MFTP: Recent Enhancements and Performance Measurements

Jonathan Hahn

RND-94-006 June 1994

hahn@nas.nasa.gov

NAS Systems Development Branch
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

Abstract

This paper discusses recently completed modifications to mftp, an enhanced version of ftp which utilizes multiple TCP data connections in parallel to increase the total effective TCP window size. A file transfer restart feature was added, internal buffering was increased, and the help facility has been enhanced. These enhancements are described in detail and performance results are presented. Mftp transfers are as fast, or faster than ftp in all cases except over UltraNet. The transfer restart feature will be of great value to users who must transfer large files. Topics for future work are suggested.

1.0 Introduction

This paper reports on follow-up work to [1]; refer to that paper for a detailed discussion of the motivations and ideas behind *mftp*. The enhancements reported in this paper improve *mftp*'s performance and usability, and position it as an important tool for file transfer within the NAS local and wide-area networks.

2.0 Enhancements

The following enhancements have been made to *mftp*:

1. File Transfer Restart
2. Improved Help Facility
3. Improved Internal Buffering
4. Bugs Fixed

2.1 File Transfer Restart

WAN network links are generally slower and less reliable than LAN links. It's not unusual for large file transfers over WANs to require one or more hours to complete. When a network disruption causes a file transfer to terminate, the transfer must be restarted from the beginning. *Mftp* has been enhanced to allow disrupted transfers to pick up where they left off. The *mftp* `restart` command has the following format:

```
restart <MARKER>
```

where *<MARKER>* is a byte offset into the file where the restarted transfer should begin. Issuing the restart command prior to a `get` or `put` causes the transfer to commence at byte offset *<MARKER>* rather than at the beginning of the file.

A sample *mftp* session with restart:

```
% mftp farhost
mftp> get bigfile
200 PORT command successful (129.99.50.17,2062)

...connection fails due to network failure before transfer completes...

% ls -l bigfile
-rw-r--r-- 1 user 10240 May 4 22:06 bigfile
% mftp farhost
mftp> restart 10240
restarting at 10240. execute get, put or append to
     initiate transfer
mftp> get bigfile
200 PORT command successful (129.99.50.17,2062)
350 Restarting at 10240. Send STORE or RETRIEVE to
     initiate transfer.
150 Opening <1> BINARY mode data connection(s) for
     bigfile (99999 bytes).
226 Transfer complete.
99999 bytes received in 0.077 seconds (1.4e+04 Kbytes/s)
% ls -l bigfile
-rw-r--r-- 1 user 99999 May 4 22:08 bigfile
%
```

Mftp does not check the integrity of the data that is skipped over on a restarted transfer. If there is any doubt (for example, if one of the machines involved in the transfer crashed), remove the suspect data and re-transfer the entire file.

2.2 Improved Help Facility

The messages in the *mftp* help facility have been expanded to full descriptions with usage instructions. Descriptions of new commands have been added.

2.3 Improved Internal Buffering

2.3.1 Buffer Size

The internal buffers in *mftp* have been increased in size to 24K from 4K bytes. This results in improved performance between LAN hosts over the previous version of *mftp:* the new *mftp* is about as fast as vendor-supplied *ftp* implementations, however, it is not as fast as UltraNet *ftp* implementations (see Figure 7). Between hosts connected over AEROnet, this version of *mftp* is about as fast as the previous version with 10 channels active. It is much faster with fewer channels.

The increased buffer size does not significantly improve the speed of 10-channel *mftp* transfers over AEROnet because the original version (with 4K buffers) adequately accommodates the bandwidth-delay product of the AEROnet links which are terrestrial and of moderate delay. Large improvement would be observed over satellite links which

are characterized by high bandwidth and long delays, but AEROnet does not employ satellite links.

2.3.2 Buffering Algorithm

The internal buffering algorithm in *mftp* has undergone a major overhaul. The algorithm in the original version is described in [1]. Briefly, file blocks are written to, and read from network data ports in round-robin fashion. If a data block is not available at the time a data port is read, a gap is created in the resulting destination file, to be filled in later when the block arrives.

