DoC ISSI Emulation and Test System (DIETS)

Requirements, Architecture and Design Overview

Version 0.7
July 18, 2006

S. Quirolgico
M. Ranganathan

Advanced Network Technologies Division
Information Technology Laboratory
National Institute of Standards and Technology

K. Behnam

Institute for Telecommunication Sciences
National Telecommunications and Information Administration

Contents

1. Definitions

- *Commercial RFSS*: ("actual" or "vendor" RFSSs) RFSSs that are implemented by vendors.
- *Scenario*: A definition of a call flow or signaling interaction between RFSSs.
- *Emulated RFSS*: A process that provides all the signaling to execute a given scenario but no actual RF interface.
- *Test RFSS*: An Emulated RFSS with a test execution facility.
- *Test Launcher* : A management client that loads test cases into emulated RFSSs and initiates the test.
- *ISSI IUT*: (or IUT) ISSI Implementation Under Test. An ISSI Implementation that is being evaluated for reference testing or subsystem interoperability.
- *Reference Testing* : Isolated testing against a standard or reference implementation.
- *Subsystem Interoperability Testing*: Testing between vendor implementations of the ISSI G interface that does not necessarily involve an intervening reference implementation.
- *Node*: An entity (typically a machine) having a specified IP address.
- *Test system*: System that comprises one or more emulated nodes with the ability to initiate and respond to signaling actions and one ISSI IUT.
- *Emulator*: Synonymous with ISSI Emulator.
- *Message trace*: Signaling trace (comprising SIP Call Setup and Push To Talk signaling) for call setup, mobility and call teardown.

2. Introduction

The DoC ISSI Emulation and Test System (DIETS) provides a set of tools for testing emerging ISSI technologies. More specifically, DIETS aims to facilitate reference testing and subsystem interoperability testing of commercial Radio Frequency Subsystems (RFSSs) and to enable controlled laboratory experiments to evaluate the ISSI design. An emulated RFSS can execute a call setup scenario and provides the necessary signaling for the call setup but does not have a functional radio interface.

In this document, we make a distinction between reference testing and subsystem interoperability tests and treat these separately.

Reference testing tests dynamically assess the interoperation of a given implementation under test by testing interactions against a standard or reference implementation. Reference testing tests necessitate the interaction of a vendor implementation with a reference RFSS implementation.

Subsystem interoperability tests, on the other hand do not necessitate the presence of a standard or reference RFSS. An subsystem interoperability test can consist of two or more vendor implemented RFSSs interacting with each other to exercise a scenario. The vendor implementations are not necessarily from the same vendor. Note that passing reference tests does not guarantee subsystem interoperability between vendor implementations nor does subsystem interoperability imply that reference testing will pass.

Conducting reference testing tests requires:

1. A test suite
2. A reference implementation.
3. A test driver to exercise the interactions in the test suite.

We will use the test cases developed by the ISSI TG and authored by EADS (henceforth referred to as the "EADS document") to define a test suite. To support such tests, the DIETS tools suite provides a test driver to initiate interactions between vendor RFSSs and a test tool which emulates the network messaging behavior of an actual RFSS.

Subsystem interoperability tests of vendor RFSSs requires tools to perform analysis of how such RFSSs interact with other RFSSs while exercising a scenario, treating the components in between as a black box.

For subsystem interoperability testing, DIETS provides non-invasive trace capture and visualization tools to analyze interactions between RFSSs.

2.1.Functional Summary

In the first phase, the DIETS project will provide the following test tools:

1. A protocol emulator to exercise the protocols defined in the ISSI Specification document. This protocol emulator is henceforth referred to as the "ISSI Emulator". The purpose of the ISSI emulator is to provide a stand alone emulation of the scenarios described in the call flows section (appendix C of the ISSI specification) and the test cases defined in the EADS test document so as to verify correctness and completeness of the ISSI protocol suite.
2. A test tool to interact with vendor developed RFSSs and verify compliance: The test component of DIETS is henceforth referred to as the "ISSI Tester". In this mode, DIETS takes a test script and stimulates responses from vendor developed RFSSs or will respond to stimuli from vendor RFSS implementations. The purpose of reference testing testing is to analyze vendor implementations in isolation by treating them as a "black box" and analyzing interactions between an ISSI Reference implementation and a vendor implementation. For this we need a test driver to drive the interactions. DIETS will provide the test driver framework to exercise all the tests in the EADS test document [Ref].
3. A packet capture and visualization tool to display the interactions between RFSSs captured during subsystem interoperability testing. Note that subsystem interoperability testing does not have to involve a reference implementation. The primary purpose of trace capture and visualization is diagnostic. DIETS will provide a set of tools that can capture (log) the interactions between vendor RFSSs and that can display these as signaling traces.

