Plexus SIP Test Harness User Manual

79-3534

Revision History

Rev	Date	Purpose	Originator
0.5	2006-08-14	Version for alpha release	QD Tool Team
0.6	2006-09-04	Version for beta-1 release	QD Tool Team
0.7	2006-09-27	Version for beta-2 release	QD Tool Team
1.0	2006-10-30	Version for 1.0 release	QD Tool Team
1.1	2006-11-17	Version for 1.1 release	QD Tool Team
1.2	2008-11-04	Add procedure for HUG test	Joey Zhang

Table of Contents

1 Introduction

1.1 Purpose

The SIP test harness tool will be used by Plexus SIP developers for feature and bug unit testing and testing automation. A regression test suite will be built up over time.

1.2 Features

Supported Features in alpha:

- GCC to SIP half-call model [什么是 half-call]
- interactive command line & scripted modes
- basic call scenarios
- test suites (batch run)
- multiple simultaneous users
- collection of logs & SIPA alarms
- user manual & example test case scripts provided
- both sequential and state-based scripting

NOTE: Due to tool server performance limitations, tests will fail occasionally if ~5 users simultaneously run test suites.

Supported Features in beta-1:

- Quick search command history (bugzilla 49492)
- Support DNS in test scripts
- Pass/Fail test case based on GCC message
- Pass/Fail test case based on SIP message
- Support multiple tests simultaneously in one user login
- Provide over 35 example scripts

NOTE:

- All test harness codes have been checked into MAIN branch. Users can check out code from CVS, and build/run test harness on their sandbox. For further information, please refer to section 2.1
- For DNS feature, test harness can't support its simultaneous test. The reason is that DNS feature needs to write test-specific DNS info into /etc/resolv.conf file, which is shared by all users. For details, please refer to section 8.4

Supported Features in beta-2:

- Support two SIPp
- SIPp screen display (#49446)
- GDB integration (#49445)
- Manual testing
- Bugs Fix (#50540, #50440, #50578, #50294, #50255)

Supported Features in 1.0:

- SIPA failover and redundancy
- Code coverage analysis
- SIPp 1.1 integration

- Different log directories for each session

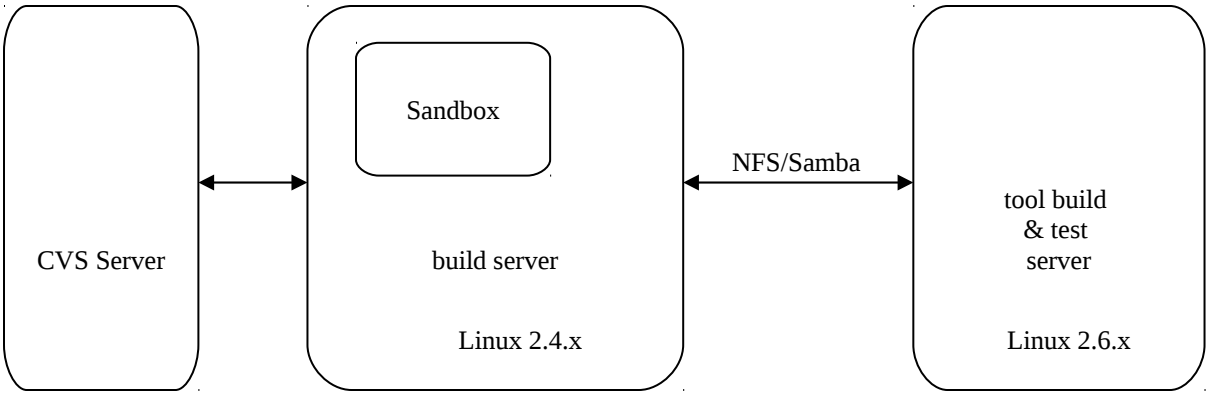
Supported Features in 1.1:

- SIPT support
- Multi-level test suite
- Valgrind integration
- Bug fixes

Supported in future releases:

- ddd integration
- GUI
- Additional example scripts

1.3 Test environment



This diagram illustrates the planned hardware and software environment for SIP test harness. The tool code will be in cvs; the user can check it out into their sandbox on a build server. The tool itself will be run from a dedicated tool server (Linux PC) in order to isolate the performance impact from the build server and utilize a newer Linux kernel. Thus to build and run the tool, the user will log into the tool server, which mounts the users sandbox via NFS. For details, please refer to chapter 8.

Since the test harness code is not yet in cvs, for the alpha release, individual accounts have been set up for the alpha testers, each with its own copy of a sandbox and the tool code. Alpha users will log in directly to the tool server.

2 Getting Started

All test harness source codes (including test scripts) have been checked into Main branch. Then what users need to do is just checking out the specific branch they want and build & run directly.

2.1 Check-out Test Harness

A new script named ***TestHarnessUpdate*** was added which is used to check out test harness only. Basically it will check out test harness code, test scripts, SIPA, Relay, logging library and some header files that are necessary to build test harness. The widely used ***TelicaUpdate*** script is not changed so even if users have run ***TelicaUpdate*** before they still need to execute ***TestHarnessUpdate*** to get test harness. Detailed check-out steps are as follows,

1) Set CVS authentication using **`cvs login`**.
2) Go to your working sandbox directory. For example, type **`cd sandbox/Main`**.
3) Type **`cvs co –r <branch> TelicaRoot/TestHarnessUpdate`**.
4) Type **`./TelicaRoot/TestHarnessUpdate –r <branch>.`**

IMPORTANT NOTE:

Check-out will take long time on QD test harness server due to network speed limitation.

Tool Source Code Location:

Existing SIPA, Relay and logging library code remain where it is currently located.

Test scripts will be in `TelicaRoot/components/test_harness/scripts/sip`.

For other components, in `TelicaRoot/components/test_harness`

```
test_harness                        - common makefile
test_harness/bin                    - executable files
test_harness/sipp1.1                - SIPp source code
test_harness/lmgcc                  - lm and gcc emulator
test_harness/expect_controller      - expect controller
test_harness/3rdparty               - all 3rd party tools
test_harness/scripts/sip            - all SIP test cases
```

2.2 Build Test Harness

1) Go to directory `TelicaRoot/components/test_harness`.
2) Type **`make`** to build release version SIPA and other executables. All the executables will be in directory `TelicaRoot/components/test_harness/bin`, including

```
sipa            - Linux version SIPA product
siptest         - expect controller executable
sipp            - SIPp executable
liblmgcc.so     - lm and gcc emulator library used by expect controller
libtoolcom.so   - common library used by expect controller
inittool.tcl    - TCL script used by expect controller
runtestsuite.tcl - TCL script used by expect controller
```

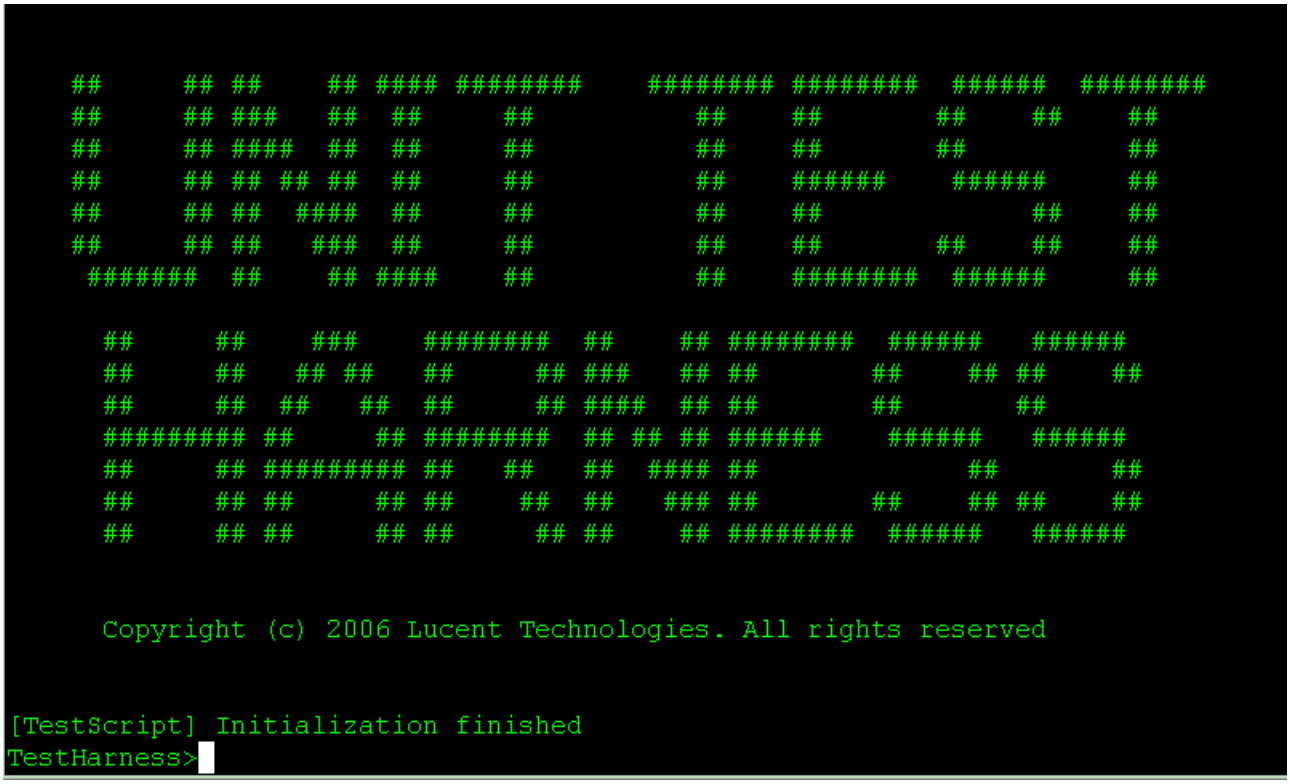
runtl1.tcl - TCL script used by expect controller
gcc.tcl - TCL script used by expect controller

3) Or use **`make all_debug`** to build debug version SIPA and other executables.

4) If you want to clean all built objects, please type **`make clean`**.

2.3 Start Test Harness

Test harness provides a single point for users to start - go to directory `test_harness/bin` and type `./siptest`. Then you could run commands using following command line interface (with `TestHarness>` prompt).



2.4 Run Sample Test

All our test scripts are stored in directory `test_harness/scripts/sip`. Test harness uses '`runtest`' and '`runtestsuite`' methods to execute them.

runtest method

`runtest` method is used to run a single test case. For each single test case there are 4 test scripts (for detailed information please refer to section 3). For example,

```
basiccall1.tc              - Main script file (in TCL format)
basiccall1.cs              - Call scenario file
basiccall1.xml             - SIPp configuration file
reg.csv                    - SIPp configuration file
```

The syntax for runtest method is runtest <main_script_filename>. Below is a screenshot.

```
TestHarness>runtest basiccall1.tc
1 test cases are running (enter "stoptest" to stop running) ...
```

You may enter “lstc” command to list all available “.tc” files.

runtestsuite method

`runtestsuite` method is used to batch run test cases. The syntax is `runtestsuite <testsuite_filename>`. The content of test suite file is a simple list of main script file names or nested suite files. Following is an example,

File testsuite1.ts,

```
basiccall1.tc
basiccall2.tc
basiccall3.tc
basiccall4.tc
nestedsuite.ts
```

Type `runtestsuite testsuite1.ts`, all above four basic cases and the cases in nestedsuite.ts will be executed.

