SIPp

Reference documentation

by Richard GAYRAUD [initial code], Olivier JACQUES [code/documentation]

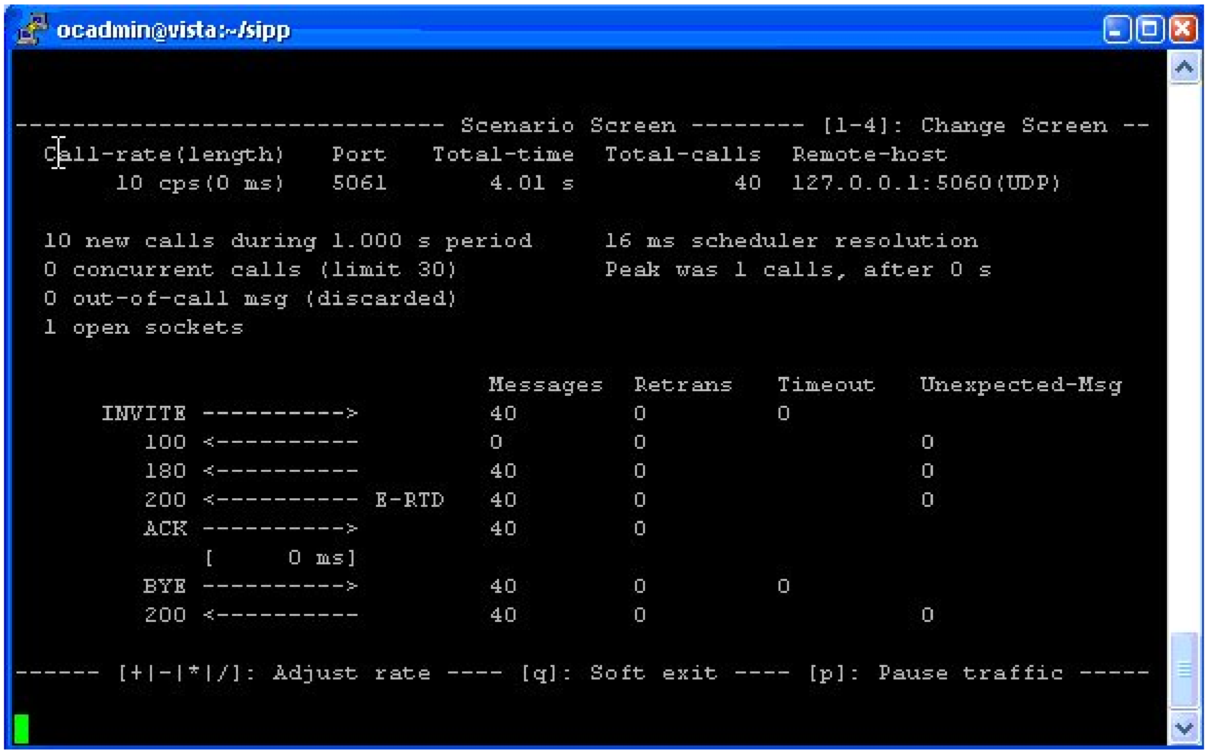
1. Foreword

SIPp is a performance testing tool for the SIP protocol. It includes a few basic SipStone user agent scenarios (UAC and UAS) and establishes and releases multiple calls with the INVITE and BYE methods. It can also reads XML scenario files describing any performance testing configuration. It features the dynamic display of statistics about running tests (call rate, round trip delay, and message statistics), periodic CSV statistics dumps, TCP and UDP over multiple sockets or multiplexed with retransmission management, regular expressions and variables in scenario files, and dynamically adjustable call rates.

SIPp can be used to test many real SIP equipements like SIP proxies, B2BUAs, SIP media servers, SIP/x gateways, SIP PBX, ... It is also very useful to emulate thousands of user agents calling your SIP system.

Want to see it?

Here is a screenshot



```
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
      10 cps(0 ms)    5061       4.01 s           40  127.0.0.1:5060(UDP)

  10 new calls during 1.000 s period      16 ms scheduler resolution
  0 concurrent calls (limit 30)           Peak was 1 calls, after 0 s
  0 out-of-call msg (discarded)
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->         40        0         0
         100 <----------         0         0                   0
         180 <----------         40        0                   0
         200 <---------- E-RTD   40        0                   0
         ACK ---------->         40        0
             [      0 ms]
         BYE ---------->         40        0         0
         200 <----------         40        0                   0

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

SIPp screenshot

And here is a video (Windows Media Player 9 codec or above required) of SIPp in action:

sipp-01.wmv

2. Installation

2.1. Getting SIPp

SIPp is released under the GNU GPL license. All the terms of the license apply. It is provided to the SIP community by Hewlett-Packard engineers in hope it can be useful.

We receive some support from our company to work on this tool freely, but **HP does not provide any support nor warranty concerning SIPp.**

2.2. Stable release

Like many other "open source" projects, there are two versions of SIPp: a stable and unstable release. Stable release: before being labelled as "stable", a SIPp release is thoroughly tested.

So you can be confident that all mentionned features will work :)

Note:
Use the stable release for your everyday use and if you are not blocked by a specific feature present in the "unstable release" (see below).

SIPp stable download page

2.3. Unstable release

Unstable release: every new features and bug fixes are checked in SIPp's CVS repository as soon as they are available. Every night, an automatic extraction is done and the source code of this release is made available.

Note:
Use the unstable release if you absolutely need a bug fix or a feature that is not in the stable release.

SIPp "unstable" download page

2.4. Available platforms

SIPp is available on almost all UNIX platforms: HPUX, Tru64, Linux (RedHat, Debian, FreeBSD), Solaris/SunOS.

A Windows port has been contributed. You can now compile SIPp under Cygwin. A binary package with a Windows installer is also available. Check the snapshot page to download it and run SIPp under Windows.

2.5. Installing SIPp

- On Linux, SIPp is provided in the form of source code. You will need to compile SIPp to actually use it. The good news is that there are **no depencies to install**. Building SIPp is straight forward.