In this document, we specify the requirements and architecture of DIETS and its support for specific objectives. Following is a detailed set of requirements for the DIETS tool suite.

2.2. Requirements

The ISSI test system will be designed to meet the following requirements:

1) *Distributed architecture*: the system must be able to support communication between distributed RFSSs running on hosts with differing IP addresses while providing a central point of monitoring and control for all the emulated RFSSs involved in a test scenario.
2) *Multiple commercial RFSSs*: the system must be able to non-obtrusively capture interactions between multiple commercial RFSSs to facilitate subsystem interoperability testing.
3) *Portable platform support*: the test system must support major operating systems including Windows XP and Unix.
4) *Extensible scripting*: The test system should support an XML based scripting environment to allow users to extend the suite of reference testing test cases.
5) *Black Box Testing*: The test system should test only external input / output message *protocol* behavior of RFSSs and not their internal behaviors.
6) *Apriori knowledge of topology* : The test system must have knowledge of the ISSI configuration for a given test (i.e., RFSS IDs, IP addresses, and ports) before test execution begins. Note that this cannot include apriori knowledge of the RTP ports used for Push to talk Signaling. These ports are dynamically assigned during call setup (ref. the ISSI messages and procedures design document).
7) *Manageability and configurability*: The distributed test system should be manageable and configurable using a central configuration and management tool. The system must provide a management client for centrally managing the distributed test system. The management client will provide a user friendly GUI-based user interface to monitor and control the test system.
8) *Trace analysis tools*: The system must provide a set of tools for analyzing message transactions (i.e., message traces) between RFSSs. These tools must be used in conjunction with the test system to support testing of RFSSs. These tools will be bundled with DIETS, but can also be used independently of DIETS. These tools must provide support for:
    a. *Client-side trace capture*: Components are needed for capturing message transactions, or "traces", between all RFSSs in the system
    b. *Server-side trace aggregation* A centralized component is needed to aggregate and display trace logs generated by distributed trace capture components associated with each RFSS in the system.

9) *Functional Extensibility*: The tool may later be extended to support performance tests. It should be architected with this potential future requirement in mind

3. Architectural Overview

The distributed DIETS architecture for reference testing is shown in Figure 1. This architecture comprises multiple RFSSs, one of which can be a IUT as well as a single central management client/controller. Note that an RFSS in the figure below can be either an emulated or commercial RFSS. Note that in various places, the EADS document states "Tests may be performed conjointly or in isolation". In this document, we refer to multiple vendor RFSS tests as Subsystem interoperability tests and single vendor RFSS tests as reference testing tests. The same scenario may be run in both cases but the set of tools are different in each case. To perform a reference testing test as defined in the EADS test document, only *one* of the RFSSs can be an IUT. The rest of the system is emulated. The same test may be run as a *subsystem interoperability* test where more than one node is an actual RFSS. The EADS document does not make a distinction between these two types of testing but we treat these as separate.

The diagram below shows emulated RFSSs interacting with each other using a defined control interface with a central *Management and Configuration client* and a single unit under test. Vendor developed RFSSs do not export a management interface and hence we do not depict the controller as interacting with a vendor developed RFSS. For input output behavior (black box) testing, the test system can be driven by stimuli (messages) from either emulated or actual RFSSs. However, assertions on messages are only checked locally at the emulated RFSSs.

