Using SIPp to run Performance Tests

References

- SIPp Project Page
- SIPStone Benchmarks
- Columbia University paper on SIPStone benchmarks
- SER test scenario
- SIPP Users Mailing List Archive
- SIP Call Flows
- SIPSAK - SIP Test Tool
- SIP Parameters

Objectives and ToDo's

I tried to create a simple test setup using SIPp to test the call processing performance of the sipX SIP proxy server version 2.8.1.

This is what I thought:

1. Adapt the SIPp cmd lines below for this scenario
2. Find out how to configure sipX (create user / passwd if necessary)
3. Find out the difference between authenticated and un-authenticated calls, so that we can test both

The current un-authenticated call rate as stated by the sipX team is 15 cps. Comparing this number with SIP proxy servers such as SER suggest that this is rather low. However, sipX is a full IP PBX with all the features. For authentication purposes it consists of two proxies: the SIP proxy and the auth proxy. Still, is it possible that just because media-server, config-server as well as the postgres database, Apache and JBoss all run on the same server makes such a difference?

Test Setup and Hardware

```
SIPp UAC ------------ sipX proxy ---------- SIPp UAS
```

sipX test scenario: (to be developed)

Receiver:
./sipp -sn uas -d 0 -p 5060 -i receiver_ip -rsa sipx_ip:5060 -trace_msg
Transmitter:
./sipp -sn uac -nr -r 1 -rp 1000 -d 0 -l 1 -p 5060 -trace_msg -i sender_ip -rsa sipx_ip:5060 receiver_ip:5060

This basic setup should work **without any configuration of sipX**, but it does not for the reasons pointed out above. It is possible to use the *transmitter* UAC scenario and place calls to a regular phone registered with sipX. The phone (tested with a Polycom SoundPoint IP501) shows the correct user agent behaviour and allows for correct call completion. The reverse, however, does not work: Using a phone (tested with Pingtel xpressa URL calling) to place calls to the SIPp UAS receiver does not work. The phone never receives the *180-Ringing* response, even though it is sent by SIPp UAS.

Notes

Configuration of sipX:

Create a user *test* with the password (PIN) *123*
sipX might only accept calls from authenticated users - needs to be verified.

In order for authentication to work it seems that the standard UAC scenario XML file needs to be extended as follows (copied from the example in the manual):

```
<recv response="407" auth="true">
</recv>

<send>
  <![CDATA[

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

  ]]>
</send>

<send retrans="500">
  <![CDATA[

    INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port]
    From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
    To: sut <sip:[service]@[remote_ip]:[remote_port]>
    Call-ID: [call_id]
    CSeq: 2 INVITE
    Contact: sip:sipp@[local_ip]:[local_port]
    [authentication username=foouser]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP4 127.0.0.1
    s=-
    t=0 0
    c=IN IP4 [media_ip]
    m=audio [media_port] RTP/AVP 0
    a=rtpmap:0 PCMU/8000

  ]]>
</send>
```

Built in scenarios are part of the executable code. In order to dump an existing scenario into a file use the following command:

```
./sipp -sd uac >uac.xml
./sipp -sd uas >uas.xml
```

External scenarios can be loaded using the -sf option

```
./sipp -sf file.xml .....
```

Example SER test scenario: taken from here

- SIPp UAS & UAC run on a laptop (PentiumM 1.8Ghz)
- ser runs on an Athlon64 3800+
- The 2 machines are connected by gigabit ethernet.

```
SIPp cmd lines:
./sipp -sn uas -d 0 -p 5060 -l laptop_ip -rsa ser_ip:5060 laptop_ip:5080
./sipp -r 200 -rp 100 -d 0 -sn uac -p 5080 -l laptop_ip -rsa ser_ip:5060 laptop_ip:5060
(sipp tries to send ~ 2000 cps: 200 calls each 100ms)
```

Receiver:

```
-sn uas default SipStone scenario
-d 0 call duration is default (0ms)
-p 5060 local port number to listen at
-l laptop_ip
-rsa ser_ip:5060 set remote sending address for sending the message
laptop_ip:5080 remote host and port number
```

