and y2), loss and retransmission rates (y5 and y10), congestion window size and increase rate (y11 and y12) and throughput among all flow classes (y17-y22). The main effects plot for PC1 indicates that slower network speed, longer propagation delay, shorter think time and more sources lead to a positive component value – a negative component value is produced by the opposite setting for these factors. From this, we may infer that higher network congestion yields a positive value for PC1 and lower network congestion yields a negative value. This inference suggests that PC1 represents the influence of network congestion (y1, y2, y5 and y10) on congestion window size (y11) and increase rate (y12), which determines throughput on flows of all classes (y17-y22).

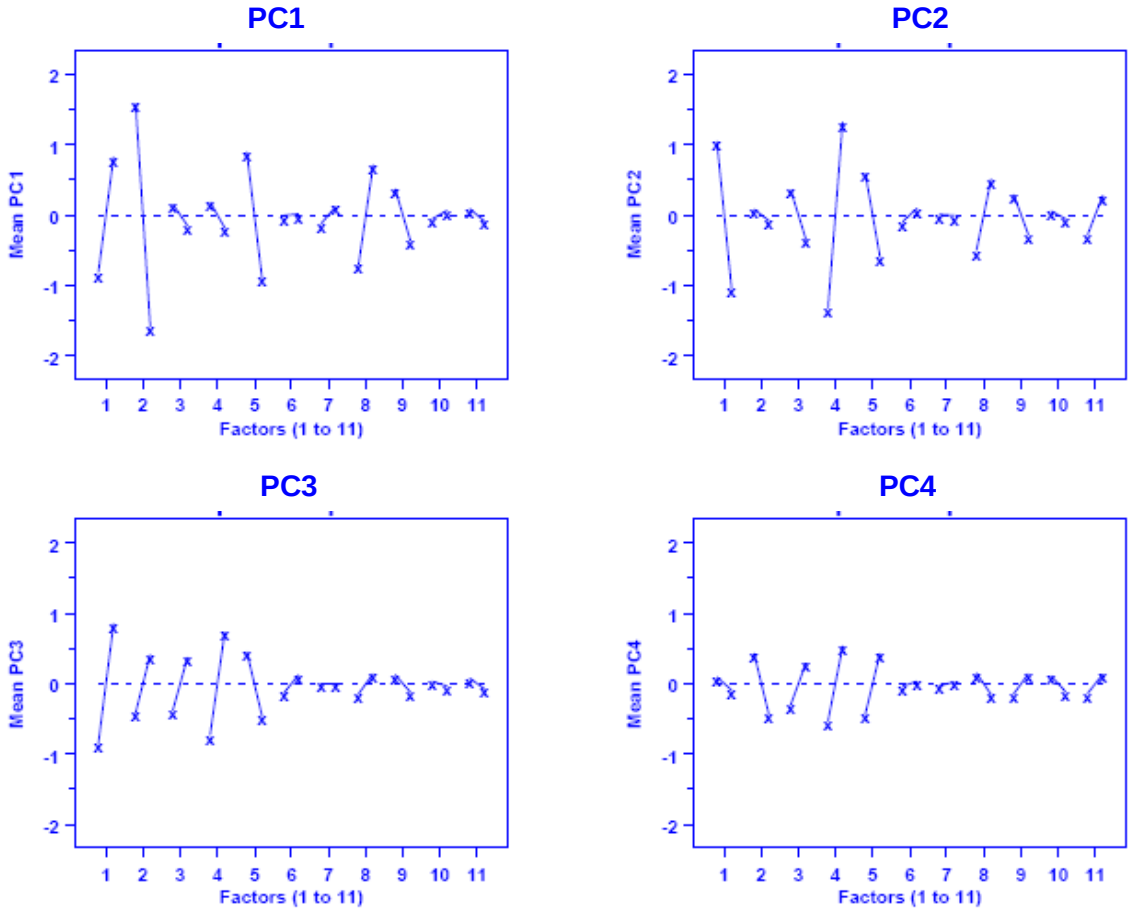


Figure C-6. Main Effects Plots for Top Four Principal Components

The second principal component appears to characterize throughput on more advantaged DD (y17), DF (y18) and FF (y20) flows. Provided congestion is not too heavy, larger file sizes (x4) and shorter propagation delays (x1) should lead to higher throughputs, especially for advantaged flows. The main effects plot for PC2 suggests that higher throughputs are coincident with a positive value of the component. At the same time, the network would transport more packets (y3 and y4) for more flows (y1 and y2), which would lead to more losses (y5) and retransmissions (y10), especially for less advantaged flows (y19, y21 and y22), which would experience more negative acknowledgments (y13) and timeouts (y14).

The third principal component appears to characterize queuing delay (y16), which is grouped together with the number of packets flowing in (y3) and out (y4) of the network. Referring to the main effects plot for PC3, longer propagation delay (x1), faster network speed (x2), larger buffer sizes and larger file sizes (x4) generate a positive value for the component. We can infer that such conditions permit the larger network buffers to hold more packets, leading to longer queuing delays.

The fourth principal component appears to characterize network throughput measured in terms of flows completed (y6) and packets transferred (y3 and y4). Shorter files sizes (x4) combine with higher network speed (x2) and shorter think times (x5) to permit more flows to be completed per unit time (y6). Comparing the main effects plot for PC4 from Fig. C-6 with the main effects plot for PC4 from Fig. 4-29 reveals similarity.

The forgoing discussion illustrates that the PCA conducted for the second sensitivity analysis produced results more difficult to interpret than was the case for the original sensitivity analysis reported in Chapter 4. For this reason, we postpone until the discussion (Sec. C.4) further consideration of the principal components.

C.3.3 Exploratory Analysis of y7-y22 Scatter Plot Bifurcation

In Fig. 4-10 of Sec. 4.1.6, we reported a scatter plot of y7 (flow completion rate) vs. y22 (throughput on NN flows) that showed a bifurcation. We used an exploratory technique, altering the plot symbols to represent minus and plus settings for each factor, to discover that the bifurcation arose due to factor x4 (average file size). Shorter file sizes resulted in higher completion rates (y7) and yet led to lower average throughputs for NN flows (y22). This made sense because shorter files spend a higher percentage of their transfer in TCP slow start, during which throughputs are lower. On the other hand, shorter files are generally transferred more quickly because they involve fewer packets. Since shorter files take less time, more flows complete per unit of time and the flow completion rate is higher. Longer files spend a higher percentage of their transfer beyond TCP slow start, during which throughputs are higher. On the other hand, longer files require transferring more packets, taking more time and completing fewer flows per unit of time. Since we increased the distance between the minus and plus file sizes (from 50/100 to 25/200), we expected the bifurcation to appear in enhanced form in this sensitivity analysis.

Fig. C-7 shows twelve y7-y22 scatter plots generated from the 64 simulations in the current experiment. The first scatter plot contains the bifurcation data. Each of the next 11 plots distinguishes the minus and plus level settings for a given factor. The plots clearly identify average file size (x4) as the factor responsible for the bifurcation. Comparing Fig. 4-10 with Fig. C-7 shows that, as expected, the bifurcation is enhanced in the current sensitivity analysis.

C.3.4 Main Effects Analysis

In this section, we provide main effects plots generated from the data captured in the current experiment. To facilitate comparison we plot main effects for the same responses used in the original sensitivity analysis, including those listed in Table 4-19, and adding the average congestion window size, used previously as an example in Fig. 4-9. Analyzing the same responses should allow us to identify similarities and differences in factor-response relationships between the two sensitivity analyses. We expect that most

of the factor-response relationships will remain unchanged, but differences in the correlation and principal components analyses suggest that some relationships might be different. We organize the exposition into four categories, responses related to congestion, responses related to delay, responses related to macroscopic throughput and responses related to throughput for advantaged flows, as shown in Table C-12.

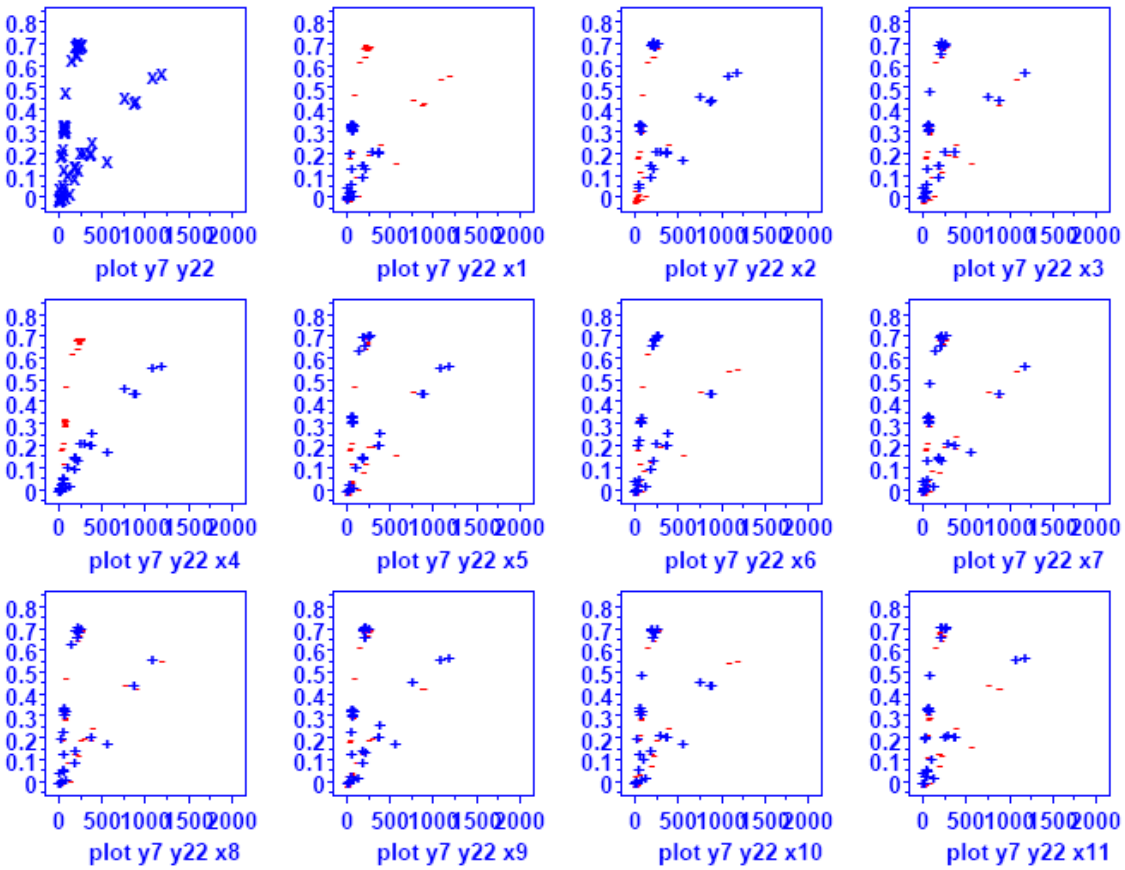


Figure C-7. Y-Y-X plot for Responses y7 and y22

Table C-12. Responses Selected for Investigation in Sensitivity Analysis

	Response	Definition
Congestion	y1	Average number of active flows
	y10	Average retransmission rate
	y11	Average congestion window size
	y22	Average instantaneous throughput for NN flows
Delay	y15	Average smoothed round-trip time
Macroscopic Throughput	y4	Average number of packet output per measurement interval
	y6	Average number of flows completed per measurement interval
Advantaged Flows	y17	Average instantaneous throughput for DD flows
	y20	Average instantaneous throughput for FF flows

C.3.4.1 Congestion-Related Responses. We begin by examining the main factors influencing the average number of active flows (y1). Fig. C-8 shows the relevant main effects plot, which can be compared with Fig. 4-18. We find the same main factors influencing the number of active flows in both experiments, though the order of the factors shifts slightly. The main factors appear to fall into three categories: (a) number of sources underneath N-class access routers, (b) duration for which flows remain active and (c) idle interval for those sources. The number of sources (x8) claims the main influence, followed by the average file size (x4). The longer it takes to transfer files, the more likely flows are to be active. The duration of transfer time is influenced not only by file size, but also by network speed[3] (x2) and congestion, which is influenced by number (x8) and distribution (x9) of sources and by average think time (x5). These relationships are evident in Figs. C-8 and 4-18.

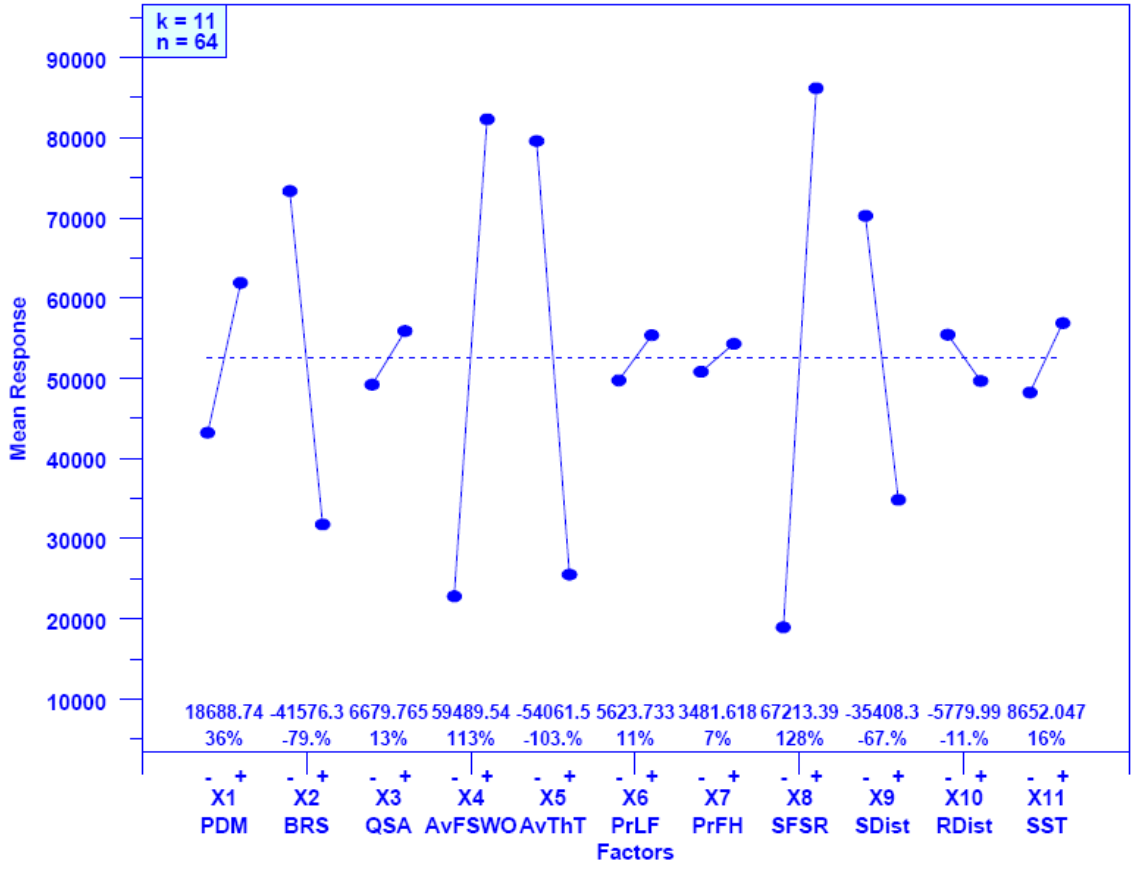


Figure C-8. Main-Effects Plot for Response y1 (Average Number of Active Flows) – x axis lists 11 model parameters with a – and + value for each parameter, y axis gives average number of active flows, two averages are given for each parameter, one average when the parameter is set to its – level and one average when the parameter is set to its + level, and a line connects the pair of average sizes for each parameter. Dashed line is the overall average number of active flows (about 52.250 x 10^3 here)

[3] Recall that in the previous sensitivity analysis we miscoded network speed (minus was higher network speed) and distribution of sources (plus was a more P2P-like traffic pattern). In this sensitivity analysis the levels were properly coded, so care should be taken in comparing the slope for these factors on the main effects plots in Chapter 4 against the main effects plots in Appendix C.