The new restart feature requires that an interrupted transfer leave a file fragment with no gaps, so the algorithm had to be modified. In the new *mftp*, data blocks are read from the data ports into buffers which are assembled onto a linked list. When a contiguous chunk of data (of predetermined size) is available on the list, it is written out to disk. If a data block should fail to appear for a while, the subsequent data blocks will be buffered until the missing data block shows up. At that time, all the buffered data up to the last full chunk will be written out to disk.

This algorithm has the advantage that it will write files without gaps, and it can buffer a large amount of data (much more than the kernel could) should one or more data channels get "really stuck"[1] for a significant period of time. Also, the ability exists to write arbitrarily large blocks when writing the destination file; this is advantageous for some file systems (e.g. Amdahl's EFS).

Disadvantages include increased computational overhead and complexity, and slower and more expensive buffering in user space than if kernel network buffering was used because user space is subject to context switches.

2.4 Bugs Fixed

2.4.1 Data Port Synchronization Bug

A bug was discovered and fixed in the original *mftp* control logic. If an *mftp* `get` or `put` was interrupted or terminated abnormally, the internal counters used to determine the data ports' TCP port numbers would get out of synchronization between *mftp* and *mftpd* resulting in all subsequent transfers hanging and failing to transfer any data.

2.4.2 Deadlock Bug

A bug was discovered and fixed in the new algorithm in which deadlock could occur during a transfer. Buffers are allocated from a fixed pool;[2] as data blocks are read from the data channels, buffers are allocated as needed. If one or more data channels should become "really stuck" the pool may be depleted of buffers such that there is no buffer

1. I.e. data doesn't arrive for a long period of time.

2. The reason for allocating out of a fixed pool, instead of just allocating memory as needed is to set a limit on the maximum size of the *mftp* and *mftpd* processes. Processes allowed to grow arbitrarily large do not behave nicely on virtual memory systems, and can cause problems for other processes and for the system in general. Also, arbitrarily large amounts of memory should not be required for *mftp* transfers; if they are, then something is wrong. Finally, if not for the fixed pool, this and other bugs would not have been discovered and fixed.

available when the stuck channel finally delivers a data block. If there is no buffer available to read that data block into, then it's gap in the file can't be filled and the other buffers can't be written out to the file: deadlock occurs.

The fix: the last buffer of the pool will only be allocated for the data corresponding to the first gap in the destination file.

2.4.3 Memory Leak

In the new algorithm, if a data channel was open, and a read on that channel returned 0 bytes, and there was no data buffered for that data channel, then the buffer allocated for the read was not deallocated. This produced a slow but steady leak of memory buffers. This bug has been fixed.

2.4.4 Memory allocated was 4 times required amount

A bug was discovered and fixed in the new algorithm in which 4 times the memory asked for was being allocated due to a typographical error.

2.4.5 Robustness Improved

Without addressing any specific bugs that have been fixed per se, the new version of *mftp* is much more robust than the old version. The performance data which follows shows that the original *mftp* would often be unable to complete any transfers with greater than a certain number of data channels. This is not a problem in the new *mftp*. It's possible that there were bugs in the old buffering algorithm which were eliminated when that code was rewritten.

2.4.6 Convex/Ultra Network Memory Exhaustion Bug

In the course of testing *mftp* on various NAS machines, a problem was encountered on all Convex machines in which an *mftp* transfer would run the system out of network memory. The symptom observed when *mftp* was run on a Convex was a total suspension of network processing while the machine was otherwise unaffected. Network processing would usually return within 10 minutes. The cause of the problem was compound:

1. ConvexOS is shipped tuned with 256 maximum mbuf clusters[3]
2. The Ultra library was allocating on the order of 128 mbuf clusters per data channel (*mftp* uses up to 10 data channels)
3. The Convex networking software doesn't handle running out of mbuf clusters gracefully

Item 2 was due to a known bug and a work-around was provided by Convex.[4]

---

3. An *mbuf cluster* is a 1024-byte memory buffer, of which there are a fixed number in kernel memory.

4. This was reported to Convex in contact report #31432. Convex responded with the following work-around: excessive network memory allocation will not take place for Ultra if the following shell environment variables are set (csh notation):

```
setenv ULTRA_SOCK_SENDSPACE 0
setenv ULTRA_SOCK_RECVSPACE 0
```

2.5 Proxy Transfers

The proxy transfer facility was fixed.

