SIPp

SIPp 2.0 reference documentation

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

Table of contents

1. Foreword

Warning:

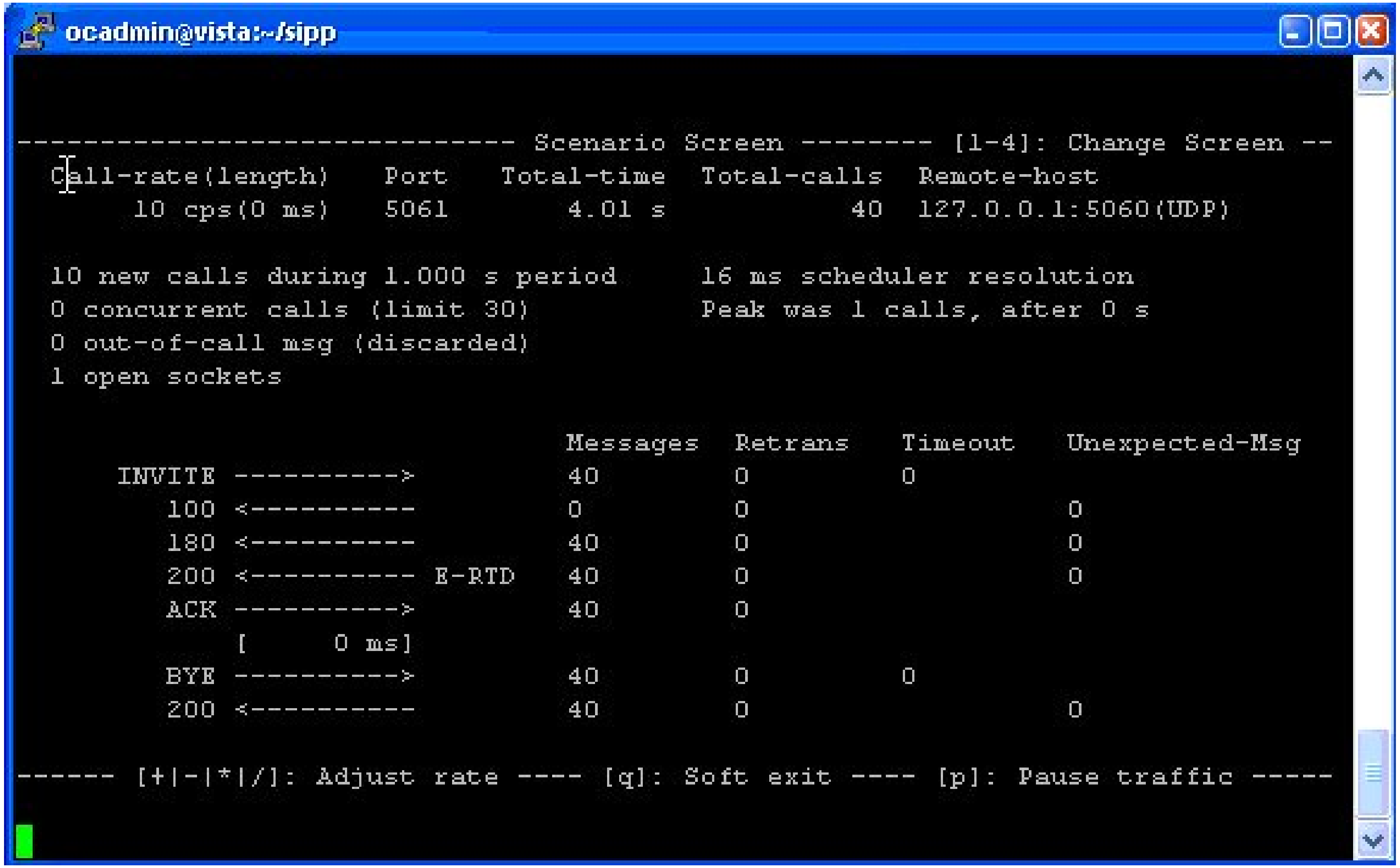
This version of the documentation is for SIPp 2.0 branch. To access the latest version of the documentation, go to this page (../doc/reference.html) .