You can enter “lsts” command to list all available “.ts” files.

manual run mode

`runtestsuite` and `runtest` methods are very suitable for automation testing. From beta-2 release test harness also support a manual run mode which will be useful for developers to do debugging. Basically it allows user to start/stop SIPA and SIPp processes manually and fully control the GCC Emulator actions. GDB could also be used to do run time debug. Please refer to section 5.2 for more details.

2.5 Examine Test Output

After the execution of each `runtest` or `runtestsuite` method, test harness will print out testing results statistics. Following is an example.

```
[TestScript] One TestCase Finished!
Test cases (1/1) completed successfully

-----Test Result------
  Total Test Cases: 1
  Completed Test Cases: 1
  Passed Test Cases: 1
  Failed Test Cases: 0
----Completed Test------
  TestCase 1: Script: success; SIPP: success
```

As described in section 2.1, all SIPA logs are generated in directory `test_harness/bin` so that developers can have additional information for investigation/debugging.

What we want to specify a bit more here is **runtl1** method. Using this method users could execute some TL1 commands (such as SIGDBG commands) under `TestHarness>` prompt. The syntax is `runtl1 {normal tl1 command}`. For example,

```
TestHarness>runtl1 {sigdbg:::::dest-a-ccs-slave sipa dbglvl 0};
```

For a list of supported TL1 commands, see section 6.2.

2.6 Where to find Log & Core files

Every time SIP Test Harness starts up, it generates new directory under "test_harness/bin/" directory.

This log directory is used to accommodate SIPA logs, Expect Controller logs, LM&GCC emulator logs, and SIPA core files. The log directory name consists of 10 digits, which are as follows.

MMDDHHMISS (Month Day Hour Minute Second).

Run "*logdir*" commands under `TestHarness>` prompt to query current log directory. Detailed info about this command, please refer to section 6.1.

2.7 Quit Test Harness

Run exit under `TestHarness>` prompt to exit the tool.

3 Test Script Writer's Guide

3.1 Introduction

This section provides instructions on scripting for SIP Test Harness Tool and assumes a basic knowledge of the unit operations. Through writing customized test script, virtually any call type can be simulated. As with any programming language, to build a script program it is necessary to outline the tasks that you want the program to execute. Once you list the tasks, you can enter them in a flow chart or state diagram. You can apply the appropriate elements of the script language to accomplish the defined task.

3.2 Basic Components of Test Script

Usually there are four scripts for a given test case, which are main script, call scenario script, SIPp XML file and SIPp CSV file. SIPp CSV file is optional and can reuse other cases'.

- Main script

  This script will control the whole test case execution flow. It will take charge of the following functions:

  - To configure SIPA debug level
  - To configure SIPA trunk group profile
  - To configure SIPA trunk group
  - To configure SIPA route information
  - To load the SIPA configuration script
  - To startup SIPp with SIPp XML file and SIPp CSV file (optional). SIPp is used as the SIP UA external to the switch that communicates with SIPA.
  - To run call scenario script

- Call scenario script

  This script is designed according to testing cases and it acts as GCC in this tool. It follows Tcl syntax and semantics. It is responsible for sending and receiving PSIF-SIP primitives. It can be written in either a sequential mode or state-based mode, selectable on a per-test-case basis.

- SIPp XML file

  This file is for SIPp process and it is call scenario file of SIPp. SIPp will send and receive SIP requests and responses according to this file. Refer to SIPp Reference Documentation v1.1 for details (see section 10).

- SIPp CSV file

  This file is also for SIPp. It is optional or can reuse others'. Usually it defines variable parameters of SIPp XML file. You can refer to SIPp Reference Documentation v1.1 for details (see section 10).

3.3 Test Script Example

Here is a sample script for a basic call. For its detailed call flow, please refer to section 7. The sequence of events are as follows:

1. SIPp-A sends INVITE
2. SIPp-A receives 100 Trying from SIPA
3. Emulator receives SipwLiConInd from SIPA
4. Emulator sends back SipwLiCnStReq to SIPA

5. SIPp-A receives 180 Ring from SIPA
6. Emulator sends back SipwLiConRsp to SIPA
7. SIPp-A receives 200 OK from SIPA
8. SIPp-A sends ACK to SIPA
9. SIPp-A sends BYE after talking to SIPA
10. Emulator receives SipwLiRelInd from SIPA
11. Emulator sends back SipwLiRelRsp to SIPA
12. SIPp-A receives 200 OK from SIPA

This section covers following script files.

- basiccall1.tc : Main script
- basiccall1.cs : call scenario file
- basiccall1.xml : SIPp XML file
- reg.csv : SIPp CSV file

3.3.1 Main Script

Following is an example of a main script file (taken from basiccall1):

```
;# main script file example
;# this main script refer to SIPA configuration file, SIPp configuration file, call
scenario file.

;# this tool will calculate port for each cases, get the hostip and script
directory
;# attention: currently below 4 global variables are mandatory
global sipp_local_port ;#global variable storing "local port of sipp"
global sipa_listen_port ;#global variable storing "Listening port of sipa"
global hostip ;#global variable storing "Host IP" our tool runs on
global scrDir ;#global variable storing "directory name script located"

;# provision SIPA
;# to set sipa debug level
runtl1 {sigdbg:::::dest-a-ccs-slave sipa dbglvl 0};
;#attention: don't wait since SIPA doesn't respond to this command
;# to configure sipa profile
runtl1 ent-prfl-sip::1:::maxfwd=100,sip3xxhandling=release
runtl1 ent-
trkgrp::9999:::sip:siproutename=test,tgprofile=1,sipDstFqdn=135.252.142.150
runtl1 ent-sip-tgmapaddr::135.252.142.150-ip-${sipp_local_port}:::tgn=9999

;# schedule SIPp startup
;# to set sipp options
set sipp_args "$hostip:$sipa_listen_port -p $sipp_local_port -mp $sipp_media_port
-sf [getscriptdir]basiccall1.xml -inf [getscriptdir]reg.csv -m 1 -trace_msg"
;# to start up SIPp after 5 seconds since SIPp should be start up later than
basiccall1.cs in this this case.
schedule 5 {startupSIPp $sipp_args}

;# load call scenario file, and start real test scenario
runcs basiccall1.cs
```

Table Number 1. basiccall1.tc

3.3.2 Call Scenario File

For this file, we can support 2 modes, one is sequential test script and the other is state-based test script. Note that, in a given test case, only one of these would be used.

1. Sequential test script sample

```
;#  call scenario script file example
;# system initialization
;# ...

global CCEVNT  ;# global variable for call control event

set state "IDLE"  ;# global variable storing "state"
set event "null"  ;# global variable storing incoming event
;# currently system use “1” to represent the call control event.

set conIndTime 20 ;#Wait for Connection Indication time out
set relIndTime 100 ;#Wait for Release Indication time out, including call hold time

;# start test case
;# wait for sipwLiConInd from SIPA. Global variable sipwLiConInd is the pointer to
received message.
waitforcond $conIndTime $CCEVNT {$event==$EVTSIPWCONIND}

;# create and set parameters for sipwLiCnStReq
;# then SIPA will translate it into 180 message to SIPp
;# gcc_suConnId is a variable maintained by expect controller. It will be
;# incremented by 1 at the beginning of each case to avoid every test case
;# use the same suConnId
set sendCnStReq [SipwLiCnStEvnt]
setGccPara SipwLiCnStEvnt $sendCnStReq "
                suConnId $gcc_suConnId
                sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]
                evntType.pres 1
                evntType.val 180"
sendCnStEvnt $sendCnStReq $EVTSIPWCNSTREQ
tsleep 1

;# create and set parameters for sipwLiConRsp
;# then SIPA will translate it into 200 OK message to SIPp
set sendConRsp [SipwLiConEvnt]
setGccPara SipwLiConEvnt $sendConRsp "
                suConnId [getGccPara SipwLiCnStEvnt $sendCnStReq suConnId]
                sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]
                sdp.eh.pres 1
                sdp.chosenOne 1
                sdp.numDecode 1
                sdp.p(0).address.ip.binary.pres         1
                sdp.p(0).address.ip.binary.val          0x0100007f
                sdp.p(0).address.ip.port.pres           1
                sdp.p(0).address.ip.port.val            0
                sdp.p(0).mediaType                      $SDP_MTYPE_AUDIO
                sdp.p(0).m.voip.codec(0).payloadType    $SDP_CODEC_PCMU
                sdp.p(0).m.voip.codec(0).codec.pres     1
                sdp.p(0).m.voip.codec(0).codec.val      $SDP_CODEC_PCMU
                sdp.p(0).m.voip.codec(0).clockRate.pres 1
                sdp.p(0).m.voip.codec(0).clockRate.val  $SDP_CLOCK_RATE_DEFAULT
                sdp.p(0).m.voip.numCodec                1
                sdp.p(0).m.voip.rtpInfo.packet.pres     1
                sdp.p(0).m.voip.rtpInfo.packet.val      40"
```

```
sendConEvnt $sendConRsp $EVTSIPWCONRSP

print "voice path established!"

;# wait for SIPp BYE message and SIPA translate it into sipwLiRelInd.
waitforcond $relIndTime $CCEVNT {$event==$EVTSIPWRELIND}
tsleep 1
;# create and set parameters for sipwLiRelRsp
;# then SIPA will translate it into 200 OK message to SIPp
set sendRelRsp [SipwLiRelEvnt]
setGccPara SipwLiRelEvnt $sendRelRsp "
                suConnId [getGccPara SipwLiCnStEvnt $sendCnStReq suConnId]
                sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]"
sendRelEvnt $sendRelRsp $EVTSIPWRELRSP
```