3.0 Performance Tests and Results

Tests were performed between SGI hosts at NAS and SUN SPARCstations at JPL and LeRC[5]. The purpose of the tests was to:

- Investigate the performance of the new *mftp* over AEROnet
- Investigate the performance of the new *mftp* over the NAS LAN
- Compare the performance of the new and old versions of *mftp*
- Compare the performance of *mftp* to *ftp*

Testing was performed during off hours, but not during any dedicated time. Data points in the following graphs were the average of 5 transfers attempts (sometimes not all 5 succeeded). Transfers were from disk files of varying sizes to `/dev/null`. The intent was to show typically achievable results. Plot naming convention:

<*source machine*>.<*destination machine*>.<*description*>.<*direction*>

Machines used in testing:

wk200	SGI 4D320VGX running IRIX 4.0.5
igson	SGI 4D440VGX running IRIX 4.0.5
lerc	SUN SPARCstation running SunOS 4.1.1
jpl	SUN SPARCstation running SunOS 4.1.1

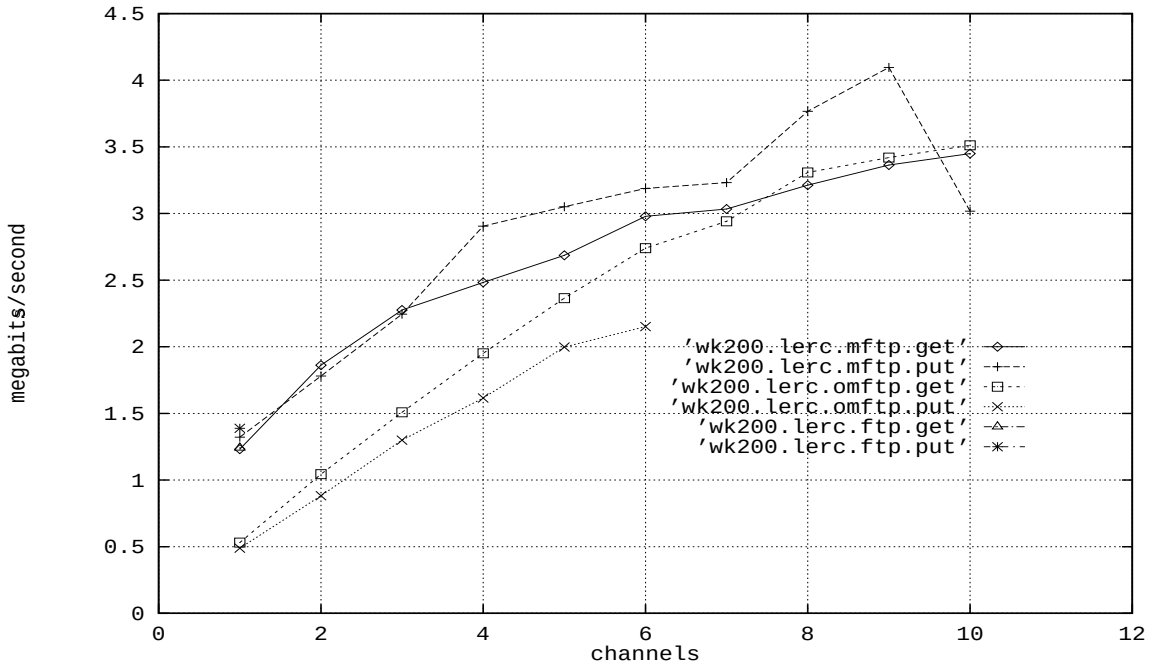
Within the plot descriptions, “mftp” denotes the new version of *mftp* and “omftp” denotes the original version. The directions are either “get” or “put” for the corresponding *mftp* operations.

3.0.1 LeRC Results

Lewis Research Center is connected to NAS over AEROnet by four T1 circuits with a combined throughput of about 6.2 megabits/sec. The delay to LeRC is typically 60ms (see Figure 3). Figure 1 shows a general upward trend in throughput as the number of channels is increased.