Fig. C-9 gives the main-effects plot for retransmission rate (y10). Comparing this with Fig. C-8 shows that the same factors influence both the number of active flows and retransmission rate. This mirrors the same relationship found in Chapter 4, where Fig. 4-18 and Fig. 4-19 also show the same influential factors. Figs. C-9 and 4-19 exhibit one main difference: buffer sizing algorithm does not play as significant a role in Fig. C-9. This makes sense because buffer sizing for the former sensitivity analysis led to very small buffer sizes under the minus setting. Buffer sizes were not as constrained under the minus setting in the current sensitivity analysis. One other difference can also be discerned: the number of sources plays a larger role and the distribution of sources a smaller role in Fig. C-9 than in Fig. 4-19. This makes sense because in the current sensitivity analysis the variation in number of sources was larger and the variation in the distribution of sources was smaller.

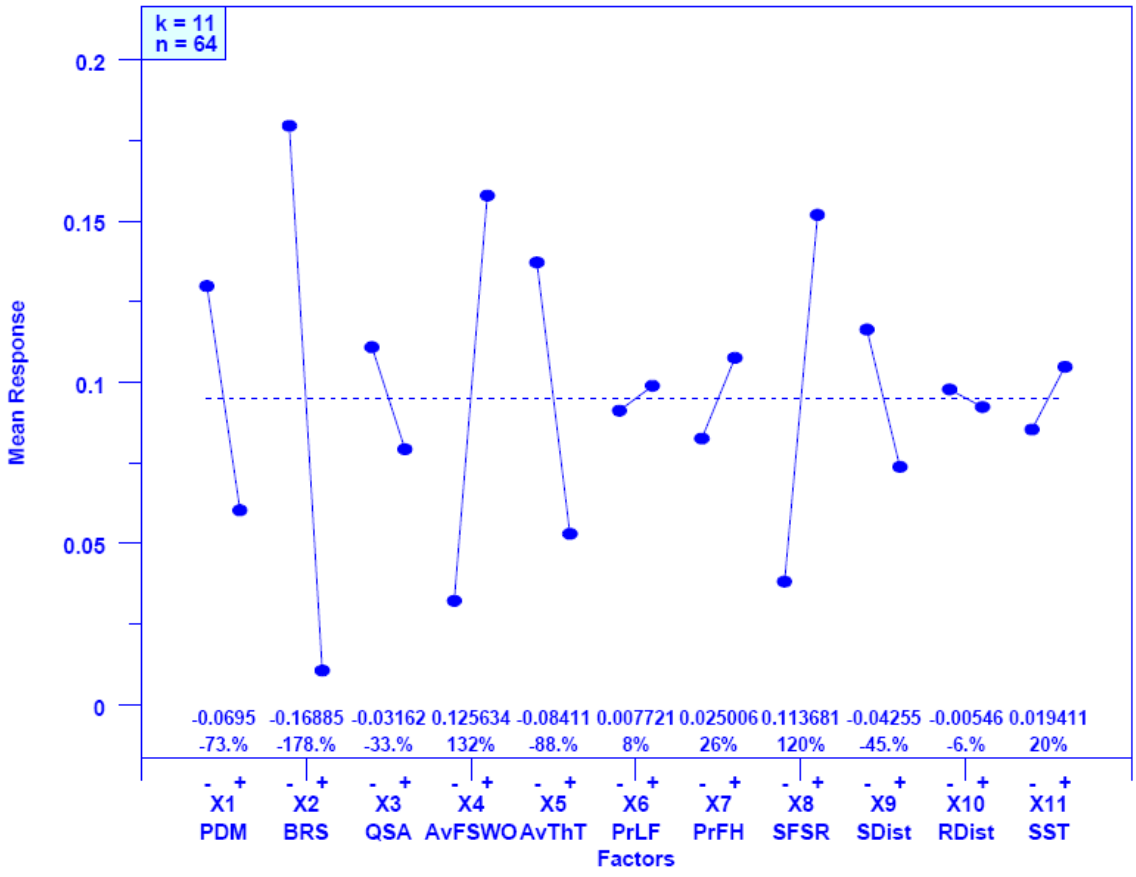


Figure C-9. Main-Effects Plot for Response y10 (Average Retransmission Rate) – y axis gives the proportion of packets resent

Fig. C-10 gives the main-effects plot for average congestion window (CWND) size. This figure can be compared with Fig. 4-9 when assessing similarities and differences among the two sensitivity analyses. Fig. C-10 reveals the CWND size is influenced mainly by two factors: average file size (x4) and network speed (x2). This differs somewhat from Fig. 4-9, which identified network speed (x2) as the main factor followed by four closely grouped factors: buffer sizing algorithm (x3), initial slow-start

threshold (x11), think time (x5) and distribution of sources (x9). We previously explained why buffer sizing algorithm and distribution of sources should be less influential with respect to congestion in the current sensitivity analysis. What about other differences between Figs. C-10 and 4-9?

Fig. C-10 identifies average file size (x4) as a significant factor, while Fig. 4-9 does not. In the current sensitivity analysis the difference between the plus and minus settings for average file size was increased substantially, which accounts for the increase in influence of factor x4. Average think time (x5) is the third most significant factor in both sensitivity analyses. What accounts for the diminished influence in the initial slow-start threshold? This appears related to increased congestion. The factor settings for the current sensitivity analysis allowed higher levels of congestion, as expressed for example by retransmission rates, which averaged around 9.5 % compared with only 8.8 % in the previous sensitivity analysis. Higher levels of congestion reduce the influence of the initial slow-start threshold because flows can incur lost packets sooner, and thus transition to congestion avoidance sooner.

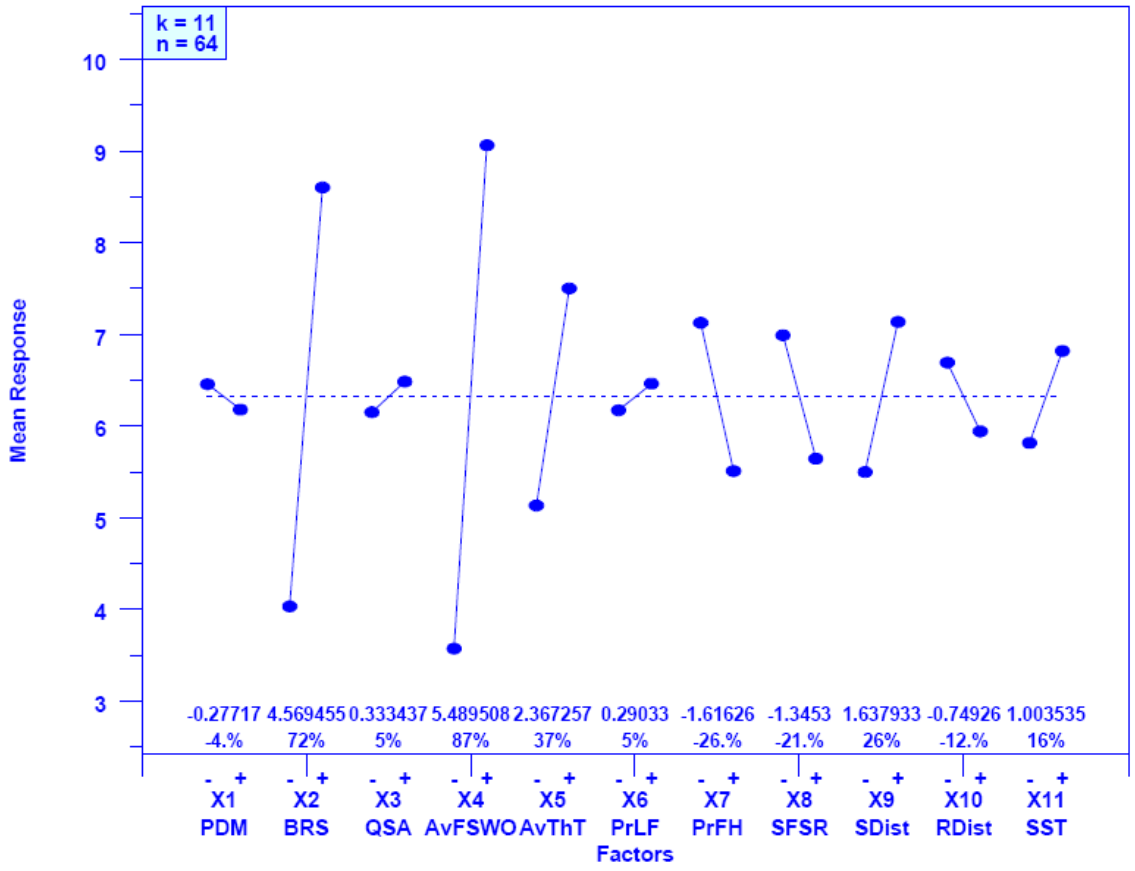


Figure C-10. Main-Effects Plot for Response y11 (Average Congestion Window Size) – y axis gives average congestion window size in packets

Fig. C-11, which can be compared with Fig. 4-20, displays main effects driving throughput on NN flows. The previous sensitivity analysis showed throughput on NN flows to be driven primarily by a relationship between available bandwidth (network speed) and number of active flows. Fig. C-11 also reflects this relationship: throughput is

higher under increased network speed (x2) when fewer flows are active. Factors leading to fewer active flows include a lower number of sources (x8 minus) and a source distribution (x9 plus) that leads to fewer NN flows, as well as longer think times (x5 plus). The main differences between Fig. C-11 and Fig. 4-20 relate to buffer sizing algorithm (previously explained) and average file size (x4). In the previous sensitivity analysis we found, surprisingly, that smaller file sizes led to higher throughputs on NN flows. This finding was surprising because larger file sizes can generally achieve higher throughputs. We attributed this to the fact that smaller files finished more quickly, which helped to reduce the number of active flows. This attribution made sense because the difference in average file size between the plus and minus settings was relatively small (50 packets), so file size could not have much influence on throughput. In the current sensitivity analysis, the difference in average file size between the plus and minus settings was significantly larger (175 packets). These larger files can achieve much higher instantaneous throughput than the much smaller files.

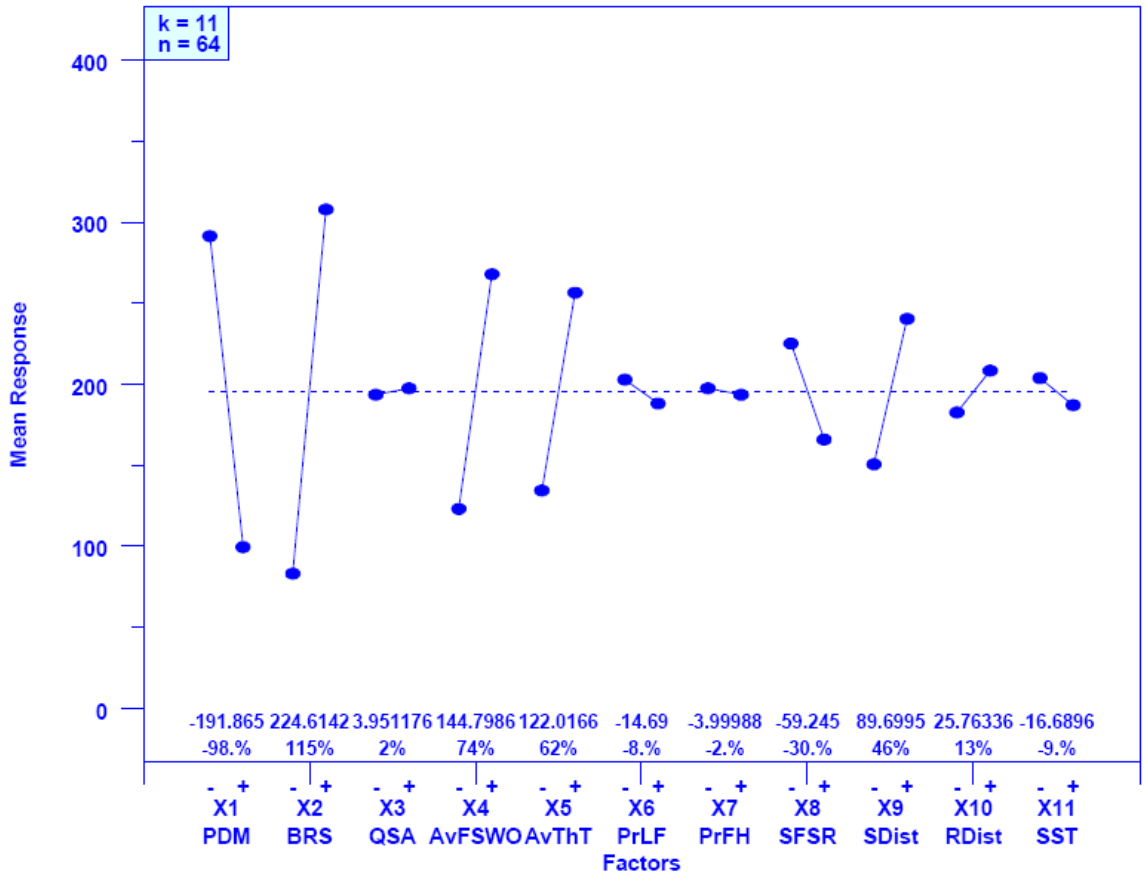


Figure C-11. Main-Effects Plot for Response y22 (Average Throughput on NN Flows) – y axis gives average goodput in pps

C.3.4.2 Delay-Related Responses. Fig. C-12 reports the influence of each input factor on response y15: average smoothed, round-trip time (SRTT). Both Fig. C-12 and the comparable Fig. 4-21 identify propagation delay (x1) and buffer sizing algorithm (x3) as the two main factors influencing SRTT. The influence of buffer sizing algorithm is less

significant in Fig. C-12 because fewer configurations exhibit small buffer sizes in the current sensitivity analysis.