Table Number 2. sequential basiccall1.cs

2. state-based test script sample

```
;# overall call scenario script file example

;# system initialization
;# ...

global CCEVNT     ;# global variable for call control event
set state "IDLE"  ;# global variable storing "state"
set event "NULL"  ;# global variable storing incoming event
settesttimer 60   ;# set maximum execution time for this test case

;# register call control event handler
;# call state machine script
;# gcc_suConnId is a variable maintained by expect controller. It will be
;# incremented by 1 at the beginning of each case to avoid every test case
;# use the same suConnId
regevent $CCEVNT {
;# call state machine script
    switch -- $state {
        "IDLE"  {
            if {$event==$EVTSIPWCONIND} {
                set stReq [SipwLiCnStEvnt]
                setGccPara SipwLiCnStEvnt $stReq "
                    suConnId $gcc_suConnId
                    sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]
                    evntType.pres 1
                    evntType.val 180"
                sendMsg SipwLiCnStEvnt $stReq $EVTSIPWCNSTREQ
                set state "SND_CNST_REQ"

                tsleep 1

                set conRsp [SipwLiConEvnt]
                setGccPara SipwLiConEvnt $conRsp "
                    suConnId [getGccPara SipwLiCnStEvnt $stReq suConnId]
                    sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]
                    sdp.eh.pres 1
                    sdp.chosenOne 1
                    sdp.numDecode 1
                    sdp.p(0).address.ip.binary.pres         1
                    sdp.p(0).address.ip.binary.val          0x0100007f
                    sdp.p(0).address.ip.port.pres           1
                    sdp.p(0).address.ip.port.val            0
```

```
                    sdp.p(0).mediaType                          $SDP_MTYPE_AUDIO
                    sdp.p(0).m.voip.codec(0).payloadType        $SDP_CODEC_PCMU
                    sdp.p(0).m.voip.codec(0).codec.pres         1
                    sdp.p(0).m.voip.codec(0).codec.val          $SDP_CODEC_PCMU
                    sdp.p(0).m.voip.codec(0).clockRate.pres     1
                    sdp.p(0).m.voip.codec(0).clockRate.val      $SDP_CLOCK_RATE_DEFAULT
                    sdp.p(0).m.voip.numCodec                    1
                    sdp.p(0).m.voip.rtpInfo.packet.pres         1
                    sdp.p(0).m.voip.rtpInfo.packet.val          40"
                sendMsg SipwLiConEvnt $conRsp $EVTSIPWCONRSP
                set state "SND_CON_RSP"

                puts "voice path is established"
            } else {
                puts "unexpected event: $event"
                quittest $TC_FAIL
            }
        }
        "SND_CON_RSP" {
            if {$event==$EVTSIPWRELIND} {
                set relRsp [SipwLiRelEvnt]
                setGccPara SipwLiRelEvnt $relRsp "
                    suConnId [getGccPara SipwLiCnStEvnt $stReq suConnId]
                    sipInstId [getGccPara SipwLiConEvnt $sipwLiConInd sipInstId]"
                sendMsg SipwLiRelEvnt $relRsp $EVTSIPWRELRSP
                set state "SUCCESS"
                quittest $TC_SUCCESS
            }
        }
        default {
            puts "unknow event!"
            quittest $TC_FAIL
        }
    }
}
```

Table Number 3. state-based basic1_state.cs

3.3.3 SIPp XML File

Following is an example of a SIPp XML file (taken from basiccall1):

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <!-- This must be the first line-->

<!-- SIPp scenario: NormalCall_1. Terminated by orignating party. Originating party
-->
<scenario name="Basic Sipstone UAC">

<!-- send a INVITE message to remote side -->
<send>
<![CDATA[
INVITE sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>
Call-ID: [call_id]
Cseq:1 INVITE
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
Content-Type:application/sdp
Content-Length:[len]
```

```
v=0
o=user1 53544765 2353687637 IN IP4 127.0.0.1
s=-
c=IN IP4 [media_ip]
t=0 0
m=audio [media_port] RTP/AVP 0
a=rtpmap:0 PCMU/8000

]]>
</send>

<!-- wait for 100 TRYING message -->
<recv response="100"> optional="true"
</recv>

<!-- wait for 180 RING message -->
<recv response="180"> optional="180"
</recv>

<!-- wait for 200 OK message -->
<recv response="200"> optional="200"
</recv>

<!-- send a ACK message -->
<send>
<![CDATA[
ACK sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>[peer_tag_param]
Call-ID: [call_id]
Cseq:1 ACK
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
Content-Length:0
]]>
</send>

<!-- stop 3 seconds -->
<pause milliseconds="3000">
</pause>

<!-- send BYE message -->
<send retrans="500">
<![CDATA[
BYE sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>[peer_tag_param]
Call-ID: [call_id]
Cseq:2 BYE
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
Content-Length:0

]]>
</send>

<!-- wait for 200 OK message -->
<recv response="200">
```

```
</recv>

</scenario>
```

Table Number 4. basiccall1.xml

3.3.4 SIPp CSV File

An example SIPp CSV file follows. Note that this file is used by all example test cases.

```
#file reg.csv
SEQUENTIAL
1122;3344
```

Table Number 5. reg.csv

3.4 Parameter Values for Messages Sent by GCC Emulator

From the above call scenario script, we see there are many parameters in primitives and they will need values to fulfill the call. This tool does support getting values from a previously received message, which will be helpful to fill the parameters.

GCC Emulator will need to generate sipwLiCnStReq, sipwLiConRsp, sipwLiRelRsp, sipwLiConReq and sipwLiRelReq. The following description focuses on setting mandatory parameter values for a SIP-SIP basic call.

1) SipwLiCnStReq

Declare a variable as SipwLiCnStEvnt type. For a basic call, the following parameters must be assigned:

suConnId: need end user to set, its value is between 1 … 4294967295(2^{32}-1)

sipInstId: need to get from sipwLiConInd (use supplied function getGccPara)

evntType: need to set to 18x

2) SipwLiConRsp

Declare a variable as SipwLiConEvnt type. For a basic call, the following parameters must be assigned:

suConnId: need to get from SipwLiCnStReq

sipInstId: need to get from sipwLiConInd

sdp.eh.pres: need to be set, it is 1

sdp.numDecode: need to be set, it determines how many media stream in sdp.p is valid.

sdp.p: need to be set, it is an array of data about media stream.

sdp.p(0).address.ip.binary.pres : need to be set, it is 1.

sdp.p(0).address.ip.binary.val : need to be set.

sdp.p(0).address.ip.port.pres: need to be set, it is 1.

sdp.p(0).address.ip.port.val: need to be set.

sdp.p(0).m.voip.rtpInfo.packet.pres: need to be set, it is 1

sdp.p(0).m.voip.rtpInfo.packet.val: need to be set

sdp.p(0).mediaType: need to be set

sdp.p(0).m.voip.numCodec: need to be set, it determines how many codec in sdp.p(0).m.voip.codec is valid.

sdp.p(0).m.voip.codec(0).payloadType: need to be set

sdp.p(0).m.voip.codec(0).codec.pres: need to be set, it is 1.

sdp.p(0).m.voip.codec(0).codec.val: need to be set.

sdp.p(0).m.voip.codec(0).clockRate.pres: need to be set, it is 1.

3) SipwLiRelRsp

Declare a variable as SipwLiRelEvnt type. For a basic call, the following parameters must be assigned:

suConnId: need to get from SipwLiCnStReq

sipInstId: need to get from sipwLiConInd

4) SipwLiConReq

Declare a variable as SipwLiConEvnt type. For a basic call, the following parameters must be assigned:

suConnId: need end user to set, its value is between 1 … 4294967295(2^{32}-1)

reqUri: set to CLD

toHdr: set to switch address

fromHdr: set to CLG address

fsIpInfo: set the trunk group ID

maxForwards: set to “69”

subject: set to an arbitrary string

sdp.eh.pres: need to be set, it is 1

sdp.numDecode: need to be set, it determines how many media stream in sdp.p is valid.

sdp.p: need to be set, it is an array of data about media stream.

sdp.p(0).address.ip.binary.pres : need to be set, it is 1.

sdp.p(0).address.ip.binary.val : need to be set.

sdp.p(0).address.ip.port.pres: need to be set, it is 1.

sdp.p(0).address.ip.port.val: need to be set.

sdp.p(0).m.voip.rtpInfo.packet.pres: need to be set, it is 1

sdp.p(0).m.voip.rtpInfo.packet.val: need to be set

sdp.p(0).mediaType: need to be set

sdp.p(0).m.voip.numCodec: need to be set, it determines how many codec in sdp.p(0).m.voip.codec is valid.

sdp.p(0).m.voip.codec(0).payloadType: need to be set

sdp.p(0).m.voip.codec(0).codec.pres: need to be set, it is 1.

sdp.p(0).m.voip.codec(0).codec.val: need to be set.

sdp.p(0).m.voip.codec(0).clockRate.pres: need to be set, it is 1.

5) SipwLiRelReq

Declare a variable as SipwLiRelEvnt type. For a basic call, the following parameters must be assigned:

suConnId: need to get from SipwLiConReq

sipInstId: need to get from sipwLiConRsp reqUri: set to CLD

3.5 Messages Received by GCC Emulator

From the above call scenario script, we can see the tool will receive primitives from SIPA. These message bodies of these primitives will be stored as global variables in the tool.

1) sipwLiConInd
2) sipwLiCnStInd
3) sipwLiConCfm
4) sipwLiRelInd
5) sipwLiRelCfm
6) sipwLiSvcInd
7) sipwLiMsgInd
8) sipwLiMsgCfm
9) sipwLiAudCfm
10) sipwLiGeoCrtCfm
11) sipwLiGeoAudInd

For details of the variable types, see section 6.3.

3.6 Fail test case by user

The previous sections provide some normal script examples. In some conditions, such as receiving unexpected GCC parameters or messages, users may want to fail the case by themselves . This section will introduce the failed cases mechanism and provide some examples.

3.6.1 Fail case by checking GCC message

1) Fail sequential cases

Referring to the assertion mechanism in C and C++, the tool provides a similar command *"tclassert"*, which is used to fail a test case explicitly by user.

The usage of this command is similar to function *assert()*. It accepts a logic expression as an argument, if the value of the expression is true, the case continues; otherwise *tclassert* will make the test case failed. Besides failing the case, expect controller also prints useful information on screen, including the logic expression, the line *tclassert* locates in. Following is an simple example:

tclassert {0>1}

A more meaningful example is also provided, please refer to the example test case names "*fail_bc1_assert.tc*" locating in the directory "*test_harness/script/sip/*". And you can also see its description in section 7.

2) Fail state-based cases

From the state-based example provided in section 3.3.2, we can see that the tool provides a command

"*quittest*" to fail the case immediately if some errors happen. The usage of this command please refer section 6.3.

In fact, *tclassert* also can be used in state-based case, but tool will not print out the corresponding line number because the script is not executed sequentially, and test harness has no idea of the line number.

3.6.2 Fail case by checking SIP message

For the above example of SIPp xml file in section 3.3.3, the content in *<recv></recv>* tag is empty. So the tool doesn't do any check about received messages. We can use regular expressions in SIPp to extract content of a SIP message or a SIP header, and check that whether these message or header match the expected pattern. If they don't match, SIPp will mark the call as failed, and the test case fails as well. The following is an example:

```
<recv response="200" start_rtd="true">
  <action>
    <ereg regexp="([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*" search_in="msg" check_it="true"
assign_to="1,2" />
    <ereg regexp=".*" search_in="hdr" header="Contact:" check_it="true" assign_to="6" />
  </action>
</recv>
```

You can check SIPp User Manual§ for basic information about how to use "regexp".

Besides the basic information documented in Ref doc, there are also some rules you must follow up:

1) Make sure the variable ids you "assign_to" are different for each "regexp", even they are in different "action" tags. The following is a wrong example:

```
<action>
    <ereg regexp= .....assign_to="1">
    <ereg regexp= .....assign_to="2">
</action>
  .....................
<action>
    <ereg regexp= .....assign_to="1">
    <ereg regexp= .....assign_to="2">
</action>
```

2) The later variable number MUST be bigger than the former ones. If you use as following, it doesn't work as you expect:

```
<action>
    <ereg regexp= .....assign_to="1">
    <ereg regexp= .....assign_to="4">
</action>
.....................
<action>
    <ereg regexp= .....assign_to="2">
    <ereg regexp= .....assign_to="3">
```

```
</action>
```

And these numbers don't need to be sequential ones, so the following also works too:

```
<action>
  <ereg regexp= .....assign_to="1">
  <ereg regexp= .....assign_to="4">
</action>
.....................
<action>
  <ereg regexp= .....assign_to="5">
  <ereg regexp= .....assign_to="8">
</action>
```

For example scripts, please refer to *"bc1_msgcheck.tc"* and *"bc2_msgcheck.tc"* in tool's directory *"test_harness/scripts/sip/"*. And you can also see their descriptions in section 7.

3.7 Test Suite File

As its name said, test suite will run batch of test cases according to some rule. Its command format is as follows:

TestHarness>runtestsuite filename

For example "runtestsuite testcases.ts". testcases.ts file is as below:

```
basiccall1.tc
basiccall2.tc
basiccall3.tc
basiccall4.tc
```

Table Number 6. testcases.ts

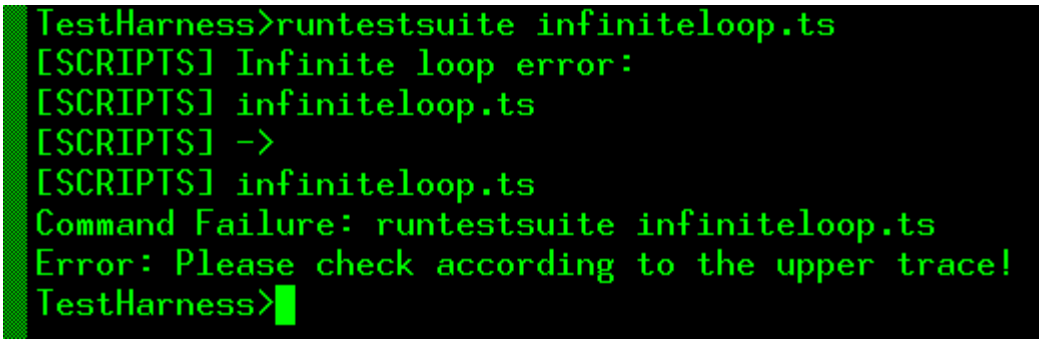
xxx.tc is the main script which was mentioned in section 3.3.1.

And the harness tool also supports multiple level test suite files. Suite file can include other suite file as part.

For example "rel10.ts" is suite file including all test cases the tool support in 1.0 release; release 1.1 not only supports all cases in 1.0 release, but also support more cases about SIPT, so "rel11.ts" can be written as below:

```
rel10.ts
bc1_sipt.tc
bc2_sipt.tc
```

If a infinite loop happens in the test suite including, the tool will show out a warning message, and print the nested trace to help user finding out the problem. For example:


TestHarness>runtestsuite infiniteloop.ts
[SCRIPTS] Infinite loop error:
[SCRIPTS] infiniteloop.ts
[SCRIPTS] ->
[SCRIPTS] infiniteloop.ts
Command Failure: runtestsuite infiniteloop.ts
Error: Please check according to the upper trace!
TestHarness>

4 Graphics User Interface

will be provided in future release

5 Advanced Features

5.1 Hot Keys

The harness accepts shell-like hot keys:

1. UP/DOWN key : get the history commands;
2. LEFT/RIGHT key: move the cursor left or right
3. BACKSPACE key: delete previous inputted character
4. TAB key: map currently existed script files. Now it supports the function of mapping specific type of files or command strings. For instance, enter “run” then press “TAB”, it will display all the commands starts with “run”; enter “runtest “, then press “TAB”, it will display all the files end with “.tc”; enter “runtestsuite”, then press “TAB”, it will display all the files end with “.ts”.

5.2 Manual Testing

In manual testing mode, user can manually start up or stop SIPA, SIPp and send GCC messages to SIPA. Beta2 release provides commands to launch and kill SIPA and SIPp processes (please refer to section 6.5). The GCC messages sent to SIPA should be in script written by user and source in the tool (using Tcl command “*source*”). Beta2 also provides two sample scripts, gcclib.tcl and cs_state.tcl to demonstrate manual testing.

The steps of manual testing are as follows:

1. Launch SIPA

   runsipa

2. Source script

   source gcclib.tcl or
   source cs_state.tcl
   gcclib.tcl and cs_state.tcl define procedures to provision SIPA and send out GCC messages. User also can source self-defined script.

3. Provision SIPA

   runtl1 <TL1 command>

   gcclib.tcl encapsulates some widely used TL1 commands in procedure “*tl1*”. So user can issue “*tl1*” command under test harness prompt to provision SIPA.

4. Launch SIPp

   runsipp
   Examples:
   runsipp basiccall1.xml reg.csv or
   runsipp basiccall1.xml reg.csv –trace_msg

5. Interact with SIPA

   gcclib.tcl provides some procedures, which can be used to send out GCC messages. The messages from SIPA will be displayed on the screen.
   sendconreq
   sendcnstreq [param] – param can be 180 or reinvite (case insensitive). The default value is 180.
   sendconrsp

sendrelreq [param] - param can be bye or cancel (case insensitive). The default value is bye.
sendrelrsp

cs_state.tcl defines a state machine. User needs to send a message to activate it. Then the state machine will send out pre-defined messages according to what it receives.

6. Dump received messages

dumpevt and other commands can be used to print the data of received message on the screen. Following is an example:

```
TestHarness>runsipa
Queue was empty
MP Logging already enabled, ./sip_protocol_errors.log
TestHarness>source gcclib.tcl
TestHarness>tl1
TestHarness>[MESSAGE] (TL1 event: 0 - SUCCESS)
[MESSAGE] (GCC event: 6 - CNSTIND)
[MESSAGE] (GCC event: 6 - CNSTIND)
[MESSAGE] (TL1 event: 0 - SUCCESS)
[MESSAGE] (TL1 event: 0 - SUCCESS)

TestHarness>runsipp basiccall2.xml reg.csv
TestHarness>sendconreq
TestHarness>[MESSAGE] (GCC event: 6 - CNSTIND)

TestHarness>[MESSAGE] (GCC event: 7 - CONCFM)

TestHarness>sendrelreq
TestHarness>[MESSAGE] (GCC event: 9 - RELCFM)
[PROCESS] SIPp(1218) now exiting...

TestHarness>dumpevt 6
CnStEvent Info:
    suConnId: 6 sipInstId: 0x01040000
    evntType: val: 180 pres: 1
```

5.2.1 HUG Test

Test Harness can also support HUG test manually. It is actually possible to run TH on 6-4-0 ACT and 7-0 STBY. It was able to have data replicated from 6-4-0 to 7-0 side. Following are the procedures.

1. Build 6-4-0 TH sandbox
2. Build 7-0 TH sandbox (including your changes)
3. Start TH on 6-4-0 side (siptest)
4. Start 6-4-0 active side (runsipa)
5. Start 7-0 stby side (runsipahug -s /home2/yongz/sandbox/MAIN/TelicaRoot/components/test_harness/bin) in that same TH window)
6. You can now do manual provisioning and make manual calls.

5.3 GDB Integration

Test Harness provides the feature of debugging SIPA by using GDB. Users can open new terminal session, and telnet the Test Harness to get the operation interface of GDB.

Notice: It's highly recommended to use GDB in manual testing mode. When users running test with the command "runtest"or "runtestsuite", the gdb is disabled automatically.

The following example demonstrates how to use GDB in manual mode. Users could use GDB by

following these steps:

1. Telnet to Test Harness

   Test Harness listens on specified TCP ports when it starts up. There are two ways to get the TCP port number. One is that test harness prints out the binding TCP port when starting up.

```
Telnet: please telnet local host at port <61005> for GDB and SIPp display.
TestHarness>
```

   Or, enter the command "*lsport*" to get the details of TCP/UDP port allocation.

   Then users could telnet to test harness by entering the command "*telnet <local-ip> <port>*" **in a new terminal window (called telnet window below)**. Here is the welcome page:

```
Options:
        <F1> GDB interface for primary SIPA (under gdb mode)
        <F2> GDB interface for standby SIPA (under gdb mode)
        <F3> call flow display for 1st SIPp
        <F4> call flow display for 2nd SIPp

Enter your choice:
```

   By entering the "**F1"** key, and then pressing **"<ENTER>**" key in telnet window, the user interface switches to GDB control interface.

2. Start SIPA with GDB mode

   Enter the command "*gdb on*" in test harness console window to enable GDB mode. Then, issue "*runsipa*" command to bring up SIPA by using GDB.

```
TestHarness>gdb on
[SCRIPT]  switch to GDB mode.
TestHarness>runsipa
```

   The users can use GDB to debug SIPA in the telnet window:

```
GNU gdb Red Hat Linux (5.3.90-0.20030710.40rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
 are
welcome to change it and/or distribute copies of it under certain condit
ions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for detai
ls.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthrea
d_db library "/lib/tls/libthread_db.so.1".

