

Using Spirent's (Avalanche) Tcl API on FreeBSD

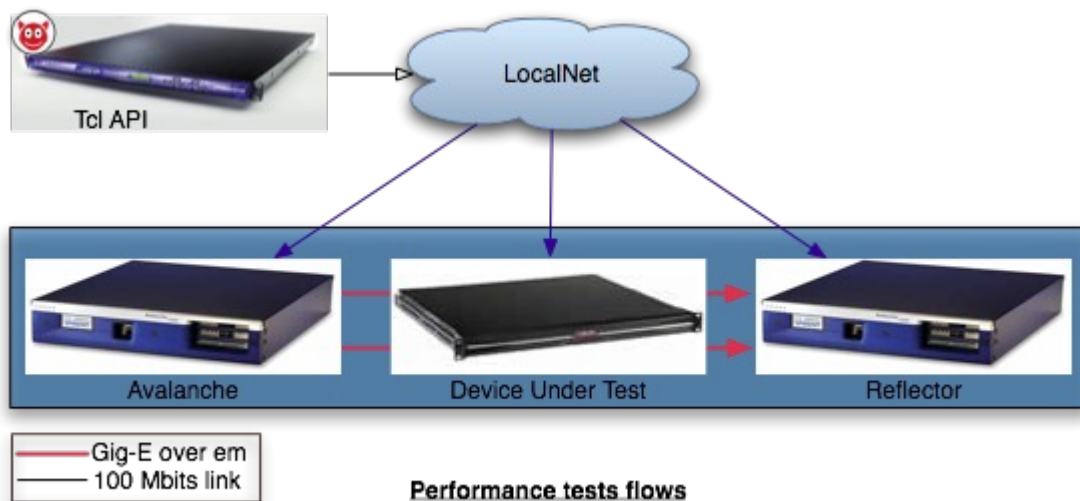
Before going further, I strongly encourage you to read the « Using the scripting API » part of the Spirent documentation.

While testing a proxy in term of performance (benchmarking) with and without some antiviral solutions, I found that using the Avalanche GUI to launch the tests was painful when trying to evaluate many cases.

Besides this, we have to use the Tcl API because it is the only way to include the invocation of the performance tests and

What follows focus on the HTTP, POP3 & SMTP protocols.

One caveat has not been resolved, we are using traditional performance tests, this prevents us from knowing if performance have been improved in the tested release.



Preparing the « Commander » host

Of course, SpirentCom does not support this work and their document [SPI01] regarding Unix is, as far as I know, incomplete.

Though, the Linux ABI support in FreeBSD, which is documented in the handbook [BSD01], is smart enough to let us do whatever we want.

```
$ brandelf -t Linux AvalancheTclAPI_Linux_7_02_0_37702.bin
# ./AvalancheTclAPI_Linux_7_02_0_37702.bin -console
```

Follow the instructions...

For some reasons the usual TCL does not work with the TclLib, you need to install the ActiveState package, which has not been package for FreeBSD...