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 -----
```

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

sipp-01.wmv (images/sipp-01.wmv)

2. Installation

2.1. Getting SIPp

SIPp is released under the GNU GPL license (http://www.gnu.org/copyleft/gpl.html) . All the terms of the license apply. It is provided to the SIP community by Hewlett-Packard (http://www.hp.com) 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 mentioned 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 (http://sourceforge.net/project/showfiles.php?group_id=104305)

2.3. Unstable release

Unstable release: all new features and bug fixes are checked in SIPp's SVN (http://sipp.svn.sourceforge.net/viewvc/sipp/sipp/trunk/) 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 (http://sipp.sourceforge.net/snapshots/)

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 download page (http://sourceforge.net/project/showfiles.php?group_id=104305) to download it and run SIPp under Windows.

Note:

SIPp works only over Windows XP and will not work on Win2000. This is because of IPv6 support. The Windows installer should prevent someone to install SIPp on Win2000.

2.5. Compiling SIPp from the source code

- Pre-requisites to compile SIPp are (see Compilation tips (http://sipp.sourceforge.net/wiki/index.php/Compilation)):
  - C++ Compiler
  - curses or ncurses library
  - For authentication and TLS support: OpenSSL >= 0.9.8
  - For pcap play support: libpcap
  - For distributed pauses: Gnu Scientific Libraries (http://www.gnu.org/software/gsl/)
- You have four options to compile SIPp:
  - **Without TLS (Transport Layer Security) and authentication support**: This is the recommended setup if you don't need to handle SIP authentication and/or TLS. In this case, there are **no depencies to install** before building SIPp. It is straight forward:

    ```
    # gunzip sipp-xxx.tar.gz
    # tar -xvf sipp-xxx.tar
    # cd sipp
    # make
    ```
  - **With TLS and authentication support**, you must have installed OpenSSL library (http://www.openssl.org/) (>=0.9.8) (which may come with your system). Building SIPp consist only in adding the "ossl" option to the make command:

    ```
    # gunzip sipp-xxx.tar.gz
    # tar -xvf sipp-xxx.tar
    # cd sipp
    # make ossl
    ```
  - **With PCAP play and without authentication support**:

    ```
    # gunzip sipp-xxx.tar.gz
    # tar -xvf sipp-xxx.tar
    # cd sipp
    # make pcapplay
    ```
  - **With PCAP play and authentication support**:

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

Note:

To enable GSL (http://www.gnu.org/software/gsl/) at compile time, you must install GSL and its include files, as well as un-comment the lines in the global.mk file of SIPp distribution. Then, re-compile SIPp.

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

> **Warning:**
> SIPp compiles under CYGWIN, provided that you installed IPv6 extension for CYGWIN (http://win6.jp/Cygwin/), as well as OpenSSL and libncurses.

- To compile SIPp on Windows with pcap (media support), you must:
  - Copy the WinPcap developer package (http://www.winpcap.org/devel.htm) to "C:\cygwin\lib\WpdPack"
  - Remove or rename "pthread.h" in "C:\cygwin\lib\WpdPack\Include", as it interfers with pthread.h from cygwin
  - Compile using either "make pcapplay_cygwin" or "pcapplay_ossl_cygwin"

2.6. Installing SIPp

Binary packages are also available for several Linux and HP_UX platforms. For each one of these platforms, the given archive contains two packages:

Those packages are incompatible to each other.

Note that there is currently no sipp-pcapplay-ossl package available for HP_UX 11.23 IA64.

First, unzip and untar the sipp archive:

```
sipp-[standard|pcapplay-ossl]-[tool version]-[OS]-[OS release version].tar.gz
```

Then, use the package installer of your platform:

- HPUX 11i/11.23:
  ```
  swinstall -s /full_path_to_the_depot/sipp-[standard|pcapplay-ossl]-[tool version]-[OS]-[OS release version]-[processor].depot
  ```
- Linux RedHat :
  ```
  rpm -ivh sipp-[standard|pcapplay-ossl]-[tool version]-[OS]-[OS release version]-[processor].rpm
  ```

Executable is located at

- /usr/local/bin/sipp

The pcapplay-ossl package also provide the following directory, containg the example pcap files:

- /usr/local/share/sipp/pcap

2.7. Increasing File Descriptors Limit

If your system does not supports enough file descriptors, you may experience problems when using the TCP/TLS mode with many simultaneous calls.

You have two ways to overcome this limit: either use the `-max_socket` command line option or change the limits of your system.

Depending on the operating system you use, different procedures allow you 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 (uac.xml.html) (original XML file (uac.xml))

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

3.2.2. UAC with media

Scenario file: uac_pcap.xml (uac_pcap.xml.html) (original XML file (uac_pcap.xml))

```
SIPp UAC            Remote
    |(1) INVITE         |
    |------------------>|
    |(2) 100 (optional) |
    |<------------------|
    |(3) 180 (optional) |
    |<------------------|
    |(4) 200            |
    |<------------------|
    |(5) ACK            |
    |------------------>|
    |                   |
    |(6) RTP send (8s)  |
    |==================>|
    |                   |
    |(7) RFC2833 DIGIT 1|
    |==================>|
    |                   |
    |(8) BYE            |
    |------------------>|
    |(9) 200            |
    |<------------------|
```

3.2.3. UAS

Scenario file: uas.xml (uas.xml.html) (original XML file (uas.xml))

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

3.2.4. regexp

Scenario file: regexp.xml (regexp.xml.html) (original XML file (regexp.xml))

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            |
    |<------------------|
```