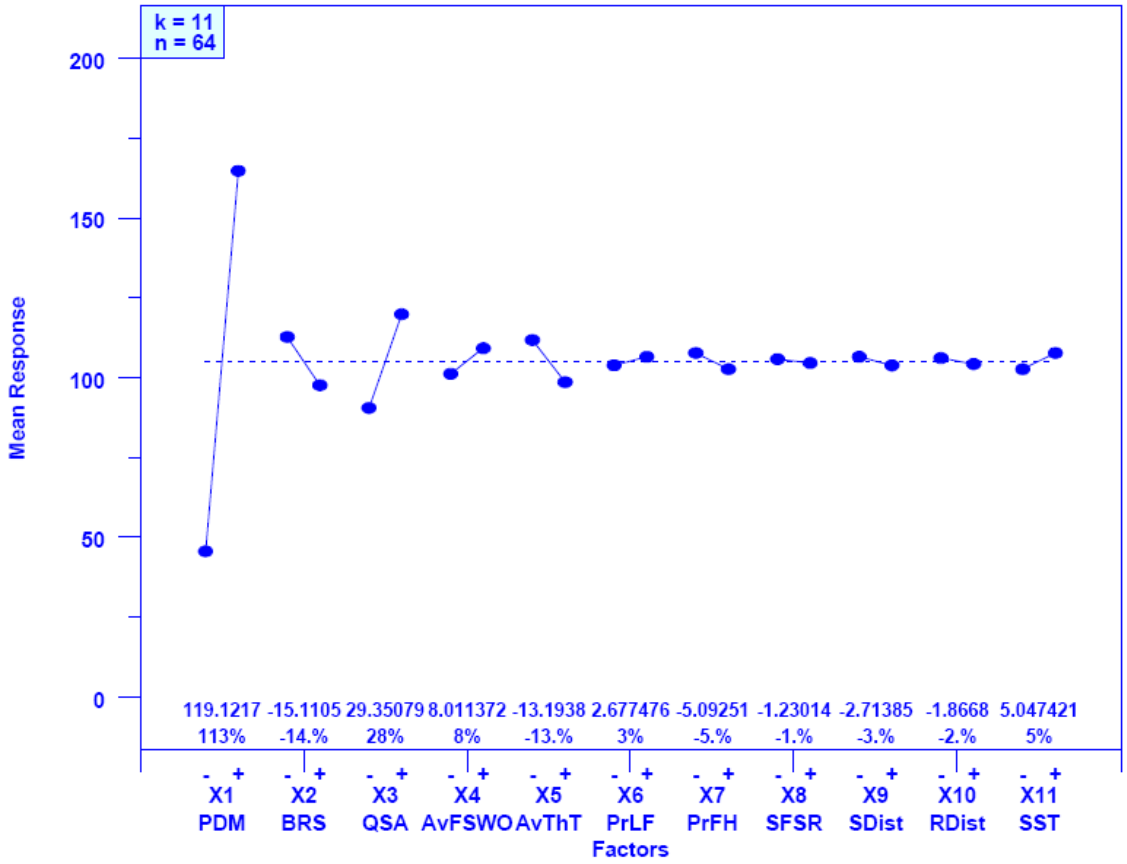


Figure C-12. Main-Effects Plot for Response y15 (Average Smoothed Round-Trip Time) – y axis gives average round-trip time in ms

C.3.4.3 Responses Related to Macroscopic Throughput. To represent macroscopic network throughput, we selected two responses: data packets output per interval (y4) and flows completed per interval (y6). The first response represents the rate at which packets are flowing through the network, while the second response represents the rate at which flows are being completed by the network. We begin by considering the rate of packet output.

Fig. C-13 identifies the same main factors influencing rate of packet output as revealed in Fig. 4-22. The main influence on the rate of packet output is network speed: higher network speed (x2 plus) means a greater rate of packet output. This stands to reason in a network with a sufficient number of active flows. The combination of shorter think times (x5 minus) and more sources (x8 plus) leads to an increase in the number of active flows and the higher network speed implies that each flow can transmit faster. Thus, the aggregate rate of packet output should be greater under these circumstances. File size is another factor significantly affecting the rate of packet output. Larger file sizes (x4 plus) lead to greater throughputs because a smaller portion of the transfer occurs during slow-start, the transfer phase during which a flow's congestion window is lowest.

Flows transferring with a larger congestion window achieve higher throughput, which helps to increase the aggregate network throughput.

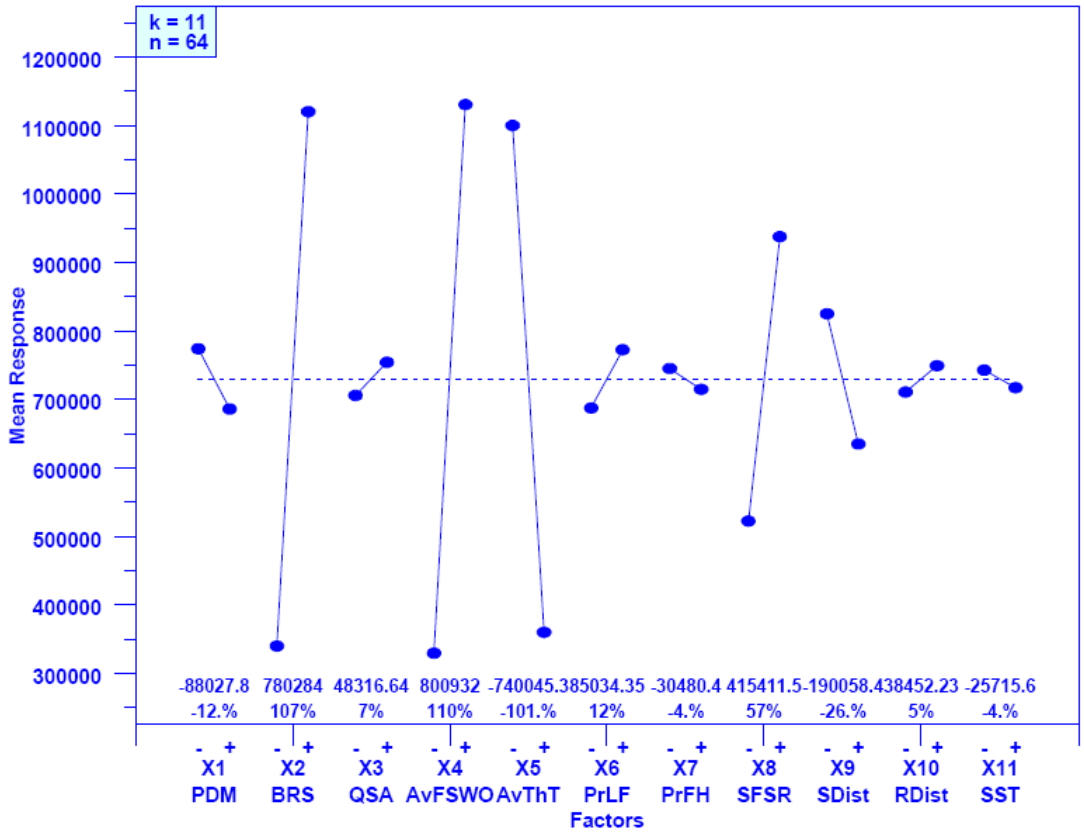


Figure C-13. Main-Effects Plot for Response y4 (Average Packets Output per Measurement Interval) – y axis gives average number of packets output per 200 ms

As shown in Fig. C-14 (and also Fig. 4-23), with one major exception, the story regarding the rate of flow completions is quite similar to the story regarding the rate of packet outputs. A sufficient number of connections (x5 minus and x8 plus) combined with higher network speed (x2 plus) contributes to a higher rate of flow completion. The exception involves file size (x4). In the case of packets output, larger file sizes (x4 plus) led to higher throughputs and thus to more packets output. On the contrary, for flows completed, a smaller average file size led to a higher completion rate. This stands to reason; smaller flows will be completed sooner. The sooner flows can be completed, the more flows can be completed per unit of time.

C.3.4.4. Responses Related to Advantaged Flow Classes. The final two responses we investigate represent throughputs achieved over advantaged flow classes, which are flows that transit between sources and receivers located under directly-connected and fast access routers. We examine the average instantaneous throughput of DD (y17) and FF (y20) flows. We begin by considering DD flows.

In the previous sensitivity analysis (see Fig. 4-24) we found that throughput on DD flows was influenced by only two factors: propagation delay (x1) and file size (x4). Shorter propagation delay (x1 minus) permitted faster feedback on DD flows, which

allowed the congestion window to increase more quickly. The rate of feedback was most important during the initial slow-start phase, where the congestion window started at a small size but doubled with each acknowledgment received. The influence of file size was also clear. Larger file sizes (x4 plus) allowed more of the packets in a file to be transferred after the flow reached its peak sending rate. Smaller file sizes (x4 minus) implied that more of the packets in a file were sent early in the slow-start phase, when a flow is building up toward its peak sending rate. Throughput early in slow-start will be much smaller than throughput after a flow reaches its peak rate.

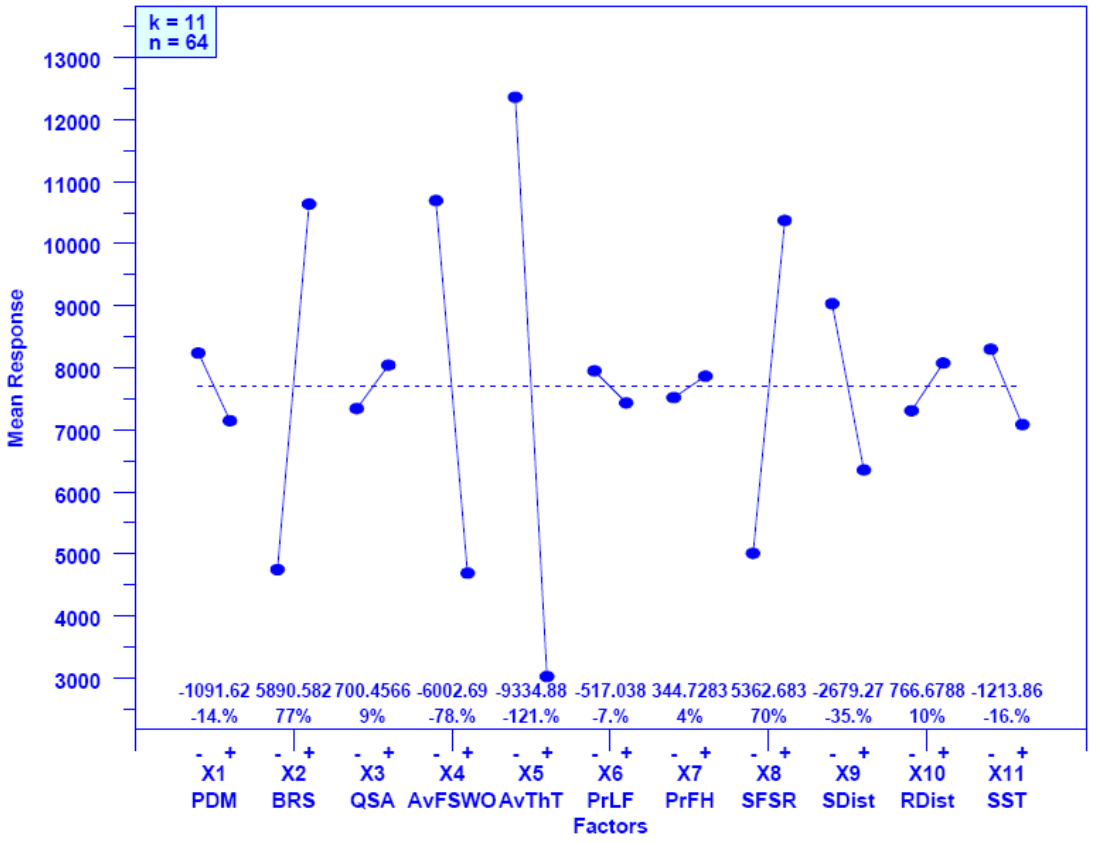


Figure C-14. Main-Effects Plot for Response y6 (Flows Completed per Measurement Interval) – y axis gives average number of flows completed per 200 ms

The current sensitivity analysis (Fig. C-15) also identifies propagation delay (x1) and file size (x4) as the two main factors influencing throughput on DD flows. Fig. C-15 also identifies network speed (x2) as a significant factor. This makes sense because the speed difference between the plus and minus settings was much larger (14×10^3 p/ms) here than in the previous sensitivity analysis (400 p/ms). The higher difference in network speed would certainly contribute to a larger throughput on DD flows. Fig. C-15 identifies a few other factors having some influence. For example, lower congestion arising from fewer sources (x8 minus) and longer think times (x5 plus) allow for higher throughputs on DD flows. Finally, the larger difference in file sizes (175 packets instead of 50 packets) enables a higher initial slow-start threshold to contribute more to higher throughputs on DD flows.

In the previous sensitivity analysis (see Fig. 4-25) we found throughput on FF flows to be influenced by a more complex mix of factors than DD flows. The significance of propagation delay (x1) and file size (x4) were two clear common factors between all advantaged flow classes. Shorter propagation delay meant quicker feedback, which led to faster increase in the congestion window for flows that were not impeded by congestion. Larger file sizes allowed more of a flow's packets to be transferred at a higher sending rate. Less advantaged (DN, FN and NN) flows were influenced mainly by congestion, so propagation delay had less affect on those flows.

