TEST METHODOLOGIES

Prevent Outages: Characterize the Unexpected under Real-world Load.

An Integrated Test Solution from Mu Dynamics and Agilent Technologies to mitigate protocol errors, day-zero attacks and hidden vulnerabilities in network equipment, and ensure uninterrupted service delivery and maximum performance under worst-case conditions.

James Maze, Director of Systems Engineering, Mu Dynamics
Peter Atanasovski, Product Manager, Agilent Technologies

Mu Dynamics™

686 W. Maude Avenue, Suite #104
Sunnyvale, CA 94085
866-276-4640 toll-free
408-329-6330 international
408-329-6317 fax

Agilent Technologies

5301 Stevens Creek Boulevard
Santa Clara, CA 95051
408-553-7777 international

Table of Contents

Executive Summary

As Network Operators look to migrate much of their infrastructure to real time IP services like VoIP, IPTV, and IMS, the reliability and security of the network has become increasingly important. The systems supporting these services are highly interconnected, and many are built on increasingly open, standardized communication protocols and software. Unfortunately, with increased complexity comes increased risk, and thus Network Operators must bear the responsibility of ensuring these networked applications and systems are going to be reliable, available and secure in a production environment. Without a comprehensive Service Assurance strategy in place, enabled by appropriate tools and test methodologies, a high-profile service outage is all but guaranteed.

This paper will explore how an integrated approach to Service Assurance testing offers unique and powerful benefits to customers. By combining the benefits of traditional load and performance testing tools from Agilent Technologies, and Mu's innovative Service Analyzer, customers can finally make critical product purchasing and deployment decisions based on real-world reliability, availability, security, performance, and Quality of Experience metrics. Services can be deployed with confidence and assurance that they will remain robust and available, despite the constant threat of exposure from attacks and other anomalous traffic conditions.

Solution Overview

Agilent Technologies and Mu Dynamics address vastly different problem spaces within network testing. At the most basic level the Agilent N2X characterizes how a system under test (SUT) behaves and performs when it receives expected input at scale; the Mu Service Analyzer helps characterize what happens when the unexpected occurs. This paper will explore how the two systems can be used together in a complementary fashion.

There are a variety of test tool solutions available from Agilent Technologies. This paper will focus on how to incorporate the Mu-4000 service analyzer with the Agilent N2X Multiservice Test Solution, the N2X Tcl API and N2X Tcl libraries that are widely used to automate testing. The Agilent N2X Tcl API can be used to completely automate every aspect of operation of the N2X test solution.

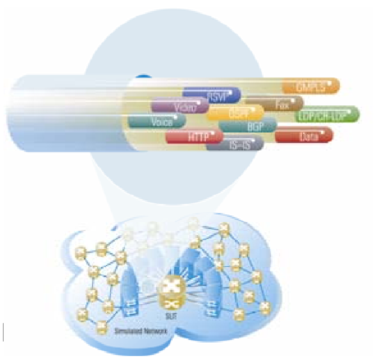
The Agilent N2X provides a platform for validating the performance and scalability characteristics of next-generation network equipment. N2X is designed to test IP forwarding devices that deliver video, VoIP, data services and business VPNs. It is used in "out-of-service" lab environments to test at real-world scale with realistic traffic.

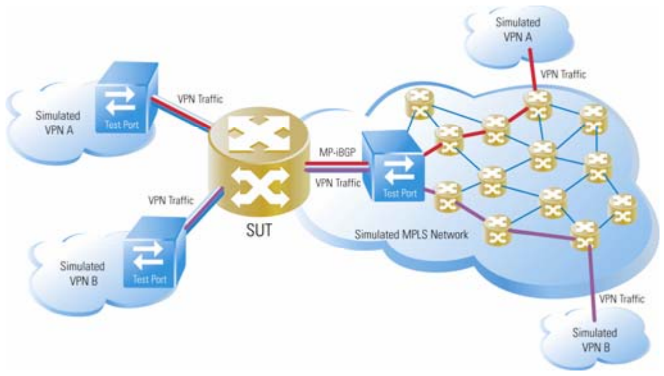
Agilent N2X emulates real network devices by statefully initiating and responding to protocol messages, allowing you to replace large testbeds of network equipment. It will characterize failure recovery times; quantify protocol performance at scale and simulate control plane flapping.

Agilent N2X will also verify that a device or network can correctly prioritize and forward millions of flows from multiple subscriber services, uphold QoS service level agreements and quantify the ultimate limits under extreme loads. It can generate millions of unicast and multicast streams; measure per-service, per-subscriber or per-VLAN performance;

validate mixed and proprietary encapsulations; test new protocols and report packet loss, latency, throughput and other performance parameters.

Integrated Traffic and Protocol Emulation using Agilent N2X:





u
 Dynamic's test equipment, in contrast, is focused on evaluating the "exception" handling capabilities of the protocol implementations of the device/system under test, and characterizing overall device robustness and security. A Mu analyzer sends a series of malformed, or mutated packets to the DUT (Device Under Test) using an automated process that is generally referred to as "Protocol Fuzzing". The Mu sends a very large number of protocol-specific test cases in a very structured and repeatable pattern. Each mutated sequence contains a single malformed packet in which a single field or value has been altered from the standard. A typical Mu analysis may consist of anywhere between tens of thousands and over a million "mutated" packet sequences.

The Mu determines which mutations impact the services supported by the DUT through an automated fault isolation process. A test begins by sending one or more valid sequences to the DUT to verify that it is operating normally, a process referred to as instrumentation. Normally, the protocol being tested is instrumented, but this may be augmented (or even replaced) by any other supported protocol if desired. If the Mu is able to communicate with the DUT using all of the "instrumentation" protocols it will then send up to 16 related (the same message and field are altered using variations of a single mutation type) mutated sequences. Following these mutations another set of valid instrumentation messages are sent. If the DUT responds as expected the process continues and the next set of mutations are sent. If the DUT fails to respond to instrumentation the Mu will periodically resend the instrumentation messages to determine when the DUT is responsive again (optionally, the Mu may take proactive steps to bring the DUT back on line). Once valid communication is re-established the Mu will resend each of the individual mutated sequences contained in the previous set surrounded by valid instrumentation packets. This is done to reproduce and validate the issue and, if possible, isolate the responsible sequence down to a single mutated packet.

The fault isolation procedure described above will identify any anomalous behavior in the control plane protocols being used for instrumentation. While this is extremely useful, we may also wish to characterize effects outside of the operation of the control plane. This is best illustrated through examples.

When testing a router the isolation procedure described above will tell us if the routing protocol, OSPF for instance, stops responding for some period of time. The Mu's system-level monitors can also detect system-level crashes and can help isolate the root cause of the failure. However, this is a limited perspective into how the router is actually functioning. Does it continue to forward packets using the latest routing table or does it immediately shut down. The answer to that question has a significant effect on the impact of the discovered vulnerability. Although it seems less likely, one can also imagine a situation in which the protocols continue to communicate but packet forwarding ceases or is significantly degraded.

Similarly, when testing a SIP call processing device such as a soft switch or session border controller (SBC) we are certainly interested in the operation of the SIP protocol as this determines our ability to make new calls. However, it would also be useful to know if the mutation had any effect on existing calls. For example, imagine we are testing a low-end soft switch capable of maintaining 500 simultaneous calls and of accepting new calls at a rate of 5 calls per second. Using only the instrumentation method of fault isolation we find that a particular mutation of the Invite message causes the SIP protocol to become unresponsive for 500 milliseconds. This might be evaluated to be a significant, but not critical vulnerability since at the call rates supported it would only effect 2-3 callers. However, if in addition to SIP being unresponsive the vulnerability also causes all existing calls to drop, this would make the problem far more severe. The event would be much more apparent to the customer and might lead to a significant increase in call volume that the device would be unable to accommodate.

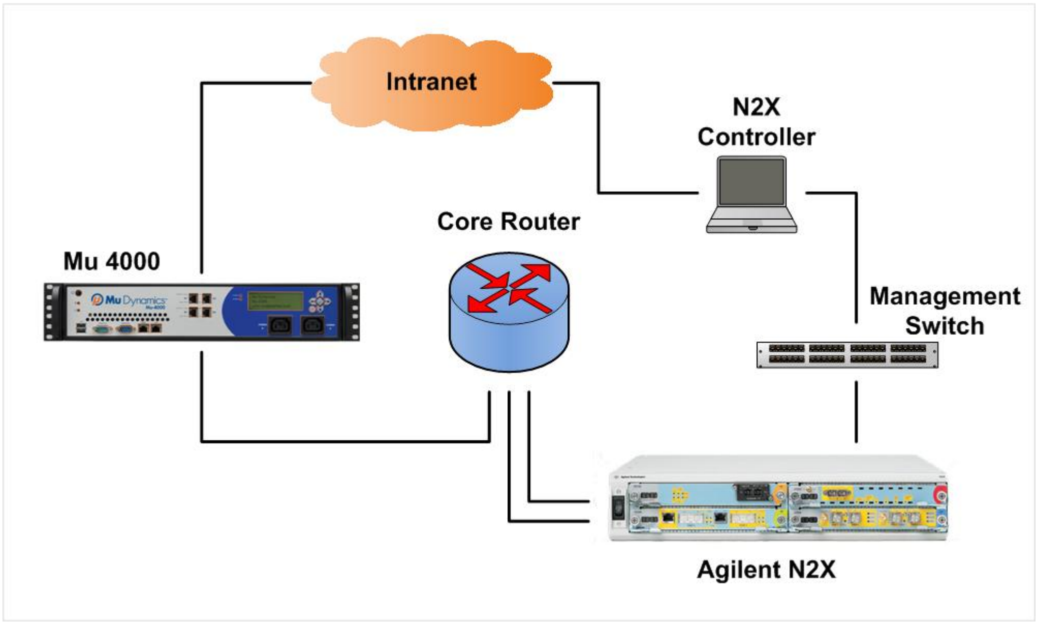
The previous two examples consider the functionality of the device being tested, but there may also be cases where the performance of the device is an important indicator of the existence or severity of a fault. We have found a number of issues related to faults causing a spike in processor or memory utilization. It is reasonable to assume that these events might impact the performance of the systems. Consider a high-end SIP soft switch capable of 500 calls per second. It would certainly be interesting to know if a particular mutation (or set of mutations) results in a 20% reduction in call processing rates.

Although the Mu appliance does not have any capability to directly test the functionality or performance of the target systems it has two features that allow any external test tool having a remotely accessible API to be incorporated in a Mu test. These features use an integrated console (either Telnet or SSH) to send commands to the target tool at strategic points during the fault isolation process. The values returned by these commands may be used as an alternate method for triggering fault isolation.

The integrated solution described in this document is intended to show how the Agilent N2X can be integrated with the Mu to produce tests that are greater then either one could accomplish in a standalone fashion. With that in mind the test scenario will be focused on a fairly basic and specific integration, and more complex test scenarios will be discussed. In the test environment below the Mu will control the Agilent N2X in order to monitor the delivery of data services during testing and provide a more holistic view into the DUT's reliability, availability, security, and performance characteristics.

Test Bed Configuration

In the network diagram shown in Figure 1 below shows a simplified lab network that contains a core router connected to the Mu-4000 which will send protocol mutations, and the Agilent N2X which will be a control plane neighbor with the M10 and will send data packets through the router. There is also an Agilent N2X controller, which the Mu-4000 will control via a Telnet channel.