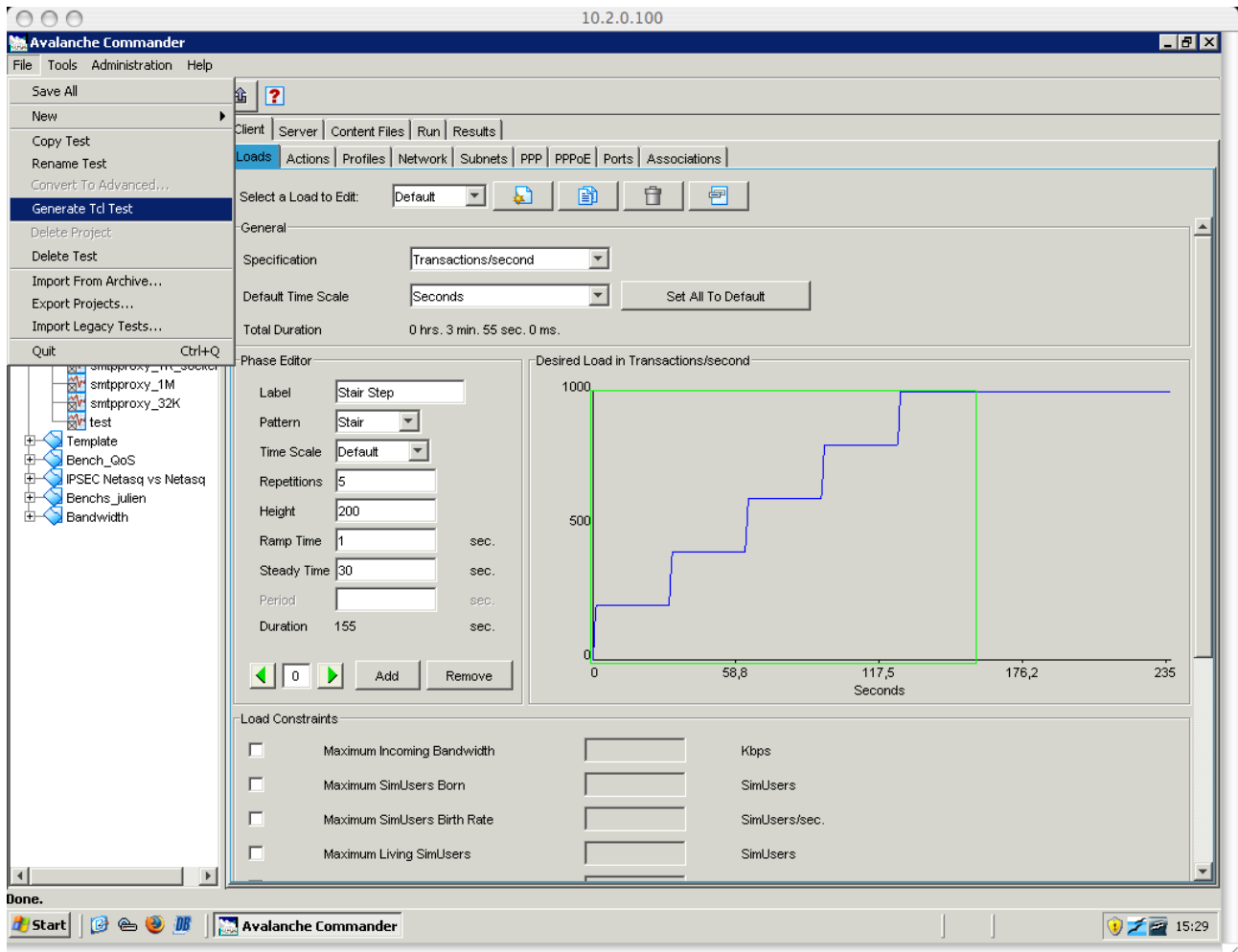
Get ActiveTcl8.4.12.0.226725- linux- ix86.tar.gz, unpack it, run the install.sh script, then add /usr/local/ActiveTcl/lib/ and SmartLib/unix/linux to your LD_LIBRARY_PATH

```
$ export LD_LIBRARY_PATH=/usr/local/ActiveTcl/lib:/opt/SpirentCommunications/Avalanche/TclAPI/SmartLib/unix/linux
```

Launching the benchmark

First, you need a test...

To quickly get started, use the Avalanche Commander to design your test and generate its Tcl counterpart.



From the Spirent Literature [SPI01]:

« Any test written with the Avalanche Commander can run with some little modification using the Scripting API. »

Modification of those scripts is beyond the scope of this document, please refer to the relevant documentation and our example (Appendix).

The API crash on Unix if you use things like:

```
smtp://172.17.0.1 FROM=<Emails^1> TO=<Emails^2> SUBJECT=<Emails^3> SMTP_BODY_FILE=<"32K_body.txt" "plain/text">
```

Then comes the invocation of a performance test. It is as simple as the following command line:

```
$ /usr/local/ActiveTcl/bin/tclsh8.4 /path/to/your/bench.tcl
```

If everything goes as expected, you should see something like:

```
xterm
daffy@wonderland:~/http_1k %> tclsh8.4 test_http10proxy_1k.tcl
Generating Test - done
Stopping Client Cluster - done
Stopping Server Cluster - done
Removing Test From Client Cluster - done
Removing Test From Server Cluster - done
Uploading Test To Client Cluster - done
Uploading Test To Server Cluster - done
Clearing Event Logs Of Client Cluster - done
Clearing Event Logs Of Server Cluster - done
Starting Server Cluster - done
Starting Client ClusterstatsCallbackProc (Cluster1); Testname = Test Name; Time Elapsed = 4
statsCallbackProc (Cluster1); Testname = Test Name; Time Elapsed = 8
- done
Waiting For Client Cluster To CompletestatsCallbackProc (Cluster1); Testname = Test Name; Time Elapsed = 12
statsCallbackProc (Cluster0); Testname = AdvancedDeviceTestExample; Time Elapsed = 4, Time Remaining = 230
http,attemptedTxnsPerSec : 177
http,attemptedTxns      : 706
http,successfulTxns    : 706
http,unsuccessfulTxns  : 0
statsCallbackProc (Cluster1); Testname = Test Name; Time Elapsed = 16
```

At the end of the test, it dumps the test results into some CSV files which can be used for detailed analysis (latency, TCP sockets states, etc...)