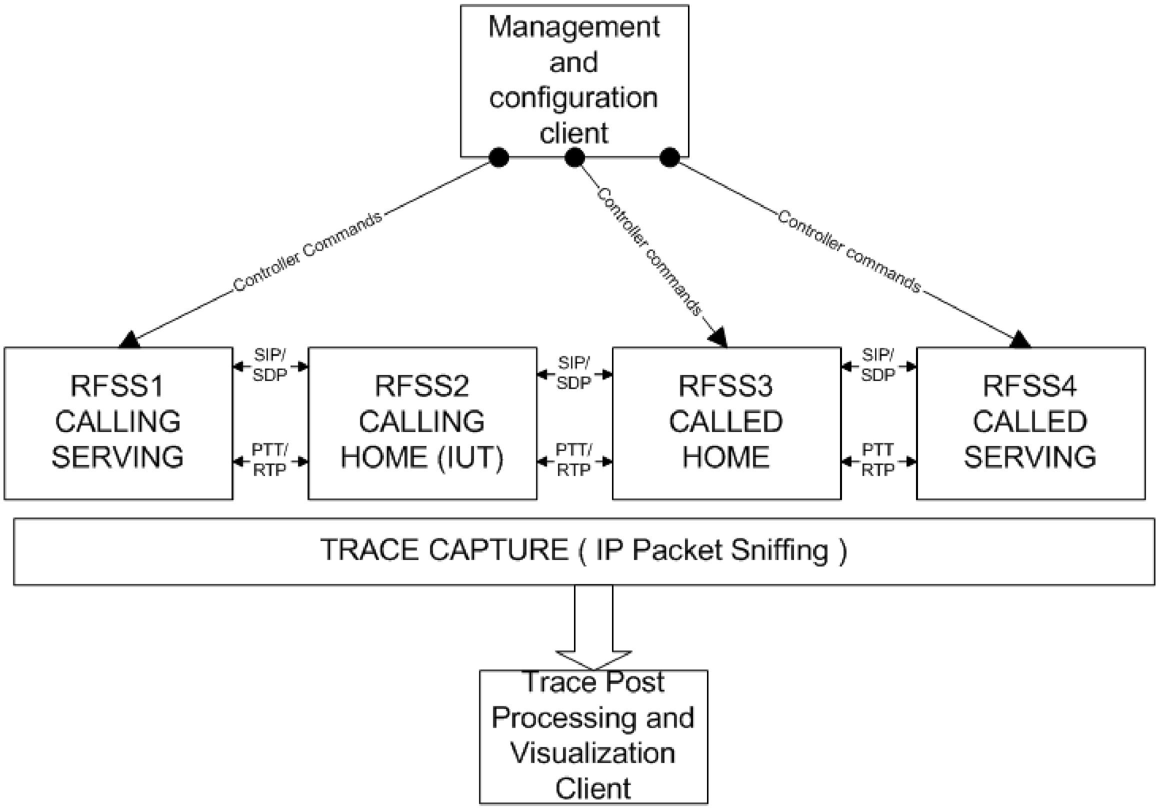


Figure 1. Distributed DIETS architecture.

4. Test System Architecture

The purpose of the test system is to stimulate responses from RFSSs under test by setting up calls with vendor implemented RFSSs in the call path to *react* to calls initiated by a vendor implementation under test and to check for expected properties of input and output messages locally at the emulated nodes. An emulated RFSS consists of two distinct parts: (1) a call Setup Emulator and (2) a Push To Talk Emulator. The figure below shows an architectural overview of an emulated RFSS. Each emulator consists of the following protocol stacks on top of which we build higher layer abstraction:

1. A SIP Stack (we use NIST-SIP for this purpose)
2. An RTP stack on top of which we implement the Push To Talk protocol of the ISSI specification.

We have based our design of the emulated RFSS roughly on same lines as the ISSI specification. The function of each of the major components of an emulated RFSS maps directly to corresponding functional descriptions in the ISSI specification. We refer the reader to the ISSI specification for details and describe their function here briefly.

Each emulator houses a Unit to Unit Call Control Manager, a Group Call Control Manager, a Unit to Unit Mobility Manager and a Group Mobility Manager. The functions of each of these entities maps to the corresponding functions in the ISSI specification. The Unit To Unit Call Control Manager performs the function of setting up calls between Subscriber Units for point to point call setup. The Group Call Control Manager performs the function of setting up calls between subscriber units and groups. Each group is pre-assigned a home RFSS (specified in the topology configuration file). The Mobility Manager tracks the movement of Emulated Subscriber Units. The Unit to Unit Mobility Manager tracks the movement of subscriber units between emulated RFSSs and the Group Mobility manager tracks the movement of members of a group as they move from RFSS to RFSS. The Transmission Control Manager sets up and manages point to point sessions for SU to SU calls as well point to multipoint sessions for group calls. Each of these components (CC Manager, Mobility Manager and Transmission Control Manager) exports an Event-Oriented interface to an emulated Subscriber Unit. Each Emulated SU has a handle to talk to the Mobility Manager, Call Control Manager and Transmission Control Manager in the emulated RFSS via a Service Access Point (SAP) interface as described in the RFSS specification.