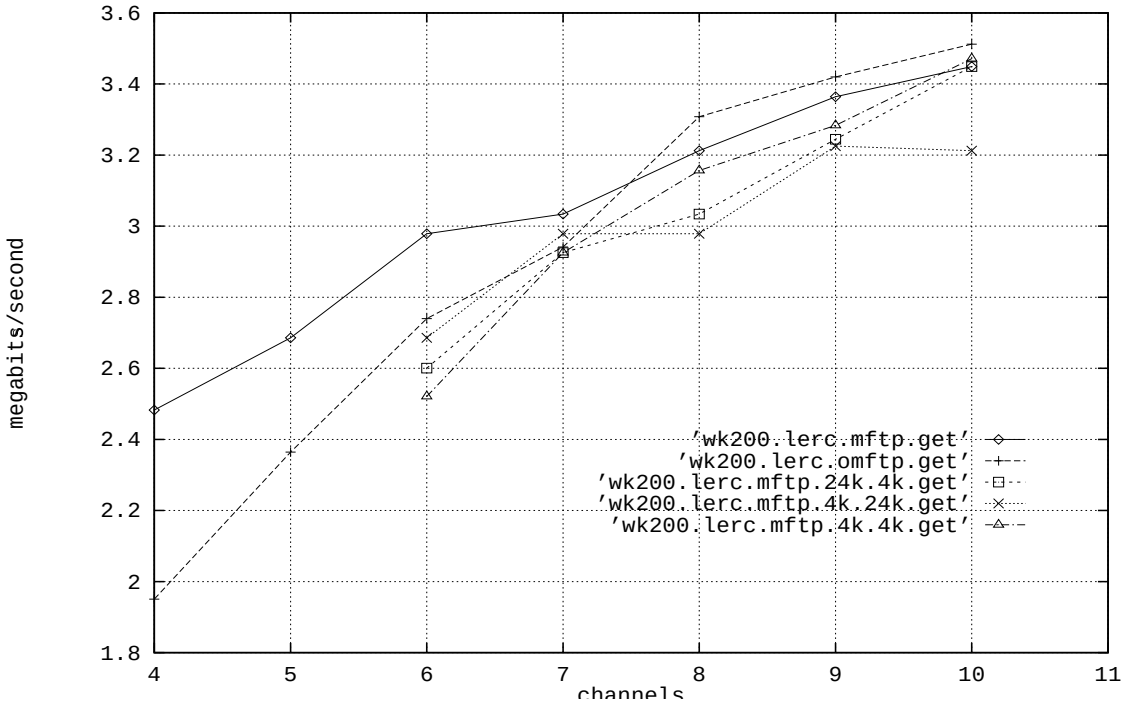
5. JPL: Jet Propulsion Laboratory in Pasadena, CA; LeRC: Lewis Research Center, Cleveland, OH. The reason for choosing JPL and LeRC for testing was the HNMS I/O modules in place at those locations. HNMS is NAS’ network management system; the IO modules are SPARCstations at remote sites.

FIGURE 1. wk200-LeRC mftp Results



In the `get` plots the new *mftp* is over twice as fast as the old version at 1 channel, but the improvement diminishes as the number of channels is increased until the old *mftp* is slightly faster, however the difference in speed at 10 channels is not significant. In the `put` plots, new *mftp* performance is better than for `get`s, but drops off at 10 channels. Old *mftp* could not successfully perform `put`s with more than 6 channels. The results for *ftp* transfers show up at 1 channel. The results of *ftp* and new *mftp* at 1 channel are comparable but *mftp* does much better with multiple channels.

FIGURE 2. wk200-LeRC mftp with Modified Buffering



In Figure 2, some experimentation was done with the buffering algorithm to see if the small, but systematic loss of throughput between the new and old versions using 6 to 10 channels was due in any part to the increase in buffer size. The size of internal buffers and the TCP send/receive window sizes were alternatively and collectively set to 4K from 24K. The results show that the effects of these changes are “in the noise” and are not significant. The systematic loss is most likely due to the increased computational overhead in the new version and not to the change in buffer size.