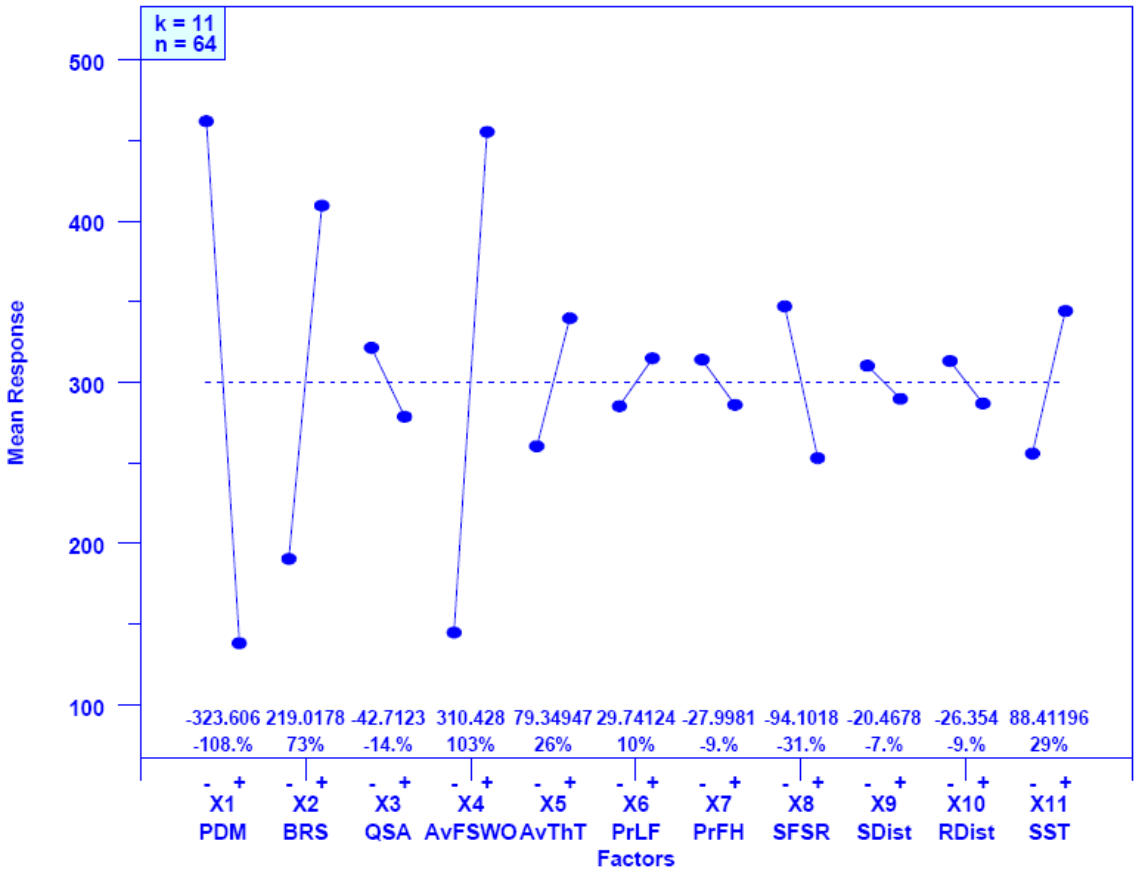


Figure C-15. Main-Effects Plot for Response y17 (Average Instantaneous Throughput on DD Flows) – y axis gives average goodput in pps

As shown in both Figs. C-16 and 4-25, FF flows, unlike DD flows, can face some congestion because selected source distributions lead to higher numbers of FN flows. Specifically, a source distribution (x9 plus) that gives the network a Web-centric characteristic leads to more FN flows, which compete for throughput with FF and DF flows. In addition, more sources (x8 plus) and lower average think time (x5 plus) lead to more active flows that can compete for throughput. Under these circumstances, higher network speed (x2 plus) allows competing flows to achieve higher throughputs. In the current sensitivity analysis buffer sizing algorithm has less influence on throughput because buffer sizes tend to be larger and initial slow-start threshold has more influence on throughput because the spread in file sizes and the higher network speed enabled increased use of initial slow-start.

C.3.5 Summary of Findings from Sensitivity Analysis

We use a rank analysis to summarize the findings from the current sensitivity analysis. We use one response to represent each characteristic: packet throughput (y4), flow completion throughput (y6), congestion (y10), delay (y15) and throughput of DD (y17) and FF (y20) flows. Table C-13 shows the results of our rank analysis, where the relative influence of each factor on each of the six responses is assigned a rank from one (most influential) to 11 (least influential) based upon the degree to which the factor altered the response when moving from a plus to a minus setting. The average rank is computed for each factor, and then the average rank is converted into an ordinal ranking based on ordering the factors from most (one) to least (11) influential. The table shows that network speed (x2) is the most influential factor, followed by file size (x4) and propagation delay (x1). Next is think time (x5), followed by number of sources (x8). These five factors (x2, x4, x1, x5 and x8) are the same five factors identified as most influential in the earlier sensitivity analysis (see Table 4-25). The only difference is one of ordering: propagation delay has jumped from fifth to third most influential.

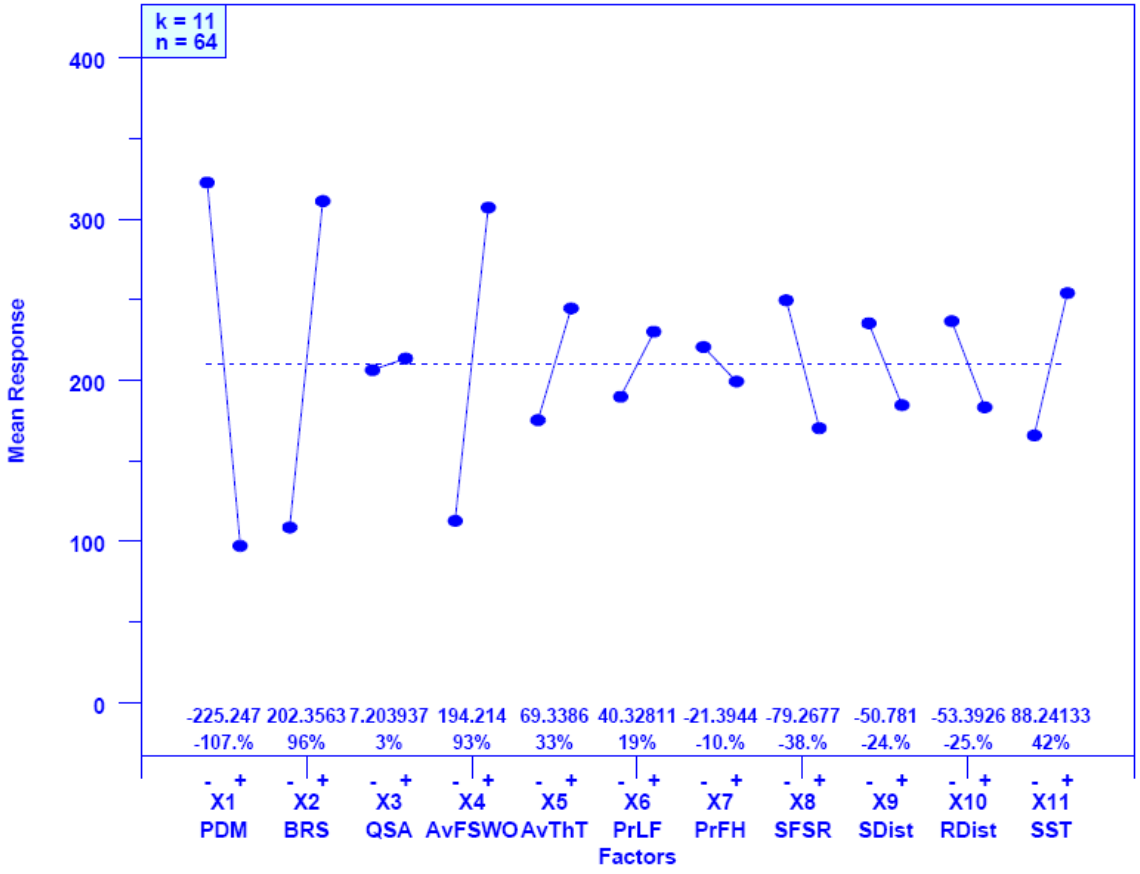


Figure C-16. Main-Effects Plot for Response y20 (Average Instantaneous Throughput on FF Flows) – y axis gives average goodput in pps

Table C-13. Rank Analysis of Sensitivity Analysis Responses

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
y4	6.5	2	8	1	3	6.5	10.5	4	5	9	10.5
y6	7	3	9	2	1	10	11	4	5	8	6
y10	5	1	7	2	4	10	8	3	6	11	9
y15	1	3	2	5	4	8.5	6.5	11	8.5	10	6.5
y17	1	3	7	2	6	8	9.5	4	11	9.5	5
y20	1	2	11	3	6	9	10	5	8	7	4
Average Rank	3.58	2.33	7.33	2.50	4.00	8.67	9.25	5.17	7.25	9.08	6.83
Ordinal Rank	3	1	8	2	4	9	11	5	7	10	6

This ordering shift can be justified. First, propagation delay affects many aspects of flow operation and the difference between the plus and minus settings in propagation delay has increased from 2 fold in the previous sensitivity analysis to 5 fold in the current sensitivity analysis. Second, the number of active sources (y1) is influenced by a relationship between think time and number of sources. The current sensitivity analysis has an order of magnitude more sources but these sources can see more than an order of magnitude greater network capacity and can have much longer think time (up to 10 instead of 5 seconds). The combination of sources with potentially longer think times and operating in a network with up to 20 times more capacity leads to a slightly higher proportion (74 % vs. 72 %) of sources in the thinking state at any given time. Since we scaled number of sources, think time and network speed to match, we would expect propagation delay to exert increased influence in the current sensitivity analysis.

In examining the less influential factors, we find that the sixth most influential parameter has become the initial slow-start threshold (x11). This makes sense because file sizes can be much larger and network speeds can be much higher in the current experiment, so more flows have opportunity to exploit the potential of a higher initial slow-start threshold.

C.3.6 Exploring Effects of Buffer Sizing

We decided to repeat the exploration of the effects of buffer sizing, which we applied to the earlier sensitivity analysis, as described in Sec. 4.7.2. Specifically, we consider the relative influence of propagation delay (x1), network speed (x2) and buffer-sizing algorithm (x3) on selected responses, chosen to represent macroscopic network behavior and user experience. To represent macroscopic behavior, we use packet throughput (y4), flow completion throughput (y6), retransmission rate (y10) and relative queuing delay (y16). To represent user experience, we use average throughput from three different flow classes: DD flows (y17), FF flows (y20) and NN flows (Y22). We aim to determine which of the three factors (x1, x2 or x3) has largest influence on the combined responses.

We use a rank analysis to study the effects of our chosen factors on our selected responses. In this particular analysis, we elected to use a larger number to indicate higher rank and a smaller number to indicate lower rank. We began by combining our three factors into a condition that can be assigned one of eight settings, as illustrated in Table C-14 (which can be compared with Table 4-27). Next, we computed the average value for each of our responses under each condition. Table C-15 (comparable to Table 4-28) displays the results of this averaging.

Table C-14. Mapping of Factor Settings to Eight Conditions (M = minus; P = plus)

	Values			
Condition	**Factor Settings x1:x2:x3**	**Propagation Delay Multiplier**	**Backbone Router Speed**	**Buffer Sizing Algorithm**
C1	**M:M:M**	**1**	**2×10^3**	***RTT*x*C*/SQRT(*n*)x2**
C2	**P:M:M**	**2**	**2×10^3**	***RTT*x*C*/SQRT(*n*)x2**
C3	**M:P:M**	**1**	**16×10^3**	***RTT*x*C*/SQRT(*n*)x2**
C4	**P:P:M**	**2**	**16×10^3**	***RTT*x*C*/SQRT(*n*) x2**
C5	**M:M:P**	**1**	**2×10^3**	***RTT*x*C*/2**
C6	**P:M:P**	**2**	**2×10^3**	***RTT*x*C*/2**
C7	**M:P:P**	**1**	**16×10^3**	***RTT*x*C*/2**
C8	**P:P:P**	**2**	**16×10^3**	***RTT*x*C*/2**

Using the average responses from Table C-15, we next rank each condition from high (8) to low (1) for each response, based on the appropriate ordering criteria. For retransmission rate (y6) and relative queuing delay (y16) a lower value would be ranked higher. For the other responses in Table C-15, a higher value would be ranked higher. After ranking the conditions with respect to each response, we compute an average ranking. The results of our ranking are shown in Table C-16 (comparable to Table 4-29).

Table C-15. Average Response Values for Each Condition

	Response						
Condition	**y4**	**y6**	**y10**	**y16**	**y17**	**y20**	**y22**
C1	**358 946.069**	**5561.8**	**0.211**	**1.6199**	**176.28**	**196.53**	**132**
C2	**316 517.564**	**3579.9**	**0.1946**	**1.3866**	**399.78**	**43.253**	**41.1**
C3	**1 308 862.84**	**8632.8**	**0.0379**	**1.9674**	**637.31**	**450.52**	**435**
C4	**839 113.623**	**11593**	**6E-05**	**1.4241**	**176.28**	**135**	**166**
C5	**347 508.785**	**4416.3**	**0.2679**	**2.9595**	**158.16**	**107.59**	**100**
C6	**336 533.732**	**5428.9**	**0.0445**	**2.0917**	**131.96**	**87.617**	**59.3**
C7	**1 080 811.29**	**14340**	**0.0024**	**2.143**	**652.1**	**535.56**	**498**
C8	**1 251 852.86**	**7983.3**	**0.0021**	**1.5581**	**172.49**	**123.35**	**132**

We can assign the average rank for each condition to the vertex of a cube, where each vertex represents a specific combination of settings for propagation delay, network speed and buffer size. Fig. C-17 (comparable to Fig. 4-32) shows the cube corresponding to Table C-16. Moving along the edges among the vertices on the cube allows us to determine changes in ranking attributable to each factor. The change in each factor (x1, x2 and x3) across all conditions is represented by a set of four different edges from among the 12 edges contained in the cube. We extract the relevant changes in ranking and display them in Table C-17 (comparable to Table 4-30).

Interpreting Table C-17 we see that changing network speed has the largest effect on the responses we selected. This agrees with the previous sensitivity analysis. Changing buffer sizing has the second largest effect, nearly the same as changing propagation delay, which has the smallest effect. Further, Fig. C-17 shows that changing from fewer to more buffers has a larger effect when network speed is low and propagation delay is short. This finding is counter to the earlier sensitivity analysis that found that changing from fewer to more buffers had a larger effect when network speed is high and

propagation delay is long. While counter to the previous findings, the effect can still be explained. In the previous sensitivity analysis, the network speeds were quite close and the small buffer size was very small for half the configurations. In the current sensitivity analysis, the network speeds are quite disparate. At the faster network speed (16×10^3 p/ms) packets rarely need to be buffered and so increasing to a larger buffer size does not matter much. At the slower network speed the potential for buffering packets increases and so the increase in buffer size has a larger influence on responses. In addition, packets can be "buffered in flight" on network transmission links in proportion to the bandwidth-delay product. For this reason, higher network speeds coupled with longer propagation delays mean more potential for in-flight buffering. Conversely, lower network speeds coupled with shorter propagation delay means less potential for in-flight buffering. For this reason, adding buffers when network speeds are lower and propagation delays are shorter should have more influence on responses. If nothing else, the two sensitivity analyses reveal a complicated interrelationship among network speed, propagation delay and buffer sizing. This interrelationship merits additional study.