Transmitter:

```
-r 200 call rate in calls per second (200 cps)
-rp 100 rate period in ms for the call rate
-d 0 call duration is default (0ms)
-sn uac default SipStone scenario
-p 5080 local port number to transmit at
-l laptop_ip
-rsa ser_ip:5060 set remote sending address for sending the message
laptop_ip:5060 remote host and port number
```

Note: the option *-l laptop_ip* for both receiver and transmitter seems to be an error. The option *-i laptop_ip* could be meant, however, the default value already provides the correct value.

SIPp Version:

I am using *sipp.1.1rc2*. download here

Reference: http://sipp.sourceforge.net/doc1.1/reference.html#installing

Compiling:

```
gunzip sipp.1.1rc2.tar.gz
tar -xf sipp.1.1rc2.tar
cd sipp
make
```

make clean (use this command if you built sipp without OpenSSL first)

```
make ossl (compile with the authentication option enabled, requires OpenSSL)
```

Open Ports:

```
nmap -sT -O host
netstat -l
```

Scenario Definition:

- Basic SipStone UAC XML scenario file
- How to create your own XML scenario files

Invite as per the default SipStone UAC scenario:

```
INVITE sip:service@remote_ip:remote_port SIP/2.0
Via: SIP/2.0/transport local_ip:local_port
From: sipp <sip:sipp@local_ip:local_port>;tag=call_number
To: sut <sip:service@remote_ip:remote_port>
Call-ID: call_id
Cseq: 1 INVITE
Contact: sip:sipp@local_ip:local_port
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: len
```

v=0

```
o=user1 53655765 2353687637 IN IP4 127.0.0.1
s=-
t=0 0
c=IN IP4 media_ip
m=audio media_port RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Note: The identifier service can be passed on the command line as *-s service_name*.

I assume that the user with the User ID *service_name* has to exist in sipX.

Authentication:

Reference: http://sipp.sourceforge.net/doc1.1/reference.html#authentication

Example Registration (w/Auth) test scenario:

Here's an example test scenario that can be used to test the sipx proxy and registrar. The scenario file *register_client.xml* simulates an authenticated registration and the external CSV file *register_client.csv* is used to substitute values for username, domain, and authentication credentials. The following command can be used to run this scenario:

```
sipp -sf register_client.xml -inf register_client.csv -r 10 -trace_err -trace_stat -nd -fd 1 -i sender_ip sipx_ip
```

{{Box File| SIPp register scenario register_client.xml|

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

<scenario name="register_client">
  <send retrans="500">
    <![CDATA[

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: <sip:[field0]@[field1]>;tag=[call_number]
      To: <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      Max-Forwards: 5
      Expires: 1800
      User-Agent: SIPp/Linux
      Content-Length: 0

    ]]>
  </send>

  <recv response="401" auth="true">
  </recv>

  <send retrans="500">
    <![CDATA[

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: <sip:[field0]@[field1]>;tag=[call_number]
      To: <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: 2 REGISTER
      Contact: sip:[field0]@[local_ip]:[local_port]
      [field2]
      Max-Forwards: 5
      Expires: 1800
      User-Agent: SIPp/Linux
      Content-Length: 0

    ]]>
  </send>

  <recv response="200">
  </recv>
</scenario>
```

}}

{{Box File| SIPp register scenario external value injection CSV register_client.csv|

```
SEQUENTIAL
user0;example.com;[authentication username=user0 password=123];
user1;example.com;[authentication username=user1 password=456];
```