3.2.5. branch

Scenario files: branchc.xml (branchc.xml.html) (original XML file (branchc.xml)) and branchs.xml (branchs.xml.html) (original XML file (branchs.xml))

Those scenarios, which work against each other (branchc for client side and branchs for server side) are explained in greater details in this section.

```
    REGISTER ---------->
         200 <----------
         200 <----------
      INVITE ---------->
         100 <----------
         180 <----------
         403 <----------
         200 <----------
         ACK ---------->
             [  5000 ms]
         BYE ---------->
         200 <----------
```

3.2.6. 3PCC

3PCC stands for 3rd Party Call Control. 3PCC is described in RFC 3725 (http://www.ietf.org/rfc/rfc3725.txt) . 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 (3pcc-A.xml.html) (original XML file (3pcc-A.xml))

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

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

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

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. 3PCC Extended

An extension of the 3pcc mode is implemented in sipp. This feature allows n twin sipp instances to communicate each other, each one of them being connected to a remote host.

The sipp instance which initiates the call is launched in "master" mode. The others are launched in "slave" mode. Twin sipp instances have names, given in the command line (for example, s1, s2...sn for the slaves and m for the master) Correspondances between instances names and their addresses must be stored in a file (provided by -slave_cfg command line argument), in the following format:

```

  s1;127.0.0.1:8080
  s2;127.0.0.1:7080
  m;127.0.0.1:6080
```

Each twin sipp instance must access a different copy of this file.

sendCmd and recvCmd have additional attributes:

```
<sendCmd dest="s1">
  <![CDATA[
    Call-ID: [call_id]
    From: m
    [$1]

   ]]>
</sendCmd>
```

Will send a command to the "s1" peer instance, which can be either master or slave, depending on the command line argument, which must be consistent with the scenario: a slave instance cannot have a sendCmd action before having any recvCmd. Note that the message must contain a "From" field, filled with the name of the sender.

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

Indicates that the twin command is expected to be received from the "m" peer instance.

Note that the master must be the launched at last.

There is no integrated scenarios for the 3pcc extended mode, but you can easily adapt those from 3pcc.

Example: the following drawing illustrate the entire procedure. The arrows that are shown between SIPp master and slaves depict only the synchronization commands exchanged between the different SIPp instances. The SIP message exchange takes place as usual.

3.4. 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 decrease 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.

You can also force SIPp to **quit** immediatly by pressing the 'Q' key. Current calls will be terminated by sending a BYE or CANCEL message (depending if the calls have been established or not). The same behaviour is obtained by pressing 'q' twice.

> **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.

3.5. Remote control

SIPp can be "remote-controlled" through a UDP socket. This allows for example

- To automate a series of actions, like increasing the call rate smoothly, wait for 10 seconds, increase more, wait for 1 minute and loop
- Have a feedback loop so that an application under test can remote control SIPp to lower the load, pause the traffic, ...

Each SIPp instance is listening to a UDP socket. It starts to listen to port 8888 and each following SIPp instance (up to 60) will listen to base_port + 1 (8889, 8890, ...).

It is then possible to control SIPp like this:

```
echo p >/dev/udp/x.y.z.t/8888 -> put SIPp in pause state (p key)
echo q >/dev/udp/x.y.z.t/8888 -> quit SIPp (q key)
```

> **Note:**
>
> All keys available through keyboard are also available in the remote control interface

You could also have a small shell script to automate a serie of action. For example, this script will increase the call rate by 10 more new calls/s every 5 seconds, wait at this call rate for one minute and exit SIPp:

```
#!/bin/sh
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 60
echo "q" >/dev/udp/127.0.0.1/8889
```

3.6. 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 the background. The PID of the SIPp process is provided. If you didn't specify 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.