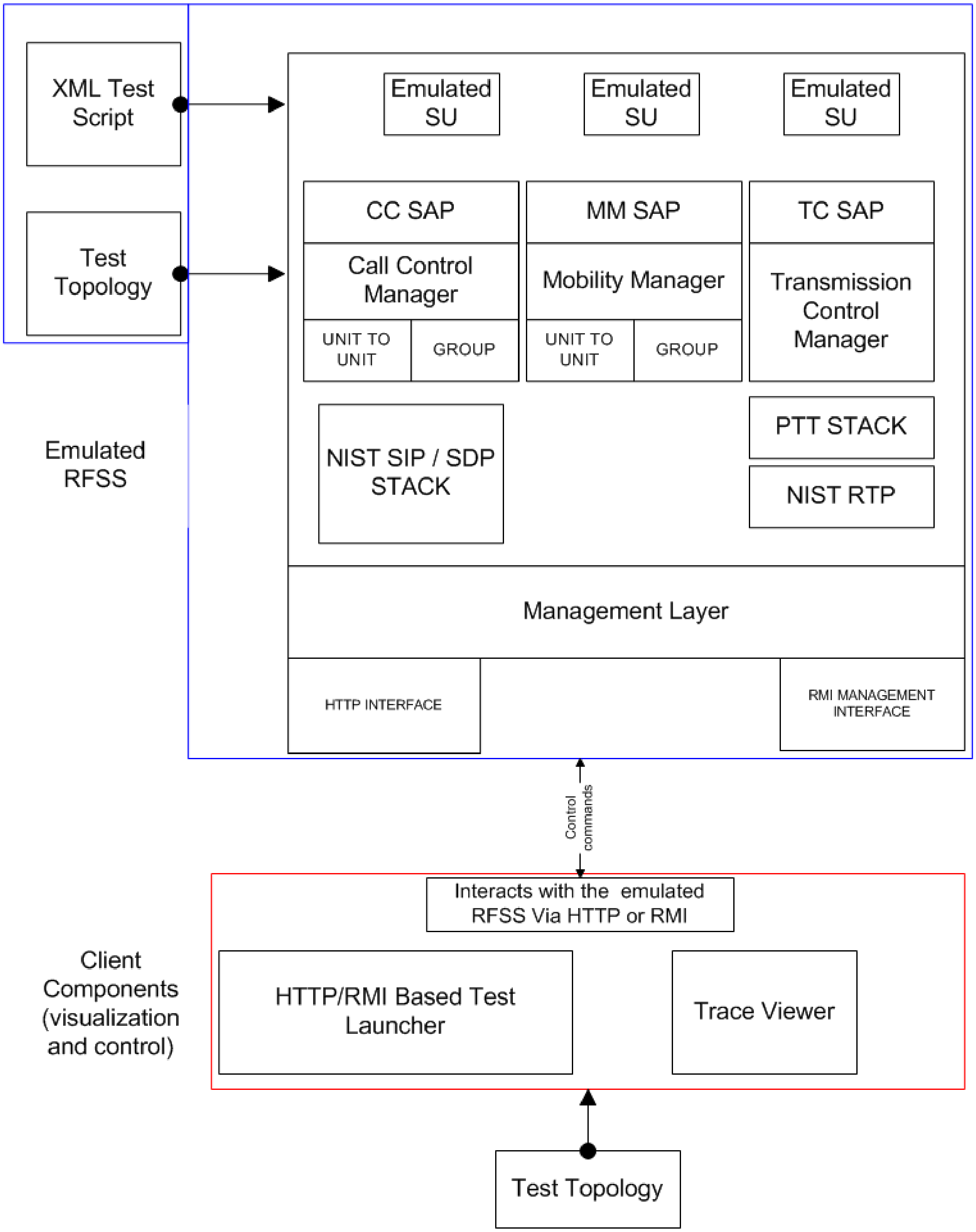


Figure 2: Software Architecture of an emulated RFSS.

4.1.Operational Description

For reference testing, we assume that a single RFSS at a time will be tested as part of a scenario. The EADS test document specifies several tests where multiple vendor developed RFSSs are involved. Although all RFSs are involved in the test, our unit test driver framework involves interacting with and analyzing responses from an isolated vendor developed unit under test. We treat multi vendor developed RFSSs separately. We call these *subsystem interoperability tests* although the EADS document does not make a clear distinction between subsystem interoperability and reference testing. We provide a test driver framework to execute an emulation of each of the test cases defined in the EADS test document. In performing a test, a single emulated RFSS may be replaced by a vendor implemented RFSS. The test consists of sending requests to the unit under test and analyzing responses from the unit under test for expected properties or by reacting to messages sent from the IUT. Note that because the ISSI specification does not include a management or external control interface for a vendor developed RFSS we cannot fully automate the testing. We can only automate those tests for which the stimulus is originating from an emulated RFSS. We make several assumptions which are outlined below.