FIGURE 3. wk200-LeRC Ping Results

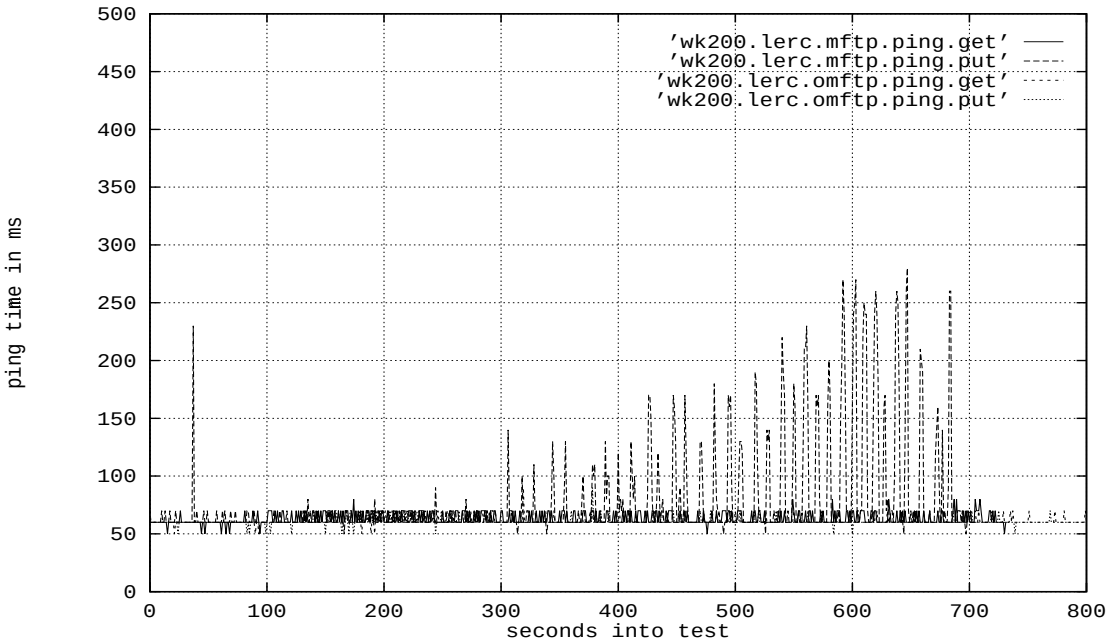
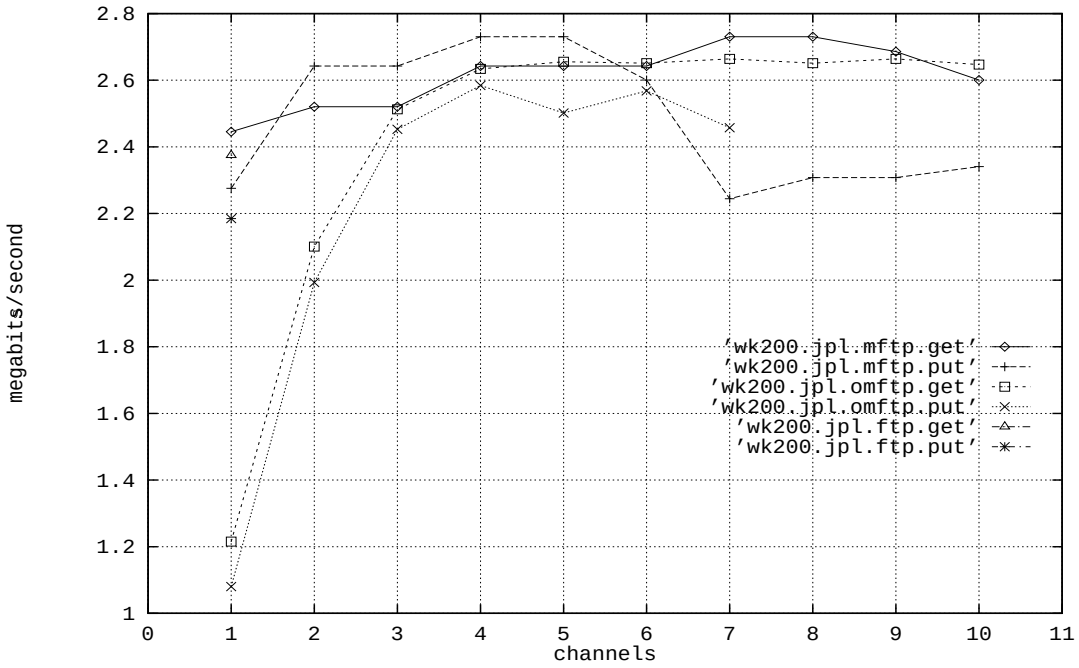


Figure 3 shows the result of *ping*s performed during the transfers shown in Figure 1. All the results indicate a lower bound on the round-trip *ping* delay of 50ms, and a more typical delay of 60ms. The results indicate an upper bound on the delay of around 70ms and stable behavior, except for new *mftp* `put` which reaches over 250ms and shows an increase in delay as the number of channels was increased. This phenomenon is even more prominent in the tests to JPL in Figure 5 below.