(gdb) (gdb)
```

3. Debug SIPA

   Please refer to GDB manual for debug commands.

4. Stop the SIPA

   There are two ways to stop SIPA. One is entering "*q*" or "*quit*" in telnet window; the other is running the command "*stopsipa*" under test harness prompt. We recommend to use the command "*stopsipa*".

Notice: We didn't support Ctrl+C in GDB window, instead, please use the command "suspendsipa" to suspend the running of sipa in harness window.

Note that you could switch to non-GDB mode by entering the command "*gdb off*" in test harness prompt.

5.4 SIPp Message Flow Display

The way users get the SIPp call flow display is quite similar to one of using GDB. Both of them need to open new terminal session, and telnet to harness tool.

First, telnet to test harness (refer to the first step in "section 5.3"), and choose the display window(F3 or F4 key). The default choice is the display of the first SIPp (F3 window). When running the test case by entering "runtest <.tc script>", the telnet window will display the SIPp output.

Steps under manual mode are the same except that the SIPp is brought up by command "runsipp <.xml file> <.csv file>".

Notice: To display SIPp message flow in auto-testing mode, users need to check the parameters of "startupSIPp" in ".tc" script file at first, to ensure that "-bg" option isn't included.

5.5 Multiple SIPp

5.5.1 Auto mode

Users may benefit a lot from multiple SIPp, for instance, testing the state of SIPA when running two different calls simultaneously. Also, for some call flows, which have more than two call legs like call transfer, multiple SIPp support is mandatory.

Test harness supports running two SIPp simultaneously. The scripts "bc_multiplesipp1.tc", "bc_multiplesipp1.xml" and "bc_multiplesipp1.cs" are the example scenario files. In the file "bc_multiplesipp1.tc", function "startupSipp" is called twice. Correspondingly, two SIPp instances are brought up.

5.5.2 Manual mode

To support the function of multiple SIPp in manual mode, harness incurs two additional commands:

One is "runmultisipp". "runmultisipp" shares the parameter format with the command "runsipp". But unlike the command "runsipp", if there is one or more SIPp instances existing, instead of killing the existing SIPp, "runmultisipp" will bring up a new SIPp instance and keep the previous started SIPp alive.

The other is "setinstid". Since GCC is unable to record the instance ID for previous processed calls, you must record them by yourself. "setinstid" needs one or two parameters: call index is mandatory; the followed instance ID is optional (if it's not present, harness will use the instance ID of current processing call). "setinstid" is defined in gcclib.tcl. Before using it, you must include it by command "source gcclib.tcl".

The procedure of running multiple SIPp in manual mode is similar to that of running single SIPp in manual mode, except that you should start SIPp with command "runmultisipp", and you must record

the information of instance ID according to the call index (they're all printed in screen). Of course, when you send GCC primitive notification with "sendconreq" or "sendrelreq", you need a parameter to indicate the call index.

Here is an example:

1. start sipa

```
TestHarness>runsipa
```

2. provision

```
TestHarness>source gcclib.tcl
TestHarness>tl1
```

3. start two SIPp

```
TestHarness>runmultisipp bc_multiplesipp1.xml reg.csv
[SCRIPTS] Call index 0 allocated for this SIPp
TestHarness>runmultisipp bc_multiplesipp1.xml reg.csv
[SCRIPTS] Call index 1 allocated for this SIPp
```

4. 4. begin with the call 1

```
TestHarness>sendconreq 1
[SCRIPTS] call 1: connect ID (7) in trkgrp 9998.
TestHarness>[MESSAGE] (GCC event: 6 - CNSTIND)
[MESSAGE] (GCC event: 7 - CONCFM)
```

5. record the instance ID for call 1

```
TestHarness>setinstid 1
[SCRIPTS] set the instance ID for call 1 to be 17039360
```

6. make the call 0 and record its instance ID

```
TestHarness>sendconreq
[SCRIPTS] call 0: connect ID (6) in trkgrp 9999.
TestHarness>[MESSAGE] (GCC event: 6 - CNSTIND)
[MESSAGE] (GCC event: 7 - CONCFM)

TestHarness>setinstid 0
[SCRIPTS] set the instance ID for call 0 to be 17235968
```

7. release call 1 and 0

```
TestHarness>sendrelreq
[SCRIPTS] call 0: connect ID (6) instance id (17235968)
TestHarness>[PROCESS] SIPp(2697) now exiting...

TestHarness>send[MESSAGE] (GCC event: 9 - RELCFM)

TestHarness>sendrelreq 1
[SCRIPTS] call 1: connect ID (7) instance id (17039360)
TestHarness>[PROCESS] SIPp(2699) now exiting...

TestHarness>[MESSAGE] (GCC event: 9 - RELCFM)
```

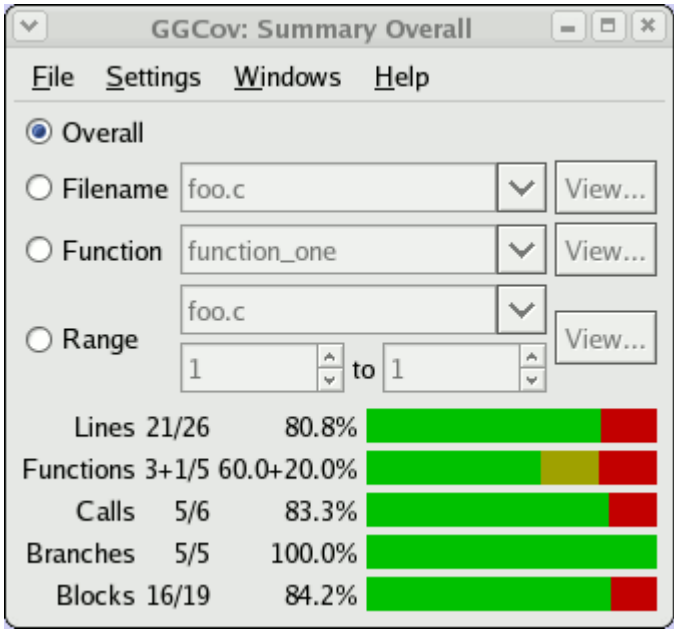
In the later version, we will provide more example scripts to demonstrate how to use multiple SIPp.

5.6 Code Coverage Statistics

1. Overview

Coverage analysis measures how much of the target code is run during a test and is a useful mechanism for evaluating the effectiveness of a system test or benchmark.

Ggcov is a GTK+ GUI for exploring test coverage data produced by C and C++ programs compiled with gcc -fprofile-arcs -ftest-coverage. So it's basically a GUI replacement for the gcov program that comes with gcc. Its latest version is 0.8. You can download it here. To install ggcov on your server, please refer section8.6 for more. The following is a graph presenting code coverage information.



This summary window shows summaries of line, call and branch coverage statistics, as percentages, counts and color coded bars. Green means the part is executed at lease once; Yellow means this part is partly executed; Red means this part is never executed.

You can find more info about its new features and introduction here.

Ggcov will find out all the coverage data files in specified directory and give a summary, which is very useful. GNU gcov can't do like this.

2. Getting started

You can try Ggcov as the following steps

1). cd to "test_harness" directory and launch "*make clean*", then launch "*make all_gcov*"

When using ggcov, you must first compile your program with two special GCC options: -fprofile-arcs -ftest-coverage. For the tool, the makefile provide a tag named "all_gcov", you can compile all the source codes in batch.

2). cd to "test_harness/bin" directory and launch *testsuite targetsuite.ts -gcov*

Run your test suite with the argument "-gcov" to let harness tool know that you want to do code coverage. Code coverage data files will be generated during this time.

3). *th_gcov –x / th_gcov -t*

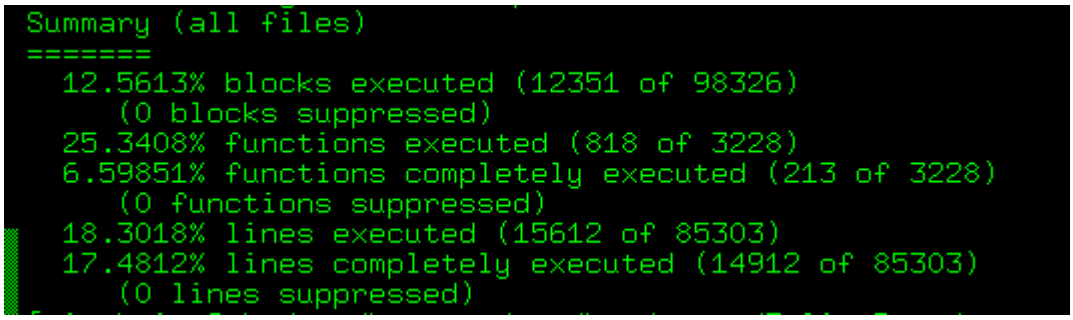
This two commands are shell commands used to simplify the ggcov options. The argument "-x" and "-t" means starting ggcov in X window mode or textual mode.

3. More details

There are two kinds of coverage data files which depend on your gcc version, gcno with higher compiler version, or .bb, .bbg files with older ones. These data files are located in the directory

"/component/signaling/sip2.3/jobs/gnu". After running of the tool (Note: running test suite with the argument "-gcov"), .gcda or .da files will be generated in the same directory, then you can try to launch Ggcov and see the code coverage statistics.

Here is the snapshot of the textual mode result:



This picture show us the textual reports about line, call and branch coverage statistics.

You can launch "man ggcov" to see the manual of Ggcov. Since the synopsis of Ggcov is a little complex, the tool provides a shell command "*th_gcov*" to simplify the usage, just as the quick start shows. Its synopsis is as the following:

1) *th_gcov*

Display the synopsis of how to use this shell file

2) *th_gcov –t*

Display coverage statistics of all SIPA source files in textual mode

3) *th_gcov –x*

Display coverage statistics of all SIPA source files in GUI mode

4) *th_gcov –t relative_source_directory*

Display coverage statistics for the specific directory in textual mode. relative_source_directory is relative directory names in "sip2.3", such as "sdp" or "tools/telica_sipa". You can use the later, for example, to just measure code coverage of the Telica SIPA code (excluding DCL).

5) *th_gcov –x relative_source directory*

Display coverage statistics for the specific directory in GUI mode

Note:

1) You can launch *th_gcov* in harness bin directory only.

2) failover cases can effect the SIPA relative codes' statistical result because SIPA is restarted during these cases running.

5.7 SIPA Failover

Test harness supports SIPA failover feature. User can use command "runsipa", "stopsipa" and "failover" (section 6.5) to test SIPA failover in auto mode as well as in manual mode. The steps to do failover in basic call are as follows:

1. Launch active and standby SIPA.

In manual mode, user can use *runsipa –A* to launch both SIPA.

In auto mode, active SIPA is launched automatically by test harness, user only needs to launch standby SIPA in .tc file using command *runsipa –s* before provisioning SIPA.

2. Establish a call.
3. Kill active SIPA process using command *stopsipa –a* or *stopsipa* or *failover*.
4. Release the call.

NOTE:

1. Command *failover* differs with *stopsipa* in *failover* also launch a new standby SIPA while *stopsipa* doesn't.

5.8 Valgrind Integration

Valgrind is a tool for debugging and profiling Linux programs. The current distribution includes four tools: a memory error detector, a cache (time) profiler, a call-graph profiler, and a heap (space) profiler. with which you can automatically detect many memory management and threading bugs, avoiding hours of frustrating bug-hunting, making your programs more stable. You can also perform detailed profiling, to speed up and reduce memory use of your programs. The most popular tool is the memory checking tool (Memcheck) which can detect many common memory errors. For more information about Valgrind, please refer here.

Normally, you can using Valgrind as the following steps:

1) Compile your program with –g to include debugging information

2) Running your program under Valgrind as following:

valgrind --tool=<tool name> [default=memcheck] <options> <your program args>

In the harness tool, you can test SIPA code as the following steps:

1) Under the harness prompt, launch "*setsipamode valgrind*"

2) Run test suite with argument "-sipa", such as "*runtestsuite rel10.ts -sipa*".

Valgrind has a tool suite which could do many code correctness and profiling checks. Tool has a variable "valgrind" with the default value "--tool=memcheck --leak-check=yes --show-reachable=yes --log-file=valgrind". You can also modify it (use set command under test harness prompt) if you want to try other options. For more information about the usage of valgrind, please refer the user manual document.

After finish the running, a log file named as "valgrind.[pid]" is generated in directory /bin. Here is a slice of the file:

```
**********************************************************************************

==6730== 5,000,100 bytes in 1 blocks are still reachable in loss record 10 of 11

…….