We assume that the topology (i.e. assignment of RFSS IDs to IP address and SIP listen port) are known *a priori* and are made available to each emulated RFSS and to the test controller at startup time. Note that there is no defined format for inputting this information into an actual vendor implemented RFSSs. Therefore, each commercial implementation under test will have to manually configure this information using its own proprietary scheme.

The test controller and each of the emulated RFSSs have access to the same topology description and to the same test database. A network node (i.e. machine with a given IP Address) can contain one or more emulated RFSSs. In order to run a test, the test launcher loads the same test script in each emulated RFSS and then instructs each emulated RFSS to run the test case. Each emulated RFSS starts executing the script at the same time. Because the execution is time based and because each emulated RFSS loads the same test script, no additional test coordination is necessary between emulated RFSSs at run-time. We assume that some vendor specific mechanism will be defined for effecting test actions on real RFSSs. For example, in testing mobility scenarios, an RFSS may have to support movement of an SU to its domain. The programmatic interface for effecting this action is not defined in the ISSI specification and hence is outside the scope of this document.

Test assertions are checked locally at each emulated RFSS. After the test is complete (determined by waiting for a pre-defined period of time), the trace viewer can retrieve the traces from the emulated RFSSs (provided the emulated RFSSs are logging traces locally). Alternately, traces can be gathered from the trace capture daemons. The logging and trace viewing architecture is discussed below.

4.2.Specifying Test Cases

Test cases are specified in XML as time-based event oriented scripts. Each nested XML tag is an “action” to be performed. Each action specifies an event to occur at a certain time into the test. For example, in the test script below, at time 0, SU “12” places a unit to unit call to SU “34” and at time 10 seconds SU “12” moves to RFSS with identifier 3. All emulated RFSS’s start executing the script at the same time when the central controller sends a command to do so. While exact synchronization is impossible, so long as the time granularity of the test system is less than the synchronization error, the system should generate reproducible results. We assume that a granularity of 1 sec. is sufficient for the purposes of specifying actions and that the emulated RFSS’s test start time can be synchronized to within a second of each other. Here is a specification for a test:

```
<test-script
            systemId="2"
            wacnId="1"
>

<!-- At time 0 SuId 12 calls SuId 34. This is the initial condition in Page 308 of the ISSI spec.  -->
<su-to-su-call-scenario
            id="su-to-su-call-0"
            time = "0"
            callingSuId= "12"
            calledSuId="34"
/>
<!-- at time 10 seconds, the suId 12 roams to rfssId 3. It takes 3 seconds for the roaming to occur -->
<roaming-scenario
            id="roaming-scenario-1"
            time = "10"
            suId = "12"
            rfssId = "5"
            delay="3"
/>

</test-script>
```

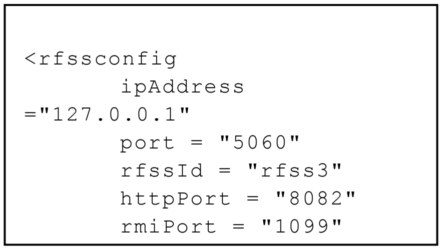
4.3.Emulating Mobility

In-call roaming is the act of an SU moving from one RFSS to another RFSS and resuming the call there. In-call roaming is emulated without any central control as follows:

- At the time at which the roaming SU departs an RFSS, the RFSS from which the SU departs removes the RFSS from its served set.
- At the time at which the emulated roaming SU is supposed to arrive at the new Serving RFSS, the new emulated serving RFSS (which is running the same script as the old serving RFSS) initiates the mobility related SIP signaling actions (i.e. ReINVITE, REGISTER, BYE) as described in the ISSI Messages and Procedures document.