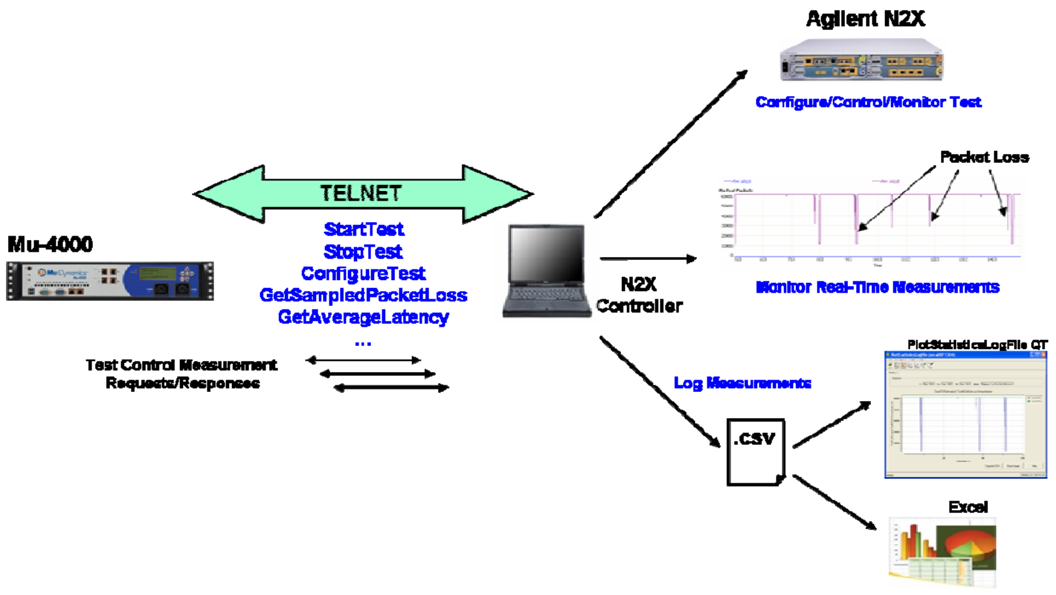


This scenario is merely one example; there are many others that could and should be tested via this test lab topology. All critical services running on the Core router should be tested in end-point mode, including BGP, OSPF, PIM, LDP/MPLS, management interfaces such as SSH, Telnet, and SNMP, as well as any other end-point security, routing, authentication, or management protocol. The M10 can also be analyzed in passthrough mode, to see how the higher-layer protocol mutations affect forwarding and inspection engines. The Agilent N2X also supports these protocols, and can measure the effect of a Mu analysis on the performance and reliability of these key services.

A key feature of the Mu analyzer with respect to integrated testing is the ability to communicate with other devices using an SSH or Telnet connection. In this case we will establish a Telnet connection to a N2X Controller, in order to automate the running of an N2X load test during a Mutation Analysis. Once connected the Mu starts the Tcl shell and sources a file that contains all of the scripts needed for the integrated test. It then runs a Tcl procedure to configure the desired test and instructs the N2X to bring the control plane up and begin transmitting data packets.

Once the N2X begins transmitting, the Mu will begin sending mutation traffic from the attack ports specified in the test configuration. At each point where the Mu “instruments” to validate that the DUT is still functioning it instructs the N2X controller to run a Tcl procedure which queries the status of the N2X load test. This script returns a message that indicates either that the load test is operating within acceptable parameters or that it has failed. It can also display the counters of the interfaces to be recorded for later use. If no anomaly is detected, the Mu continues with the next set of mutated sequences. If the script instead indicates that the DUT is no longer capable of handling the offered load it will trigger the Mu to enter “Fault Isolation”. This occurs whether or not the Mu’s own fault identification mechanisms are triggered.

Figure 2 illustrates how the Mu-4000 will interact with the N2X controller, how the Mu and the N2X will test the performance and robustness of the Core router, and how results will be correlated to form a more complete picture of how the router will behave in the real world when subjected to a variety of traffic conditions.



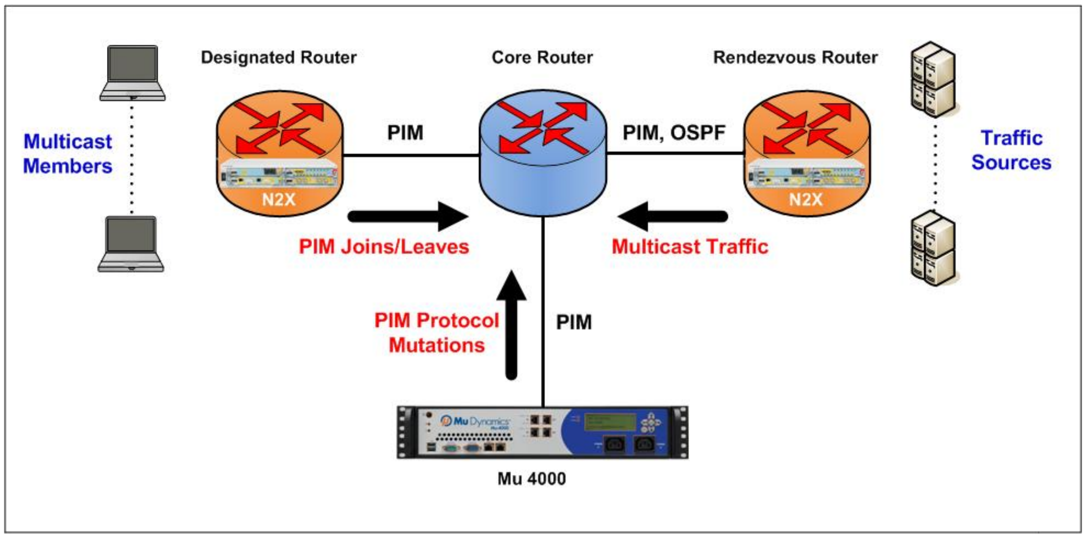
Test Integration Example – Core router PIM analysis and data throughput

As an example of Mu/Agilent N2X integration, we will first consider the testing of a core router, which will participate in the forwarding of data via Protocol Independent Multicast (Sparse Mode). The N2X will act as the PIM Rendezvous Point (RP) on one side of the M10, and the Designated Router on the other side, and will simulate traffic sources and multicast members connecting through the Core router.

The Mu-4000 will send malformed/mutated PIM messages to the M10, and will look for any interruptions in service. The Mu will also leverage integrated monitors to detect “hard faults” such as process re-starts or system crashes.

The Agilent N2X will simultaneously send data through the M10 (PIM Joins/Leaves and Multicast traffic), and will measure and log packet loss and latency statistics and measure the overall performance of the router. OSPF will also be running between the M10 and the N2X to establish a route table for connectivity.

Figure 3 represents the PIM analysis scenario:

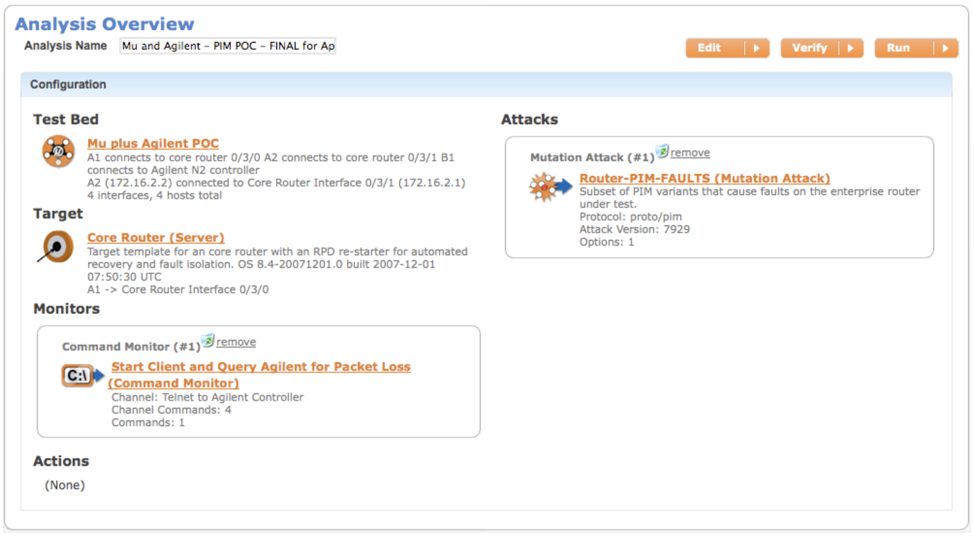


Agilent N2X Integration

The integrated Mu/Agilent N2X test is constructed from the four modules shown below. Three of these (Test Bed, Target and Attacks) define the Mu test. The remaining one (Monitors) control the Agilent N2X. An additional monitor will measure CPU utilization of the M10 router during the analysis.

Analysis Overview

The Analysis Overview shows how the various configuration elements fit together in order to make up a meaningful analysis. We will explore each configuration element in detail below.



Analysis Overview

Analysis Name: Mu and Agilent – PIM POC – FINAL for Ap

Configuration

Test Bed

Mu plus Agilent POC
A1 connects to core router 0/3/0 A2 connects to core router 0/3/1 B1 connects to Agilent N2 controller
A2 (172.16.2.2) connected to Core Router Interface 0/3/1 (172.16.2.1)
4 interfaces, 4 hosts total

Target

Core Router (Server)
Target template for an core router with an RPD re-starter for automated recovery and fault isolation. OS 8.4-20071201.0 built 2007-12-01 07:50:30 UTC
A1 -> Core Router Interface 0/3/0

Monitors

Command Monitor (#1)

Start Client and Query Agilent for Packet Loss (Command Monitor)
Channel: Telnet to Agilent Controller
Channel Commands: 4
Commands: 1

Actions

(None)

Attacks

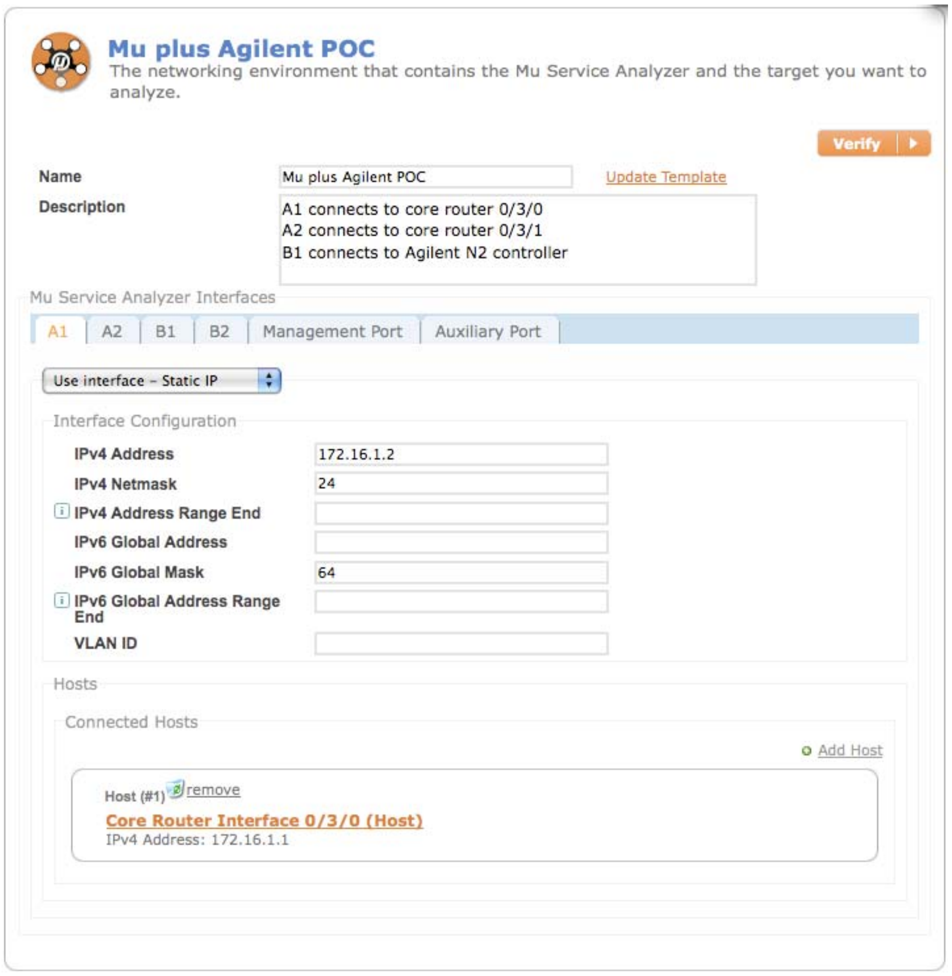
Mutation Attack (#1)

Router-PIM-FAULTS (Mutation Attack)
Subset of PIM variants that cause faults on the enterprise router under test.
Protocol: proto/pim
Attack Version: 7929
Options: 1

Test Bed

The Test Bed section defines the objects that the Mu can interact with and determines what network settings the attack ports should have at run time. Everything that the Mu needs to communicate with must be defined in the Test Bed. The first step is to define the address that the Mu will source to communicate to other network objects. In this test bed the interfaces A1, A2, B1, and Management ports are used. All other Mu interfaces are set to “do not use”.