Table C-16. Ranking for Each Condition vs. Each Response

	Condition							
Response	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**
y4	4	1	8	5	3	2	6	7
y6	4	1	6	7	2	3	8	5
y10	2	3	5	8	1	4	6	7
y16	5	8	4	7	1	3	2	6
y17	4	6	7	5	2	1	8	3
y20	6	1	7	5	3	2	8	4
y22	5	1	7	6	3	2	8	4
Average Rank	4.3	3.0	6.3	6.1	2.1	2.4	6.6	5.1

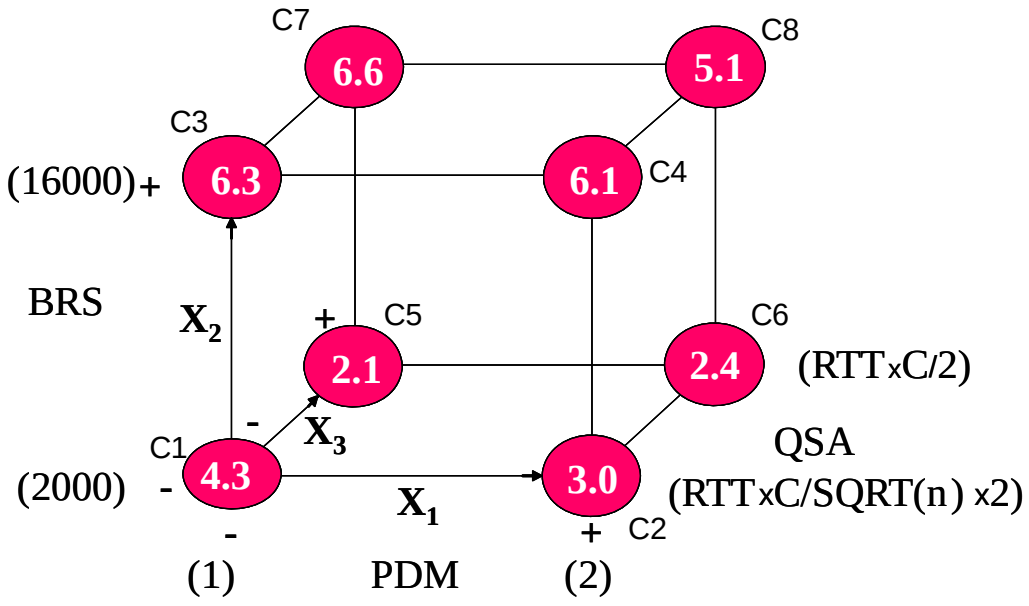


Figure C-17. Average Condition Ranking Displayed on Vertices of a Cube

Table C-17. Changes in Ranking Attributable to Each Factor

	Propagation Delay (x1)	Network Speed (x2)	Buffer Sizing (x3)
Edge 1	1.3	2.0	2.2
Edge 2	0.3	3.1	0.6
Edge 3	0.2	4.5	0.3
Edge 4	1.5	2.7	1.0
Average	0.8	3.1	1.0

C.4 Discussion

The supplementary sensitivity analysis confirmed the primary findings about main effects that were revealed in the previous sensitivity analysis, covered in Chapter 4. Both analyses found that system response was driven by the same primary input factors: network speed, file size, propagation delay, think time and number of sources. Propagation delay was found to be more influential in the current sensitivity analysis because the plus-minus level difference in propagation delay was substantially increased over the previous sensitivity analysis. Initial slow-start threshold moved to sixth (from eighth) because more flows had the opportunity to exploit a high initial slow-start threshold under the current sensitivity analysis, which allowed larger file sizes and much higher network speeds. The influence of buffer-sizing algorithm switched to eighth (from seventh) because in the current sensitivity analysis only 12.5 % of the configurations created average buffer sizes below 10^3 packets, compared with 50 % of the configurations in the previous sensitivity analysis. In comparing the main-effects plots, both sensitivity analyses identified the same input factors driving each system response. For particular responses, as explained above, small differences in factor influences could be discerned – all such differences were justified.

With respect to correlation among responses, both sensitivity analyses identified the same six response pairs to be correlated with magnitude greater than 0.95. The supplementary sensitivity analysis identified a seventh response pair (y17-y18) correlated above 0.95. In the range of 0.90 to 0.95, the supplementary sensitivity analysis found only three correlated pairs, while the previous sensitivity analysis indentified 10 such pairs. Among the seven missing pairs, four appeared at lower strength in the supplementary sensitivity analysis.

Two main differences were noted among correlations in the sensitivity analyses. First, the supplementary analysis found throughputs in all flow classes were correlated, whereas the previous analysis found throughput on DD flows and to be uncorrelated with throughput on other flow classes, and also found throughputs on DF and FF flows to be correlated with each other but uncorrelated with the throughputs of other flow classes. Comparing the relevant main-effects plots for these responses found that the supplementary analysis showed all flow throughputs to be driven by the same top three factors: propagation delay, network speed and file size. The order shifted among the three flow classes: propagation delay the main influence on DD and FF flows and network speed the main influence on NN flows. While the main influence (propagation delay or network speed) also appeared in the previous sensitivity analysis, the second most influential factors differed for DD flows (average file size), FF flows (source distribution) and NN flows (average think time). This shows that throughputs for FF and NN flows

were driven more by congestion in the first sensitivity analysis and less so in the supplementary analysis. Further, in the supplementary analysis, increased differences in network speed and propagation delay had a greater influence on flow throughputs than congestion. Guided by this finding, we defined our experiments comparing congestion control algorithms to include situations where congestion played a significant role, as well as situations where congestion was less significant.

A second main difference in correlation appeared in the supplementary sensitivity analysis. In the previous analysis, round-trip time (y15) and queuing delay (y16) were moderately correlated (0.70), while the supplementary analysis found no correlation (-0.08). This correlation change is largely due to changes in the relationship between propagation delay and buffer sizing between the two analyses. In the supplementary sensitivity analysis, propagation delay had a much bigger influence on SRTT, which increases 113 % when moving from a minus to plus setting vs. only 52 % in the first sensitivity analysis. As a result, higher propagation delays in the supplementary analysis increased the numerator in the computation for relative queuing delay (y16), which reduces y16 more than is the case in the previous sensitivity analysis. As a result, in the supplementary analysis with generally increasing SRTT (driven by both propagation delay and buffer size), y16 increases for 32 conditions (smaller propagation delay) then drops drastically upon reaching the larger propagation delay, increasing again along with increasing buffer size. These two functions, round-trip time (y15) and queuing delay (y16), do not exhibit the same relationship under the settings associated with the previous sensitivity analysis, where a generally increasing SRTT mirrors a generally increasing buffer size, except for occasional dips associated with specific parameter combinations. From this, we concluded that we should not estimate queuing delay by dividing SRTT by average propagation delay. For the experiments comparing congestion control algorithms we decided to estimate queuing delay by subtracting the average round-trip propagation delay from the average SRTT.

Similar to results from the correlation analysis, results from the principal components analysis (PCA) exhibited several differences between the two sensitivity analyses we conducted. Principal components analyses are notoriously difficult to interpret in many domains, especially when investigating a complex system with many factors. In the initial sensitivity analysis, we identified four principal components: congestion, delay, throughput on advantaged flows and network-wide throughput (flows and packets). For the supplementary PCA, identifying the top four principal components proved more difficult. As we explained above, the supplementary sensitivity analysis led us to group responses comprising four principal components that influence: throughput on all flow classes, throughput on advantaged flow classes, delay and network-wide throughput (flows and packets). The top four components bear general similarity among the two analyses, though they differ in the grouping of specific responses. The first component in both analyses could be said to characterize congestion and throughput for most typical users of the network. Delay is also a common component, playing more prominence in the first sensitivity analysis because half the configurations had very small buffers (compared to only 1/8 of the configurations in the supplementary analysis). The remaining two principal components were similar between the two analyses: throughput on advantaged flows and network-wide throughput.

We also compared two exploratory analyses: (1) seeking the source of a bifurcation in a (y7-y22) scatter plot and (2) investigating the relative influence of buffer sizing on system responses. As shown above, both sensitivity analyses revealed average file size (x4) as the cause of the y7-y22 bifurcation. Further, as expected, the supplementary sensitivity analysis led to an increased angle of bifurcation. With respect to the relative influence of buffer size, both sensitivity analyses found network speed to have much greater influence on system response than either propagation delay or buffer size. The supplementary and initial sensitivity analyses did vary with respect to the conditions under which increasing buffer size had larger influence on system responses, but these differences were explainable.

C.5 Conclusions

In this appendix, we conducted a supplementary sensitivity analysis following the general plan presented in Chapter 4, but changing the level settings for the 11 parameters. Specifically, we increased network speed and size by about an order of magnitude, we stretched the range of parameter values covered by the plus and minus settings of each factor, and we shifted the traffic patterns slightly to generate more DD flows and to give a higher prominence to Web browsing activity over P2P exchanges. We subjected the results to the same analyses applied in Chapter 4. Comparing our findings with those from Chapter 4, we identified general agreement in the main factors driving model responses. Where differences arose, we were able to attribute them to changes in level settings or relationships among level settings. On the whole, the results from this supplementary sensitivity analysis increased our confidence in the MesoNet simulation model. In addition, we gained increased confidence in our analysis methods, which were able to identify differences arising from parameter variations, as well as relationships remaining invariant across our two sensitivity analyses. Finally, comparing results from the two sensitivity analyses guided us to change our technique for estimating queuing delay and to select experiment designs to include configurations with little congestion, as well as configurations with significant congestion.

Appendix D – 10-Step Graphical Analysis Technique

Appendix D 10-Step Graphical Analysis Technique

We adopted a (NIST-developed) 10-step graphical analysis technique to evaluate the behavior of our model. The analysis technique uses 10 different plot types, listed in Table D-1, which allow us to identify the main factors influencing system behavior, to discover interactions among factors, to assess statistical significance of the factors, to propose linear models that match the data, to identify best and worse combinations of factors and to suggest additional factor settings that can drive system responses in particular directions. In this section, we introduce the technique through a sample sequence of the 10 plot types, using as an example one response variable, y11, average congestion window (CWND). The sample plots were taken from our study.

Table D-1. Identity and Purpose of 10 Plots in the 10-Step Graphical Analysis

Plot	Purpose
Ordered Data Plot (D.1)	Reveal how combinations of parameter settings influence response
Multi-factor Scatter Plot (D.2)	Reveal influence of individual parameter levels on response distribution
Main Effects Plot (D.3)	Reveal individual parameters having greatest influence on response
Interaction Effects Matrix (D.4)	Reveal degree of influence of parameter pairs on response
Block Plot (D.5)	Test robustness of statistically significant parameters in light of secondary or nuisance factors
Youden Plot (D.6)	Reveal parameters and parameter pairs with greatest influence on response
|Effects| Plot (D.7)	Reveal magnitude of a change in response due to specific parameters and parameter interactions
Half-Normal Probability Plot of |Effects| (D.8)	Separate influential parameters and parameter interactions from those that are not influential
Cumulative Residual SD Plot (D.9)	Provide information sufficient to construct a linear model to represent response data
Contour Plot (D.10)	Suggest how alterations in parameter settings could influence system response in predictable directions.

The transmission control protocol (TCP) manages a congestion window (CWND) variable that represents the number of packets that can be sent prior to receipt of an acknowledgment. The larger the CWND, the more packets that can be sent per unit time and thus the greater will be the transmission rate. For that reason, a network with a high average CWND (y11) will be able to transmit more packets than a network with a lower average CWND. In general, a CWND is reduced when packets are lost, usually due to

congestion. Lowering the CWND slows the rate of packet transmissions in the network and thus should reduce congestion. Subsequent to a reduction, TCP increases the CWND linearly and so the rate of packet transmissions in the network should also increase. Once the transmission rate becomes too high, packets are lost and the CWND is reduced and the rate of transmission slows and so on. Thus the average CWND size might be used to represent the level of congestion in a network. Here, we analyze average CWND size with 10 plots, each representing one of the plot types listed in Table D-1. The changes in CWND were driven by a sensitivity analysis experiment, described in Chapter 4.

D.1 Ordered Data Plot. Fig. D-1 shows an ordered data plot, which graphs a system response (y axis) against every combination of factors (x axis) investigated in an experiment. The data is arrayed from smallest (on the left) to largest (on the right) response value. Our sensitivity analysis used 64 combinations of the 11 factors (recall Fig. 4-1) and so the plot contains 64 points. The upper left-hand corner of the plot shows the number of factors (k =11) and the number of combinations of factors (n = 64) in the experiment that generated the data. Below the x axis, each point is labeled with the specific combination of factor (x1 to x11) levels (- or +) that led to the response.