3.7. 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 (a DTD that may help you write SIPp scenarios does exist and has been tested with jEdit - this is described in a later section). 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 T1 timer value, as described in SIP RFC 3261, section 17.1.1.2.	`<send retrans="500">`: will initiate T1 timer to 500 milliseconds (RFC3261 default).
	start_rtd	Starts one of the 5 "**R**esponse **T**ime **D**uration" timer. (see statistics section).	`<send start_rtd="2">`: the timer number 2 will start when the message is sent.
	rtd	Stops one of the 5 "**R**esponse **T**ime **D**uration" timer.	`<send rtd="2">`: the timer number 2 will stop when the

			message is sent.
	crlf	Displays an empty line **after** the arrow for the message in main SIPp screen.	`<send crlf="true">`
	lost	Emulate packet lost. The value is specified as a percentage.	`<send lost="10">`: 10% of the message sent are actually not sent :).

	next	You can put a "next" in a send to go to another part of the script when you are done with sending the message. See conditional branching section for more info.	Example to jump to label "12" after sending an ACK:
			`<send next="12">`
			`<![CDATA[`
			`ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0`
			`Via: ...`
			`From: ...`
			`To: ...`
			`Call-ID: ...`
			`Cseq: ...`
			`Contact: ...`
			`Max-Forwards: ...`
			`Subject: ...`
			`Content-Length: 0`
			`]]>`
			`</send>`

	test	You can put a "test" next to a "next" attribute to indicate that you only want to branch to the label specified with "next" if	Example to jump to label "6" after sending an ACK only if variable 4 is set: `<send next="6" test="4">`

		the variable specified in "test" is set (through regexp for example). See conditional branching section for more info.	`<![CDATA[`
			
			`ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0`
			`Via: ...`
			`From: ...`
			`To: ...`
			`Call-ID: ...`
			`Cseq: ...`
			`Contact: ...`
			`Max-Forwards: ...`
			`Subject: ...`
			`Content-Length: 0`
			
			`]]>`
			`</send>`

	counter	Increments the counter given as parameter when the message is sent. A total of 5 counter can be used. The counter are saved in the statistic file.	`<send counter="1">`: Increments counter #1 when the message is 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 request="ACK">`: SIPp will expect an "ACK" SIP message.
	optional	Indicates if the message to receive is optional. In case of an	`<recv response="100" optional="true">`:

		optional message and if the message is actually received, it is not seen as a unexpected message.	The 100 SIP message can be received without being considered as "unexpected".
	crlf	Displays an empty line **after** the arrow for the message in main SIPp screen.	`<recv crlf="true">`
	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">`.
	auth	Authentication. if this attribute is set to "true", then the "Proxy-Authenticate:" header of the message received is stored and is used to build the **[authentication]** keyword.	`<recv response="407" auth="true">`.
	start_rtd	Starts one of the 5 "**R**esponse **T**ime **D**uration" timer. (see statistics section).	`<recv start_rtd="4">`: the timer number 4 will start when the message is received.
	rtd	Stops one of the 5 "**R**esponse **T**ime **D**uration" timer.	`<recv rtd="4">`: the timer number 4 will stop when the message is received.

	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}\.){3}[0-9]{1,3}:[0-9]*" search_in="msg" check_it="true" assign_to="1,2"/>`
			`    </action>`
			`  </recv>`

	next	You can put a "next" in an optional receive to go to another part of the script if you receive that message. See conditional branching section for more info.	Example to jump to label "5" when receiving a 403 message:
			`    <recv response="100" optional="true">`
			`    </recv>`
			`    <recv response="180" optional="true">`
			`    </recv>`
			`    <recv response="403" optional="true" next="5">`
			`    </recv>`
			`    <recv response="200">`
			`    </recv>`

	test	You can put a "test" in an optional receive to go to another part of the script if you receive that message only if	Example to jump to label "5" when receiving a 403 message only if variable 3 is set:

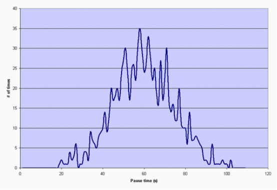
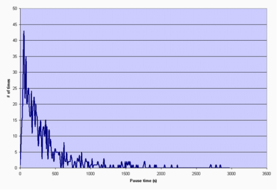
		the variable specified by "test" is set. See conditional branching section for more info.	`<recv response="100" optional="true">`
			`</recv>`
			`<recv response="180" optional="true">`
			`</recv>`
			`<recv response="403" optional="true" next="5" test="3">`
			`</recv>`
			`<recv response="200">`
			`</recv>`

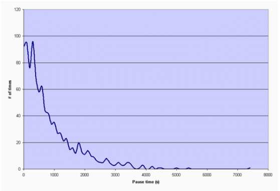
	chance	In combination with "test", probability to actually branch to another part of the scenario. Chance can have a value between 0 (never) and 1 (always). See conditional branching section for more info.	`<recv response="403" optional="true" next="5" test="3" chance="0.90">`
			`</recv>`
			90% chance to go to label "5" if variable "3" is set.