A1 connects to the Core router under test via interface 0/3/0:



Mu plus Agilent POC

The networking environment that contains the Mu Service Analyzer and the target you want to analyze.

Name: Mu plus Agilent POC

Description:
A1 connects to core router 0/3/0
A2 connects to core router 0/3/1
B1 connects to Agilent N2 controller

Mu Service Analyzer Interfaces

Use interface - Static IP

Interface Configuration

IPv4 Address	172.16.1.2
IPv4 Netmask	24
IPv4 Address Range End	
IPv6 Global Address	
IPv6 Global Mask	64
IPv6 Global Address Range End	
VLAN ID	

Hosts

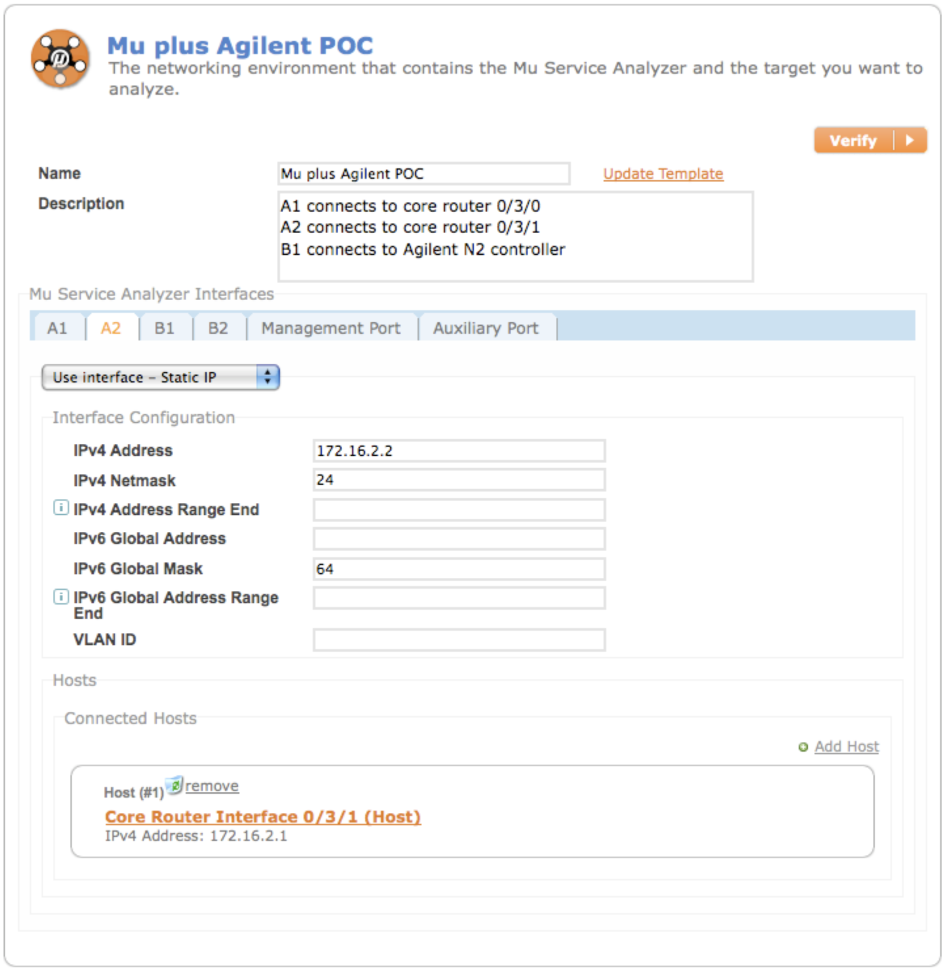
Connected Hosts

Host (#1)

Core Router Interface 0/3/0 (Host)

IPv4 Address: 172.16.1.1

A2 connects to the Core router under test via interface 0/3/1



Mu plus Agilent POC

The networking environment that contains the Mu Service Analyzer and the target you want to analyze.

Name	Mu plus Agilent POC
Description	A1 connects to core router 0/3/0 A2 connects to core router 0/3/1 B1 connects to Agilent N2 controller

Mu Service Analyzer Interfaces

Use interface - Static IP

Interface Configuration

IPv4 Address	172.16.2.2
IPv4 Netmask	24
IPv4 Address Range End	
IPv6 Global Address	
IPv6 Global Mask	64
IPv6 Global Address Range End	
VLAN ID	

Hosts

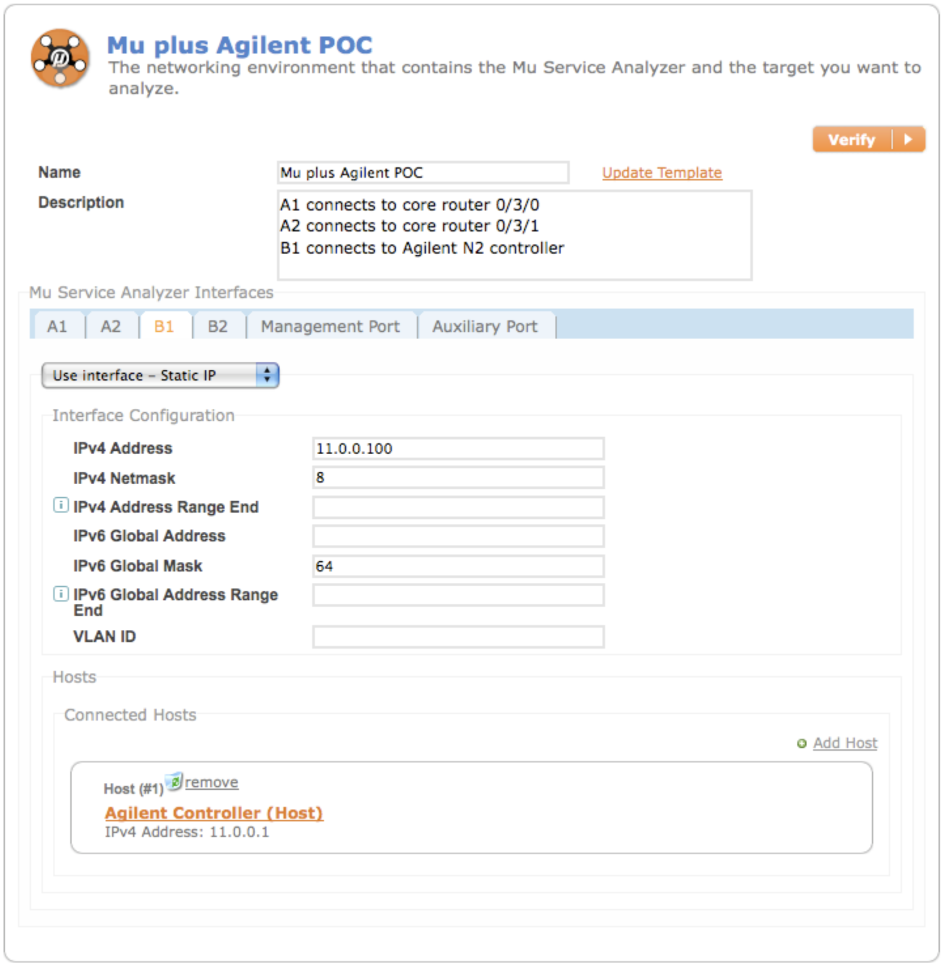
Connected Hosts

Host (#1) remove

Core Router Interface 0/3/1 (Host)

IPv4 Address: 172.16.2.1

B1 connects to the Agilent N2X controller:



Mu plus Agilent POC

The networking environment that contains the Mu Service Analyzer and the target you want to analyze.

Name	Mu plus Agilent POC
Description	A1 connects to core router 0/3/0 A2 connects to core router 0/3/1 B1 connects to Agilent N2 controller

Use interface - Static IP

Interface Configuration

IPv4 Address	11.0.0.100
IPv4 Netmask	8
IPv4 Address Range End	
IPv6 Global Address	
IPv6 Global Mask	64
IPv6 Global Address Range End	
VLAN ID	

Hosts

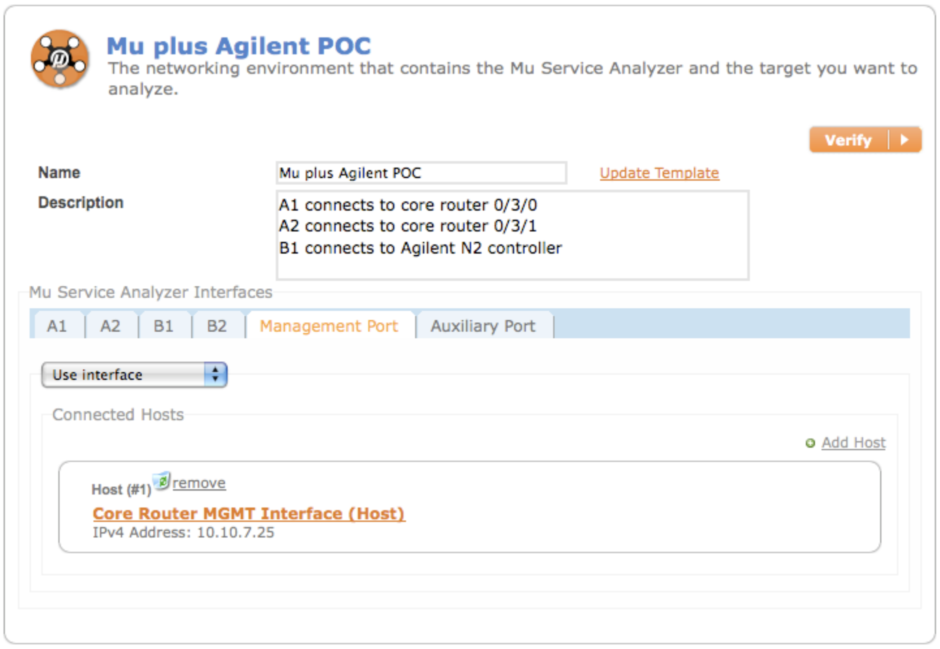
Connected Hosts

Host (#1) remove

Agilent Controller (Host)

IPv4 Address: 11.0.0.1

The Mu Management port is configured to talk with the Core router management port:



Mu plus Agilent POC

The networking environment that contains the Mu Service Analyzer and the target you want to analyze.