```
# gunzip sipp-xxx.tar.gz
# tar -xvf sipp-xxx.tar
# cd sipp
# make
```

- On Windows, SIPp is provided both with the source and the pre-compiled executable. Just execute the installer to have SIPp installed.

2.6. Increasing File Descriptors Limit

If your system does not supports enough file descriptors, you may experience problems when using the TCP mode with many simultaneous calls. Depending on the operating system you use, different procedures allow to increase the maximum number of file descriptors:

- On Linux 2.4 kernels the default number of file descriptors can be increased by modifying the `/etc/security/limits.conf` and the `/etc/pam.d/login` file.

  Open the `/etc/security/limits.conf` file and add the following lines:

  ```
  soft nofile 1024
  hard nofile 65535
  ```

  Open the `/etc/pam.d/login` and add the following line

  ```
  session required /lib/security/pam_limits.so
  ```

  The system file descriptor limit is set in the `/proc/sys/fs/file-max` file. The following command will increase the file descriptor limit:

  ```
  echo 65535> /proc/sys/fs/file-max
  ```

  To increase the number of file descriptors to its maximum limit (65535) set in the `/etc/security/limits.conf` file, type:

  ```
  ulimit -n unlimited
  ```

  Logout then login again to make the changes effective.

- On HP-UX systems the default number of file descriptors can be increased by modifying the system configuration with the sam utility. In the Kernel Configuration menu, select Configurable parameters, and change the following attributes:

  ```
  maxfiles : 4096
  maxfiles_lim : 4096
  nfiles : 4096
  ninode : 4096
  max_thread_proc : 4096
  nkthread : 4096
  ```

3. Using SIPp

3.1. Main features

SIPp allows to generate one or many SIP calls to one remote system. The tool is started from the command line. In this example, two SIPp are started in front of each other to demonstrate SIPp capabilities.

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
```

3.2. Integrated scenarios

Integrated scenarios? Yes, there are scenarios that are embedded in SIPp executable. While you can create your own custom SIP scenarios (see how to create your own XML scenarios), a few basic (yet useful) scenarios are available in SIPp executable.

3.2.1. UAC

Scenario file: uac.xml (original XML file)

```
SIPp UAC             Remote
    |(1) INVITE         |
    |------------------>|
    |(2) 100 (optional) |
    |<------------------|
    |(3) 180 (optional) |
    |<------------------|
    |(4) 200            |
    |<------------------|
    |(5) ACK            |
    |------------------>|
    |                   |
    |(6) PAUSE          |
    |                   |
    |(7) BYE            |
    |------------------>|
    |(8) 200            |
    |<------------------|
```

3.2.2. UAS

Scenario file: uas.xml (original XML file)

```
Remote               SIPp UAS
    |(1) INVITE         |
    |------------------>|
    |(2) 180            |
    |<------------------|
    |(3) 200            |
    |<------------------|
    |(4) ACK            |
    |------------------>|
    |                   |
    |(5) PAUSE          |
    |                   |
    |(6) BYE            |
    |------------------>|
    |(7) 200            |
    |<------------------|
```

3.2.3. regexp

Scenario file: regexp.xml (original XML file)

This scenario, which behaves as an UAC is explained in greater details in this section.

```
SIPp regexp         Remote
    |(1) INVITE         |
    |------------------>|
    |(2) 100 (optional) |
    |<------------------|
    |(3) 180 (optional) |
    |<------------------|
    |(4) 200            |
    |<------------------|
    |(5) ACK            |
    |------------------>|
    |                   |
    |(6) PAUSE          |
    |                   |
    |(7) BYE            |
    |------------------>|
    |(8) 200            |
    |<------------------|
```

3.2.4. 3PCC

3PCC stands for 3rd Party Call Control. 3PCC is described in RFC 3725. While this feature was first developped to allow 3PCC like scenarios, it can also be used for every case where you would need one SIPp to talk to several remotes.

In order to keep SIPp simple (remember, it's a test tool!), one SIPp instance can only talk to one remote. Which is an issue in 3PCC call flows, like call flow I (SIPp beeing a controller):

```
             A              Controller               B
             |(1) INVITE no SDP  |                   |
             |<------------------|                   |
             |(2) 200 offer1     |                   |
             |------------------>|                   |
             |                   |(3) INVITE offer1  |
             |                   |------------------>|
             |                   |(4) 200 OK answer1 |
             |                   |<------------------|
             |                   |(5) ACK            |
             |                   |------------------>|
             |(6) ACK answer1    |                   |
             |<------------------|                   |
             |(7) RTP            |                   |
             |.......................................|
```

Scenario file: 3pcc-A.xml (original XML file)

Scenario file: 3pcc-B.xml (original XML file)

Scenario file: 3pcc-C-A.xml (original XML file)

Scenario file: 3pcc-C-B.xml (original XML file)

The 3PCC feature in SIPp allows to have two SIPp instances launched and synchronised together. If we take the example of call flow I, one SIPp instance will take care of the dialog with remote A (this instance is called 3PCC-C-A for 3PCC-Controller-A-Side) and another SIPp instance will take care of the dialog with remote B (this instance is called 3PCC-C-B for 3PCC-Controller-B-Side).

The 3PCC call flow I will, in reality, look like this (Controller has been divided in two SIPp instances):

```
             A             Controller A         Controller B            B
             |(1) INVITE no SDP  |                  |                   |
             |<------------------|                  |                   |
             |(2) 200 offer1     |                  |                   |
             |------------------>|                  |                   |
             |                sendCmd  (offer1)     |                   |
             |                   |----------------->|                   |
             |                   |               recvCmd                |
             |                   |                  |(3) INVITE offer1  |
             |                   |                  |------------------>|
             |                   |                  |(4) 200 OK answer1 |
             |                   |                  |<------------------|
             |                   |               sendCmd                |
             |                   |     (answer1)    |                   |
             |                   |<-----------------|                   |
             |                 recvCmd              |(5) ACK            |
             |                   |                  |------------------>|
             |(6) ACK answer1    |                  |                   |
             |<------------------|                  |                   |
             |(7) RTP            |                  |                   |
             |..........................................................|
```

As you can see, we need to pass informations between both sides of the controller. SDP "offer1" is provided by A in message (2) and needs to be sent to B side in message (3). This mechanism is implemented in the scenarios through the <sendCmd> command. This:

```
<sendCmd>
  <![CDATA[
    Call-ID: [call_id]
    [$1]

   ]]>
</sendCmd>
```

Will send a "command" to the twin SIPp instance. Note that including the Call-ID is mandatory in order to correlate the commands to actual calls. In the same manner, this:

```
<recvCmd>
  <action
     <ereg regexp="Content-Type:.*"
           search_in="msg"
           assign_to="2"/>
  </action>
</recvCmd>
```

Will receive a "command" from the twin SIPp instance. Using the regular expression mechanism, the content is retrieved and stored in a call variable ($2 in this case), ready to be reinjected

```
  <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
      [$2]

    ]]>
  </send>
```

In other words, sendCmd and recvCmd can be seen as synchronization points between two SIPp instances, with the ability to pass parameters between each other.