	counter	Increments the counter given as parameter when the message is received. A total of 5 counter can be used. The counter are saved in the statistic file.	`<recv counter="1">`: Increments counter #1 when the message is received.
	regexp_match	Boolean. Indicates if 'request' ('response' is not available) is given as a regular expression. If so, the recv command will	Example of a recv command that matches MESSAGE or PUBLISH or SUBSCRIBE requests: `<recv`

		match against the regular expression. This allows to catch several cases in the same receive command.	`request="MESSAGE|PUBLISH|SUBSCRIBE"`
			`crlf="true"`
			`regexp_match="true">`
			`</recv>`

<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.
	min	Indicates a minimum value for a pause. A random pause is executed between min and max values, using a uniform distribution.	`<pause min="2000" max="5000"/>` for pauses between 2 and 5 seconds.
	max	Indicates a maximum value for a pause. A random pause is executed between min and max values, using a uniform distribution..	`<pause min="2000" max="5000"/>` for pauses between 2 and 5 seconds.
	normal	If true, use a normal distribution pause with a mean and standard deviation (if GSL is available at compile time).	`<pause normal="true" mean="60000" stdev="15000"/>` provides a normal pause with a mean of 60 seconds (i.e. 60,000 ms) and a standard deviation of 15 seconds. The mean and standard deviation are specified as integer milliseconds. The

			distribution will look like: 
	lognormal	If true, the pause is specified in terms of the mean and standard deviation of the normal distribution that is exponentiated. (if GSL is available at compile time).	`<pause lognormal="true" mean="12.28" stdev="1" />` creates a distribution's whose natural logarithm has a mean of 12.28 and a standard deviation of 1. The mean and standard deviation are specified as double values (in milliseconds). The distribution will look like: 
	exponential	If true, the pause is specified using an exponential distribution, with an integer mean. (if GSL is available at compile time).	`<pause exponential="true" mean="900000"/>` creates an exponentially distributed pause with a mean of 15 minutes. The distribution will

			look like: 
	Weibull	If true, the pause is specified using a Weibull distribution, with integer scale and shape. (if GSL is available at compile time).	`<pause weibull="true" lambda="3" k ="4"/>` creates an Weibull distributed pause with scale of 3 and a shape of 4.(See Weibull on Wikipedia (http://en.wikipedia.org/wiki/Weibull) for an explanation of the distribution.)
	crlf	Displays an empty line **after** the arrow for the message in main SIPp screen.	`<pause crlf="true">`
	next	You can put a "next" in a pause to go to another part of the script when you are done with the pause. See conditional branching section for more info.	Example to jump to label "7" after pausing 4 seconds: `<pause milliseconds="4000" next="7"/>`

<nop>	action	The nop command doesn't do anything at SIP level. It is only there to specify an action to execute. See Actions section for possible actions.	Execute the play_pcap_audio/video action:
			`<nop>`
			`  <action>`
			`    <exec play_pcap_audio="pcap/g711a.pcap"/>`

			`  </action>`
			`</nop>`

	start_rtd	Starts one of the 5 "**R**esponse **T**ime **D**uration" timer. (see statistics section).	`<nop start_rtd="1">`: the timer number 1 starts when nop is executed.
	rtd	Stops one of the 5 "**R**esponse **T**ime **D**uration" timer.	`<nop rtd="1">`: the timer number 1 will stops when nop is executed.

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

	dest	3pcc extended mode only: the twin sipp instance which the command will be sent to	`<sendCmd dest="s1">`: the command will be sent to the "s1" twin instance

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

	src	3pcc extended mode only: indicate the twin	`<recvCmd src = "s1">`: the command

		sipp instance which the command is expected to be received from	will be expected to be received from the "s1" twin instance
<label>	id	A label is used when you want to branch to specific parts in your scenarios. The "id" attribute is an integer where the maximum value is 19. See conditional branching section for more info.	Example: set label number 13: `<label id="13"/>`
<Response Time Repartition>	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.
<Call Length Repartition>	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

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...

3.7.1. 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: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      t=0 0
      c=IN IP[media_ip_type] [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. You can add a computed offset [remote_port+3] to this value.
[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_ip_type]	-	Depending on the address type of **-i** parameter (IPv4 or IPv6), local_ip_type will have value "4" for IPv4 and "6" for IPv6.
[local_port]	Random	Will take the value of **-p** parameter. You can add a computed offset [local_port+3] to this value.
[len]	-	Computed length of the SIP body. To be used in "Content-Length" header. You can add a computed offset [len+3] to this value.
[call_number]	-	Index. The call_number starts from "1" and is incremented by 1 for each call.
[cseq]	-	Generates automatically the CSeq number. The initial value is 1 by default. It can be changed by using the `-base_cseq` command line option.
[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.