Name	Mu plus Agilent POC
Description	A1 connects to core router 0/3/0 A2 connects to core router 0/3/1 B1 connects to Agilent N2 controller

Mu Service Analyzer Interfaces

Use interface

Connected Hosts

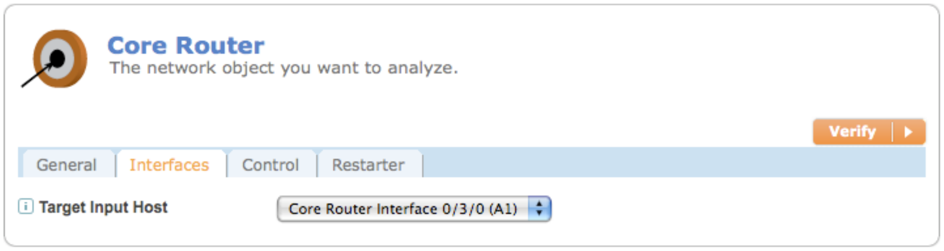
Host (#1) remove

Core Router MGMT Interface (Host)

IPv4 Address: 10.10.7.25

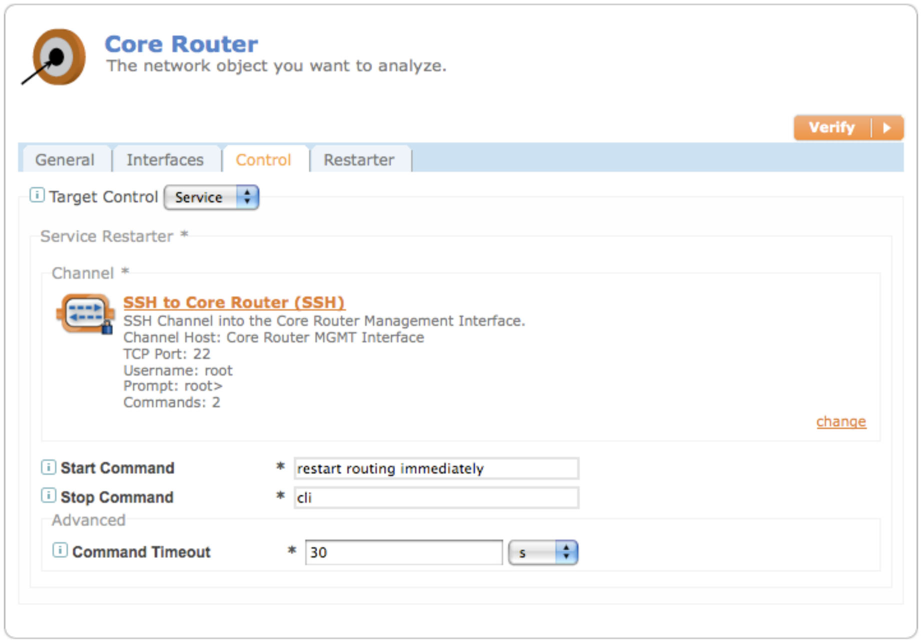
Target

The Target section allows the user to select the “Host” to be tested from a list of previously defined hosts. We simply have to select the target from the pull down menu located on the Interface tab as shown below:



Target Input Host: Core Router Interface 0/3/0 (A1)

The control for this target allows for the target to be restarted in the event that target becomes unresponsive. In this case the restart action is a CLI command that is entered via an SSH channel into the Core router under test.



Target Control: Service

Service Restarter *

Channel *

SSH to Core Router (SSH)
SSH Channel into the Core Router Management Interface.
Channel Host: Core Router MGMT Interface
TCP Port: 22
Username: root
Prompt: root>
Commands: 2

Start Command * restart routing immediately

Stop Command * cli

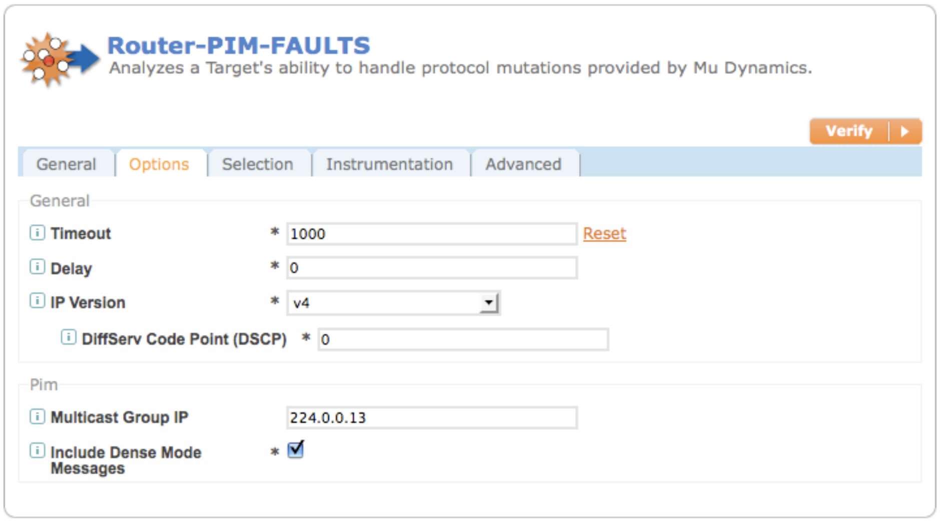
Advanced

Command Timeout * 30 s

Attack

The attack section defines the protocol that will be used to test the exception handling capabilities of the DUT, and any parameters required to allow valid protocol interaction with the DUT. The user may also choose to define one or more protocols that will be used to instrument the DUT. “Instrumentation” is the Mu’s internal method for determining if the DUT remains in a functional state following the reception of malformed packet sequences. A valid exchange is directed at the DUT and its responses are evaluated. If the responses indicate that the protocol is running the device is assumed to be operational and the next set of mutations is sent; otherwise the Mu will enter fault isolation. By default the protocol being tested is used for instrumentation, however, this behavior may be over ridden or additional protocols may used as well.

For simplicity we will accept the default instrumentation and forgo a detailed discussion of the options as the focus once again is the integration of the Mu with the Agilent N2X. Below is a capture of the attack configuration options:



Router-PIM-FAULTS

Analyzes a Target's ability to handle protocol mutations provided by Mu Dynamics.

General

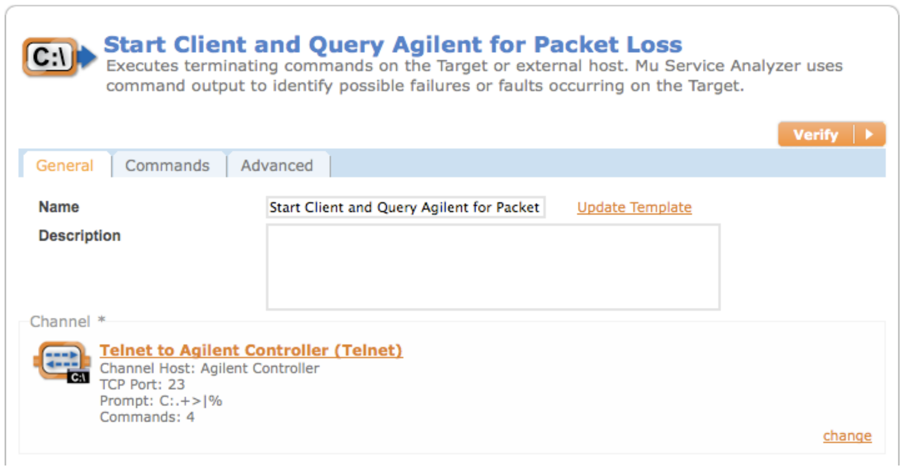
Timeout	*	1000
Delay	*	0
IP Version	*	v4
DiffServ Code Point (DSCP)	*	0

Pim

Multicast Group IP		224.0.0.13
Include Dense Mode Messages	*	✓

Monitors

The Monitors section is used to define interactions with devices outside the Mu that are being used to determine the health of the DUT during the course of the mutation test. Many times this is done through direct interaction with the DUT, but for the purpose of this App Note we will use it to run the Agilent N2X Tcl procedures to look for packet loss and latency issues on the DUT. The main setup window for the Monitors section is shown below:



Start Client and Query Agilent for Packet Loss

Executes terminating commands on the Target or external host. Mu Service Analyzer uses command output to identify possible failures or faults occurring on the Target.

Name: Start Client and Query Agilent for Packet

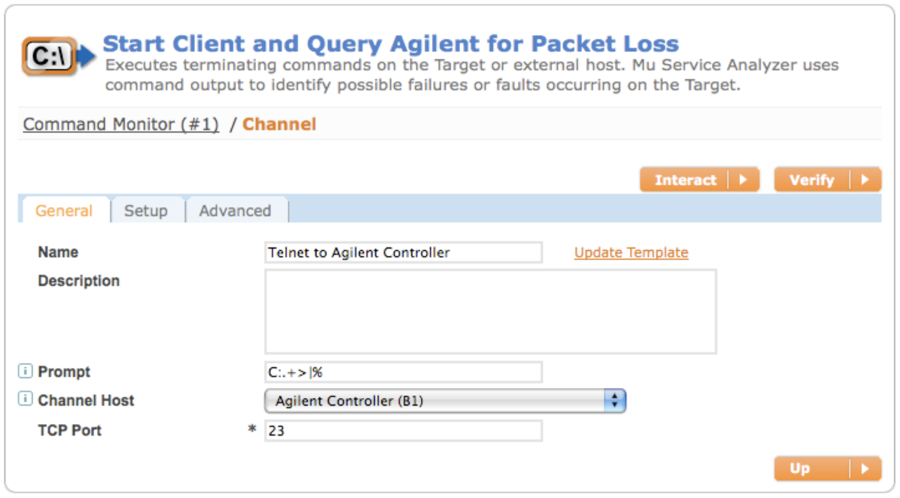
Description:

Channel *

Telnet to Agilent Controller (Telnet)
Channel Host: Agilent Controller
TCP Port: 23
Prompt: C:.+>|%
Commands: 4

Setting up the Mu to control the Agilent N2X is a three-step process. First we must establish a Telnet session with the N2X controller. This is done through the Channel specification.

We need to specify the Mu port we will use to establish the telnet channel, a regular expression which will return true when the channel returns a prompt, the TCP Port number used for telnet (which defaults to 23):



Start Client and Query Agilent for Packet Loss

Executes terminating commands on the Target or external host. Mu Service Analyzer uses command output to identify possible failures or faults occurring on the Target.

Command Monitor (#1) / Channel

Name: Telnet to Agilent Controller

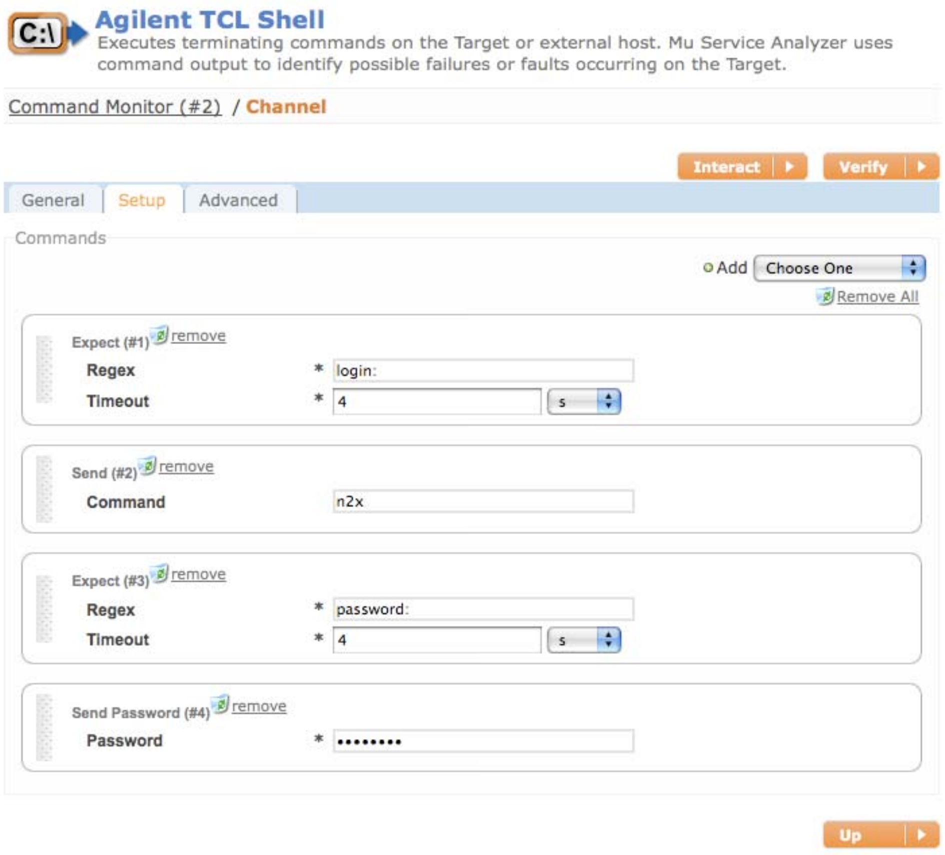
Description:

Prompt: C:.+>|%

Channel Host: Agilent Controller (B1)

TCP Port *: 23

In order to navigate the login process that occurs over telnet we must enter some setup actions. We will look for the word “login:” then send the username “n2x”, then wait for the response “password:” and then transmit the password for the user n2x. At this point we should expect a “>” prompt to be considered successful. The other prompt specified is “%” which becomes valid in the Tcl shell.