Another scenario that has been reported to be do-able with the 3PCC feature is the following:

- A calls B. B answers. B and A converse
- B calls C. C answers. C and B converse
- B "REFER"s A to C and asks to replace A-B call with B-C call.
- A accepts. A and C talk. B drops out of the calls.

3.3. Traffic control

SIPp generates SIP traffic according to the scenario specified. You can control the number of calls (scenario) that are started per second. This can be done either:

- Interactively, by pressing keys on the keyboard
  - '+' key to increase call rate by 1
  - '-' key to decrease call rate by 1
  - '*' key to increase call rate by 10
  - '/' key to increase call rate by 10
- At starting time, by specifying parameters on the command line:

- "-r" to specify the call rate in number of calls per seconds
- "-rp" to specify the "**r**ate **p**eriod" in milliseconds for the call rate (default is 1000ms/1sec). This allows you to have n calls every m milliseconds (by using `-r n -rp m`).

Note:

Example: run SIPp at 7 calls every 2 seconds (3.5 calls per second)

```
./sipp -sn uac -r 7 -rp 2000 127.0.0.1
```

You can also **pause** the traffic by pressing the 'p' key. SIPp will stop placing new calls and wait until all current calls go to their end. You can **resume** the traffic by pressing 'p' again.

To **quit** SIPp, press the 'q' key. SIPp will stop placing new calls and wait until all current calls go to their end. SIPp will then exit.

Note:

TIP: you can place a defined number of calls and have SIPp exit when this is done. Use the `-m` option on the command line.

3.4. Running SIPp in background

SIPp can be launched in background mode (`-bg` command line option).

By doing so, SIPp will be detached from the current terminal and run in background. The PID of the SIPp process is provided. If you didn't specified a number of calls to execute with the `-m` option, SIPp will run forever.

There is a mechanism implemented to stop SIPp smoothly. The command `kill -SIGUSR1 [SIPp_PID]` will instruct SIPp to stop placing any new calls and finish all ongoing calls before exiting.

3.5. Create your own XML scenarios

Of course embedded scenarios will not be enough. So it's time to create your own scenarios. A SIPp scenario is written in XML (although there are currently no DTD to help you write SIPp scenarios). A scenario will always start with:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic Sipstone UAC">
```

And end with:

```
</scenario>
```

Easy, huh? Ok, now let's see what can be put inside. You are not obliged to read the whole table now! Just go in the next section for an example.

Command	Attribute(s)	Description	Example
<send>	retrans	Used for UDP transport only: it specifies the interval, in milliseconds, between each retransmission. There is a total of 7 retransmissions before aborting the call.	`<send retrans="500">`: will retransmit the message every 500 milliseconds.
	start_rtd	Specifies the message on which SIPp starts the "**R**esponse **T**ime **D**uration" timer. It is used to compute time between message exchanges. By default, start_rtd is set on the first message of the scenario.	`<send start_rtd="true">`: the timer will start when the enclosed message is sent.
	rtd	Specifies the message on which SIPp stops the "**R**esponse **T**ime **D**uration" timer. By default, rtd is set on the last message of the scenario.	`<send rtd="true">`: the timer will stop when the enclosed message is sent.
	lost	Emulate packet lost. The value is specified as a percentage.	`<send lost="10">`: 10% of the message sent are actually not sent :).
<recv>	response	Indicates what SIP message code is expected.	`<recv response="200">`: SIPp will expect a SIP message with code "200".
	request	Indicates what SIP message request is expected.	`<recv response="ACK">`: SIPp will expect an "ACK" SIP message.
	optional	Indicates if the message to receive is	`<recv response="100"`

		optional. In case of an optional message and if the message is actually received, it is not seen as a unexpected message.	`optional="true">`: The 100 SIP message can be received without being considered as "unexpected".
	rrs	**R**ecord **R**oute **S**et. if this attribute is set to "true", then the "Record-Route:" header of the message received is stored and can be recalled using the **[routes]** keyword.	`<recv response="100" rrs="true">`.
	start_rtd	Specifies the message on which SIPp starts the "**R**esponse **T**ime **D**uration" timer. It is used to compute time between message exchanges. By default, start_rtd is set on the first message of the scenario.	`<recv start_rtd="true">`: the timer will start when the enclosed message is sent.
	rtd	Specifies the message on which SIPp stops the "**R**esponse **T**ime **D**uration" timer. By default, rtd is set on the last message of the scenario.	`<recv rtd="true">`: the timer will stop when the enclosed message is sent.
	lost	Emulate packet lost. The value is specified as a percentage.	`<recv lost="10">`: 10% of the message received are thrown away.
	action	Specify an action when receiving the message. See Actions section for possible actions.	Example of a "regular expression" action: `<recv response="200">` `<action>` `<ereg regexp="([0-9]{1,3}\.)` `search_in="msg"` `check_it="true"` `assign_to="1,2"/>` `</action>`

			`</recv>`

<sendCmd>	<![CDATA[]]>	Content to be sent to the twin 3PCC SIPp instance. The Call-ID must be included in the CDATA.	
			```
			<sendCmd>
			  <![CDATA[
			    Call-ID: [call_id]
			    [$1]

			   ]]>
			</sendCmd>
			```

<recvCmd>	action	Specify an action when receiving the command. See Actions section for possible actions.	Example of a "regular expression" to retrieve what has been send by a sendCmd command:
			```
			<recvCmd>
			  <action
			     <ereg regexp="Content-Typ
			           search_in="msg"
			           assign_to="2"/>
			  </action>
			</recvCmd>
			```

<pause>	milliseconds	Specify the pause delay, in milliseconds. When this delay is not set, the value of the `-d` command line parameter is used.	`<pause milliseconds="5000">`: pause the scenario for 5 seconds.
<ResponseTimeRepartition>	value	Specify the intervals, in milliseconds, used to distribute the values of response times.	`<ResponseTimeRepartition value="10, 20, 30"/>`: response time values are distributed between 0 and 10ms, 10 and 20ms, 20 and 30ms, 30 and beyond.
<CallLengthRepartition>	value	Specify the intervals, in milliseconds, used to distribute the values of the call length measures.	`<CallLengthRepartition value="10, 20, 30"/>`: call length values are distributed between 0 and 10ms, 10 and 20ms, 20 and 30ms, 30 and beyond.

Table 1: List of commands with their attributes

As you can see, there are not so many commands: send, recv, sendCmd, recvCmd, pause, ResponseTimeRepartition and CallLengthRepartition. To make things even clearer, nothing is better than an example...

3.5.1. Structure of client (UAC like) XML scenarios

A client scenario is a scenario that starts with a "send" command. So let's start:

```
<scenario name="Basic Sipstone UAC">
  <send>
    <![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: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: 136

      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>
```

Inside the "send" command, you have to enclose your SIP message between the "<![CDATA" and the "]]>" tags. Everything between those tags is going to be sent toward the remote system. You may have noticed that there are strange keywords in the SIP message, like **[service], [remote_ip], ...** Those keywords are used to indicate to SIPp that it has to do something with it.

Here is the list:

Keyword	Default	Description
[service]	service	Service field, as passed in the **`-s service_name`**
[remote_ip]	-	Remote IP address, as passed on the command line.
[remote_port]	5060	Remote IP port, as passed on the command line.
[transport]	UDP	Depending on the value of **-t**

		parameter, this will take the values "UDP" or "TCP".
[local_ip]	Primary host IP address	Will take the value of **-i** parameter.
[local_port]	Random	Will take the value of **-p** parameter.
[call_number]	-	Index. The call_number starts from "1" and is incremented by 1 for each call.
[call_id]	-	A call_id identifies a call and is generated by SIPp for each new call. **In client mode, it is mandatory to use the value generated by SIPp in the "Call-ID" header.** Otherwise, SIPp will not recognise the answer to the message sent as being part of an existing call.
[media_ip]	-	Depending on the value of **-mi** parameter, it is the local IP address for RTP echo.
[media_port]	-	Depending on the value of **-mp** parameter, it set the local RTP echo port number. Default is none. RTP/UDP packets received on thatport are echoed to their sender.
[last_*]	-	The '[last_*]' keyword is replaced automatically by the specified header if it was present in the last message received (except if it was a retransmission). If the header was not present or if no message has been received, the '[last_*]' keyword is discarded, and all bytes until the end of the line are also discarded. If the specified header was present several times in the message, all occurences are concatenated

		(CRLF separated) to be used in place of the '[last_*]' keyword.
[field0-n]	-	Used to inject values from an external CSV file. See "Injecting values from an external CSV during calls" section.
[$n]	-	Used to inject the value of call variable number n. See "Actions" section

Table 1: Keyword list

Now that the INVITE message is sent, SIPp can wait for an answer by using the "recv" command.

```
  <recv response="100"> optional="true"
  </recv>

  <recv response="180"> optional="true"
  </recv>

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

As you can see, 100 and 180 messages are optional, and 200 is mandatory. **In a "recv" sequence, there must be one mandatory message**.

Now, let's send the ACK:

```
  <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>
```

We can also insert a pause. The scenario will wait for 5 seconds at this point.

```
  <pause milliseconds="5000"/>
```

And finish the call by sending a BYE and expecting the 200 OK:

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

       BYE 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: 2 BYE
       Contact: sip:sipp@[local_ip]:[local_port]
       Max-Forwards: 70
       Subject: Performance Test
       Content-Length: 0

    ]]>
   </send>

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

And this is the end of the scenario:

```
</scenario>
```

As you can see, creating your own SIPp scenarios is not a big deal. If you want to see other examples, use the `-sd` parameter on the command line to display embedded scenarios.

3.5.2. Structure of server (UAS like) XML scenarios

A server scenario is a scenario that starts with a "recv" command. The syntax and the list of available commands is the same as for "client" scenarios.

But you are more likely to use [last_*] keywords in those server side scenarios. For example, a UAS example will look like:

```
  <recv request="INVITE">
  </recv>

  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>
```

The answering message, 180 Ringing in this case, is built with the content of headers received in the INVITE message.

3.5.3. Actions

In a "recv" or "recvCmd" command, you have the possibility to execute an action. The only possible action at the moment is a "regular expression".

3.5.3.1. Regular expressions

Using regular expressions in SIPp allows to

- Extract content of a SIP message or a SIP header and store it for future usage (called re-injection)
- Check that a part of a SIP message or of an header is matching an expected expression

Regular expressions used in SIPp are defined per Posix Extended standard (POSIX 1003.2). If you want to learn how to write regular expressions, I will recommend this regexp tutorial.

Here is the syntax of the regexp action:

Keyword	Default	Description
regexp	None	Contains the regexp to use for matching the received message or header. MANDATORY.
search_in	msg	can have 2 values: "msg" (try to match against the entire message) or "hdr" (try to match against a specific SIP header).
header	None	Header to try to match against. Only used when the search_in tag is set to hdr. MANDATORY IF search_in is equal to hdr.
check_it	false	if set to true, the call is marked as failed if the regexp doesn't match.
assign_to	None	contain the variable id (integer) or a list of variable id which will be used to store the result of the matching process between the regexp and the message. This variable can be re-used at a later time in the scenario

		using '[$n]' syntax where n is the variable id.

Table 1: regexp action syntax

Note that you can have several regular expressions in one action.

The following example is used to:

- First action:
  - Extract the first IPv4 address of the received SIP message
  - Check that we could actually extract this IP address (otherwise call will be marked as failed)
  - Assign the extracted IP address to call variables 1 and 2.
- Second action:
  - Extract the Contact: header of the received SIP message
  - Assign the extracted Contract: header to variable 6.

```
<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" a
    <ereg regexp=".*" search_in="hdr" header="Contact:" check_it="true" assign_to="6" /
  </action>
</recv>
```