		Note: [call_id] can be pre-pended with an arbitrary string using '///'. Example: Call-ID: ABCDEFGHIJ///[call_id] - it will still be recognized by SIPp as part of the same call.
[media_ip]	-	Depending on the value of **-mi** parameter, it is the local IP address for RTP echo.
[media_ip_type]	-	Depending on the address type of **-mi** parameter (IPv4 or IPv6), media_ip_type will have value "4" for IPv4 and "6" for IPv6. Useful to build the SDP independently of the media IP type.
[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 that port are echoed to their sender. You can add a computed offset [media_port+3] to this value.
[auto_media_port]	-	Only for pcap. To make audio and video ports begin from the value of **-mp** parameter, and change for each call using a periodical system, modulo 10000 (which limits to 10000 concurrent RTP sessions for pcap_play)
[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
[authentication]	-	Used to put the authentication header. This field can have parameters, in the following form: [authentication username=myusername password=mypassword]. If no username is provided, the value from -s command line parameter (service) is used. If no password is provided, the value from -ap command line parameter is used. See "Authentication" section
[pid]	-	Provide the process ID (pid) of the main SIPp thread.

[routes]	-	If the "rrs" attribute in a recv command is set to "true", then the "Record-Route:" header of the message received is stored and can be recalled using the [routes] keyword
[next_url]	-	If the "rrs" attribute in a recv command is set to "true", then the [next_url] contains the contents of the Contact header (i.e within the '<' and '>' of Contact)
[branch]	-	Provide a branch value which is a concatenation of magic cookie (z9hG4bK) + call number + message index in scenario.
[msg_index]	-	Provide the message number in the scenario.
[cseq]	-	Provides the CSeq value of the last request received. This value can be incremented (e.g. [cseq+1] adds 1 to the CSeq value of the last request).

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>
```

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>
```

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.

Note that SIPp now supports short form headers (e.g "Call-id:" can be replaced by "i:").

3.7.2. 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.

3.7.3. Actions

In a "recv" or "recvCmd" command, you have the possibility to execute an action. Several actions are available:

- Regular expressions (ereg)
- Log something in aa log file (log)
- Execute an external (system), internal (int_cmd) or pcap_play_audio/pcap_play_video command (exec)

3.7.3.1. 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) (http://www.opengroup.org/onlinepubs/007908799/xbd/re.html) . If you want to learn how to write regular expressions, I will recommend this regexp tutorial (http://analyser.oli.tudelft.nl/regex/index.html.en) .

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.
case_indep	false	To look for a header ignoring case . Only used when the search_in tag is set to hdr.
occurence	1	To find the nth occurence of a header. Only used when the search_in tag is set to hdr.
start_line	false	To look only at start of line. Only used when the search_in tag is set 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(s) of the matching process between the regexp and the message. Those variables can be re-used at a later time either by using '[$n]' in the scenario to inject the value of the variable in the messages or by using the content of the variables for conditional branching. The first variable in the variable list of assign_to contains the entire regular expression matching. The following variables contain the sub-expressions matching. Example:
		`<ereg regexp="o=([[:alnum:]]*) ([[:alnum:]]*) ([[:alnum:]]*)" search_in="msg" check_it=i"true" assign_to="3,4,5,8"/>`
		If the SIP message contains the line
		`o=user1 53655765 2353687637 IN IP4 127.0.0.1`
		variable 3 contains "o=user1 53655765 2353687637", variable 4 contains "user1", variable 5 contains "53655765" and variable 8 contains "2353687637".

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" assign_to="1,2" />
    <ereg regexp=".*" search_in="hdr" header="Contact:" check_it="true" assign_to="6" />
  </action>
</recv>
```

3.7.3.2. Log a message

The "log" action allows you to customize your traces. Messages are printed in the <scenario file name>_<pid>_logs.log file. Any keyword is expanded to reflect the value actually used.

Warning:

Logs are generated only if -trace_logs option is set on the command line.

Example:

```
   <recv request="INVITE" crlf="true" rrs="true">
     <action>
         <ereg regexp=".*" search_in="hdr" header="Some-New-Header:" assign_to="1" />
          <log message="From is [last_From]. Custom header is [$1]"/>
     </action>
   </recv>
```

3.7.3.3. Execute a command

The "exec" action allows you to execute "internal", "external", "play_pcap_audio" or "play_pcap_video" commands.

Internal commands

Internal commands (specified using int_cmd attribute) are stop_call, stop_gracefully (similar to pressing 'q'), stop_now (similar to ctrl+C).

Example that stops the execution of the script on receiving a 603 response:

```
   <recv response="603" optional="true">
     <action>
          <exec int_cmd="stop_now"/>
      </action>
   </recv>
```

External commands

External commands (specified using command attribute) are anything that can be executed on local host with a shell.

Example that execute a system echo for every INVITE received:

```
   <recv request="INVITE">
     <action>
          <exec command="echo [last_From] is the from header received >> from_list.log"/>
      </action>
   </recv>
```

PCAP (media) commands

PCAP play commands (specified using play_pcap_audio / play_pcap_video attributes) allow you to send a pre-recorded RTP stream using the pcap library (http://www.tcpdump.org/pcap3_man.html) .

Choose **play_pcap_audio** to send the pre-recorded RTP stream using the "m=audio" SIP/SDP line port as a base for the replay.

Choose **play_pcap_video** to send the pre-recorded RTP stream using the "m=video" SIP/SDP line port as a base.

The play_pcap_audio/video command has the following format: play_pcap_audio="[file_to_play]" with:

- file_to_play: the pre-recorded pcap file to play

> **Note:**
> The action is non-blocking. SIPp will start a light-weight thread to play the file and the scenario with continue immediately. If needed, you will need to add a pause to wait for the end of the pcap play.

Example that plays a pre-recorded RTP stream:

```
<nop>
  <action>
    <exec play_pcap_audio="pcap/g711a.pcap"/>
  </action>
</nop>
```

3.7.4. Injecting values from an external CSV during calls

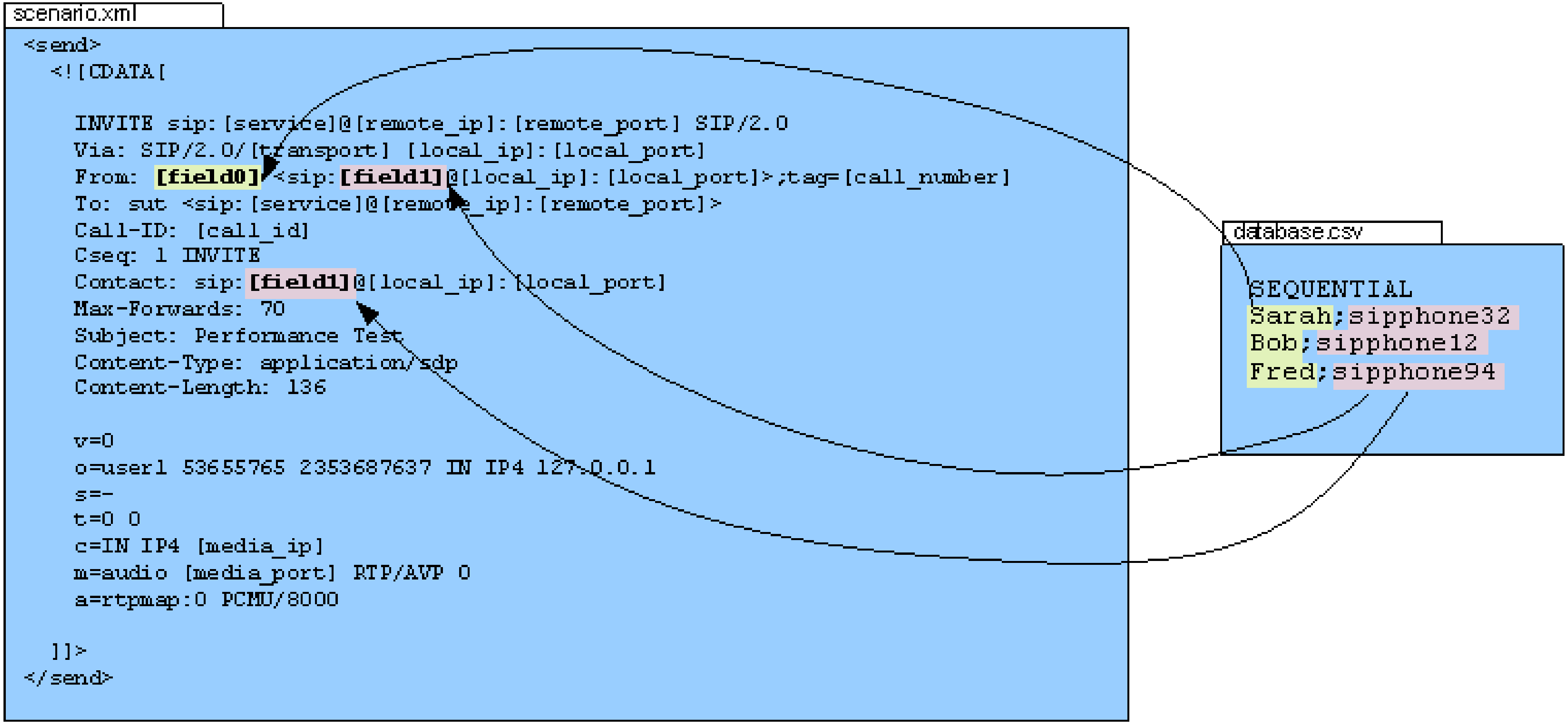