Agilent TCL Shell

Executes terminating commands on the Target or external host. Mu Service Analyzer uses command output to identify possible failures or faults occurring on the Target.

Command Monitor (#2) / **Channel**

Expect (#1) remove

Regex *	login:
Timeout *	4 s

Send (#2) remove

Command	n2x

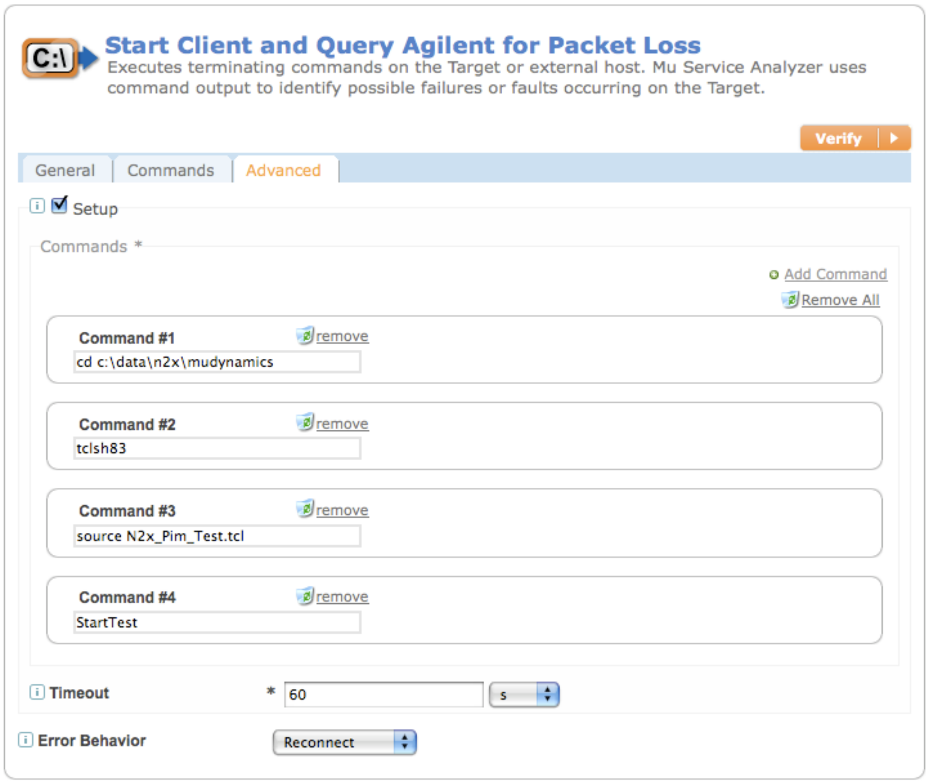
Expect (#3) remove

Regex *	password:
Timeout *	4 s

Send Password (#4) remove

Password *	•••••••••

After the channel configuration is completed, the second step is to start the Tcl shell, configure the test on the Agilent N2X and begin transmitting traffic. This is done through the Setup tab that allows us to create an Expect script to interact with the channel as shown below:


Start Client and Query Agilent for Packet Loss
Executes terminating commands on the Target or external host. Mu Service Analyzer uses command output to identify possible failures or faults occurring on the Target.

Setup ✓

Commands *

Command #1: cd c:\data\n2x\mudynamics

Command #2: tclsh83

Command #3: source N2x_Pim_Test.tcl

Command #4: StartTest

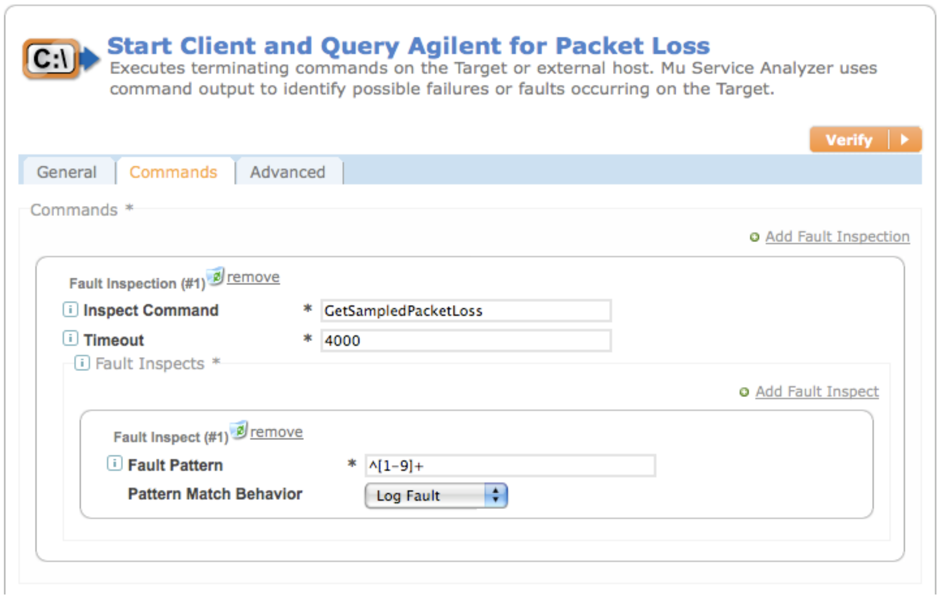
Timeout * 60 s

Error Behavior: Reconnect

Command #1 changes directories to the location of the Agilent Tcl files.
Command #2 starts the Tcl shell
Command #3 sources our Tcl procedures so that they can be used
Command #4 configures and launches the Agilent N2X load test

Please see Appendix B for details on the above procedures. The full Tcl script “N2x_Pim_Test.tcl” has been provided for your reference and as a starting point for a “production” implementation.

The final step in configuring the Command Monitor is to define the command which will be sent each time the Mu wishes to check the status of the DUT. This is done through the **Commands** tab of the Monitors window that is shown below:


Start Client and Query Agilent for Packet Loss
Executes terminating commands on the Target or external host. Mu Service Analyzer uses command output to identify possible failures or faults occurring on the Target.
Verify
General
Commands
Advanced
Add Fault Inspection
remove
Add Fault Inspect
remove

Commands *

Fault Inspection (#1)

Inspect Command * GetSampledPacketLoss

Timeout * 4000

Fault Inspects *

Fault Inspect (#1)

Fault Pattern * ^[1-9]+

Pattern Match Behavior Log Fault

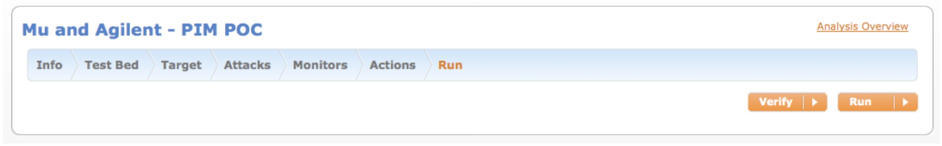
Inspect Command:

GetSampledPacketLoss

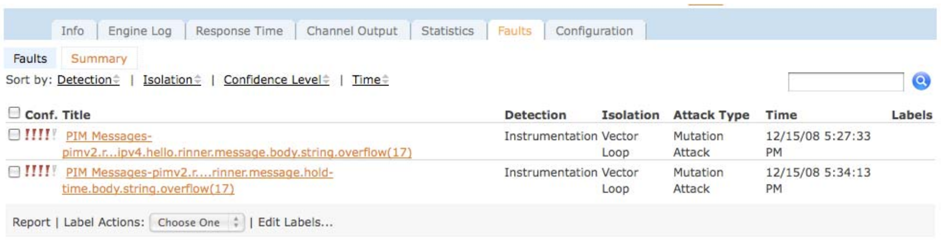
With the command GetSampledPacketLoss, a Tcl procedure is initiated that will measure the number of packets lost during the PIM load test, between successive requests. The allowed tolerance of packet loss is up to 9 packets, and so the Fault Pattern is set to trigger whenever packet loss is 10 packets or greater. This can of course be customized to any packet loss tolerance level, and can be extended to include additional metrics beyond just packet loss (i.e. latency)

Running the Test

After completing this configuration (or loading a saved configuration) the integrated Mu/Agilent N2X test is run exactly like a standard Mu test. Click on the **Run** tab and click the Run button as shown:



The screen shot below shows the **Faults** tab of a completed analysis:


Info
Engine Log
Response Time
Channel Output
Statistics
Faults
Configuration
Faults
Summary
Sort by: Detection | Isolation | Confidence Level | Time
Report | Label Actions: Choose One | Edit Labels...

Conf.	Title	Detection	Isolation	Attack Type	Time	Labels
!!!!!	PIM Messages-pimv2.r...ipv4.hello.rinner.message.body.string.overflow(17)	Instrumentation	Vector Loop	Mutation Attack	12/15/08 5:27:33 PM	
!!!!!	PIM Messages-pimv2.r....rinner.message.hold-time.body.string.overflow(17)	Instrumentation	Vector Loop	Mutation Attack	12/15/08 5:34:13 PM	

Note that the attached Monitor traces show clear packet loss occurring whenever a fault was identified:

17:27:25C.0 GetSampledPacketLoss
0
17:27:30C.0 GetSampledPacketLoss
0
17:27:35C.0 GetSampledPacketLoss
51962.0

17:34:05C.0 GetSampledPacketLoss
0
17:34:10C.0 GetSampledPacketLoss
0
17:34:15C.0 GetSampledPacketLoss
5948.0

This result demonstrates the power of the integration, as each test tool offers a unique perspective and metric that by itself only tells part of the story. When the combined date is woven together, however, we now have a clear indication of the real-world impact (significant packet loss) of the faults identified by the Mu Service Analyzer.

Appendix A – Mu Analysis Template

The following is the Mu Analysis Template used for this integrated test as an xml file.