3.5.4. Injecting values from an external CSV during calls

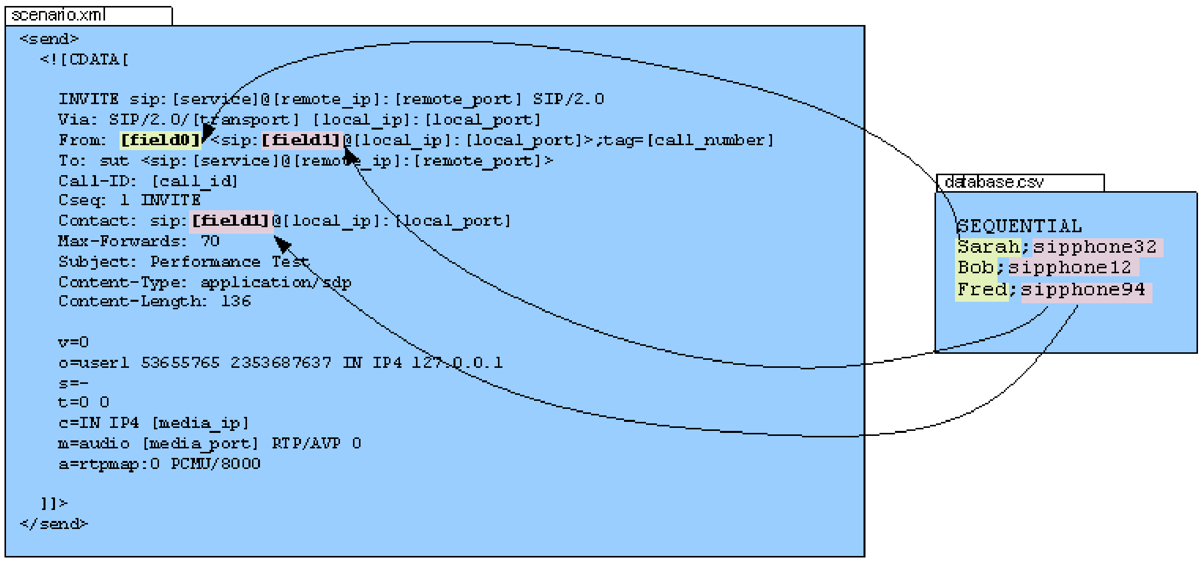
You can use "`-inf file_name`" as a command line parameter to input values into the scenarios. The first line of the file should say whether the data is to be read in sequence (SEQUENTIAL) or random (RANDOM) order. Each line corresponds to one call and has one or more ';' delimited data fields and they can be referred as [field0], [field1], ... in the xml scenario file. Example:

```
SEQUENTIAL
#This line will be ignored
Sarah;sipphone32
Bob;sipphone12
#This line too
Fred;sipphone94
```

Will be read in sequence (first call will use first line, second call second line). At any place where the keyword "[field0]" appears in the scenario file, it will be replaced by either "Sarah", "Bob" or "Fred" depending on the call. At any place where the keyword "[field1]" appears in the scenario file, it will be replaced by either "sipphone32" or "sipphone12" or "sipphone94" depending on the call. At the end of the file, SIPp will re-start from the beginning. The file is not limited in size.

The CSV file can contain comment lines. A comment line is a line that starts with a "#".

As a picture says more than 1000 words, here is one:



```
<send>
  <![CDATA[

    INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port]
    From: [field0] <sip:[field1]@[local_ip]:[local_port]>;tag=[call_number]
    To: sut <sip:[service]@[remote_ip]:[remote_port]>
    Call-ID: [call_id]
    Cseq: 1 INVITE
    Contact: sip:[field1]@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Type: application/sdp
    Content-Length: 136

    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>
```

```
SEQUENTIAL
Sarah;sipphone32
Bob;sipphone12
Fred;sipphone94
```

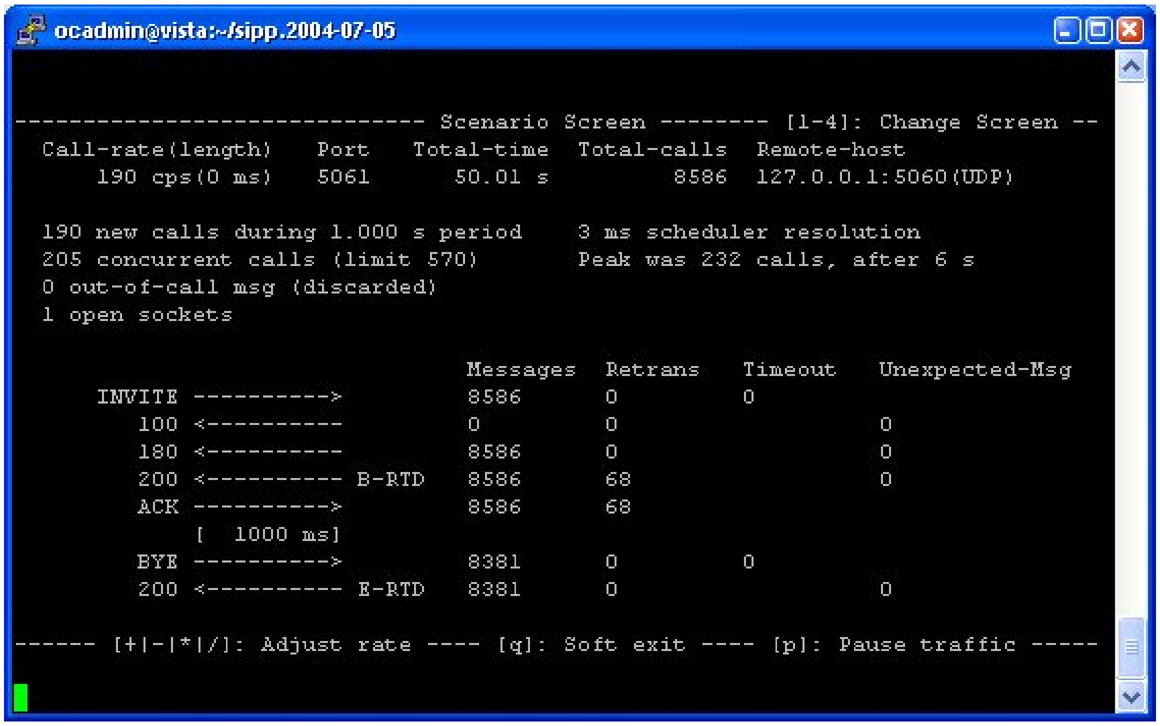
Field injection

Think of the possibilities of this feature. They are huge.

3.6. Screens

Several screens are available to monitor SIP traffic. You can change of screen by pressing 1, 2, 3 or 4 keys on the keyboard.