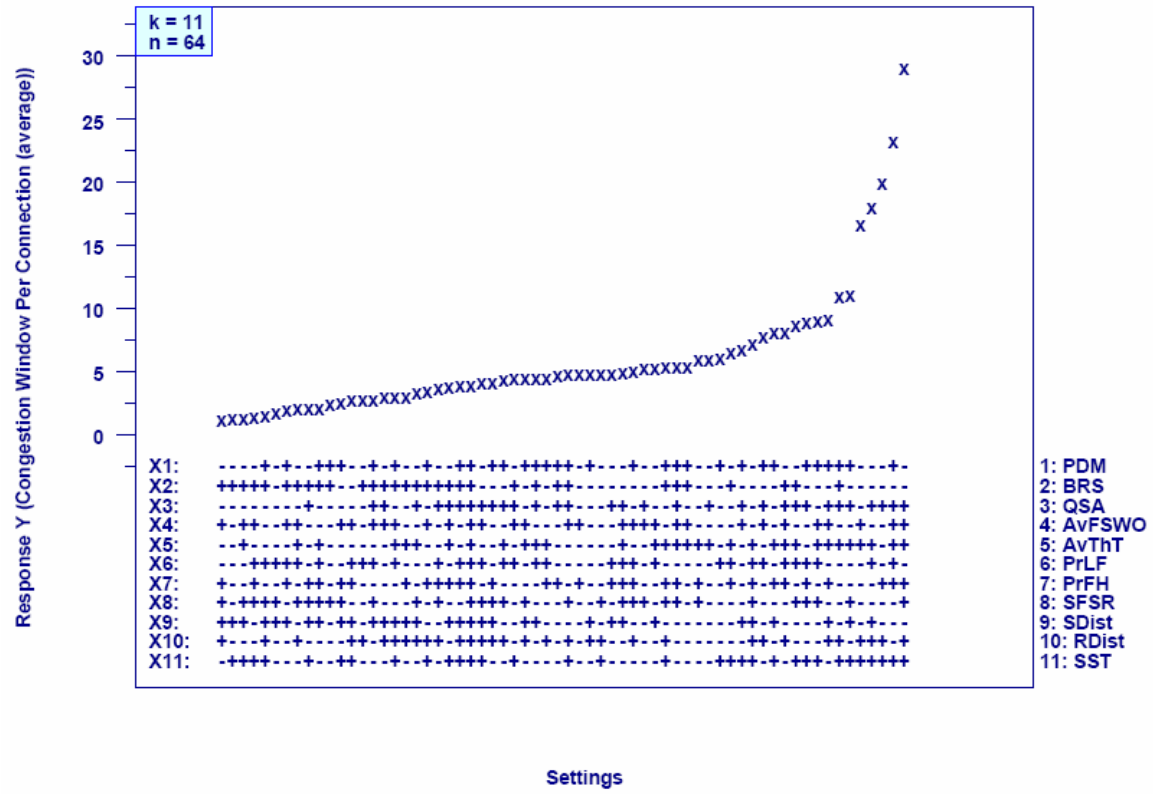


Figure D-1. Sample Ordered Data Plot

A legend to the right of the plot gives shorthand names to identify each of the 11 factors. Here the factors include: propagation delay (PDM/x1), network speed (BRS/x2), buffer-sizing algorithm (QSA/x3), average file size (AvFSWO/x4), average think time (AvThT/x5), probability that a user downloads a larger file (PrLF/x6), probability that a

source resides on a fast host (PrFH/x7), scaling factor for the number of sources and receivers (SFSR/x8), distribution of sources (SDist/x9) and receivers (RDist/x10) throughout the topology, and initial slow-start threshold (SST/x11).

The plot can reveal the combination of factor settings that lead to the smallest (left-most) and largest (right-most) response from the system. In addition, the plot can reveal combinations of factor settings that appear to have greatest influence on the response. From Fig. D-1, we might conclude that the five right-most combinations had more significant influence (than other combinations) in increasing the average CWND per connection. Examining the factor settings associated with these data points reveals some common factors among them. For example, these points all have higher[1] network speed (x2 = -) and higher initial slow-start threshold (x11 = +) and four of the five points have larger (x3 = +) buffer sizes. Looking across the row of settings for network speed (factor x2) one can see that higher network speeds (x2 = -) seem to result more often in higher average CWND. Similar views can be taken of the other factors.

As a result of viewing the ordered data plot, an experimenter begins to see how various factors could be driving system response. From Fig. D-1 alone, an experimenter knowledgeable in the domain could begin to get a sense that faster networks with less congestion, higher initial slow-start threshold and larger buffer sizes lead to higher average CWND. More detailed information on the influence of these and other factors becomes available from subsequent plots.

D.2 Multi-factor Scatter Plot. Fig. D-2 shows a sample multi-factor scatter plot, which groups responses for each factor by setting (+ or -) and then plots the responses (y axis) together with the average response (dashed horizontal line). The plot also gives the number of factors (k = 11) and observations (n = 64). The x axis shows each factor (x1 to x11) as two vertical scatter plots, one when the factor setting is a minus and one when the factor setting is a plus. Thus, each individual scatter plot has half of the observations (here 32 of 64). The plot shows the distribution of response values and identifies the minimum and maximum values. From Fig. D-2 one can see that average CWND tends to be under 10 for most observations, which might suggest that many of the experiment combinations constrained CWND. The plot also reveals a clear setting for each factor in order for CWND to achieve its maximum value (near 30). This combination of factors will correspond with the right-most combination of factors in the ordered data plot.

Interpreting Fig. D-2, an experimenter can see the following settings leading to highest average CWND: shorter propagation delay (x1 = -), higher network speed (x2 = -), larger buffer sizes (x3 = +), larger file sizes (x4 = +), longer think times (x5 = +), higher probability of transferring larger files (x6 = -)[2], lower probability of fast hosts (x7 = +)[3], more sources (x8 = +), less uniform distribution of sources (x9 = -), more uniform distribution of receivers (x10 = +), and higher initial slow-start threshold (x11 = +). The experimenter might wonder which of the factors and settings are most influential. The next plot provides this information.

[1] Recall the miscoding of factor x2: minus is higher network speed and plus is lower network speed.
[2] The coding for factor x6 was reversed (plus was a lower probability and minus was a higher probability).
[3] The coding for factor x7 was also reversed (plus was a lower probability and minus was a higher probability).

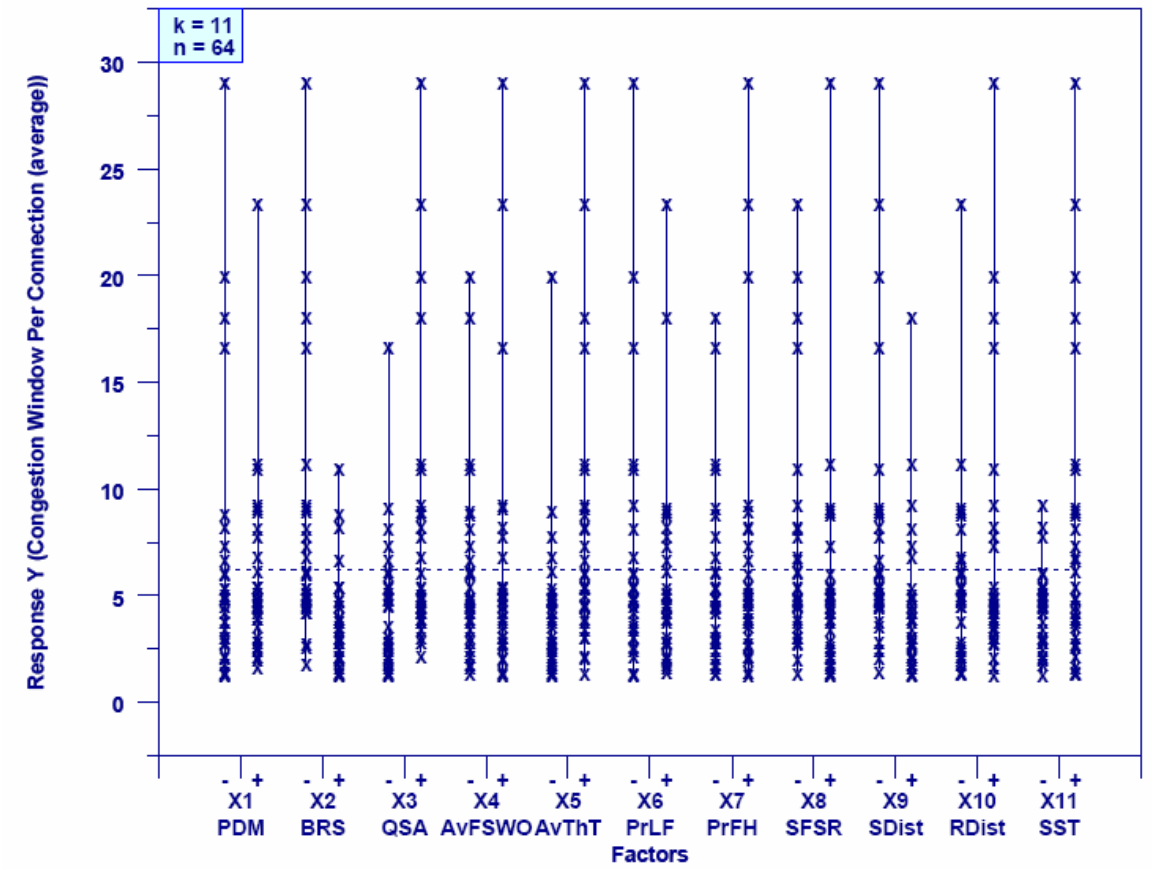


Figure D-2. Sample Multi-factor Scatter Plot

D.3 Main Effects Plot. Fig. D-3 gives a sample main effects plot, which is the most essential plot to identify the factors and settings driving a system's response. The basic framing of the plot is similar to that of the multi-factor scatter plot; however, each vertical scatter plot is replaced by an average of the response. For each factor, the averages are connected with a line that indicates the magnitude and direction the response changes when moving from a minus to a plus setting for the factor. On the x axis, each factor is annotated with the absolute change in response and the change relative to (i.e., as a % of) the mean response.

Fig. D-3 reveals that the most influential factor in determining CWND is network speed (70 % of the mean) followed by three closely grouped factors: buffer-sizing algorithm (54 %), initial slow-start threshold and think time (each 53 %). The distribution of sources also has a significant (50 %) influence. Notice that the plot reveals a smaller number of sources and receivers (x8 = -) leading to a (1.7 packet) larger average CWND than a larger number of sources – this is true despite the fact that the ordered data plot and scatter plot showed that the largest CWND was achieved when the number of sources was at its higher setting. In fact, a domain expert will understand that fewer sources sharing the same network means that each source may transmit faster, which is reflected in a larger CWND. Thus, here the main effects plot clearly reveals the true nature of the influence of the factors and settings on the response.

In thinking about the main effects, an experimenter with domain knowledge might be quite pleased with the meaning of these results regarding the validity of the model. Fewer, simultaneously active, flows (x5 = +, x8 = - and x9 = -), higher network speeds (x2 = -) together with more buffers (x3 = +) should permit higher CWND. Under these circumstances, the ability to increase the CWND to a higher threshold via initial slow-start (x11 = +) should also lead to higher CWND, because CWND increases fastest during initial slow start.

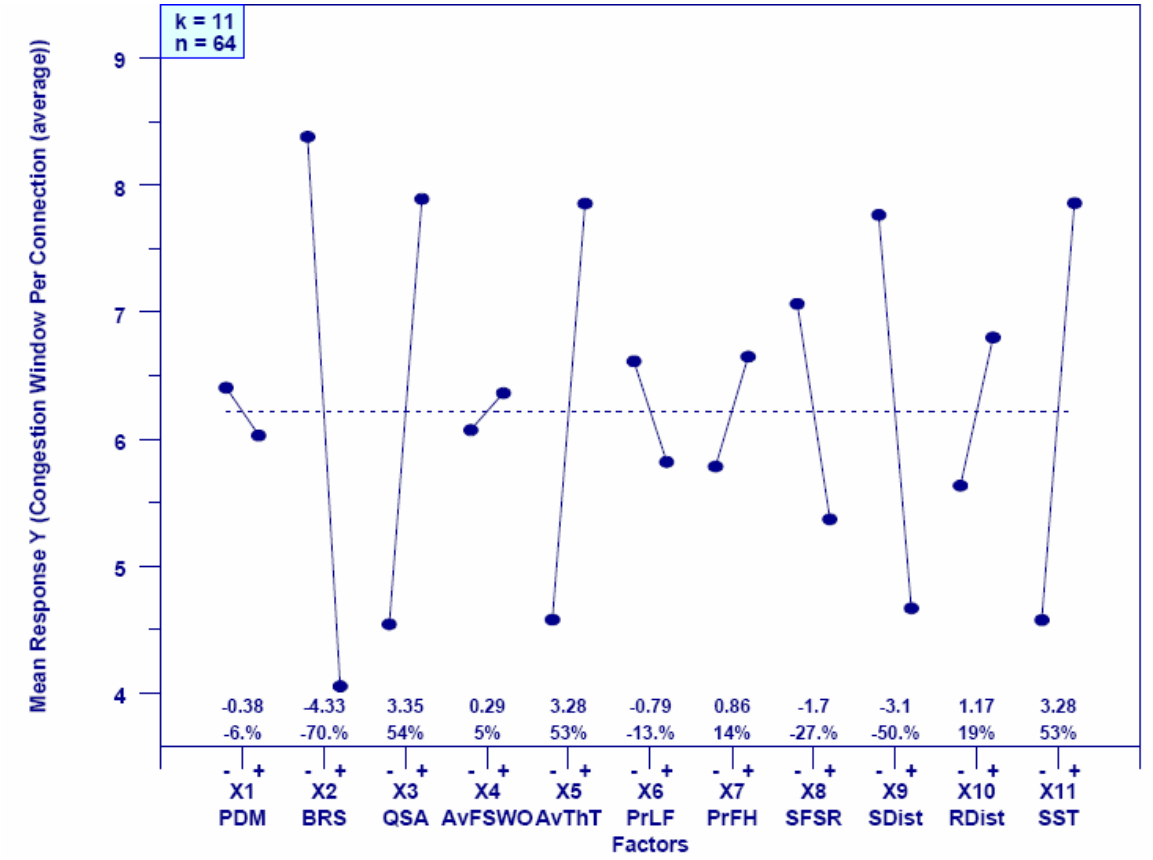


Figure D-3. Sample Main Effects Plot

D.4 Interaction Effects Matrix. Figure D-4 shows a sample interaction effects matrix. The purpose of this plot is to determine if interactions among factors have a significant influence on the response. If this plot reveals no such interaction effects, then an experimenter can conclude that the system response is driven primarily by main effects. The plot might also reveal interactions that an experimenter expected based on domain knowledge. On the other hand, the plot could reveal significant, unexpected effects due to interactions, which requires further investigation by the experimenter.