FIGURE 4. wk200-JPL mftp Results



3.1 JPL Results

Jet Propulsion Laboratory is connected to NAS over AEROnet by two T1 circuits with a combined throughput of about 3.1 megabits/sec. The delay to JPL is a minimum of 20ms (see Figure 5). The *mftp* results for tests between NAS and JPL are shown in Figure 4. There are many similarities to the LeRC plots:

- new *mftp* is much faster than old *mftp* for 1 channel with the difference diminishing as the number of channels is increased
- new *mftp* is comparable in speed with *ftp*
- old *mftp* `puts` fail with more than 7 channels
- new *mftp* `puts` behave slightly erratically at the top end

A significant difference is that there is no upward trend as with LeRC. This is because the bandwidth-delay product was largely accommodated by old *mftp* with around 3 channels and by new *mftp* with around 2 channels. There's a slight improvement to be had by increasing channels above these amounts. But as can be seen, increasing the number of channels can also cause degradation in performance, most likely due to additional software overhead and context switching.

Another difference is that the JPL tests achieved around 85% of the theoretical maximum available throughput, while the LeRC tests achieved only around 60%. This is

more likely a result of the routers' inefficiency multiplexing over multiple T1s, than an attribute of *mftp*.

FIGURE 5. wk200-JPL Ping Results

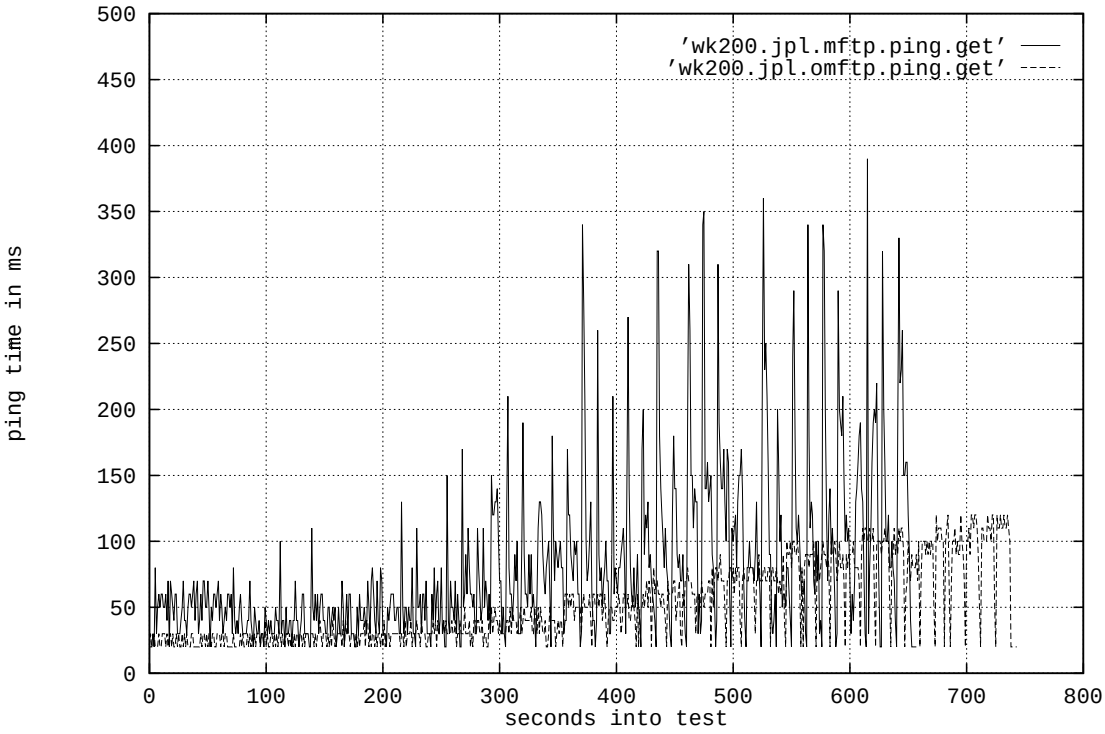
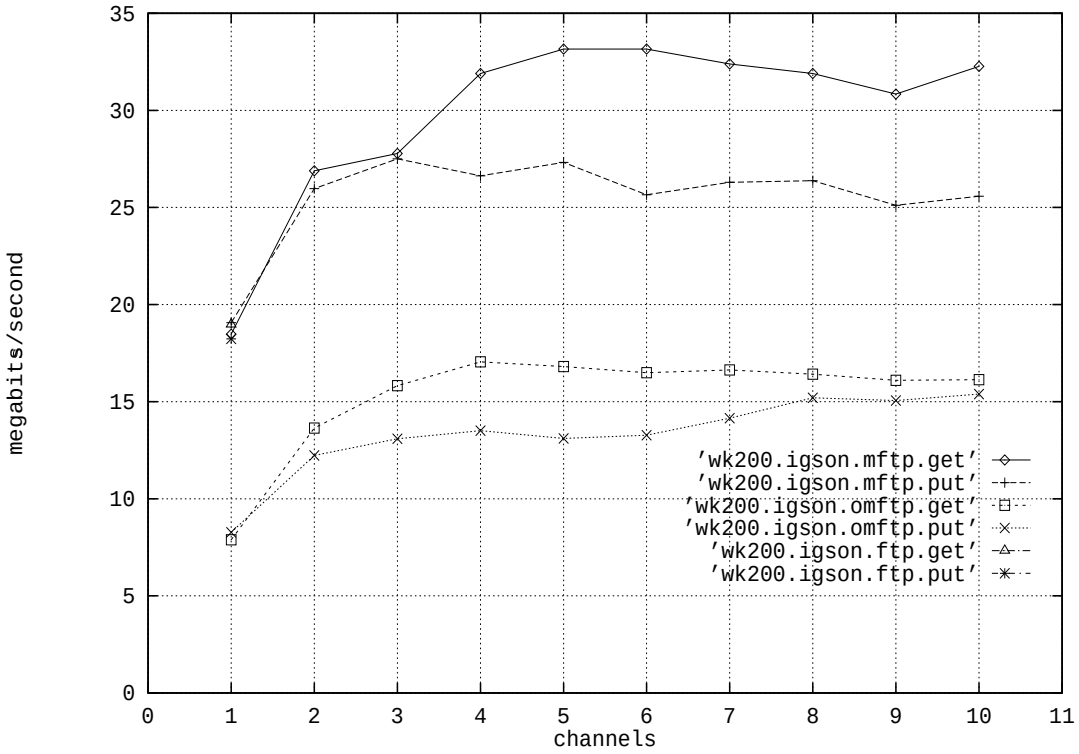