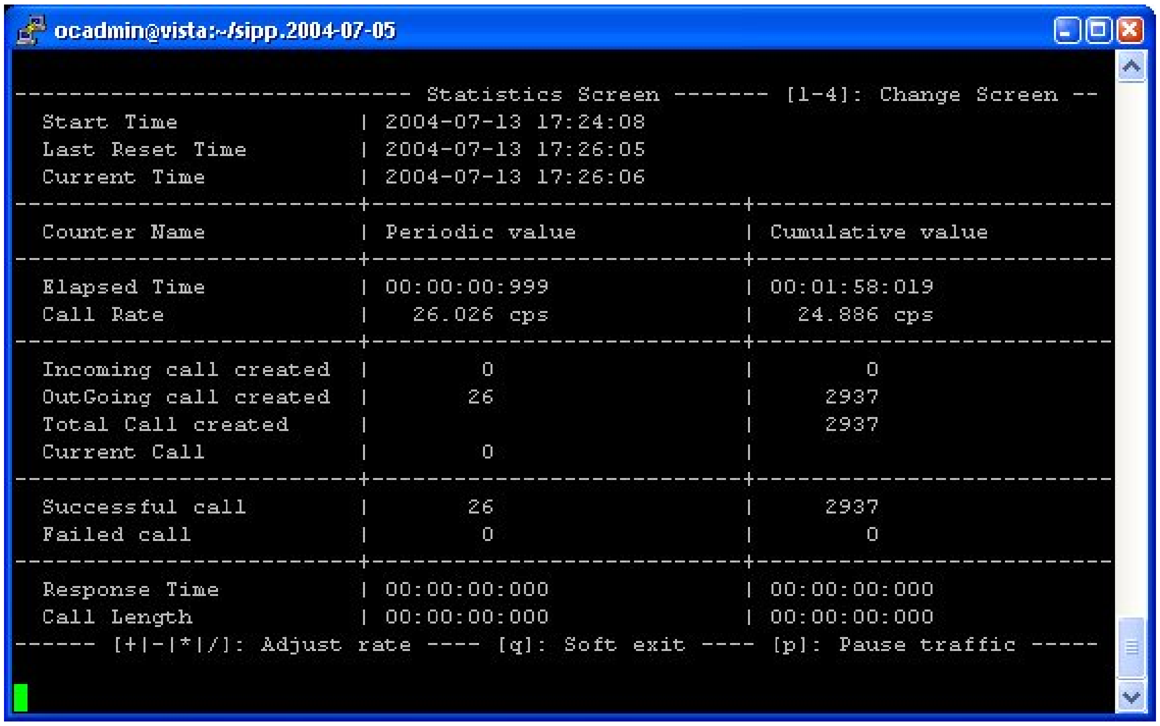
- Key '1': Scenario screen. It displays a call flow of the scenario as well as some important informations.



```
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
      190 cps(0 ms)   5061      50.01 s         8586  127.0.0.1:5060(UDP)

  190 new calls during 1.000 s period    3 ms scheduler resolution
  205 concurrent calls (limit 570)       Peak was 232 calls, after 6 s
  0 out-of-call msg (discarded)
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->         8586      0         0
         100 <----------         0         0                   0
         180 <----------         8586      0                   0
         200 <---------- B-RTD   8586      68                  0
         ACK ---------->         8586      68
             [  1000 ms]
         BYE ---------->         8381      0         0
         200 <---------- E-RTD   8381      0                   0

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Scenario screen

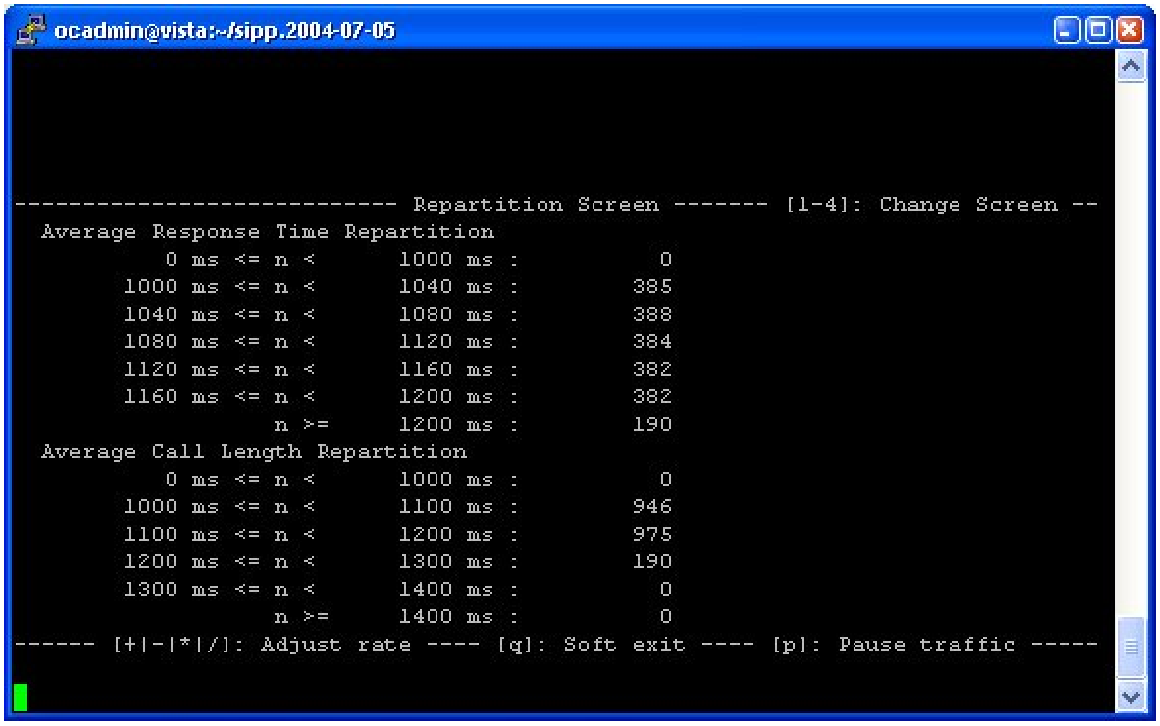
- Key '2': Statistics screen. It displays the main statistics counters. The "Cumulative" column gather all statistics, since SIPp has been launched. The "Periodic" column gives the statistic value for the period considered (specified by `-f frequency` command line parameter).