==6730==    by 0x4031DAB: start_thread (in /lib/tls/libpthread-0.60.so)

==6730==    by 0x41939E9: clone (in /lib/tls/libc-2.3.2.so)


==6730== 68 bytes in 1 blocks are definitely lost in loss record 3 of 11

……..
```

```
==6730==    by 0x823DADC: MPlogInit (mv_mplogUtils.c:1175)
==6730==    by 0x818F8E9: main (main.c:950)

==6730== 204 bytes in 3 blocks are possibly lost in loss record 4 of 11
……..
==6730==    by 0x823B293: logInit (logUtils.c:623)
==6730==    by 0x818F6A9: main (main.c:744)

==6730== 5,120,512 bytes in 1 blocks are possibly lost in loss record 11 of 11
…….
==6730==    by 0x823B293: logInit (logUtils.c:623)
==6730==    by 0x818F6A9: main (main.c:744)

==6730== LEAK SUMMARY:
==6730==    definitely lost: 68 bytes in 1 blocks.
==6730==      possibly lost: 5,120,716 bytes in 4 blocks.
==6730==    still reachable: 7,144,974 bytes in 248 blocks.
==6730==         suppressed: 0 bytes in 0 blocks.
*********************************************************************************
```

- The 6730 is the process ID; it is usually unimportant.
- There are several kinds of leaks;

  - "definitely lost": your program is leaking memory—fix it!

  - "possibly lost": your program is leaking memory, unless you're doing funny things with pointers(such as moving them to point to the middle of a heap block).

  - "still reachable": This usually indicates programming sloppiness. Since the block is still pointed at, the programmer could, at least in principle, free it before program exit. Because these are very common and arguably not a problem, Memcheck won't report such blocks unless *--show-reachable=yes* is specified.

  For more description of error messages, you can refer valgrind user manual.

  NOTE:

  1) Programs running under Valgrind run significantly more slowly.

  2) From above example we could see, Valgrind reported 68 bytes memory leak in the logging thread. We double checked the logging library code but could not find the root cause. Don't know if it is a real plexus bug or Valgrind mis-description.

- The stack trace only tells us where the leaked memory was allocated, it cannot tell you why the memory leaked, so you need to check by yourself.

  Here is a synthetic bug to demonstrate Valgrind functions.

  In the source file "signaling/sip2.3/tools/telica_sipa/main.c", a malloc() sentence is called without free():

```
malloc (1000*sizeof(int));
```

Then we get the following test result:

****

==31115== 4,000 bytes in 1 blocks are definitely lost in loss record 7 of 12

==31115== at 0x401A6EE: malloc (vg_replace_malloc.c:149)

==31115== by 0x81918BF: main (main.c:555)

==31115== LEAK SUMMARY:

==31115== definitely lost: 4,068 bytes in 2 blocks.

****

4000 more memory leak is detected. And we can also see that the memory leak happens at the line 555 of the main.c file.

5.9 SIPT Support

Test harness could also support SIPT testing. In order to support SIPT message, you must provision the associated tl1 commands(sipt profile, sipt trunk).

The script must be written according to the specified format. The isup raw data must be encapsulated with {isupraw} and segregated with space for every two bytes. For example, the raw data of IAM is 010021000900020907 011022222222221d02008000, so you should encapsulate it with this format {isupraw}01 00 21 00 09 00 02 09 07 01 10 22 22 22 22 22 1d 02 00 80 00{isupraw}. Any isup raw data can be encapsulated in the xml file and be sent successfully.

Below is an example of SIPp xml file embedded with sipt message.(taken from bc1_sipt.tc).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- SIPp scenario: Basic call Terminated by orignating party. Originating party -->
<scenario name="basiccall1">

<!-- send a INVITE message to remote side -->
<send>
<![CDATA[
INVITE sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>
Call-ID: [call_id]
Cseq:1 INVITE
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
MIME-Version: 1.0
Content-Length: [len]
Content-Type: multipart/mixed; boundary=unique-boundary-1

--unique-boundary-1
Content-Type: application/sdp

v=0
o=- jpeterson 2890844526 2890842807 IN IP4 127.0.0.1
s=-
c=IN IP4 [media_ip]
t=0 0
m=audio 6000 RTP/AVP 0
```

```
a=rtpmap:0 PCMU/8000

--unique-boundary-1
Content-Type: application/ISUP; version=itu; base=itu-t92+
Content-Disposition: signal; handling=required

{isupraw}01 00 21 00 09 00 02 09 07 01 10 22 22 22 22 22 1d 02 00 80 00{isupraw}
--unique-boundary-1

]]>
</send>

<!-- wait for 100 TRYING message -->
<recv response="100"> optional="true"
</recv>

<!-- wait for 180 RING message -->
<recv response="180"> optional="180"
</recv>

<!-- wait for 200 OK message -->
<recv response="200"> optional="200"
</recv>

<!-- send a ACK message -->
<send>
<![CDATA[
ACK sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>[peer_tag_param]
Call-ID: [call_id]
Cseq:1 ACK
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
Content-Length:0
]]>
</send>

<!-- stop 3 seconds -->
<pause milliseconds="3000">
</pause>

<!-- send BYE message -->
<send retrans="500">
<![CDATA[
BYE sip:[field1]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From:sipp <sip:[field0]@[local_ip]:[local_port]>;tag=[call_number]
To:sut <sip:[field1]@[remote_ip]:[remote_port]>[peer_tag_param]
Call-ID: [call_id]
Cseq:2 BYE
Contact: sip:[field0]@[local_ip]:[local_port]
Max-Forwards:70
Subject: Performance Test
MIME-Version: 1.0
Content-Length: [len]
Content-Type: multipart/mixed; boundary=unique-boundary-1

--unique-boundary-1
Content-Type: application/sdp
```

```
v=0
o=- jpeterson 2890844526 2890842807 IN IP4 127.0.0.1
s=-
c=IN IP4 [media_ip]
t=0 0
m=audio 6000 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--unique-boundary-1
Content-Type: application/ISUP; version=itu; base=itu-t92+
Content-Disposition: signal; handling=required

{isupraw}0c 02 00 03 87 90 a2{isupraw}
--unique-boundary-1

]]>
</send>

<!-- wait for 200 OK message -->
<recv response="200">
</recv>


</scenario>
```

5.10 Log Parser

6 Global Variables & Command Reference

Though commands can be entered in both scripts and at the tool command line, for ease of understanding we have divided them according to common usage below. Note that Commands and variables are case sensitive.

6.1 Interactive Commands

`help`

Display help information.

`setscriptdir <directory>`

By default, the test script directory (for both test cases and suites) is in "`test_harness/scripts/sip/`" directory. This command can be used to change it if desired (i.e. it is optional).

`scriptdir`

Display current test script directory.

`logdir`

Display current log and SIPA core directory.

`lstc`

list main scripts of all test cases in current test script directory.

`lsts`

list all test suites in current test script directory.

`runtest <main_script_file>`

runtest command is used to launch one test case or batch of test cases. The "runtest <main script file name>" command is used to run one test case. runtest command can also accept a list of test cases file names. For example, "runtest <main script file name>,<main script file name>,…".

`runtestsuite <test_suite_file> <args>`

runtestsuite makes the execution of multiple test cases easier. User can specify the test case list in test suite file, and run this test suite.

By default, the tool will kill SIPA at the end of test case, and restart it at the beginning of a case. In order to support code coverage, which requests for the SIPA keeping exist during the test suite running, and SIPA can't be killed in force, the tool provide a parameter whose value can be "-sipa" and "-gcov". "-sipa" means SIPA will exist until the entire test suite finishes; "-gcov" means not only keep SIPA existing during test suite, but also exit SIPA graceful by launching TL1 command "sw-toprotn-eqpt".

For test suite file format, please refer to section 3.6

`teststat`

shows if a test case/suite is running or not

`stoptest`

stop current running test case or test suite

`exit`

quit SIP test harness

`displayon`

turns on display of expect controller debugging logs to screen. This command is typically used by test harness tool developers, not users.

`displayoff`

turns off display of expect controller debugging logs to screen.

`setsipamode <gdb/valgrind/normal>`

Use parameter ‘valgrind’ to switch to the mode of starting sipa with valgrind.

Use parameter ‘gdb’ to switch to the mode of starting sipa with gdb. Please be noted here, this option will not take effect if you are using runtest or runtestsuite command. That is to say, you should set sipa to this mode only if you are running manual test.

Default mode is ‘normal’, in which no gdb or valgrind will be involved.

`lsport`

list the ports assigned to current user and their usage description.

`lsproc`

list the current running processes and their process ID.

6.2 Main Script Commands

`tsleep <time in seconds>`

suspend the test for a period of time. This command won’t block the interactive test commands input.

It only affects the running test case.

`runtl1 <{tl1 command}> <[destsipa]>`

This command is a Tcl extension command used to parse TL1 commands. The argument “destsipa” is a tag used to describe which SIPA is provisioned. Its default value is “-b”, which means provision both active and standby SIPA. It also can be “-a” and “-s”, which means only provision active or standby.

Example:

runtl1 { ent-trkgrp::1:::sip:name=pod2,tgprofile=1,sipdstfqdn=10.86.2.3,sipdstfqdnport=5060; }

So far we support 8 provisioning commands, all SIGDBG commands, and 1 commands for failover:

`ED-DNS-SYS`

`ED-SIP-SYS`

`ED-AAG-SYS`

`ENT/ED/DLT-TRKGRP`

`ENT/ED/DLT-PRFL-SIP`

`ENT/ED/DLT-PRFL-SIPT`

`ENT/ED/DLT-SIP-ROUTE`

`ENT/ED/DLT-SIP-TGMAPADDR`

`SIGDBG-DBGLVL/IPSTRC/IPSFLUSH/DUMPRAW/STTRC……`

`SW-TOPROTN-EQPT`

NOTE: The tool just ignore exact checking the SIGDBG input, because SIP developer may create his own parameters. So user must take response himself about the input.

`startupSIPp <options>`

start up SIPp process with its options. For details about options, please refer to SIPp reference document. The following is an example of options.