Figure 5 shows the results of *ping*s performed during the transfers shown in Figure 4. (For clarity, only the `get` plots are shown.) The results indicate a lower bound on the round-trip *ping* delay of 20ms. The round-trip delay increases as the number of channels are increased, as was seen in Figure 3, but to a much greater degree. New *mftp* experiences greater delays than old *mftp*, at times reaching over 350ms. This is over an order of magnitude greater delay than is normally experienced and is a surprising and as-yet unexplained phenomenon.

FIGURE 6. SGI-SGI FDDI mftp Results



3.1.1 LAN Results

The introduction of 24K buffers in *mftp* has resulted in greatly improved LAN performance. Figure 6 shows the results of transfers performed between two SGI workstations over FDDI. New *mftp* performs at vendor-supplied *ftp* levels for 1 channel and provides around 50% improvement in throughput for any number of channels. Also, 10 channels provides about a 50% improvement in throughput over 1 channel.

FIGURE 7. SGI-SGI Ultra mftp Results

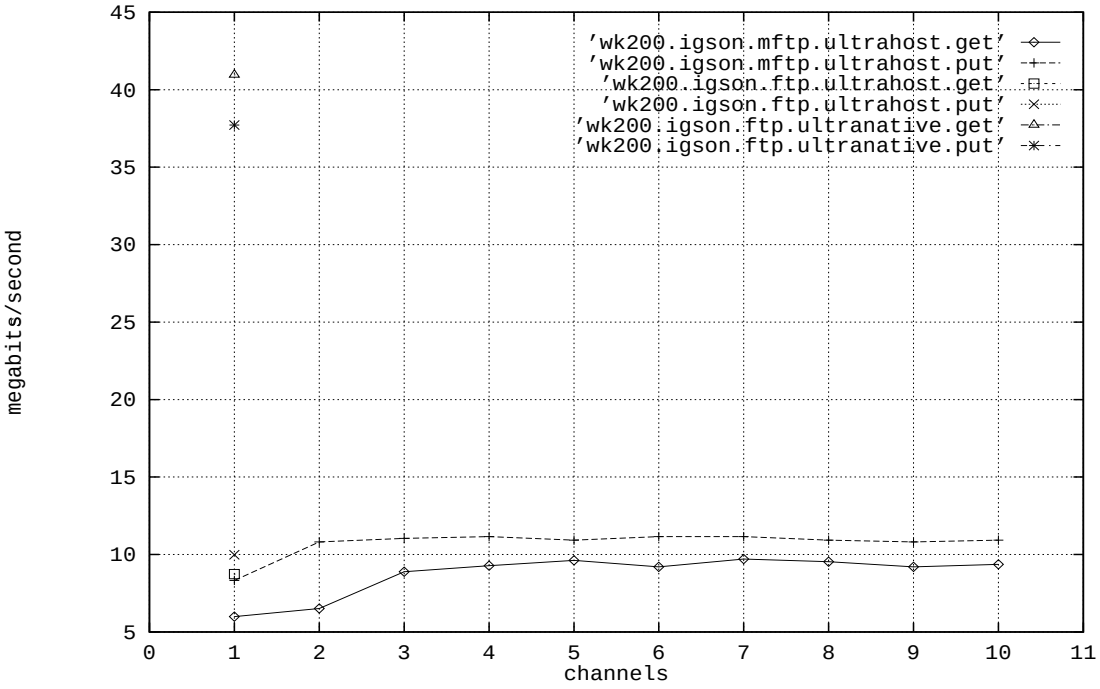


Figure 7 shows *mftp*'s performance over the UltraNet between two SGI computers. There are three scenarios: *mftp* over Ultra host-stack, *ftp* over Ultra host-stack, and *ftp* over Ultra native. (Neither the new nor the old version of *mftp* currently works over Ultra native.) The host-stack performance of *mftp* is below that of *ftp*.

4.0 Conclusion

mftp has been improved in performance and usability. While it does not provide the best performance in all cases, it improves performance in enough cases that it is worthy of consideration by NAS users. The best performance is delivered by *mftp* when the proper number of channels is selected, and this number differs between different pairs of hosts. 10 channels do not always provide the best performance.

The file restart feature makes *mftp* the tool of choice for transfers of large files over wide-area networks such as AEROnet.

Summary of *mftp* performance results:

- New *mftp* is about as fast as *ftp* (except over UltraNet) for 1 channel, and can improve on *ftp*'s performance with multiple channels. This applies over the NAS LAN and over AEROnet.

- New *mftp* is much faster than old *mftp* over the NAS LAN, and over AERONet for small numbers of channels. For large numbers of channels, old *mftp* is slightly faster over AERONet.
- New *mftp* would be expected to be much faster than old *mftp* over satellite links due to increased buffering and the large bandwidth-delay product.

Suggested topics for future work:

- *mftp* should be made to work with UltraNet, especially native UltraNet.
- Experimentation with adaptive algorithms: *mftp* could determine the optimal number of channels and buffering attributes for each transfer.
- The slope of the plots in Figure 1 suggest that increasing the number of channels for NAS/LeRC transfer beyond 10 might offer improved performance.
- Investigate increasing delay phenomenon shown in Figure 3 and Figure 5.
- Investigate bandwidth utilization difference between JPL and LeRC and other AERONet sites.
- Investigate overlapping disk and network activity by means of multiple processes or light-weight threads.

5.0 Acknowledgments

John Lekashman provided the motivation and insight to enhance *mftp*. Michelle Koblas designed the enhancements to *mftp* and did most of the implementation.

6.0 References

[1] Iannucci, D. and Lekashman, J., *MFTP: Virtual TCP Window Scaling Using Multiple Connections*, NASA Technical Report RND-92-002, April 1992.

[2] Authement, Eural, *AERONet: A Network Built to Standards*, /u/nas/news, Vol. 6, No. 9, October 1991.