}}

sipX Log Viewer

sip log files in sipX:
mediaserver.log
sds.log
sipauthproxy.log
sipproxy.log
sipregistrar.log
sipstatus.log

syslogviewer is a filter that removes the escaping for newlines and otherwise makes the log files more readable. Use it in a pipe:

```
grep <call-id> /var/log/sipxpbx/sipproxy.log | syslogviewer | less
```

syslog2siptrace is a translator that is used to convert log files into a the xml format used by sipviewer, which displays calls in a very readable ladder diagram.

```
grep <call-id> /var/log/sipxpbx/sipproxy.log | syslog2siptrace > log.xml
sipviewer log.xml
```

syslogviewer reformats sip logs a bit better, replacing '\r\n' with real carriage returns

```
USAGE:
syslogviewer < sip log file
```

syslog2siptrace turns sip log into XML. Only consumers today of XML is sipviewer

```
USAGE:
syslog2siptrace < sip.log > /tmp/sip.log.xml
```

sipviewer is a graphical SIP message viewer

```
USAGE:
syslog2siptrace < sip.log > /tmp/sip.log.xml
sipviewer /tmp/sip.log.xml
```

SIPp Usage

```
sipp remote_host.remote_port options
```

Available options:

```
-v : Display version and copyright information.
```

-bg : Launch SIPp in background mode.

```
-p local_port : Set the local port number. Default is a
random free port chosen by the system.
```

-i local_ip : Set the local IP address for 'Contact:',

```
'Via:', and 'From:' headers. Default is
primary host IP address.
```

-inf file_name : Inject values from an external CSV file during calls

into the scenarios.
First line of this file say whether the data is
to be read in sequence (SEQUENTIAL) or random
(RANDOM) order.
Each line corresponds to one call and has one or
more ';' delimited data fields. Those fields can be
referred as field0, field1, ... in the xml
scenario file.

-d duration : Controls the length (in milliseconds) of

calls. More precisely, this controls
the duration of 'pause' instructions in
the scenario, if they do not have a
'milliseconds' section. Default value is 0.

-r rate (cps) : Set the call rate (in calls per seconds).

This value can be changed during test by
pressing '+','_','*' or '/'. Default is 10.
pressing '+' key to increase call rate by 1,
pressing '-' key to decrease call rate by 1,
pressing '*' key to increase call rate by 10,
pressing '/' key to decrease call rate by 10.
If the -rp option is used, the call rate is
calculated with the period in ms given
by the user.

-rp period (ms) : Specify the rate period in milliseconds for the call

rate.
Default is 1 second.
This allows you to have n calls every m milliseconds
(by using -r n -rp m).
Example: -r 7 -rp 2000 h1. > 7 calls every 2 seconds.

-max_socket max : Set the max number of sockets to open simultaneously.

This option is significant if you use one socket
per call. Once this limit is reached, traffic is
distributed over the sockets already opened.
Default value is 50000.

-base_cseq n : Start value of cseq for each call.

-sf filename : Loads an alternate xml scenario file.
To learn more about XML scenario syntax,
use the -sd option to dump embedded
scenarios. They contain all the necessary
help.

-sn name : Use a default scenario (embedded in

the sipp executable). If this option is omitted,
the Standard SipStone UAC scenario is loaded.
Available values in this version:

'uac' : Standard SipStone UAC (default).

'uas' : Simple UAS responder.
'regexp' : Standard SipStone UAC - with
regexp and variables.
'branchc' : Branching and conditional
branching in scenarios - client.
'branchs' : Branching and conditional
branching in scenarios - server.

Default 3pcc scanerios (see -3pcc option):

'3pcc-C-A' : Controller A side (must be started
after all other 3pcc scenarios)
'3pcc-C-B' : Controller B side.
'3pcc-A' : A side.
'3pcc-B' : B side.

-sd name : Dumps a default scenario (embeded in

the sipp executable)

-t u1 : Set the transport mode:

u1: UDP with one socket (default),
un: UDP with one socket per call,
t1: TCP with one socket,
tn: TCP with one socket per call,

It appears that you installed the

sippcomp.so plugin. 2 additionnal
transport modes are available:

c1: u1 + compression,

cn: un + compression.

-trace_msg : Displays sent and received SIP messages in

<scenario file name>_<ppid>_messages.log

-trace_screen : Dump statistic screens in the

<scenario_name>_<ppid>_screens.log file when
quitting SIPp. Useful to get a final status report
in background mode (-bg option).