Those this is not the awaited behaviour (or I may have missed something...), to work around, we modify that part of the script that dumps the stats because we do not want our results to be overwritten every tests.

WorkSuite Manager

From the Spirent documentation:

« A Work Suite is designed to iterate over one or more other tests and to change test parameters between iterations. »

Now, we have got the opportunity to validate all performance aspects of our proxy by invoking only one Tcl script. Let's imagine, we want to get some results for the POP3 and HTTP protocols, with or without ClamAV; the WorkSuite will modify the desired load for us, that is amazing !

Well, unfortunately, it is not possible to export a WorkSuite for an usage with the Tcl API, so this feature is useless in our case ;-)

We have written a quick Tcl Script which unrolls some tests for tproxyd and modify the firewall configuration (using nsrpc) between each iteration and/or tests.

How to export the results in HTML

At the moment, there are no automatic way to deal with this, next release of the API may

support this feature.

Of course, we can write a generic parser for the CSV files but this may take some times...

So for now, the best practice to get a graphical representation of those test results is to use the Analyser.

Fetching the test results

In the first time, we were thinking of using a perl script to extract the last values (supposed to be the highest) from the comma separated files but this approach was not generic and will require a script per test.

Indeed, each time we change the test duration, the number of stairs' repetitions or their heights, the structure of the CSV file changes too.

The Tcl API offers facilities through functions like filterList and RegisterStatsCallback.

FilterList is the patterns matching part.

TOBECOMPLETED

References

[SPI01] Scripting_API_Avalanche.pdf

[BSD01] http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/linuxemu-advanced.html

Appendix: Sample Spirent Test

```
#
# Define a callback procedure
#
proc statsCallbackProc {clusterID data} {
    array set dataArray $data

    puts -nonewline "statsCallbackProc ($clusterID)"

    if {[info exists dataArray(timeElapsed)] && [info exists dataArray(timeRemaining)]} {
        puts -nonewline ": Testname = $dataArray(testName); Time Elapsed =
$array(timeElapsed), Time Remaining = $dataArray(timeRemaining)"
    } elseif {[info exists dataArray(timeElapsed)]} {
        puts -nonewline ": Testname = $dataArray(testName); Time Elapsed =
$array(timeElapsed)"
    }

    catch {
        puts ""
        puts "pop3,attemptedSessionsPerSec      :
$array(pop3,attemptedSessionsPerSec)"
        puts "pop3,successfulSessionsPerSec      :
$array(pop3,successfulSessionsPerSec)"
        puts "pop3,unsuccessfulSessionsPerSec      :
$array(pop3,unsuccessfulSessionsPerSec)"
        puts "pop3,activeSessions                  : $dataArray(pop3,activeSessions)"
    }
    puts ""
}

set testStarted "false"
```

```

if {[catch {
    # Insert this scripts parent directory into the tcl path
    set auto_path [linsert $auto_path 0 [file join [file dirname [info script]]
"/opt/SpirentCommunications/Avalanche/TclAPI"]]

    # Load the required package
    package forget SPI_AV
    package require SPI_AV

    # Initialize the API
    set rootCommanderDirectory ""
    SPI_AV::InitializeAPI $rootCommanderDirectory

    SPI_AV::AddLicense "/home/daffy/license.xml"

    # Source the config file
    source [file join [file dirname [info script]] "config_pop3proxy_1K.txt"]

    # Create the client and server clusters
    set clientUnits [list $ava_10_2_100_100_CPU_0]
    set clientClusterID [SPI_AV::ClusterController::CreateCluster "client"
"MyClientCluster" $clientUnits]

    set serverUnits [list $ref_10_2_100_101_CPU_0]
    set serverClusterID [SPI_AV::ClusterController::CreateCluster "server"
"MyServerCluster" $serverUnits]

    #
    set testDirectory "/home/daffy/Spirent_Tests"
    set testId "pop3_1k"

    #
    puts -nonewline "Generating Test"; flush stdout
    SPI_AV::ClusterController::GenerateClusteredTest clusterConfig $testDirectory $testId
    puts " - done"

    # Do not remove this line - It is used by the code to write provision command for a
SmartBits Test
    #

    # Stop the clusters
    #
    puts -nonewline "Stopping Client Cluster"; flush stdout
    SPI_AV::ClusterController::StopClusteredTest $clientClusterID
    SPI_AV::ClusterController::WaitForClusteredTestCompletion $clientClusterID
    puts " - done"

    puts -nonewline "Stopping Server Cluster"; flush stdout
    SPI_AV::ClusterController::StopClusteredTest $serverClusterID
    SPI_AV::ClusterController::WaitForClusteredTestCompletion $serverClusterID
    puts " - done"

    # Remove any previous copy of the test from the clusters
    #
    puts -nonewline "Removing Test From Client Cluster"; flush stdout
    SPI_AV::ClusterController::RemoveClusteredTest $clientClusterID $testId
    puts " - done"

    puts -nonewline "Removing Test From Server Cluster"; flush stdout
    SPI_AV::ClusterController::RemoveClusteredTest $serverClusterID $testId
    puts " - done"

    #
    # Upload the test to the clusters
    #
    puts -nonewline "Uploading Test To Client Cluster"; flush stdout
    SPI_AV::ClusterController::UploadClusteredTest $clientClusterID $testDirectory
$testId
    puts " - done"
}]}