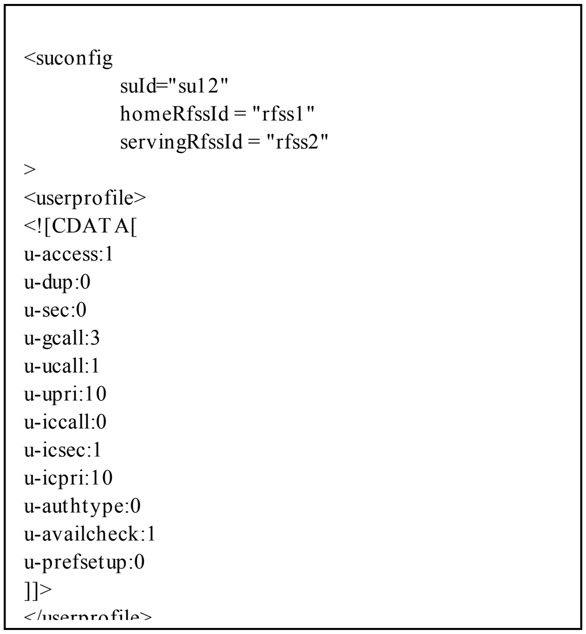
4.4.Topology Specification

The topology of the test system is specified by an XML file which is supplied to each emulated RFSS. The XML file specifies the WACN, System, RFSS ID, the IP address and port at which each RFSS is expected to receive SIP signaling messages for every real and emulated RFSS in the system. It also specifies a set of subscriber units and the initial Serving RFSSs for each emulated SU. Here is an example topology file tag that specifies the configuration of an emulated RFSS :



```
<rfssconfig
       ipAddress
="127.0.0.1"
       port = "5060"
       rfssId = "rfss3"
       httpPort = "8082"
       rmiPort = "1099"
```

The configuration above indicates that the IP address of the RFSS (where it receives SIP and RTP messages) is 127.0.0.1. The port at which it receives SIP messages is 5060. Note that the RTP (PTT) ports are not specified at this time – they are randomly assigned at call setup time. The RFSS ID assigned to the RFSS is 3. In addition, for an emulated RFSS, note that the HTTP port at which the RFSS can be contacted. The following is the configuration for a subscriber unit. Note that this includes the user profile for the subscriber unit and the RFSS that is initially its Serving RFSS.



```
<suconfig
          suId="su12"
          homeRfssId = "rfss1"
          servingRfssId = "rfss2"
>
<userprofile>
<![CDATA[
u-access:1
u-dup:0
u-sec:0
u-gcall:3
u-ucall:1
u-upri:10
u-iccall:0
u-icsec:1
u-icpri:10
u-authtype:0
u-availcheck:1
u-prefsetup:0
]]>
</userprofile>
```

Similarly, Groups are specified a-priori along with the Home RFSS of the group and the set of subscriber units that belong to the group as follows:

```
<sgconfig
          sgId= "group56"
          homeRfssId = "rfss3"
>
<subscriber
          suId = "su12"
/>
<subscriber
          suId = "su34"
/>
<subscriber
          suId = "su56"
/>
<groupprofile>
<![CDATA[
g-access:1
g-agroup:0B4561A27271
g-pri:2
g-ecap:1
g-eprempt:0
g-rfhangt:0
g-ccsetupt:0
g-intmode:0
g-sec:1
g-ic:0
g-icsecstart:0
]]>
</groupprofile>
```

All of this information is made available to every RFSS at startup. The tests use SU and RFSS identifiers that are specified in this XML document.

4.5.Verifying correct operation of Units Under Test

Refrerence tests will be based upon interactions of a vendor developed RFSS with an emulated one. Compliance with the specification is checked by assertion testing at the emulated RFSSs. For example, for simple SU to SU call setup, the INVITE must be sent out, arrive at the destination and the Calling Serving RFSS sending the INVITE must later get an OK. Each SIP header for the messages sent and received are known. The Calling and Called emulated RFSSs check for the fields to make sure that they comply with the specification. Calls are expected to complete in a fixed time bound. If not, a timeout error occurs and the test fails. While testing against an actual implementation, we apply the same assertion based approach to testing at the emulated nodes. Clearly, no such checking is possible at the actual RFSSs under test because we do not have access to the internals. However, if we assume that there is a single RFSS under test, we can check messages originating from the Implementation Under Test.

5. Message Logging

The purpose of message logging and trace display is for subsystem interoperability testing. Subsystem interoperability testing does not need to involve a Reference emulator or test driver. Subsystem interoperabilty tests can use the same scenarios as are used for reference testing. The test driver does not need to be involved in the test. For subsystem interoperability analysis, message logging and visualization is a useful diagnostic tool.

Logging requires (1) Gathering of the relevant data to be logged (2) Displaying information about sent/received messages in a simple to understand and intuitive fashion. There are two components – packet capture and trace visualization.