-trace_timeout : Displays call ids for calls with timeouts in

<scenario file name>_<ppid>_timeout.log

-trace_stat : Dumps all statistics in <scenario_name>_<ppid>.csv

file. Use the '-h stat' option for a detailed
description of the statistics file content.

-stf file_name : Set the file name to use to dump statistics

-trace_err : Trace all unexpected messages in
<scenario file name>_<ppid>_errors.log.

-trace_logs : Allow tracing of <log> actions in

<scenario file name>_<ppid>_logs.log.

-trace_rtt : Allow tracing of all response times in

<scenario file name>_<ppid>_rtt.csv.

-rtt_freq freq : freq is mandatory. Dump response times

every freq calls in the log file defined
by -trace_rtt. Default value is 200.

-s service_name : Set the username part of the resquest URI.

Default is 'service'.

-ap password : Set the password for authentication challenges.

Default is 'password'

-tls_cert name : Set the name for TLS Certificate file.

Default is 'cacert.pem'

-tls_key name : Set the name for TLS Private Key file.

Default is 'cakey.pem'

-f frequency : Set the statistics report frequency on screen

(in seconds). Default is 1.

-fd frequency : Set the statistics dump log report frequency

(in seconds). Default is 60.

-l calls_limit : Set the maximum number of simultaneous

calls. Once this limit is reached, traffic
is decreased until the number of open calls
goes down. Default: (3 * call_duration (s) * rate).

-m calls : Stop the test and exit when 'calls' calls are

processed.

-mp local_port : Set the local RTP echo port number. Default

is none. RTP/UDP packets received on that
port are echoed to their sender.
RTP/UDP packets coming on this port + 2
are also echoed to their sender (used for
sound and video echo).

-mi local_rtp_ip : Set the local IP address for RTP echo.

-3pcc ip:port : Launch the tool in 3pcc mode ("Third Party
call control"). The passed ip address
is depending on the 3PCC role.

- When the first twin command is 'sendCmd' then
  this is the address of the remote twin socket.
  Example: 3PCC-C-A scenario.
- When the first twin command is 'recvCmd' then
  this is the address of the local twin socket.
  Example: 3PCC-C-B scenario.

-nr : Disable retransmission in UDP mode.

-nd : No Default. Disable all default behavior of SIPp
which are the following:

- On UDP retransmission timeout, abort the call by
  sending a BYE or a CANCEL
- On unexpected BYE send a 200 OK and close the call
- On unexpected CANCEL send a 200 OK and close the call
- On unexpected PING send a 200 OK and continue the call
- On any other unexpected message, abort the call by
  sending a BYE or a CANCEL

-rsa host:port : Set the remote sending address to host:port.

```
for sending the messages.
```

Signal handling:

```
SIPp can be controlled using posix signals. The following signals
are handled:
USR1: Similar to press 'q' keyboard key. It triggers a soft exit
of SIPp. No more new calls are placed and all ongoing calls
are finished before SIPp exits.
Example: kill -SIGUSR1 732
USR2: Triggers a dump of all statistics screens in
<scenario_name>_<ppid>_screens.log file. Especially useful
in background mode to know what the current status is.
Example: kill -SIGUSR2 732
```

Exit code:

```
Upon exit (on fatal error or when the number of asked calls (-m
option) is reached, sipp exits with one of the following exit
code:
0: All calls were successful
1: At least one call failed
97: exit on internal command. Calls may have been processed
99: Normal exit without calls processed
-1: Fatal error
```

Example:

```
Run sipp with embedded server (uas) scenario:
./sipp -sn uas
On the same host, run sipp with embedded client (uac) scenario
./sipp -sn uac 127.0.0.1
```

Ethereal ==

Ethereal needs to be started with root priviledges in order to get access to the network interface. This only works if the *DISPLAY* variable is set correctly.

```
export DISPLAY=:0.0
```

</html>