It can be imported into any Mu using the Templates->Import option. After you save the template it will appear as an Analysis Template with the name “Residential Gateway”

```
<?xml version="1.0" encoding="UTF-8"?>
<mu_config version="3.0">
    <templates>
        <analysis name="Mu and Agilent - PIM POC - FINAL for AppNote" uuid="f8a7654c-
51ce-43b9-94de-eac35512d412">
            <description>This is a modified POC that sources the tcl script on the
Agilent controller, and then executes individual functions to query metrics from the
Agilent to look for data loss or other events, and will correlate those with the Mu test
cases.</description>
            <analysis_status>NEW</analysis_status>
            <analyzer_mode>Client</analyzer_mode>
            <attacks>
                <mutation_attack name="Router-PIM-FAULTS">
                    <description>Subset of PIM variants that cause faults on the
enterprise router under test.</description>
                    <analyzer_mode>Client</analyzer_mode>
                    <additional_instrumentation/>

<additional_instrumentation_isolation_enabled>true</additional_instrumentation_isolation_
enabled>
                    <address_looping_enabled>false</address_looping_enabled>
                    <excludes/>
                    <includes/>
                    <options>
                        <option>
                            <name>io.timeout</name>
                            <value>1000</value>
                        </option>
                    </options>
                    <protocol>proto/pim</protocol>
                    <variant_run_limit>
                        <variant_count>25</variant_count>
                    </variant_run_limit>
                    <start_from_variant>1</start_from_variant>
                    <suite_instrumentation_enabled>true</suite_instrumentation_enabled>
                    <throttle_timeout>0</throttle_timeout>
                    <variant_fault_limit>LIMIT_1</variant_fault_limit>
                    <version>7929</version>
                </mutation_attack>
            </attacks>
            <event_actions/>
            <monitors>
                <command_monitor name="Start Client and Query Agilent for Packet Loss">
                    <telnet_channel name="Telnet to Agilent Controller">
                        <capture>true</capture>
                        <commands>
                            <expect_command>
                                <regex>login:</regex>
                                <timeout>4000</timeout>
                            </expect_command>
                            <send_command>
                                <command>n2x</command>
                            </send_command>
                            <expect_command>
                                <regex>password:</regex>
                                <timeout>4000</timeout>
                            </expect_command>
                            <password_command>
                                <password>n2x12345</password>
                            </password_command>
                        </commands>
                        <prompt>C:.+&gt;|%</prompt>
                        <host ref="Agilent Controller"/>
                        <tcp_port>23</tcp_port>
                    </telnet_channel>
                    <commands>
                        <command>
                            <fault_inspects>
                                <fault_inspect>
                                    <fault_pattern>^[1-9]+</fault_pattern>

<pattern_match_behavior>Log_fault</pattern_match_behavior>
                                </fault_inspect>
                            </fault_inspects>
                            <send>GetSampledPacketLoss</send>
                            <timeout>4000</timeout>
                        </command>
                    </commands>
                    <error_behavior>reconnect</error_behavior>
                    <setup>
                        <commands>
                            <command>cd c:\data\n2x\mudynamics</command>
                            <command>tclsh83</command>
                            <command>source N2x_Pim_Test.tcl</command>
                            <command>StartTest</command>
                        </commands>
                        <timeout>60000</timeout>
                    </setup>
                </command_monitor>
            </monitors>
            <server_target name="Core Router">
                <description>Target template for an core router with an RPD re-starter
for automated recovery and fault isolation.

OS 8.4-20071201.0 built 2007-12-01 07:50:30 UTC</description>
                <analyzer_mode>Client</analyzer_mode>
                <max_boot_time>180000</max_boot_time>
                <min_boot_time>120000</min_boot_time>
                <restart_delay>2000</restart_delay>
                <restart_fail_behavior>Pause_analysis</restart_fail_behavior>
                <service_target_control>
                    <process_restarter>
                        <ssh_channel name="SSH to Core Router">
                            <description>SSH Channel into the Core Router Management
Interface.</description>
                            <capture>true</capture>
                            <commands>
                                <expect_command>
                                    <regex>root@%</regex>
                                    <timeout>4000</timeout>
                                </expect_command>
                                <send_command>
                                    <command>cli</command>
                                </send_command>
                            </commands>
                            <prompt>root&gt;</prompt>
                            <host ref="Core Router MGMT Interface"/>
                            <password>happy1</password>
                            <tcp_port>22</tcp_port>
                            <username>root</username>
                        </ssh_channel>
                        <command_timeout>30000</command_timeout>
                        <start>restart routing immediately</start>
                        <stop>cli</stop>
                    </process_restarter>
                </service_target_control>
                <target_in ref="Core Router Interface 0/3/0"/>
            </server_target>
            <testbed name="Mu plus Agilent POC">
                <description>A1 connects to core router 0/3/0
A2 connects to core router 0/3/1
```

```
B1 connects to Agilent N2 controller</description>
                <mu_ifs>
                    <attack_if>
                        <hosts>
                            <host>
                                <name>Core Router Interface 0/3/0</name>
                                <v4_addr>172.16.1.1</v4_addr>
                            </host>
                        </hosts>
                        <port>a1</port>
                        <v4_addr>172.16.1.2</v4_addr>
                        <v4_mask>24</v4_mask>
                        <v6_global_mask>64</v6_global_mask>
                    </attack_if>
                    <attack_if>
                        <hosts>
                            <host>
                                <name>Core Router Interface 0/3/1</name>
                                <v4_addr>172.16.2.1</v4_addr>
                            </host>
                        </hosts>
                        <port>a2</port>
                        <v4_addr>172.16.2.2</v4_addr>
                        <v4_mask>24</v4_mask>
                        <v6_global_mask>64</v6_global_mask>
                    </attack_if>
                    <mgmt_if>
                        <hosts>
                            <host>
                                <name>Core Router MGMT Interface</name>
                                <v4_addr>10.10.7.25</v4_addr>
                            </host>
                        </hosts>
                        <port>mgmt</port>
                    </mgmt_if>
                    <attack_if>
                        <hosts>
                            <host>
                                <name>Agilent Controller</name>
                                <v4_addr>11.0.0.1</v4_addr>
                            </host>
                        </hosts>
                        <port>b1</port>
                        <v4_addr>11.0.0.100</v4_addr>
                        <v4_mask>8</v4_mask>
                        <v6_global_mask>64</v6_global_mask>
                    </attack_if>
                </mu_ifs>
            </testbed>
        </analysis>
    </templates>
</mu_config>
```