```
------------------------------ Statistics Screen ------- [1-4]: Change Screen --
  Start Time               | 2004-07-13 17:24:08
  Last Reset Time          | 2004-07-13 17:26:05
  Current Time             | 2004-07-13 17:26:06
-------------------------+---------------------------+--------------------------
  Counter Name             | Periodic value            | Cumulative value
-------------------------+---------------------------+--------------------------
  Elapsed Time             | 00:00:00:999              | 00:01:58:019
  Call Rate                |   26.026 cps              |   24.886 cps
-------------------------+---------------------------+--------------------------
  Incoming call created    |        0                  |        0
  OutGoing call created    |       26                  |     2937
  Total Call created       |                           |     2937
  Current Call             |        0                  |
-------------------------+---------------------------+--------------------------
  Successful call          |       26                  |     2937
  Failed call              |        0                  |        0
-------------------------+---------------------------+--------------------------
  Response Time            | 00:00:00:000              | 00:00:00:000
  Call Length              | 00:00:00:000              | 00:00:00:000
------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Statistics screen

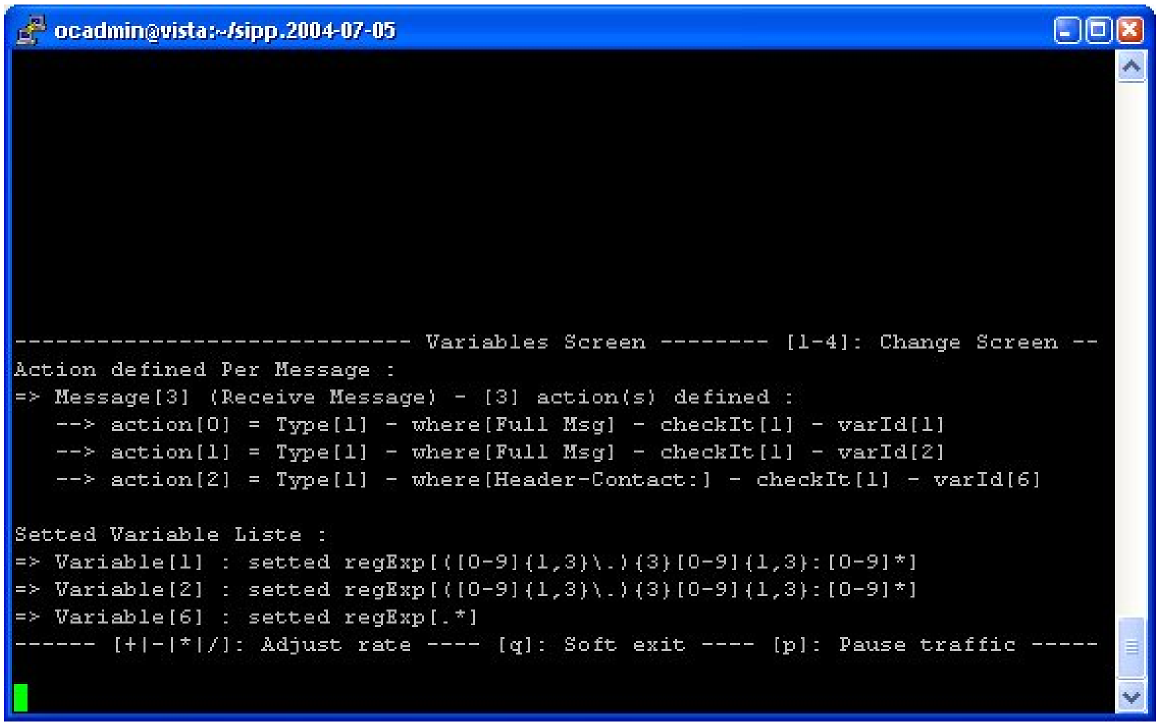
- Key '3': Repartition screen. It displays the distribution of response time and call length, as specified in the scenario.



```
---------------------------- Repartition Screen ------- [1-4]: Change Screen --
  Average Response Time Repartition
           0 ms <= n <      1000 ms :          0
        1000 ms <= n <      1040 ms :        385
        1040 ms <= n <      1080 ms :        388
        1080 ms <= n <      1120 ms :        384
        1120 ms <= n <      1160 ms :        382
        1160 ms <= n <      1200 ms :        382
                    n >=    1200 ms :        190
  Average Call Length Repartition
           0 ms <= n <      1000 ms :          0
        1000 ms <= n <      1100 ms :        946
        1100 ms <= n <      1200 ms :        975
        1200 ms <= n <      1300 ms :        190
        1300 ms <= n <      1400 ms :          0
                    n >=    1400 ms :          0
------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Repartition screen

- Key '4': Variables screen. It displays informations on actions in scenario as well as scenario variable informations.



```
------------------------------ Variables Screen -------- [1-4]: Change Screen --
Action defined Per Message :
=> Message[3] (Receive Message) - [3] action(s) defined :
   --> action[0] = Type[1] - where[Full Msg] - checkIt[1] - varId[1]
   --> action[1] = Type[1] - where[Full Msg] - checkIt[1] - varId[2]
   --> action[2] = Type[1] - where[Header-Contact:] - checkIt[1] - varId[6]

Setted Variable Liste :
=> Variable[1] : setted regExp[([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*]
=> Variable[2] : setted regExp[([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*]
=> Variable[6] : setted regExp[.*]
------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Variables screen

3.7. Transport modes

SIPp has several transport modes. The default transport mode is "UDP mono socket".

3.7.1. UDP mono socket

In UDP mono socket mode (`-u1` command line parameter), one IP/UDP socket is opened between SIPp and the remote. All calls are placed using this socket.

Note:
This mode is generally used for emulating a relation between 2 SIP servers.

3.7.2. UDP multi socket

In UDP multi socket mode (`-un` command line parameter), one IP/UDP socket is opened for each new call between SIPp and the remote.

Note:
This mode is generally used for emulating user agents calling a SIP server.

3.7.3. TCP mono socket

In TCP mono socket mode (`-t1` command line parameter), one IP/TCP socket is opened between SIPp and the remote. All calls are placed using this socket.

Note:
This mode is generally used for emulating a relation between 2 SIP servers.

3.7.4. TCP multi socket

In TCP multi socket mode (`-tn` command line parameter), one IP/TCP socket is opened for each new call between SIPp and the remote.

Note:
This mode is generally used for emulating user agents calling a SIP server.

3.8. Handling media with SIPp - RTP echo

SIPp is originally a signalling plane traffic generator. There is a limited support of media (RTP).

The "RTP echo" feature allows SIPp to listen to one local IP address and port (specified using `-mi` and `-mp` command line parameters) for RTP media. Everything that is received on this address/port is echoed back to the sender.

This allows you to have a media channel opened even if no media can be sent from SIPp.

3.9. Exit codes

To ease automation of testing, upon exit (on fatal error or when the number of asked calls (`-m` command line option) is reached, sipp exits with one of the following exit codes:

- 0: All calls were successful
- 1: At least one call failed
- 99: Normal exit without calls processed
- -1: Fatal error

Depending on the system that SIPp is running on, you can echo this exit code by using "`echo ?`" command.

3.10. Statistics

3.10.1. Available counters

The `-trace_stat` option dumps all statistics in the pid_scenario_name.csv file. The dump starts with one header line with all counters. All following lines are 'snapshots' of statistics counter given the statistics report frequency (-fd option). When SIPp exits, the last values of the statistics are also dumped in this file.