5.1.Packet Capture Component

To adequately test actual RFSSs, information about their sent and received messages must be captured. Components for capturing such information should be non-invasive and allow users to easily configure and "plug into" the test framework. Emulated RFSSs incorporate application level logging of formatted SIP and PTT messages. There are two approaches that may be considered for capturing information from an actual RFSS:

1. *Capturing from emulated RFSS*: Here, an emulated RFSS captures information received from an actual RFSS as well as messages sent from the emulated RFSS to the actual RFSS; that is, information about messages sent and received by an actual RFSS is captured by a (peer) emulated RFSS. Although this approach might be suitable for cases where an actual RFSS interacts directly with an emulated RFSS, *it does not work between two or more actual RFSSs*. In order to monitor messages between actual RFSSs, a more general approach is required.

2. *Packet capture application*: Here, a standalone application is used to capture (sniff) messages sent and received by RFSSs involved in a test scenario. This application runs separately from the RFSS in the background, capturing specific messages and processing them. There is a front end trace visualization component which retrieves traces from the packet capture daemon, collates and sorts them and displays them in an easy to understand fashion.

Of these alternatives, the packet capture application is the least invasive and is the easiest for users to install and configure. It is, however complex to implement because of the dynamic nature of the protocol. We describe this in greater detail below.

A packet capture application is used to capture SIP and RTP messages sent from, or received by, any actual (or emulated) RFSS as shown in Figure 2.

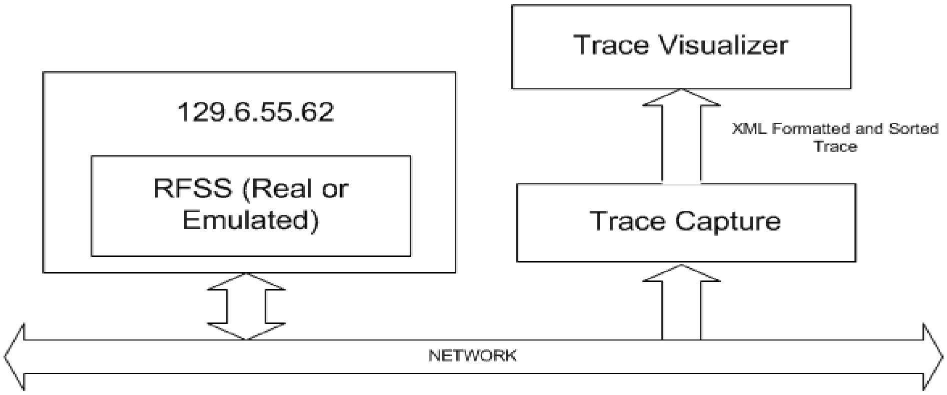


Figure 2. Packet capture application

An ISSI packet capture application comprises a number of components as shown in Figure 3. The figure represents the processing steps involved in packet capture and post processing.