Appendix B – Tcl script procedures for Agilent N2X

```
#---------------------------------------------------------------------------
# Title:             N2x_Pim_Test.tcl
#---------------------------------------------------------------------------
# Author:            Peter Atanasovski
# Created:           26 Nov 2008
# Last modified:     26 Feb 2009
#---------------------------------------------------------------------------
# Requires: Tcl/Tk 8.3 (or later)
#---------------------------------------------------------------------------
#  Copyright (C) 2008-2009 Agilent Technologies
#
#  All copies of this program, whether in whole or in part, and whether
#  modified or not, must display this and all other embedded copyright
#  and ownership notices in full.
#---------------------------------------------------------------------------
# Description :
#
#   Agilent N2X Tcl program to:
#   - configure PIM test scenario
#   - start/stop traffic
#   - retrieve/monitor N2X measurements
#
# Preconditions :
#   - install Agilent N2X Tcl library AgtRt900 (version 4.0.26 or newer)
#
# Usage :
#   $ tclsh83
#   % source N2x_Pim_Test.tcl
#   % StartTest
#   % GetSampledPacketLoss
#   % ...
#   % StopTest
#
#---------------------------------------------------------------------------

# path update for AgtRt900 installation
lappend auto_path "~/Agilent/N2X/tcl/extras"
lappend auto_path "C:/Program Files/Agilent/N2X/tcl/extras"

# source Tcl packages
package require AgtRt900

#---------------------------------------------------------------------------
# Parameters
#---------------------------------------------------------------------------

array set gScriptData \
    [list \
        firstMcastGroup             "225.0.0.1" \
        instPacketLossThreshold     5 \
        n2xServerName               localhost \
        n2xSessionHandle            0 \
        n2xSessionLabel             "N2x_Pim_Test" \
        n2xVersion                  latest \
        numMcastGroups              5 \
        numTxSourceIp               1 \
        pimSinkHandle               0 \
        pimSourceHandle             0 \
        pimSourceMemberList         {} \
        pimSinkMemberList           {} \
        portStatsHandle             0 \
        samplingInterval            1 \
        sinkPrefixLength            24 \
        sourcePrefixLength          24 \
        sutName                     "Core-Router" \
        sutRouterId                 "172.16.1.1" \
        sutSourceIp                 "172.16.4.1" \
        sutSourcePort               "fe/3/0" \
        sutSinkIp                   "172.16.3.1" \
        sutSinkPort                 "fe/2/0" \
        testDuration                60 \
        testLoad                    100 \
        testLoadUnits               PERCENTAGE_LINK_BANDWIDTH \
        testMode                    AGT_TEST_CONTINUOUS \
        tstSourceIp                 "172.16.4.2" \
        tstSourcePort               "101/1" \
        tstSourcePortHandle         0 \
        tstSourcePortPersonality    AGT_PERSONALITY_TRI_RATE_ETHERNET_X \
        tstSinkIp                   "172.16.3.2" \
        tstSinkPort                 "101/2" \
        tstSinkPortHandle           0 \
        tstSinkPortPersonality      AGT_PERSONALITY_TRI_RATE_ETHERNET_X \
        txSourceIp                  "30.1.2.1" \
        txSourceIpModifier          1 \
    ]

# N2X stats storage
# - each of the form [list portName1 interval1 value1 portName2 interval2 value2 ...]
array set gPortPreviousStatsData \
    [list \
        AGT_TEST_PACKETS_TRANSMITTED        {} \
        AGT_TEST_PACKETS_RECEIVED           {} \
        AGT_TEST_TRANSMIT_THROUGHPUT        {} \
        AGT_TEST_RECEIVE_THROUGHPUT         {} \
        AGT_PACKET_AVERAGE_LATENCY          {} \
        AGT_PACKET_MAXIMUM_LATENCY          {} \
        AGT_MISDIRECTED_PACKETS_RECEIVED    {} \
    ]

#---------------------------------------------------------------------------
# Procedures
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# AddPorts { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Add test ports to an N2X test session.
#---------------------------------------------------------------------------
proc AddPorts { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    AgtTsuTraceMessage "$procName: Adding test ports..."
    set hTestPortList [::AgtRt900::AddPorts \
                        -testportnamelist [list $gScriptData(tstSourcePort)
$gScriptData(tstSinkPort)] \
                        -sutportnamelist [list $gScriptData(sutSourcePort)
$gScriptData(sutSinkPort)] \
                        -testeripv4list [list $gScriptData(tstSourceIp)
$gScriptData(tstSinkIp)] \
                        -sutipv4list [list $gScriptData(sutSourceIp)
$gScriptData(sutSinkIp)] \
                        -prefixlengthlist [list $gScriptData(sourcePrefixLength)
$gScriptData(sinkPrefixLength)] \
                        -personalitylist [list $gScriptData(tstSourcePortPersonality)
$gScriptData(tstSinkPortPersonality)] \
                        -sutnamelist [list $gScriptData(sutName) $gScriptData(sutName)]]

    # get port handles if specified ports already added
    if { ![llength $hTestPortList] } {
        set hTestPortList [AgtTsuConvertPortNameToHandle [list
$gScriptData(tstSourcePort) $gScriptData(tstSinkPort)]]
    }

    set gScriptData(tstSourcePortHandle)    [lindex $hTestPortList 0]
    set gScriptData(tstSinkPortHandle)      [lindex $hTestPortList 1]
}

#-----------------------------------------------------------------------
# ClearMeasurementData { }
#-----------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Clear measurement data storage.
#-----------------------------------------------------------------------
proc ClearMeasurementData { } {
    variable gScriptData
    variable gPortPreviousStatsData

    set procName [AgtTsuProcedureName]

    set testPortList [list $gScriptData(tstSourcePort) $gScriptData(tstSinkPort)]
    foreach statsName [array names gPortPreviousStatsData *] {
        set valueList {}
        foreach portName $testPortList {
            lappend valueList $portName 0 0
        }
        set gPortPreviousStatsData($statsName) $valueList
    }
}

#-----------------------------------------------------------------------
# CloseSession { }
#-----------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Close the N2X test session.
#-----------------------------------------------------------------------
proc CloseSession { } {
    variable gScriptData

    AgtTsuDisconnect
    AgtTsuCloseSession $gScriptData(n2xSessionHandle)
}

#-----------------------------------------------------------------------
# ConfigureLinkLayer { }
#-----------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure the link layer on the test ports.
#-----------------------------------------------------------------------
proc ConfigureLinkLayer { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]
    set hTestPortList       [list $gScriptData(tstSourcePortHandle)
$gScriptData(tstSinkPortHandle)]
    set tstIpList           [list $gScriptData(tstSourceIp) $gScriptData(tstSinkIp)]
    set sutIpList           [list $gScriptData(sutSourceIp) $gScriptData(sutSinkIp)]
    set prefixLengthList    [list $gScriptData(sourcePrefixLength)
$gScriptData(sinkPrefixLength)]
    foreach hPort $hTestPortList tstIp $tstIpList sutIp $sutIpList prefixLength
$prefixLengthList {

        # Ethernet interface
        if { [AgtTsuIsEthernetPort $hPort] } {
            ::AgtRt900::Ethernet::InitialiseInterface \
                -porthandle $hPort \
                -enablearp true \
                -mediatype RJ45

            # reconfigure Tester and SUT IPv4 addresses (i.e. when connecting to existing
session)
            AgtTsuSetSutAndTesterIpAddress $hPort $sutIp $tstIp $prefixLength

            # send ARP requests
            ::AgtRt900::Ethernet::ResolveAddressesUsingArp -porthandle $hPort
        }
    }
    AgtTsuTraceMessage "$procName: Test port link layer configured."
}

#---------------------------------------------------------------------------
# ConfigureMeasurements { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure measurements.
#---------------------------------------------------------------------------
proc ConfigureMeasurements { } {
    variable gScriptData
    variable gPortPreviousStatsData

    set procName [AgtTsuProcedureName]

    # store information for monitoring port stats
    set hTestPortList [list $gScriptData(tstSourcePortHandle)
$gScriptData(tstSinkPortHandle)]
    ::AgtRt900::Stats::AddPortStatisticsRecord -porthandlelist $hTestPortList

    # select stats for ports
    set statsList [array names gPortPreviousStatsData *]
    ::AgtRt900::Stats::SetDefaultPortStatistics -statisticslist $statsList

    # configure stats collection
    ::AgtRt900::Stats::ProcessStatisticsRecordsAndSelect -typelist [list PORT]

    # store stats handle
    set gScriptData(portStatsHandle) [lindex $::AgtRt900::libData(portStatsHandleList) 0]

    # configure N2X stats log file
    ::AgtRt900::ApiTsuInvoke AgtStatisticsLog SetLogFile
"$gScriptData(n2xSessionLabel).csv"
    ::AgtRt900::ApiTsuInvoke AgtStatisticsLog SelectPorts $hTestPortList
    ::AgtRt900::ApiTsuInvoke AgtStatisticsLog SelectStatistics $statsList
    ::AgtRt900::ApiTsuInvoke AgtStatisticsLog EnableLogging

    AgtTsuTraceMessage "$procName: Measurement system configured."
}
```

```
#---------------------------------------------------------------------------
# ConfigureOspf { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure OSPF.
#---------------------------------------------------------------------------
proc ConfigureOspf { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # OSPF session
    set hOspf [::AgtRt900::Ospf::AddSession \
                -interfacehandle $gScriptData(tstSourcePortHandle) \
                -testeripv4 $gScriptData(tstSourceIp) \
                -sutipv4 $gScriptData(sutSourceIp) \
                -prefixlength $gScriptData(sourcePrefixLength) \
                -sutrouterid $gScriptData(sutRouterId)]

    # OSPF router
    ::AgtRt900::Ospf::AddRouters \
        -portnamelist $gScriptData(tstSourcePort) \
        -numrouters 1 \
        -startrouterid $gScriptData(txSourceIp)

    AgtTsuTraceMessage "$procName: OSPF emulation configured - session $hOspf on port
$gScriptData(tstSourcePort)."
}

#---------------------------------------------------------------------------
# ConfigurePim { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure PIM.
#---------------------------------------------------------------------------
proc ConfigurePim { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # add multicast groups
    set hMcastGroupList {}
    for {set i 0} {$i < $gScriptData(numMcastGroups)} {incr i} {
        lappend hMcastGroupList [::AgtRt900::Pim::AddMulticastGroupPool \
                                    -numaddresses 1 \
                                    -firstip [AgtTsuGetNextIpRoute
$gScriptData(firstMcastGroup) 32 $i] \
                                    -name "MulticastGroup_$i"]
    }

    # add source address pools
    set hSourcePool [::AgtRt900::Pim::AddSourceAddressPool \
                        -numaddresses $gScriptData(numTxSourceIp) \
                        -firstip $gScriptData(txSourceIp) \
                        -prefixlength 32 \
                        -name "SourcePool_1" \
                        -modifier $gScriptData(txSourceIpModifier)]

    set hSourcePoolList {}
    for {set i 0} {$i < $gScriptData(numMcastGroups)} {incr i} {
        lappend hSourcePoolList $hSourcePool
    }

    # configure PIM on source port
    set gScriptData(pimSourceHandle) [::AgtRt900::Pim::AddSession \
                    -interfacehandle $gScriptData(tstSourcePortHandle) \
                    -testerip
"$gScriptData(tstSourceIp)/$gScriptData(sourcePrefixLength)" \
                    -enable true]

    set sessionInfo [::AgtRt900::Pim::ConfigureSession \
                        -sessionhandle $gScriptData(pimSourceHandle) \
                        -addpoollist $hMcastGroupList \
                        -addsourcepoollist $hSourcePoolList \
                        -membershipmode RECEIVE \
                        -rpentitytype C_RP \
                        -rpentityaddress $gScriptData(tstSourceIp) \
                        -enable true]
    set hMemberPoolList [lindex $sessionInfo 1]

    # configure PIM on sink port
    set gScriptData(pimSinkHandle) [::AgtRt900::Pim::AddSession \
                    -interfacehandle $gScriptData(tstSinkPortHandle) \
                    -testerip "$gScriptData(tstSinkIp)/$gScriptData(sinkPrefixLength)" \
                    -remoterpaddress $gScriptData(tstSourceIp) \
                    -enable true]

    ::AgtRt900::Pim::ConfigureSession \
        -sessionhandle $gScriptData(pimSinkHandle) \
        -addpoollist $hMcastGroupList \
        -addsourcepoollist $hSourcePoolList \
        -remoterpaddress $gScriptData(tstSourceIp) \
        -enable true

    AgtTsuTraceMessage "$procName: PIM emulation configured - source session
$gScriptData(pimSourceHandle) on port $gScriptData(tstSourcePort); sink session
$gScriptData(pimSinkHandle) on port $gScriptData(tstSinkPort)"
}

#---------------------------------------------------------------------------
# ConfigureTest { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure the test scenario.
#---------------------------------------------------------------------------
proc ConfigureTest { } {
    variable gScriptData

    ConnectToSession
    AddPorts
    ConfigureLinkLayer
    RemoveExistingConfiguration
    ConfigurePim
    ConfigureOspf
    StartRouting
    ConfigureTraffic
    ConfigureMeasurements

    set hTestPortList [list $gScriptData(tstSourcePortHandle)
$gScriptData(tstSinkPortHandle)]
    JoinAllGroups $hTestPortList
}

#---------------------------------------------------------------------------
```

```
# ConfigureTraffic { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Configure traffic.
#---------------------------------------------------------------------------
proc ConfigureTraffic { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # add traffic profile
    set hProfile [lindex [::AgtRt900::Traffic::AddProfile \
                            -sourceporthandle $gScriptData(tstSourcePortHandle) \
                            -load $gScriptData(testLoad) \
                            -loadunits $gScriptData(testLoadUnits)] 1]
    set hStreamGroupList {}
    for {set i 0} {$i < $gScriptData(numMcastGroups)} {incr i} {

        # add stream group for each multicast group
        set sgInfo [::AgtRt900::Traffic::AddStreamGroup \
                        -profilehandle $hProfile \
                        -packetlengthrecord [list FIXED AGT_PACKET_LENGTH_MODE_IP_PACKETS
64 64 1] \
                        -expdestportlist $gScriptData(tstSinkPort)]
        set hSg     [lindex $sgInfo 0]
        set hPdu    [lindex $sgInfo 1]
        lappend hStreamGroupList $hSg

        # configure PDU
        set srcIpData [list ipv4 source_address [list FIXED $gScriptData(txSourceIp)]]
        set dstIpData [list ipv4 destination_address [list FIXED [AgtTsuGetNextIpRoute
$gScriptData(firstMcastGroup) 32 $i]]]
        set fieldDataList [list $srcIpData $dstIpData]
        ::AgtRt900::Traffic::ModifyPdu \
            -sourceporthandle $gScriptData(tstSourcePortHandle) \
            -streamgrouphandle $hSg \
            -pduhandle $hPdu \
            -fielddatalist $fieldDataList
    }
    AgtTsuTraceMessage "$procName: Traffic configured."
}

#---------------------------------------------------------------------------
# ConnectToSession { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Create/connect to an N2X test session.
#---------------------------------------------------------------------------
proc ConnectToSession { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    set n2xInfo [AgtTsuConnectToSessionByName \
                    -createIfNeeded \
                    -sessionVersion $gScriptData(n2xVersion) \
                    -sessionLabel $gScriptData(n2xSessionLabel) \
                    -serverName $gScriptData(n2xServerName)]
    set gScriptData(n2xSessionHandle) [lindex $n2xInfo 0]

    AgtTsuTraceMessage "$procName: Connected to N2X test session
'$gScriptData(n2xSessionLabel)' (handle = $gScriptData(n2xSessionHandle)) on server
$gScriptData(n2xSessionLabel)"
}

#---------------------------------------------------------------------------
# GetAverageLatency {  }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   rxAvgLatency    - average latency
#
# Purpose:
#   Get the cumulative average latency.
#---------------------------------------------------------------------------
proc GetAverageLatency { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # get statistics data
    set statsDataRcd    [AgtTsuInvoke AgtStatistics GetStatistics
$gScriptData(portStatsHandle)]
    set interval        [lindex $statsDataRcd 0]
    set statsRequestRcd [list [list $gScriptData(tstSinkPort)
AGT_PACKET_AVERAGE_LATENCY]]

    # get measurement value
    set requestedDataRcd    [GetPortStatisticsValue $statsDataRcd $statsRequestRcd]
    set rxAvgLatency        [lindex [lindex $requestedDataRcd 0] 1]

    #return [list $interval $rxAvgLatency]
    return $rxAvgLatency
}

#---------------------------------------------------------------------------
# GetPacketLoss { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   packetLoss      - packet loss
#
# Purpose:
#   Get the cumulative packet loss.
#---------------------------------------------------------------------------
proc GetPacketLoss { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # get statistics data
    set statsDataRcd    [AgtTsuInvoke AgtStatistics GetStatistics
$gScriptData(portStatsHandle)]
    set interval        [lindex $statsDataRcd 0]
    set statsRequestRcd [list [list $gScriptData(tstSourcePort)
AGT_TEST_PACKETS_TRANSMITTED] \
                              [list $gScriptData(tstSinkPort) AGT_TEST_PACKETS_RECEIVED]]

    # get measurement value
    set requestedDataRcd    [GetPortStatisticsValue $statsDataRcd $statsRequestRcd]
    set txStatValue         [lindex [lindex $requestedDataRcd 0] 1]
    set rxStatValue         [lindex [lindex $requestedDataRcd 1] 1]
    set packetLoss          [expr $txStatValue - $rxStatValue]

    # if packet loss is within certain boundary (i.e. to account for packets in transit),
then report 0
    if { $packetLoss < $gScriptData(instPacketLossThreshold) } {
        set packetLoss 0
    }

    #return [list $interval $packetLoss]
    return $packetLoss
}

#---------------------------------------------------------------------------
# GetMaximumLatency {  }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   rxMaxLatency    - maximum latency
#
# Purpose:
#   Get the cumulative maximum latency.
#---------------------------------------------------------------------------
proc GetMaximumLatency { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # get statistics data
    set statsDataRcd    [AgtTsuInvoke AgtStatistics GetStatistics
$gScriptData(portStatsHandle)]
    set interval        [lindex $statsDataRcd 0]
    set statsRequestRcd [list [list $gScriptData(tstSinkPort)
AGT_PACKET_MAXIMUM_LATENCY]]

    # get measurement value
    set requestedDataRcd    [GetPortStatisticsValue $statsDataRcd $statsRequestRcd]
    set rxMaxLatency        [lindex [lindex $requestedDataRcd 0] 1]

    #return [list $interval $rxMaxLatency]
    return $rxMaxLatency
}

#---------------------------------------------------------------------------
# GetPortStatisticsValue { statsDataRcd statsRequestRcd }
#---------------------------------------------------------------------------
# Parameters:
#   statsDataRcd        - stats data record as returned by 'AgtStatistics GetStatistics'
#   statsRequestRcd     - request record containing statistics of interest of the form:
#                         [list {port_name1 stat_name1} {port_name1 stat_name2} ...]
#
# Returns:
#   requestedDataRcd    - requested data of the form:
#                         [list {port_name1 stat_value1} {port_name1 stat_value2} ...]
#
# Purpose:
#   Retrieve the requested port statistics values.
#---------------------------------------------------------------------------
proc GetPortStatisticsValue { statsDataRcd statsRequestRcd } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # Get statistics
    # - returns an ordered list of values { port1_stat1 port1_stat2 ... port1_statN
port2_stat1 port2_stat2 ...}
    # - Note, these are accumulated values
    set interval    [lindex $statsDataRcd 0]
    set resultsInfo [lindex $statsDataRcd 1]

    set requestedDataRcd {}
    foreach statsConfig $statsRequestRcd  {
        set portName    [lindex $statsConfig 0]
        set statsName   [lindex $statsConfig 1]
        set hPort       [AgtTsuConvertPortNameToHandle $portName]

        # get port index from selection
        set hSelectedPortList [AgtTsuInvoke AgtStatistics ListSelectedPorts
$gScriptData(portStatsHandle)]
        set portIndex [lsearch -exact $hSelectedPortList $hPort]
        if { $portIndex == -1 } {
            AgtTsuShowMessage FATAL "$procName: Port $portName has not been configured
for statistics collection, aborting..."
        }

        # get statistics index from selection
        set selectedStatList    [AgtTsuInvoke AgtStatistics ListSelectedStatistics
$gScriptData(portStatsHandle)]
        set statIndex           [lsearch -exact $selectedStatList $statsName]
        set numStats            [llength $selectedStatList]
        if { $statIndex == -1 } {
            AgtTsuShowMessage FATAL "$procName: Measurement $statsName has not been
selected for statistics collection, aborting..."
        }

        # get stats for specified port
        set valueIndex  [expr ($portIndex * $numStats) + $statIndex]
        set statsValue  [lindex $resultsInfo $valueIndex]

        # store requested stats data
        lappend requestedDataRcd [list $portName $statsValue]
    }

    return $requestedDataRcd
}

#---------------------------------------------------------------------------
# GetSampledPacketLoss { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   packetLoss      - packet loss
#
# Purpose:
#   Get the sampled packet loss - number of packets lost between this
#   request and the previous request.
#---------------------------------------------------------------------------
proc GetSampledPacketLoss { } {
    variable gScriptData
    variable gPortPreviousStatsData

    set procName [AgtTsuProcedureName]

    # get previous values
    set prevTxValueList $gPortPreviousStatsData(AGT_TEST_PACKETS_TRANSMITTED)
    set portIndex       [lsearch -exact $prevTxValueList $gScriptData(tstSourcePort)]
    set prevTxStatValue [lindex $prevTxValueList [expr $portIndex + 2]]

    set prevRxValueList $gPortPreviousStatsData(AGT_TEST_PACKETS_RECEIVED)
    set portIndex       [lsearch -exact $prevRxValueList $gScriptData(tstSinkPort)]
    set prevRxStatValue [lindex $prevRxValueList [expr $portIndex + 2]]

    set prevInterval    [lindex $prevRxValueList [expr $portIndex + 1]]

    # get current values
    set statsDataRcd    [AgtTsuInvoke AgtStatistics GetStatistics
$gScriptData(portStatsHandle)]
    set currInterval    [lindex $statsDataRcd 0]
    set statsRequestRcd [list [list $gScriptData(tstSourcePort)
AGT_TEST_PACKETS_TRANSMITTED] \
                              [list $gScriptData(tstSinkPort) AGT_TEST_PACKETS_RECEIVED]]
    # check if we have a new sample set
    if { $prevInterval == $currInterval } {
        ::AgtRt900::ShowMessage -msgtype WARNING -popup 0 \
            -msg "$procName: Current measurement sample $currInterval is not new,
ignoring..."
        return 0
    }

    set requestedDataRcd    [GetPortStatisticsValue $statsDataRcd $statsRequestRcd]
    set currTxStatValue     [lindex [lindex $requestedDataRcd 0] 1]
    set currRxStatValue     [lindex [lindex $requestedDataRcd 1] 1]

    # calculate values between last two requests
    set txStatValue     [expr $currTxStatValue - $prevTxStatValue]
    set rxStatValue     [expr $currRxStatValue - $prevRxStatValue]

    # get packet loss for sample
    set packetLoss      [expr $txStatValue - $rxStatValue]

    # if packet loss is within certain boundary (i.e. packets in transit), then report 0
    if { $packetLoss < $gScriptData(instPacketLossThreshold) } {
        set packetLoss 0
    }

    # store new values
    UpdateMeasurementData AGT_TEST_PACKETS_TRANSMITTED $gScriptData(tstSourcePort)
$currInterval $currTxStatValue
    UpdateMeasurementData AGT_TEST_PACKETS_RECEIVED $gScriptData(tstSinkPort)
$currInterval $currRxStatValue

    # get sample period
    set samplePeriod [expr $gScriptData(samplingInterval) * [expr $currInterval -
$prevInterval]]

    #return [list $samplePeriod $packetLoss]
    return $packetLoss
}

#---------------------------------------------------------------------------
# JoinAllGroups { hPortList }
#---------------------------------------------------------------------------
# Parameters:
#   hPortList   - list of test port handles
#
# Returns:
#   nothing
#
# Purpose:
#   Join all multicast groups.
#---------------------------------------------------------------------------
proc JoinAllGroups { hPortList } {

    ::AgtRt900::Pim::JoinAllMulticastGroups -porthandlelist $hPortList
}

#---------------------------------------------------------------------------
# LeaveAllGroups { hPortList }
#---------------------------------------------------------------------------
# Parameters:
#   hPortList   - list of test port handles
#
# Returns:
#   nothing
#
# Purpose:
#   Leave all multicast groups.
#---------------------------------------------------------------------------
proc LeaveAllGroups { hPortList } {

    ::AgtRt900::Pim::LeaveAllMulticastGroups -porthandlelist $hPortList
```

```
}

#--------------------------------------------------------------------------
# RemoveExistingConfiguration { }
#--------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Remove existing routing/traffic configuration.
#--------------------------------------------------------------------------
proc RemoveExistingConfiguration { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    set hTestPortList [list $gScriptData(tstSourcePortHandle)
$gScriptData(tstSinkPortHandle)]

    AgtTsuRemoveAllOspfSessions $hTestPortList
    AgtTsuRemoveAllPimSessions $hTestPortList
    AgtTsuRemoveAllMulticastGroupPools
    ::AgtRt900::Pim::RemoveAllSourceAddressPools
    AgtTsuRemoveAllTraffic

    AgtTsuTraceMessage "$procName: Existing configuration removed."
}

#--------------------------------------------------------------------------
# StartRouting { }
#--------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Start routing and wait for control plane to come up.
#--------------------------------------------------------------------------
proc StartRouting { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    set testPortList [list $gScriptData(tstSourcePort) $gScriptData(tstSinkPort)]
    set rcdList [::AgtRt900::StartRoutingServices \
                    -portnamelist $testPortList \
                    -protocollist [list OSPF PIM] \
                    -maxwaittime 5]
    ::AgtRt900::Private::ProcessStartRoutingServicesResult -recordlist $rcdList
}

#--------------------------------------------------------------------------
# StartTest { }
#--------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Start a test.
#--------------------------------------------------------------------------
proc StartTest { } {
    variable gScriptData
    set procName [AgtTsuProcedureName]

    # ignore if test is already running
    if { [::AgtRt900::ApiTsuInvoke AgtTestController GetTestState] != "AGT_TEST_STOPPED"
} {
        return
    }

    # clear stats data storage
    ClearMeasurementData

    # start test
    ::AgtRt900::StartTest \
        -testmode $gScriptData(testMode) \
        -samplinginterval $gScriptData(samplingInterval) \
        -testduration $gScriptData(testDuration)}

#---------------------------------------------------------------------------
# StopTest { }
#---------------------------------------------------------------------------
# Parameters:
#   none
#
# Returns:
#   nothing
#
# Purpose:
#   Stop a test.
#---------------------------------------------------------------------------
proc StopTest { } {
    variable gScriptData

    set procName [AgtTsuProcedureName]

    # stop test
    ::AgtRt900::StopTest
}

#---------------------------------------------------------------------------
# UpdateMeasurementData { statsName portName interval statsValue }
#---------------------------------------------------------------------------
# Parameters:
#   statsName   - statistics name as returned by EAgtStatistics
#   portName    - test port name
#   interval    - sample interval
#   statsValue  - statistics value
#
# Returns:
#   nothing
#
# Purpose:
#   Update measurement data storage.
#---------------------------------------------------------------------------
proc UpdateMeasurementData { statsName portName interval statsValue } {
    variable gScriptData
    variable gPortPreviousStatsData

    set procName [AgtTsuProcedureName]

    set testPortList [list $gScriptData(tstSourcePort) $gScriptData(tstSinkPort)]
    set valueList {}
    foreach portName $testPortList {
        lappend valueList $portName $interval $statsValue
    }
    set gPortPreviousStatsData($statsName) $valueList
}

#---------------------------------------------------------------------------
# Initialisation
#---------------------------------------------------------------------------
```

```
# set script debug level
# 0 - Off
# 1 - Low
# 2 - Medium
# 3 - High
AgtTsuDebugLevel 0

# enable API and statistics logging
::AgtRt900::EnableApiLogging -directory [file dirname [info script]]
::AgtRt900::Stats::EnableStatisticsLogging -directory [file dirname [info script]]

#---------------------------------------------------------------------------
# MAIN
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Configure test scenario
ConfigureTest

AgtTsuTraceMessage "\nNumber of System API errors - [AgtRt900::GetApiErrorCount]"
AgtTsuTraceMessage "Number of Test Run errors - [AgtRt900::GetErrorCount]"
AgtTsuTraceMessage "\nConfiguration Done!"
```