"$hostip:$sipa_listen_port -p $sipp_local_port -mp $sipp_media_port -sf [getscriptdir]basiccall2.xml -inf [getscriptdir]reg.csv -m 1"

`schedule <time in seconds> <Tcl script>`

Schedule to run the specified Tcl script after the specified number of seconds.

`runcs <call scenario file>`

this command is used to include call scenario script in main script. Use this instead of Tcl's "source" command.

`settesttimer <timeout in seconds>`

set the maxium running time for current test case. If the test case can't be completed within this time, Expect controller will terminate this test case and report test case time out error message.

Note that this command is only used in "State-Based test script". For "Sequential test script", it should not use this command.

`print <log message>`

User can use this command to print log message in test script file.

6.3 Call Scenario Script Variables and Commands

`sipwLiConInd`

This variable will point to the latest received EVTSIPWCONIND primitive, which is SipwLiConEvnt type.

`sipwLiCnStInd`

This variable will point to the latest received EVTSIPWCNSTIND primitive, which is SipwLiCnStEvnt type.

`sipwLiConCfm`

This variable will point to the latest received EVTSIPWCONCFM primitive, which is SipwLiCnCfmEvnt type.

`sipwLiRelInd`

This variable will point to the latest received EVTSIPWRELIND primitive, which is SipwLiRelEvnt type.

`sipwLiRelCfm`

This variable will point to the latest received EVTSIPWRELCFM primitive, which is SipwLiRelCfm type.

`sipwLiSvcInd`

This variable will point to the latest received EVTSIPWSVCIND primitive, which is SipwLiSvcEvnt type.

sipwLiMsgInd

This variable will point to the latest received EVTSIPWMSGIND primitive, which is SipwLiNcEvnt type.

sipwLiMsgCfm

This variable will point to the latest received EVTSIPWMSGCFM primitive, which is SipwLiNcEvnt type.

sipwLiAudCfm

This variable will point to the latest received EVTSIPWAUDCFM primitive, which is SipwLiAudCfmEvnt type.

sipwLiGeoCrtCfm

This variable will point to the latest received EVTSIPWGEOCRTCFM primitive, which is SipwLiGeoCreateCfmEvnt type.

sipwLiGeoAudInd

This variable will point to the latest received EVTSIPWGEOAUDIND primitive, which is SipwLiGeoAudStChgIndEvnt type.

Macros, Structures and Unions

All macros, structures and unions defined in the header files included in test_harness/lmgcc/lmgcc.i are global.

sendConEvnt <event type>

This command is to send connection message to Relay.

event is the pointer to connection data structures.

type is the primitive types

User can use this command as follows:

sendConEvnt $sipwLiConEvnt $EVTSIPWCONRSP

sendCnStEvnt <event type>

This command is to send connection status message to Relay.

event is the pointer to connection status data structures.

type is the primitive types

User can use this command as follows:

sendCnStEvnt $sipwLiCnStEvnt $EVTSIPWCNSTREQ

sendRelEvnt <event type>

This command is to send release message to Relay.

event is the pointer to release data structures.

type is the primitive types

User can use this command as follows:

sendRelEvnt $sipwLiRelEvnt $EVTSIPWRELRSP

sendSvcEvnt <event type>

This command is to send service message to Relay.

event is the pointer to service data structures.

type is the primitive types

User can use this command as follows:

sendSvcEvnt $sipwLiSvcEvnt $EVTSIPWSVCREQ

`sendNcEvnt <event type>`

This command is to send MSG message to Relay.

event is the pointer to Nc data structures.

type is the primitive types

User can use this command as follows:

sendNcEvnt $sipwLiNcEvnt $EVTSIPWMSGREQ

`sendAudReqEvnt <event type>`

This command is to send AUD message to Relay.

event is the pointer to audit data structures.

type is the primitive types

User can use this command as follows:

sendAudReqEvnt $sipwLiAudReqEvnt $EVTSIPWAUDREQ

`sendGeoCreReqEvnt <event type>`

This command is to send geographic request message to Relay.

event is the pointer to geographic request data structures.

type is the primitive types

User can use this command as follows:

sendGeoCreReqEvnt $sipwLiGeoCreReqEvnt $EVTSIPWGEOCREATEREQ

`sendGeoAudReqEvnt <event type>`

This command is to send geographic audit request message to Relay.

event is the pointer to geographic audit request data structures.

type is the primitive types

User can use this command as follows:

sendGeoAudReqEvnt $sipwLiGeoAudReqEvnt $EVTSIPWGEOAUDREQ

`sendMsg <structure event type>`

This command is to send message to Relay.

Structure is data structures name. Currently this system can support SipwLiConEvnt, SipwLiCnStEvnt, SipwLiConCfm, SipwLiRelEvnt, sendSvcEvnt, SipwLiNcEvnt, SipwLiAudReqEvnt, SipwLiGeoCreateEvnt and SipwLiGeoAudStChgReqEvnt.

event is the pointer to data structures.

type is the primitive types

User can use this command as follows:

sendMsg SipwLiRelEvnt $sipwLiRelEvnt $EVTSIPWRELRSP

getGccPara <structure event parameter>

This command is to get the parameter values command will get the field values from event.

Structure is data structures name. Currently this system can support all PSIF-SIP primitives and their nested data structures.

Event is the pointer to a primitive whose one of nested fields' value will be got.

parameter is the field whose value will be got.

User can use this command as follows:

getGccPara SipwLiConEvnt sipwLiConInd evntType.val

setGccPara <structure event {parameter value}>

This command is to set the parameter values of primitives between GCC Emulator and SIPA.

Structure is data structures name. Currently this system can support all PSIF-SIP primitives and their nested data structures.

Event is the pointer to a primitive whose one of nested fields' value will be set.

Parameter is the field to be set to value.

To set character string value to field of character array type, use S()to enclose the string.

To set number string in hexadecimal format to field of character array type, use H() to enclose the string.

User can use this command as follows

setGccPara SipwLiConEvnt sipwLiConRsp “suConnId 5

reqUri.eh.pres 1

reqUri.reqUriVal.val S(11221122)

sipt.encIsupMsg.val H(010008000a)”

loglevel <level>

This command is to set the debug level of LM and GCC emulator. It can also run in interactive mode.

The level value ranges from 0 to 10. 0 will turn off the debug messages in lmgcc.dbg. 10 will print the detail logs in lmgcc.dbg. Logs will have increasing detail as it is increased from 1 to 10.

User can use this command as follows:

loglevel 5

errlevel <level>

This command is to set the error log level of LM and GCC emulator.

Its level value ranges from 0 to 10. 0 will trun off the error log messages in lmgcc.err. 10 will print the detail errors information in lmgcc.err. Errors will be more and more detail from 1 to 10

User can use this command as follows:

errlevel 10

assign <dstStruct dstEvnt srcStruct srcEvnt>

This command is to set destination event values from source event values.

dstStruct is destination data structures name. Currently this system can support SipwLiConEvnt, SipwLiCnStEvnt, SipwLiConCfm, SipwLiRelEvnt and their nested data structures.

dstEvnt is the pointer to primitives whose parameters will be assigned

srcStruct is source data structures name. Currently this system can support SipwLiConEvnt, SipwLiCnStEvnt, SipwLiConCfm, SipwLiRelEvnt and their nested data structures.

srcEvnt is the pointer to primitives whose parameters is to assign to dsrEvnt if srcEvnt and dstEvnt have same member.

User can use this command as follows:

assign SipwLiRelEvnt $sipwLiRelRsp SipwLiConEvnt $sipwLiConInd

dummySdp

This command is to set basic values to SDP structure.

This command returns a SDP structure, which can be assigned to sdp field in an event. This SDP has only one media, AUDIO and one VOIP codec, PCMU. Its clock rate is 8000. Its media IP and port are 127.0.0.1 and 0.

User can use this command as follows:

SetGccPara SipwLiConEvnt sipwLiConRsp “

suConnId 6

sdp [dummySdp]”

regevent <event_type> <Tcl script>

register expect controller event handler. If user adopts “state-based” scripts, the call event handler should be provided. The harness user can customize any actions with Tcl script. Expect controller would run the corresponding Tcl script once an event comes in.

Currently, we use following event types. But from users’ perspective, they only need to use “CCEVNT”.

1 – GCC Event. This is defined as “CCEVNT” variable.

2 – LM Alarm Event

3 – LM Provision Event

deregevent <event type>

de-register expect controller event handler. The harness user can de-register expect controller event handler simply by running this command with event type. From users’ perspective, this commands is rarely used.

listevent

list all registered expect controller event handler. From users’ perspective, this commands is rarely used.

waitforcond <time out in seconds> <event type> <condition>

This is used in “sequential test script”. Typical “event type” is GCC event. “condition” can be written in Tcl script. The “time out” argument specifies how long it will take to wait for this event. After the event comes in, Expect controller will evaluate the specified “condition” to see whether it is TRUE. If it’s TRUE, the test would continue to next command. Otherwise, the test case would terminate with failure result.

Currently, we define following event types. From users’ perspective, they only need to use “CCEVNT” type to register a call control event handler.

1 – GCC Event. This is defined as “CCEVNT” variable.

2 – LM Alarm Event

3 – LM Provision Event

`quittest <return value>`

This command is used to quit current test case with specified return value. The return value could be “TC_SUCCESS” or “TC_FAIL”. “TC_SUCCESS” indicates current test cause is successful, and “TC_FAIL” means current test case fails. Note that this command is used in “State-based” call scenario script.

`tclassert <assertion expression>`

This command is just like the assert mechanism in C and C++. it is used to throw a explicit assertion for fail specific test case. If the expression is false, a explicit error will be raised to Tcl execution, then print out the error expression and the line number where the assertion error happens in script, and the case will be failed.

`enassert`

This command is used to enable Tcl assertion. The default is “enabled”

`disassert`

This command is used to disable Tcl assertion.

`Failover`

This command is used to make two SIPA failover. It kills active SIPA process and launch a standby SIPA process.

6.4 Tcl Commands

Refer to http://mobility.ih.lucent.com/~rainbow/tcl/man/tcl_man.html.

6.5 Commands for Manual Testing

`runsipa <arg>`

This command is used to launch active and standby SIPA process. The “arg” can be “-A”, “-a”(default) and “-s”.

“-A”: launch active and standby SIPA processes if no SIPA exists.

“-a”: launch active SIPA process if no active SIPA exists.

“-s”: launch standby SIPA process if an active SIPA exists and no standby SIPA exists.

`stopsipa <arg>`

This command is used to kill existing SIPA process. The “arg” can be “-A”, “-a”(default) and “-s”.

“-A”: Kill active and standby SIPA processes if they exist.

“-a”: Kill active SIPA process if it exists.

“-s”: Kill standby SIPA process if it exists.

`suspendsipa`

This command is only valid in gdb mode. When the SIPA started by GDB is running, through entering the command “suspendsipa” in harness, the harness will stop the running of SIPA. At this time, you may input in telnet window for gdb usage.