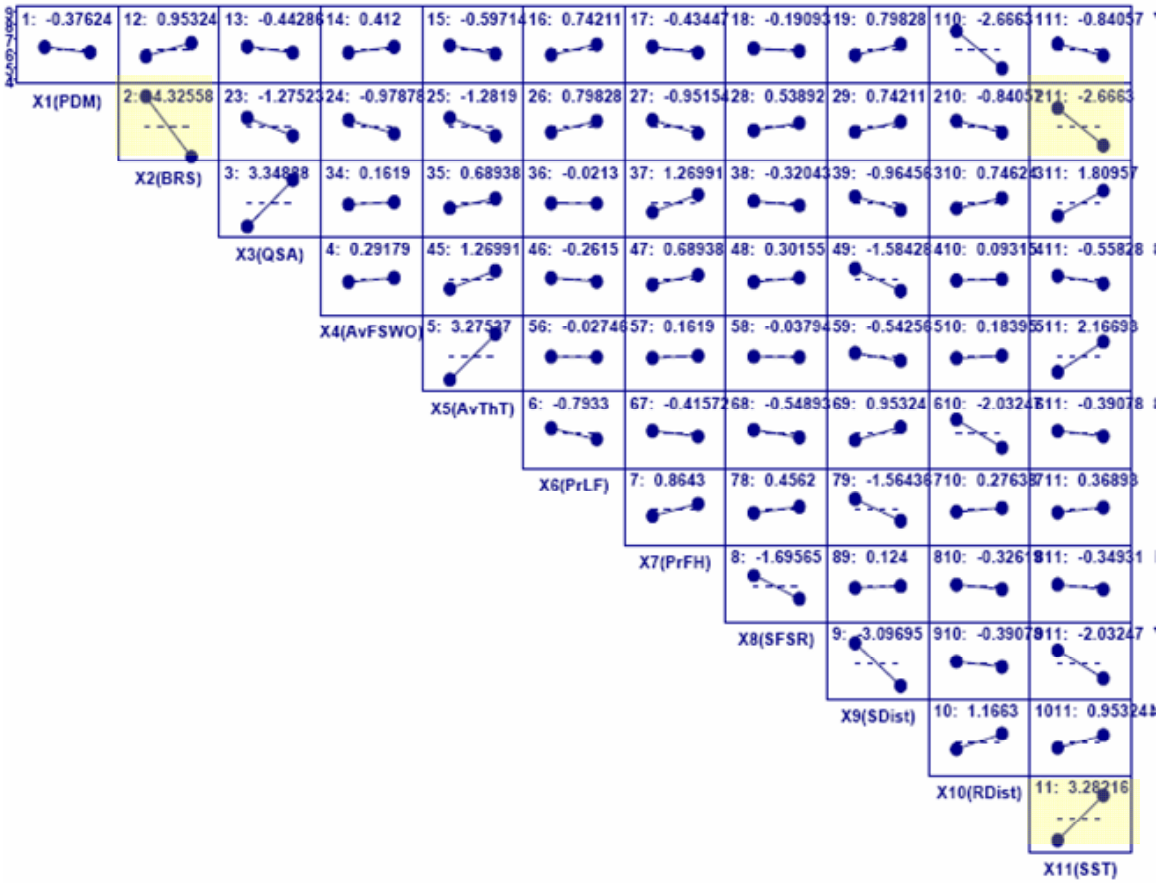


Figure D-4. Sample Interaction Effects Matrix

The interaction effects matrix takes the form of a half matrix containing rows and columns of sub-plots, where each sub-plot shows how the average response changes when moving from a minus to a plus setting for some combination of factors. The left-most sub-plot in each row (also the bottom-most sub-plot in each column) gives the main effects plot (from Fig. D-3) for a specific factor (x1 in the top row and x11 in the bottom row). Each of the remaining sub-plots in a given row (or column) show how the average response changes when moving from a minus to a plus setting for two factors (the factor beginning the row or column and each of the other factors).

An experimenter may scan the matrix starting from each main effects plot. Scan up (the related column) and also right across (the related row) to compare the influence of each main factor to the influence of possible two-factor interactions. Scanning the matrix in Fig. D-4 shows that, for the most significant main effects (x2, x3, x5, x8, x9 and x11), the influence of the main effect is greater than the influence of any interactions. So, for example, consider the three sub-plots highlighted in Fig. D-4. The upper left-hand sub-plot reports that changing network speed (x2) changes CWND size by 4.3 packets and the lower right-hand sub-plot reports that changing initial slow-start threshold (x11) changes CWND size by 3.3 packets, while the third highlighted sub-plot reports that changing x2 and x11 together changes CWND size by only 2.7 packets. Similar results can be found

when comparing the influence of other factors with their two-factor interactions. This suggests that system response is driven by main effects and not two-factor interactions.

D.5 Block Plots. Block plots provide an elementary test of statistical significance for the influence of main effects. Given a full factorial experiment design, one can compare the average response for a minus setting of a factor to a plus setting under all possible combinations of other factors. This can provide a large amount of visual information, so typically only a subset of these plots is generated. Further, given an orthogonal fractional factorial (OFF) experiment design a reduced amount of information is available for generating block plots, so such plots are not as useful for OFF experiment designs. Still, block plots can prove useful in confirming findings about main effects.

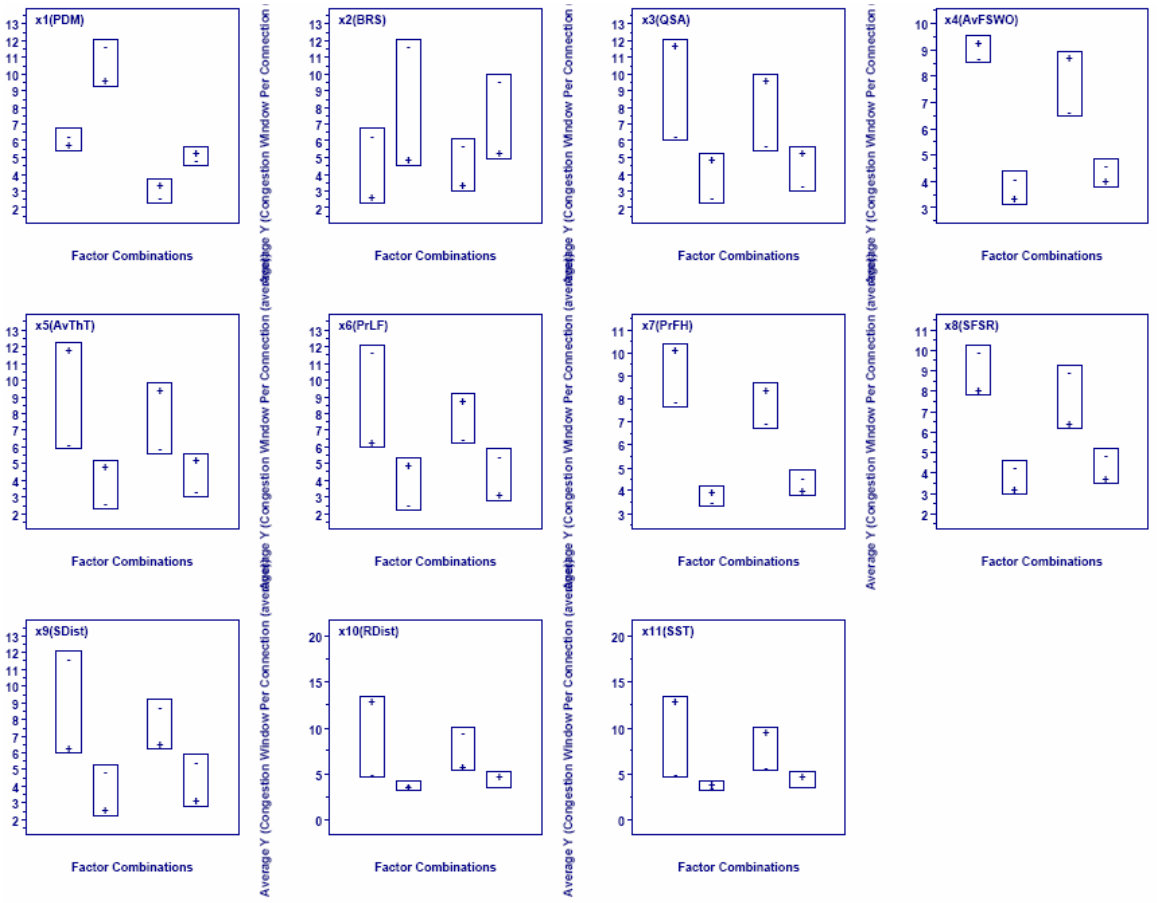


Figure D-5. Sample Block Plots

Fig. D-5 shows sample block plots for each of the 11 factors used in the sensitivity analysis of MesoNet. Each plot shows the average response for four combinations of secondary factors when the main factor of the plot is set to a minus and a plus. The block plots reinforce the findings of the main effects plot: the most significant factors are network speed (x2), buffer sizes (x3), think time (x5), number (x8) and distribution (x9) of sources and initial slow-start threshold (x11). This is revealed by the fact that one particular setting for each of these factors always leads to a higher value for

CWND. This is true no matter what combination of other factors is used. Results are mixed for the other five factors.

D.6 Youden Plot. A Youden plot, see Fig. D-6, graphs the average response for each factor (and two-factor interaction) when the factor (or factors) are set to a minus against the average response when set to a plus. In Fig. D-6 the average CWND for minus settings are plotted on the x axis and for plus settings on the y axis. For unimportant factors, the values should be nearly the same (and appear in the center of the graph). For important factors, values should lie toward the upper-left and lower-right corners of the graph.

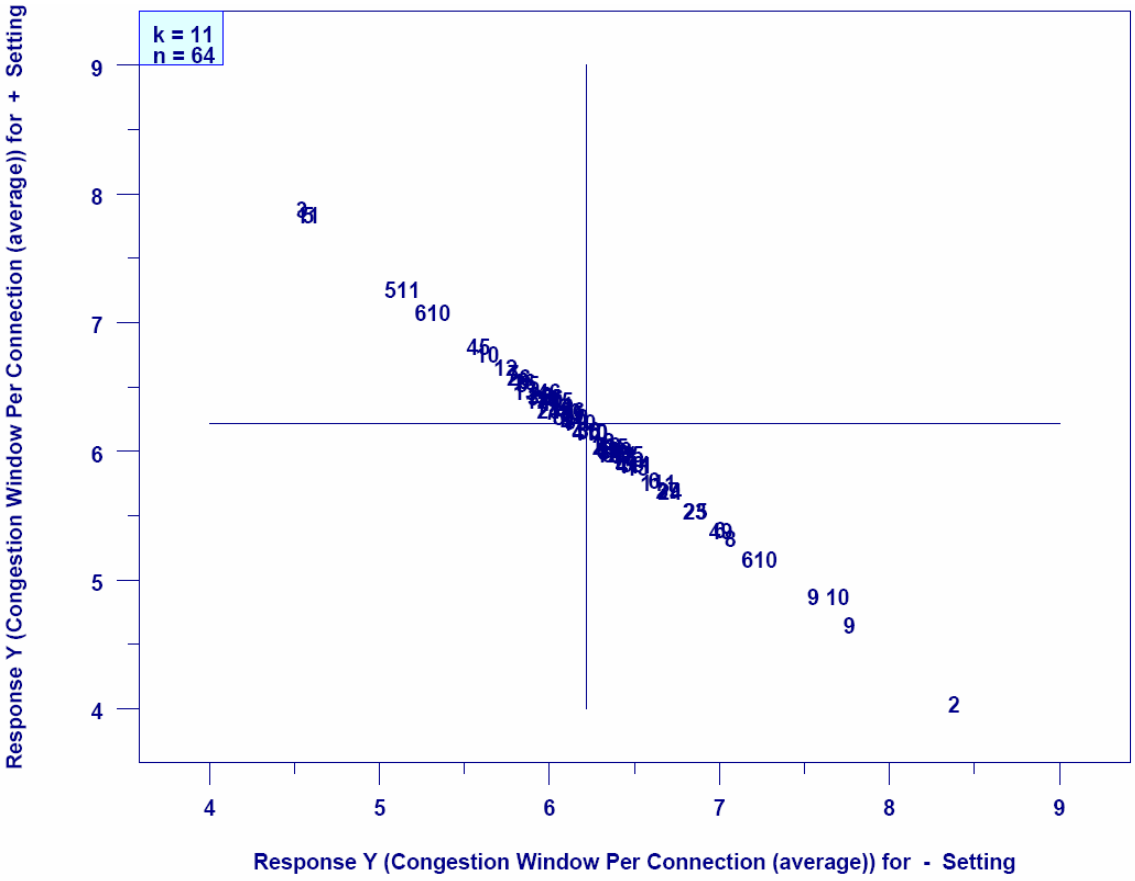


Figure D-6. Sample Youden Plot

Examining Fig. D-6 reveals that the most important factors are network speed (x2), buffer size (x3), think time (x5), source distribution (x9) and initial slow-start threshold (x11). The number of sources (x8) is not as important. Recall that the main effects plot identified x8 as of less importance than the other effects. The Youden plot supports the earlier finding. The plot also reveals some information about the influence of interactions. The distribution of receivers (x10) has a combined effect with the distribution of sources (x9) and the think time (x5) has a combined effect with initial

slow-start threshold (x11). These interaction effects (also revealed in the interaction effects matrix, Fig. D-4) are less important than the main effects.

D.7 |Effects| Plot. The |Effects| plot, Fig. D-7, displays the absolute magnitude of a change in response due to specific factors and interactions. The x axis identifies the factors or interactions for which the corresponding magnitude of the change in response is plotted on the y axis. The factors are ordered by decreasing magnitude from left-to-right on the x axis. This plot should confirm the information given in previous plots regarding the influence of factors and interactions. The average value for the response is given in the upper left-hand corner of the plot. The plot may be augmented (as here) with a rank-ordered list of factors (and interactions) and the associated (signed) magnitude of the effects.