```

```

puts -nonewline "Uploading Test To Server Cluster"; flush stdout
SPI_AV::ClusterController::UploadClusteredTest $serverClusterID $testDirectory
$testId
puts " - done"

#
# Register a statsCallback for each cluster
#
set clientStatsCallbackID [SPI_AV::ClusterController::RegisterStatsCallback
$clientClusterID statsCallbackProc [list "testName" "time*" "pop3"]]
set serverStatsCallbackID [SPI_AV::ClusterController::RegisterStatsCallback
$serverClusterID statsCallbackProc [list "testName" "time*" "pop3"]]

#
# Clear the Event Log on all devices participating in the test
#
puts -nonewline "Clearing Event Logs Of Client Cluster"; flush stdout
SPI_AV::ClusterController::ClearEventLogs $clientClusterID
puts " - done"

puts -nonewline "Clearing Event Logs Of Server Cluster"; flush stdout
SPI_AV::ClusterController::ClearEventLogs $serverClusterID
puts " - done"

set testStarted "true"

#
# Start the test on each cluster
#
puts -nonewline "Starting Server Cluster"; flush stdout
SPI_AV::ClusterController::StartClusteredTest $serverClusterID $testId
puts " - done"

puts -nonewline "Starting Client Cluster"; flush stdout
SPI_AV::ClusterController::StartClusteredTest $clientClusterID $testId
puts " - done"

#
# Wait for the client cluster to finish
#
puts -nonewline "Waiting For Client Cluster To Complete"; flush stdout
SPI_AV::ClusterController::WaitForClusteredTestCompletion $clientClusterID
puts " - done"

#
# Stop the server cluster (and wait for it to stop)
#
puts -nonewline "Stopping Server Cluster"; flush stdout
SPI_AV::ClusterController::StopClusteredTest $serverClusterID
SPI_AV::ClusterController::WaitForClusteredTestCompletion $serverClusterID
puts " - done"

#
# Transfer the results of the client cluster
#

set resultsDirectory [file join $testDirectory $testId "results"]

puts -nonewline "Transferring Client Cluster Results"; flush stdout
SPI_AV::ClusterController::GetClusteredTestResults $clientClusterID $testId [file
join $resultsDirectory]
puts " - done"

#
# Transfer the results of the server cluster
#
puts -nonewline "Transferring Server Cluster Results"; flush stdout
SPI_AV::ClusterController::GetClusteredTestResults $serverClusterID $testId [file
join $resultsDirectory]
puts " - done"

```

```

} catchError]] {
  puts "EXCEPTION: $catchError"
  if $testStarted then {
    SPI_AV::ClusterController::AbortClusteredTest $clientClusterID
    SPI_AV::ClusterController::AbortClusteredTest $serverClusterID
  }
  SPI_AV::Dump
}

catch {
  #
  # Unregister the previously registered callbacks
  #
  SPI_AV::ClusterController::UnregisterStatsCallback $clientClusterID
  $clientStatsCallbackID
  SPI_AV::ClusterController::UnregisterStatsCallback $serverClusterID
  $serverStatsCallbackID
}

# Do not remove this line - It is used by the code to write reset cards command for a
SmartBits Test
if { $testStarted == "true" } {
#
}

SPI_AV::CleanUp
puts "Done!"

```