Appendix C – Mu-4000 Features and Benefits

This section will summarize the features and benefits of the Mu Service Analyzer that will be leveraged to provide maximum integration value to the customer.

Some of the primary benefits of the Mu Service Analyzer include:

- **Millions of variations on service traffic –** a unique, stateful protocol modeling engine interactively explores the target attack surface with millions of dynamically generated non-conformant variations of protocol traffic tailored to the targets' exact capabilities.

- **Automation** – automates the negative/robustness testing process, including device monitoring and recovery, fault detection and isolation, and post-fault data collection and reporting. Can be fully integrated into any test automation harness via a scriptable Automation API.

- **Fault Isolation** – automatically pinpoints the test case or traffic scenario responsible for a fault condition, and correlates and captures the data needed to take remediation action.

- **Service Monitoring** – profiles how the various negative test cases impact the availability and responsiveness of the services under test, or any other service running on the device. Automatically records response-time degradations and service-affecting latency spikes that could cause serious issues in the production network.

- **System Monitoring** – monitors the device under test via CLI, SNMP, Syslog Console, Telnet, or SSH, to detect subtle fault conditions such as CPU spikes or memory leaks that may impact service quality or robustness.

- **Reporting** – The Mu-4000 generates actionable business reports along with the technical details needed to take remediation action to minimize any identified weaknesses.