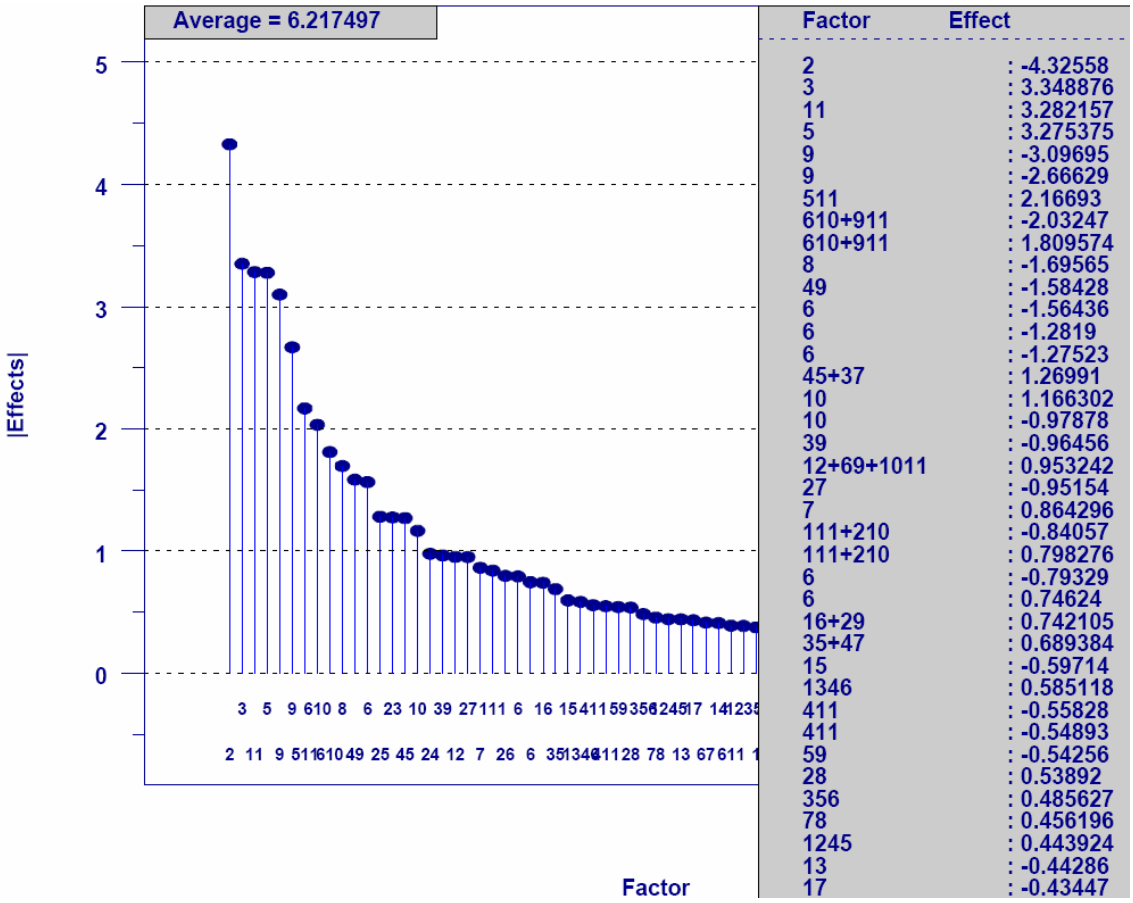


Figure D-7. Sample |Effects| Plot

Fig. D-7 confirms earlier findings: the main factors influencing CWND include (in order) network speed (x2), buffer size (x3), initial slow-start threshold (x11), think time (x5) and source distribution (x9). The associated list identifies x9 twice, but this is a mistake in labeling. Consulting the interaction effects matrix (Fig. D-4) reveals that the

second x9 should be an x2-x11 interaction. This demonstrates the cross-checking value of the redundancy included in the 10-step graphical analysis technique.

D.8 Half-Normal Probability Plot of |Effects|. A half-normal probability plot of |Effects|, Fig. D-8, classifies effects as important or unimportant. The x axis of a half-normal probability plot represents the ordering of a theoretical half-normal distribution of values. The y axis represents the |Effects|, which are plotted in order from least to greatest. Unimportant effects would tend to have mean difference centered on zero, sp when plotting the |Effects| for such data, one would expect plotted values to begin around zero and ascend linearly. Linear data emanating from the origin implies that the effects have a non-significant (zero) value. Data departing from linearity indicate a statistically significant effect.

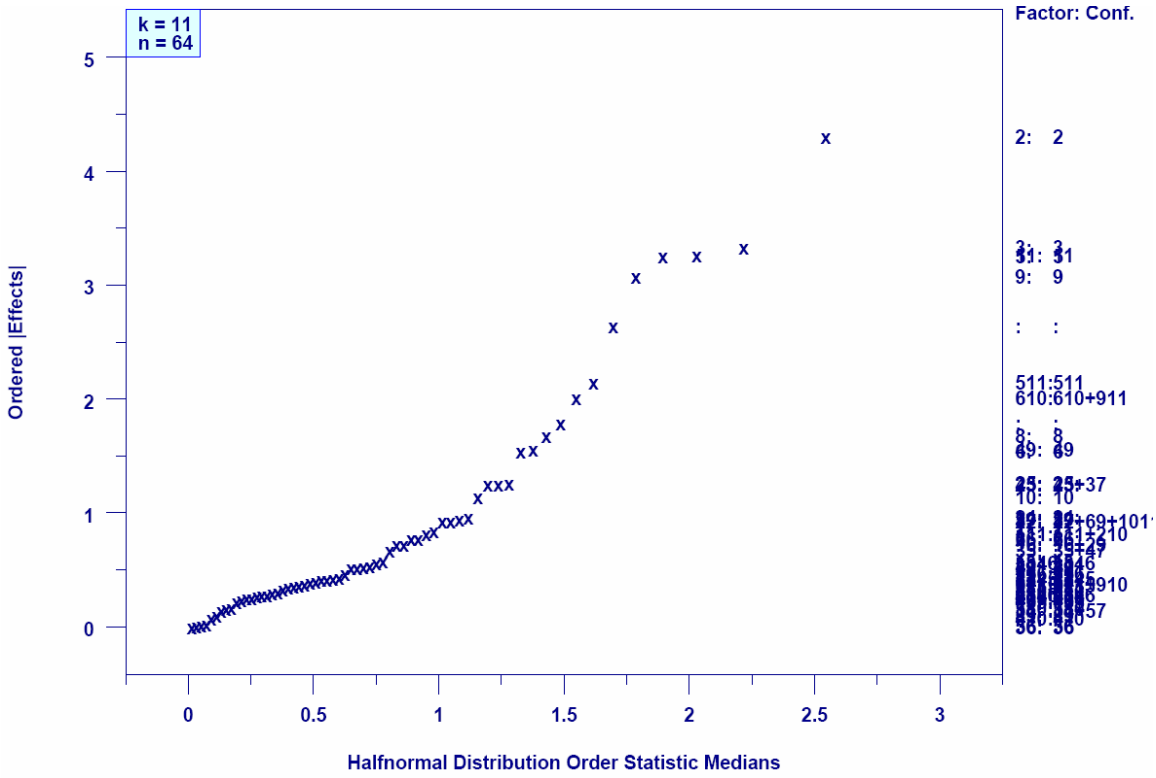


Figure D-8. Sample Half-Normal Probability Plot of |Effects|

As illustrated in Fig. D-8 the values to the right of the plot identify which factor (or interaction) is responsible for the plotted value on the y axis. Interpreting Fig. D-8 indicates the value of CWND is driven mainly by five factors: network speed (x2), buffer size (x3), think time (x5), initial slow-start threshold (x11) and source distribution (x9). The plot shows relative importance: x2, followed by the grouping of x3, x5, x11 and then finally x9. This finding is consistent with information obtained from previous plots.

D.9 Cumulative Residual Standard Deviation Plot. The cumulative residual standard deviation (SD) plot (Fig. D-9) provides information sufficient to construct a linear model to represent experiment data. The y axis of the plot gives the residual error between the data and a fitting function when adding terms representing the set of factors and interactions represented on the x axis. The first term on the x axis is the residual SD when describing the data using only the grand average over all factors. Then, the residual standard deviations are plotted as the influence of each factor is added in order of decreasing reduction in SD. The factors are identified on the x axis.

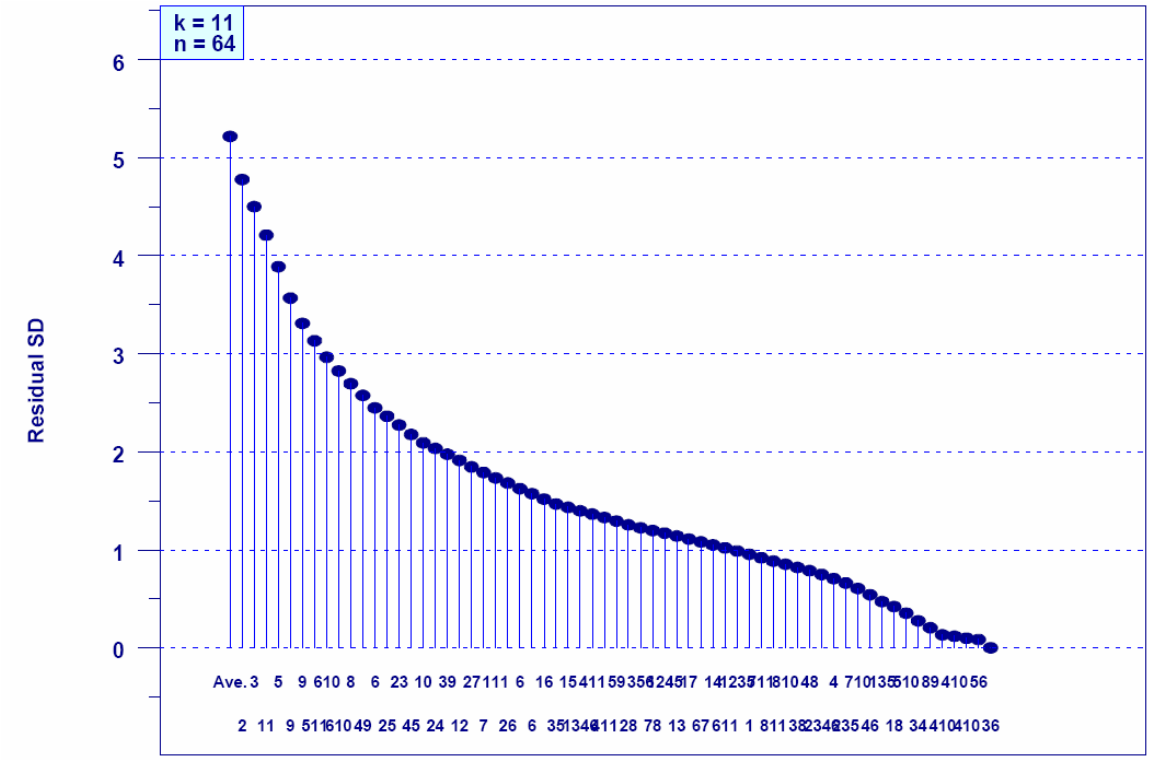


Figure D-9. Sample Cumulative Residual Standard Deviation Plot

Typically, one hopes for a model where the most important factors can explain most of the standard deviation in the data. If this holds, then a fairly simple linear model can describe the data. A plot demonstrating such a case would exhibit a large reduction in SD after the main factors are included in the model. In our example, when we include the five most important factors (x2, x3, x5, x9 and x11), Fig. D-9 shows a remaining error of about 3.5. The encouraging news from Fig. D-9 is that the five most important factors reduce the SD the most (and in the expected order). On the other hand, the residual SD is still rather high after including these factors, so the resulting linear model could likely not be used for interpolation or prediction.

D.10 Contour Plot of Two Dominant Factors. The contour plot aims to suggest other factor settings that could alter system response in predictable directions. While one could plot each pair of factors together, a typical approach is to construct a contour plot from the two most important factors. Fig. D-10 shows a sample contour plot

from the two main factors, x2 (network speed) and x3 (buffer size), influencing CWND. The most important factor is plotted on the x axis and the second most important factor is plotted on the y axis. The axes are labeled with an origin (0) and then the two settings (-1 and +1) for each factor. A point is placed at each combination of factors (x2, x3) = {(-1, -1), (+1, -1), (+1, +1), (-1, +1)}. These four points are connected with a dashed line to form a rectangle. Each point is labeled with the value of one of the specific response variables for the associated combination of the two factors. Based on the fitted model developed when generating the cumulative residual SD plot, contour lines are added to form a contour plot.

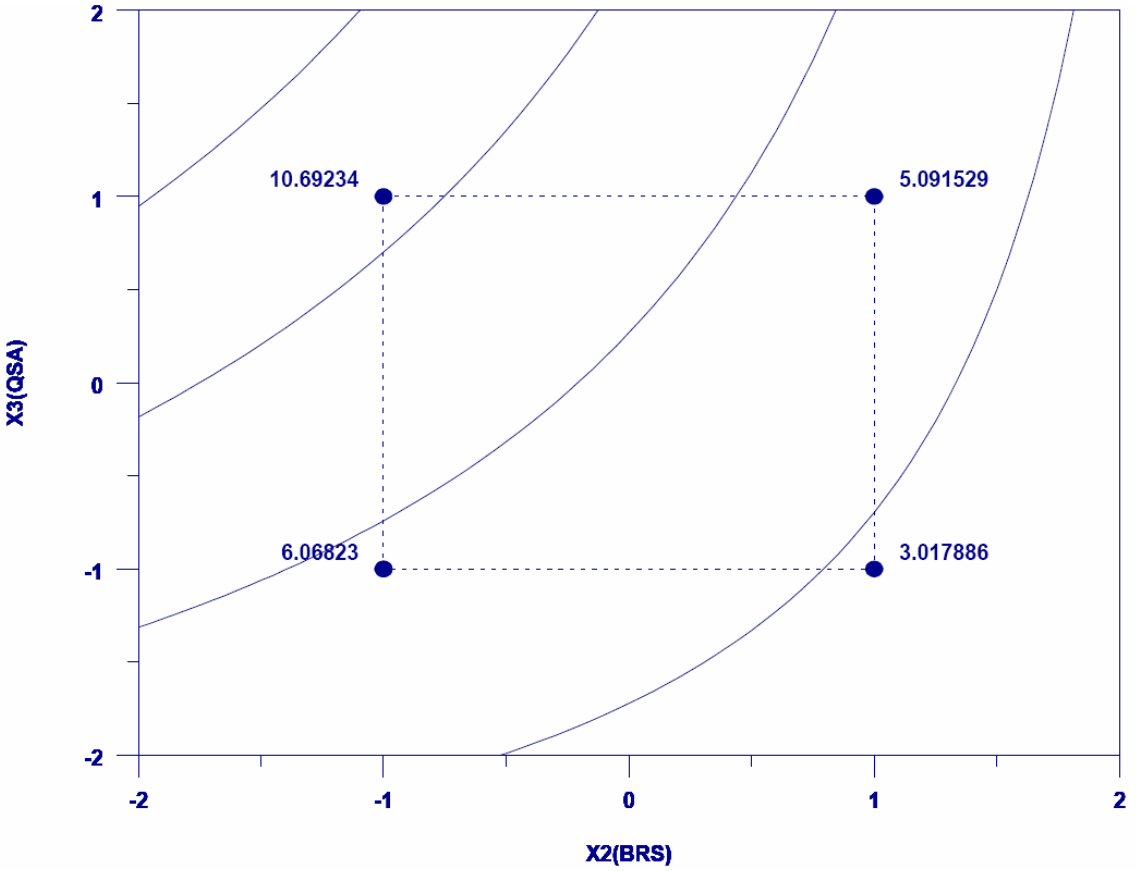


Figure D-10. Sample Contour Plot of Two Dominant Factors

As Fig. D-10 shows, the combination of (-1, +1) – higher network speed and larger buffer size – produces the largest CWND (10.69…). The contour lines indicate that increasing network speed and buffer size would lead to larger CWND values, while decreasing network speed and buffer size would lead to smaller CWND values. Under some experiments and conditions, the contour plot could reveal directions in which to alter factor settings to create more optimal responses or suggest how responses might change as factor settings are altered.