`runsipp <XML file> [other arguments]`

This command is used to launch SIPp process.

XML file is the SIPp script file.

Other argument is optional or CSV file and other SIPp arguments. But local IP and port are designated by the tool, and user should not provide them.

This SIPp would exit after processing 1 call. That is, it’s launched with argument “-m 1”.

User can use this command as follows:

runsipp basiccall1.xml

runsipp basiccall1.xml reg.csv

runsipp basiccall1.xml reg.csv -trace_msg

`runmultisipp <XML file> [other arguments]`

This command is used to launch SIPp process, unlike the command “runsipp”, it won’t kill all previous existing SIPp instances before start a new one.

XML file is the SIPp script file.

Other argument is optional or CSV file and other SIPp arguments. But local IP and port are designated by the tool, and user should not provide them.

This SIPp would exit after processing 1 call. That is, it’s launched with argument “-m 1”.

User can use this command as follows:

runmultisipp basiccall1.xml

runmultisipp basiccall1.xml reg.csv

runmultisipp basiccall1.xml reg.csv -trace_msg

`stopsipp`

This command is used to kill all existing SIPp processes.

`stopprocs`

This command is used to kill all existing SIPA and SIPp processes.

`dumpevt <event number>`

This command is used to display the data of received messages on the screen.

`dump_conevt <struct>`

This command is used to display the data of SipwLiConEvnt structure on the screen.

dump_cnstevt <struct>

This command is used to display the data of SipwLiCnStEvnt structure on the screen.

dump_concfmevt <struct>

This command is used to display the data of SipwLiCnCfmEvnt structure on the screen.

dump_relevt <struct>

This command is used to display the data of SipwLiRelEvnt structure on the screen.

dump_svcevt <struct>

This command is used to display the data of SipwLiSvcEvnt structure on the screen.

dump_ncevt <struct>

This command is used to display the data of SipwLiNcEvnt structure on the screen.

dump_audreqevt <struct>

This command is used to display the data of SipwLiAudReqEvnt structure on the screen.

dump_audcfmevt <struct>

This command is used to display the data of SipwLiAudCfmEvnt structure on the screen.

dump_geocrtevt <struct>

This command is used to display the data of SipwLiGeoCreateEvnt structure on the screen.

dump_geocrtcfmevt <struct>

This command is used to display the data of SipwLiGeoCreateCfmEvnt structure on the screen.

dump_geoaudreqevt <struct>

This command is used to display the data of SipwLiGeoAudStChgReqEvnt structure on the screen.

dump_geoaudindevt <struct>

This command is used to display the data of SipwLiGeoAudStChgIndEvnt structure on the screen.

7 Example Test Scripts

For the test cases description, please refer to the file "_scriptindex.txt" in the script directory, which is `TelicaRoot/components/test_harness/scripts/sip/`.

8 Tool Server Setup

This section is intended for the administrators of the tool server in each development location. Tool users do not need to concern themselves with it.

8.1 Tool server software packages

The SIP test harness is based on Linux, we recommend to adopt Fedora Core 3 or above release version. The tool build/test server needs to install software or packages below:

Notice: We suggest to compile the harness code with gcc-3-2-3; In higher gcc version, for instance, gcc-3-4-6, some compile warnings may be arisen and cause the failure of compile.

Software	**Package name (recommended version)**
Tcl-8.3 or above	Tcl-8.3.5-92.i386.rpm (Internal package in FC3)
Tcl-devel-8.3.5-92	Tcl-devel-8.3.5-92.i386.rpm
Tclx-8.3	Tclx-8.3-92.i386.rpm
Tclx-devel-8.3.5	Tclx-8.3.5-4.i386.rpm
Swig-1.3	Swig-1.3.24-2.i386.rpm
Ggcov-0.8	ggcov-fc3-0.8-1.i386.rpm

8.2 Compile environment

All releases are compiled successfully by gcc-3.2.3-20 in Linux kernel 2.4 and 2.6

8.3 IP Address configuration

Currently, the tool server requires only one IP address. Port allocation for multiple users (eg for SIPp, SIPA, etc) is managed by the tool.

8.4 Configure NFS

NFS is short for Network File System. It's a mechanism like remote file share in windows system. User can visit remote server's files through "mount" the destination file point to local directory. In SIP Test Harness, we apply this technique to fetch the resource code of SIPA and harness from build server, then compile and run the harness in local tool/build server. Users should configure as following steps:

1) Install NFS server software in build server

   We recommend to use nfs-utils-1.0.5-3 package or above version. If there already has nfs server been installed, you can jump directly to next step.

2) Configure NFS server (in build server)

   a) Create a new account in build server (or use an existing one)

      Users in tool build/test server will visit share directory with access right of this account.

   b) Add the line below at the end of file "/etc/exports" (If this file doesn't exist, please make sure NFS server has been installed correctly)

<share_directory> <remote_ip>(rw,sync,root_squash,all_squash,anonuid=<uid>,anongid=<gid>)

<remote_ip>: The IP of tool build/test server

<share_directory>: code directory in build server

<uid>: The UID of the user in step a)

<gid>: The GID of the user in step a)

Example:

/home4/tmp/
135.252.142.150(rw,sync,root_squash,all_squash,anonuid=600,anongid=2000)

c) Restart nfs server

Go to /etc/init.d/, and run “nfs restart”, nfs is the NFS Server’s application name. Different NFS servers may have different names.

3) Configure NFS Client (in tool build/test server)

To visit shared directory in build server temporarily, please enter the command:

mount –t nfs –o rw <nfs-server-ip>:<directory-path> <local-path>

<directory-path>: the value <share_directory> in 2) -b)

<local-path>: an existing directory, end point for mount, users can visit share directory by visiting this directory.

Example:

Mount –t nfs –o rw 135.252.142.150:/home/tmp /mnt/share

Through these three steps, users can visit /mnt/share to visit the code in build server. Then, he can also compile the files and run the application locally.

8.5 DNS support

Since in the Beta-1 version the harness has supported the DNS address. Users can enter TL1 command to set DNS server and automatically translate between DNS address and IP address.

For the configuration, harness users need the read/write privilege for the file /etc/resolv.conf. Therefore, root user needs to open the access right by “chmod” command (for instance, chmod 666 /etc/resolv.conf). Then, The corresponding TL1 command may visit and modify this file.

In current Beta-1 version, we provide a local DNS server for test usage, and the address is 135.252.129.38. The DNS server will parse the domain names with postfix of “harness.lucent.com”.

For how to configure the DNS address by using TL1 command, please refer to example user case – dns.tc.

8.6 Ggcov install

1. Download Ggcov here

2. Installation

If you download the .tar.gz file, you can install it as follow:

1). *tar –zxvf targetfile.tar.gz*

This step will create a directory with the same name. There is a “README” file in this directory, it is

very helpful to read it before you install the software.

2). Run *"./configure"*to create the makefile

3). "*make"* to compile the file

4). "*make install*" to install the software

If you download the rpm package, you can install it as follow:

rpm -ivh your-package.rpm

-i means "install" this package

-v means adding some output information

-h means hashing a progress bar during the installation

8.7 Valgrind Installation

Please refer to follow link for how to install Valgrind latest release.

http://www.valgrind.org/downloads/current.html#current

9 FAQ

9.1 Why can't get core dump file generated by SIPA [how to make it??]

use "*ulimit -a*" to check whether the environment configuration is right.

If not, please use "*ulimit –c unlimited*" to enable core dump generation.

9.2 Why add "tsleep 5" at the beginning of "main script"

wait for SIPA to startup completely, and then start to prevision SIPA.

9.3 Why add "tsleep 2" at the end of "main script"

wait for SIPp to terminate completely. Otherwise, SIPp will be killed forcedly by SIP Test Harness, and current test case will fail.

9.4 Why fail to provision SIPA

1). use "*ps –ef|grep sipa*" to see whether there is unexpected SIPA process. If it exists, please kill it.

2). use "*ps –ef|grep sipp*" to see whether there is unexpected SIPp process. If it exists, please kill it.

3). use "*tail –f sipa.out*" to see whether SIPA startups normally

9.5 What should be done if siptest gets cored

1). use "*ps –ef|grep sipa*" to see whether there is unexpected SIPA process. If it exists, please kill it.

2). use "*ps –ef|grep sipp*" to see whether there is unexpected SIPp process. If it exists, please kill it.

9.6 Why can't get GDB display on telnet window

use "gdb" to see whether siptest is running in "GDB ON" mode. If not, please use "gdb on" to enable GDB mode

9.7 How to recall previous commands

The harness accepts shell-like hot keys:

* UP/DOWN key : get the history commands;

* LEFT/RIGHT key: move the cursor left or right

* BACKSPACE key: delete previous inputted character

* TAB key: map currently existed script files. Now it supports the function of mapping specific type of files or command strings. For instance, enter "run" then press "TAB", it will display all the commands starts with "run"; enter "runtest ", then press "TAB", it will display all the files end with ".tc"; enter "runtestsuite", then press "TAB", it will display all the files end with ".ts".

9.8 How to build SIPA of debug version

1). Use 'make clean' in test_harness directory

2). Use 'make all_debug' in test_harness directory

3). Modify nbase.ini in bin directory – changing nbb_trace_level = xx

9.9 How to update SIPTH according to sip feature

I will give an example to describe how to update TH code according to FID 2373. You can refer to bug 61632 for detailed code changes. It will add 4 parameters for prfl-sip.

1). Update the function LmSipPrflDataEntSnd and LmSipPrflDataEdSnd in lm.c under /test_harness/lmgcc/, fill the associated parameters in the data structure PrflSip, you can refer to function clPrflSipSipCfg in signaling/lm/clam/cl_prflsip2.c.

2). Update tl1_glbvar.tcl and tl1_utils.tcl under test_harness/expect_controller/tcl, add the data members in PrflSip in tl1_glbvar.tcl and add the new parameters to the t1l commands ent/ed/dlt-prfl-sip in tl1_utils.tcl.

3). Update gcc.tcl under test_harness/expect_controller/tcl if the parameters are needed in associated GCC events. Add the new parameters in each related gcc events in gcc.tcl.

4). Make clean and make under test_harness.

10 Glossary

Acronym	Description
GCC	Generic Call Control
LM	Layer Management
SIGDBG	Signaling Debug
SIPA	SIP Agent
SWIG	Simplified Wrapper and Interface Generator
TL1	Transaction Language One

11 References

11.1 Lucent Technologies Documentation

1. Plexus Test Harness Architecture Document (Plexus 79-5027)
2. Plexus PSIF-SIP Software Design (Plexus 79-3242)
3. Plexus SIP Test Harness Architecture

11.2 Third-party Specifications

1. Tcl manual pages: http://mobility.ih.lucent.com/~rainbow/tcl/man/tcl_man.html.
2. DCL N-BASE Version 2.0 Porting Guide
3. GCC Manuals
4. SIPp Reference documentation v1.1