This file can be easily imported in any spreadsheet application, like Excel.

In counter names, (P) means 'Periodic' - since last statistic row and (C) means 'Cumulated' - since sipp was started.

Available statistics are:

- StartTime: Date and time when the test has started.
- LastResetTime: Date and time when periodic counters where last reseted.
- CurrentTime: Date and time of the statistic row.
- ElapsedTime: Elapsed time.
- CallRate: Call rate (calls per seconds).
- IncomingCall: Number of incoming calls.
- OutgoingCall: Number of outgoing calls.
- TotalCallCreated: Number of calls created.
- CurrentCall: Number of calls currently ongoing.
- SuccessfulCall: Number of successful calls.
- FailedCall: Number of failed calls (all reasons).
- FailedCannotSendMessage: Number of failed calls because Sipp cannot send the message (transport issue).
- FailedMaxUDPRetrans: Number of failed calls because the maximum number of UDP retransmission attempts has been reached.
- FailedUnexpectedMessage: Number of failed calls because the SIP message received is not expected in the scenario.
- FailedCallRejected: Number of failed calls because of Sipp internal error. (a scenario sync command is not recognized or a scenario action failed or a scenario variable assignment failed).
- FailedCmdNotSent: Number of failed calls because of inter-Sipp communication error (a scenario sync command failed to be sent).
- FailedRegexpDoesntMatch: Number of failed calls because of regexp that doesn't match

(there might be several regexp that don't match during the call but the counter is increased only by one).

- FailedRegexpHdrNotFound: Number of failed calls because of regexp with hdr option but no matching header found.
- OutOfCallMsgs: Number of SIP messages received that cannot be associated to an existing call.
- AutoAnswered: Number of unexpected specific messages received for new Call-ID. The message has been automatically answered by a 200 OK Currently, implemented for 'PING' message only.

3.10.2. Importing statistics in spreadsheet applications

3.10.2.1. Example: importation in Microsoft Excel

Here is a video (Windows Media Player 9 codec or above required) on how to import CSV statistic files in Excel and create a graph of failed calls over time.

sipp-02.wmv

3.11. Error handling

SIPp has advanced feature to handle errors and unexpected events. They are detailed in the following sections.

3.11.1. Unexpected messages

- When a SIP message that **can** be correlated to an existing call (with the `Call-ID:` header) but is not expected in the scenario is received, SIPp will send a CANCEL message if no 200 OK message has been received or a BYE message if a 200 OK message has been received. The call will be marked as failed. If the unexpected message is a 4XX or 5XX, SIPp will send an ACK to this message, close the call and mark the call as failed.
- When a SIP message that **can't** be correlated to an existing call (with the `Call-ID:` header) is received, SIPp will send a BYE message. The call will not be counted at all.
- When a SIP "PING" message is received, SIPp will send an ACK message in response. This message is not counted as being an unexpected message. But it is counted in the "AutoAnswered" statistic counter.
- An unexpected message that is not a SIP message will be simply dropped.

3.11.2. Retransmissions (UDP only)

A retransmission mechanism exists in UDP transport mode. To activate the retransmission mechanism, the "send" command must include the "retrans" attribute.

When it is activated and a SIP message is sent and no ACK or response is received in answer to this message, the message is re-sent.

`<send retrans="500">`: will retransmit the message every 500 milliseconds.

Even if retrans is specified in your scenarios, you can override this by using the `-nr` command line option to globally disable the retransmission mechanism.

3.11.3. Log files (error + log + screen)

There are several ways to trace what is going on during your SIPp runs.

- You can log sent and received SIP messages in <pid>_<name_of_the_scenario>_messages.log by using the command line parameter `-trace_msg`. The messages are time-stamped so that you can track them back.
- You can trace all unexpected messages or events in <pid>_<name_of_the_scenario>_errors.log by using the command line parameter `-trace_err`.
- You can save in a file the statistics screens, as displayed in the interface. This is especially useful when running SIPp in background mode.
  This can be done in two ways:
  - When SIPp exits to get a final status report (-trace_screen option)
  - On demand by using USR2 signal (example: `kill -SIGUSR2 738`)
- You can log all call ids for calls that timeout (the maximum number of retransmissions for UDP transport is reached) by using the command line parameter `-trace_timeout`

3.12. Online help (-h)

The online help, available through the -h option is duplicated here for your convenience

```
Usage:

  sipp remote_host[:remote_port] [options]

  Available options:

   -v               : Display version and copyright information.

   -bg              : Launch the tool 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 ==> 7 calls every 2 seconds.

   -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 (UDP only).
                        'regexp'   : Standard SipStone UAC - with
                                       regexp and variables.

                      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|un|t1|tn] : 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.

   -trace_msg       : Displays sent and received SIP messages in
                      <ppid>_<scenario file name>_messages.log

   -trace_screen    : Dump statistic screens in the
                      <ppid>_<scenario_name>_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
                      <ppid>_<scenario file name>_timeout.log

   -trace_stat      : Dumps all statistics in <ppid>_<scenario_name>.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
                      <ppid>_<scenario file name>_errors.log.

   -s service_name  : Set the username part of the resquest URI.
                      Default is 'service'.

   -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.

   -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.

   -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
         <ppid>_<scenario_name>_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
   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
```

4. Useful tools aside SIPp

4.1. JEdit

JEdit (http://www.jedit.org/) is a GNU GPL text editor written in Java, and available in almost all platforms. It's extremely powerful and can be used to edit SIPp scenarios with syntax checking if you put the DTD (sipp.dtd) in the same directory as your XML scenario.

4.2. Ethereal/tethereal

Ethereal (http://www.ethereal.com/) is a GNU GPL protocol analyzer. It supports SIP.

4.3. SIP callflow

When tracing SIP calls, it is very useful to be able to get a call flow from an ethereal trace. The "callflow" tool allows you to do that in a graphical way: http://callflow.sourceforge.net/

An equivalent exist if you want to generate HTML only call flows http://www.iptel.org/~sipsc/

5. Getting support

You can likely get email-based support from the sipp users community. The mailing list address is sipp-users@lists.sourceforge.net. To protect you from SPAM, this list is restricted (only people that actually subscribed can post). Also, you can browse the SIPp mailing list archive: http://lists.sourceforge.net/lists/listinfo/sipp-users

6. Contributing to SIPp

Of course, we welcome contributions! If you created a feature for SIPp, please send the "diff" output (`diff -bruN old_sipp_directory new_sipp_directory`) so that it can be integrated in SIPp.