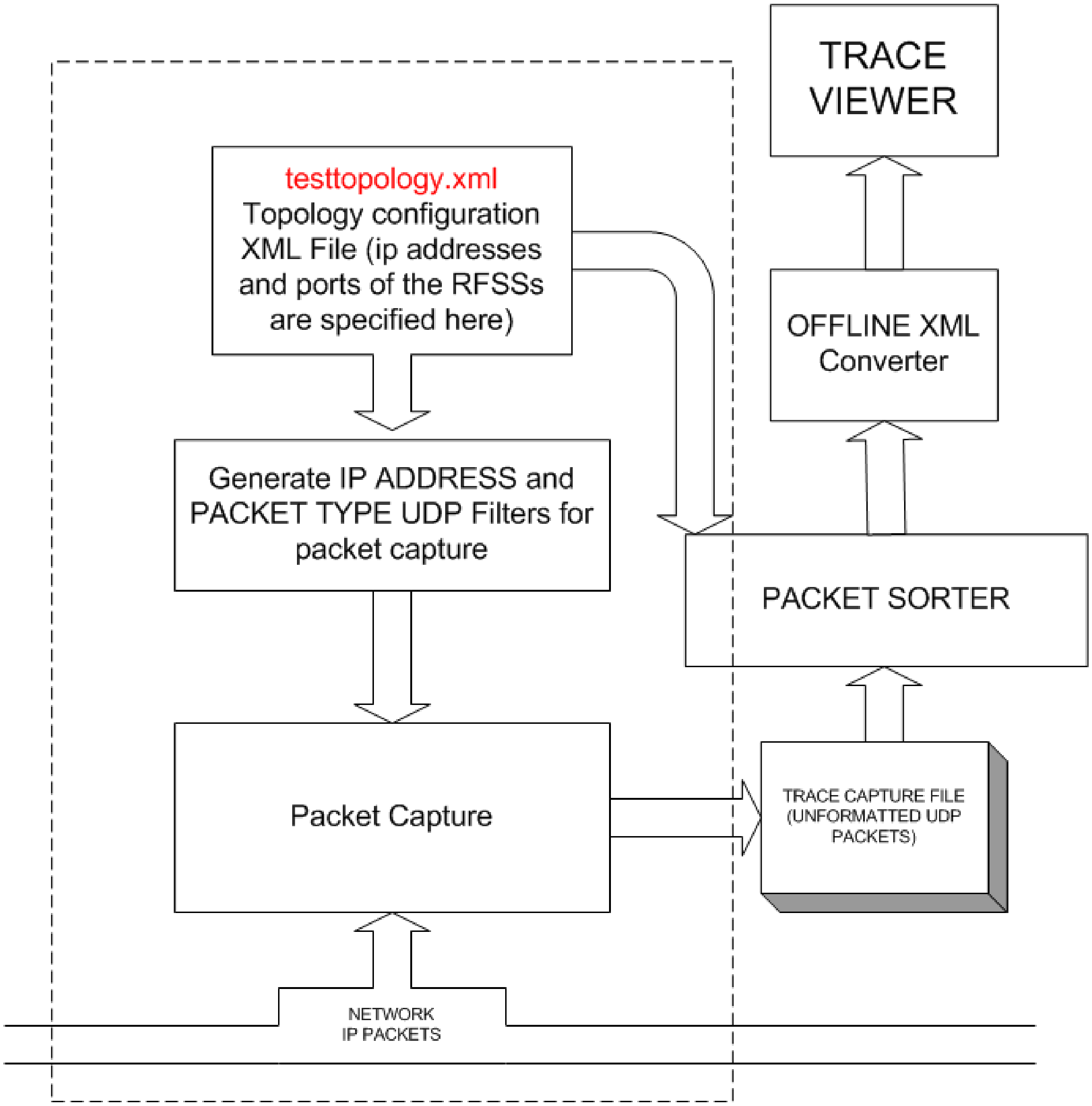

TRACE VIEWER
testtopology.xml
Topology configuration XML File (ip addresses and ports of the RFSSs are specified here)
OFFLINE XML Converter
Generate IP ADDRESS and PACKET TYPE UDP Filters for packet capture
PACKET SORTER
Packet Capture
TRACE CAPTURE FILE (UNFORMATTED UDP PACKETS)
NETWORK IP PACKETS

Figure 3. Packet capture components.

To access incoming or outgoing SIP and RTP packets from a (local) RFSS, a packet capture application requires a subcomponent for accessing raw sockets. There are several options here for packet capture. (1) *libpcap* or *WinPcap* libraries can be used to access raw sockets and capture byte streams from UNIX or Windows platforms, respectively. (2) Ethereal can be used for packet capture. In either case, packet capture is passive. There is a single packet capture daemon for a given test network. The captured packets are logged, processed offline (i.e., after running the tests), and formatted into a XML structure that can be displayed by the Trace Viewer application. In order to reduce the amount of online processing (i.e., during a test), online processing is restricted to a simple filtering function. Figure 3 shows a block diagram of a packet capture tool and the various steps that need to be performed. We explain this architecture in detail below.

The packet sniffer needs to have knowledge about the endpoints (IP addresses) and ports to which packets are headed. For this purpose, it takes an XML file that describes the test topology. The XML file contains the IP addresses and ports where the participating RFSSs are expecting to receive SIP signaling traffic. Tools such as ethereal can filter packets based upon filter rules (i.e. port, IP Address etc). Unfortunately the ports for the PTT traffic is not known apriori. This port exchange is part of the SIP call setup. Thus Ethereal has to log all the SIP and PTT UDP traffic between the IP addresses involved using only IP address and packet type (PTT or SIP) for filter. During offline processing, the topology is used again, using the SIP ports, which are static and defined in the topology file, to sort the SIP traffic. Each SIP request and response is parsed and the ports for the push to talk UDP packets are extracted from the SDP of the SIP request/response. Using these ports, the RTP traffic for push-to-talk signaling can be extracted from the message log and formatted as XML traces. Once this processing has been completed, two files are generated – one for push to talk signaling and another for SIP call setup signaling. These can then be input to the Trace viewer tool and displayed in an intuitive fashion. Offline processing is sufficient for the test frame work.

5.1.1.Analyzer

TBD

5.1.2.Communicator

TBD

5.2.Trace Aggregation and Visualization Tool

6. Central Controller

TBD