You can use "`-inf file_name`" as a command line parameter to input values into the scenarios. The first line of the file should say whether the data is to be read in sequence (SEQUENTIAL) or random (RANDOM) order. Each line corresponds to one call and has one or more ';' delimited data fields and they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

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

3.7.5. Conditional branching

3.7.5.1. Conditional branching in scenarios

It is possible to execute a scenario in a non-linear way. You can jump from one part of the scenario to another for example when a message is received or if a call variable is set.

You define a label (in the xml) as `<label id="n"/>` Where n is a number between 1 and 19 (we can easily have more if needed). The label commands go anywhere in the main scenario between other commands. To any action command (send, receive, pause, etc.) you add a next="n" parameter, where n matches the id of a label. **When it has done the command** it continues the scenario from that label. This part is useful with optional receives like 403 messages, because it allows you to go to a different bit of script to reply to it and then rejoin at the BYE (or wherever or not).

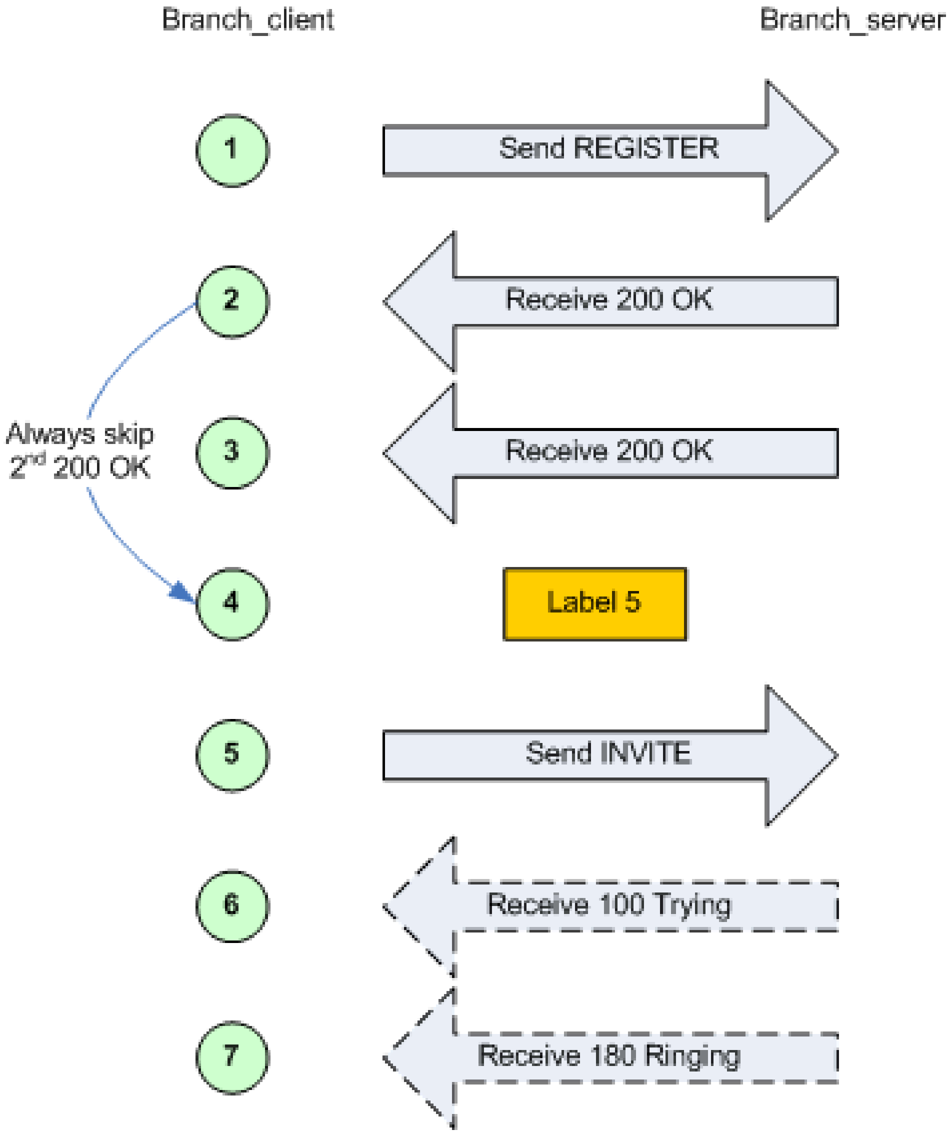
Alternatively, if you add a **test="m"** parameter to the next, it goes to the label only if variable [$m] is set. This allows you to look for some string in a received packet and alter the flow either on that or a later part of the script.

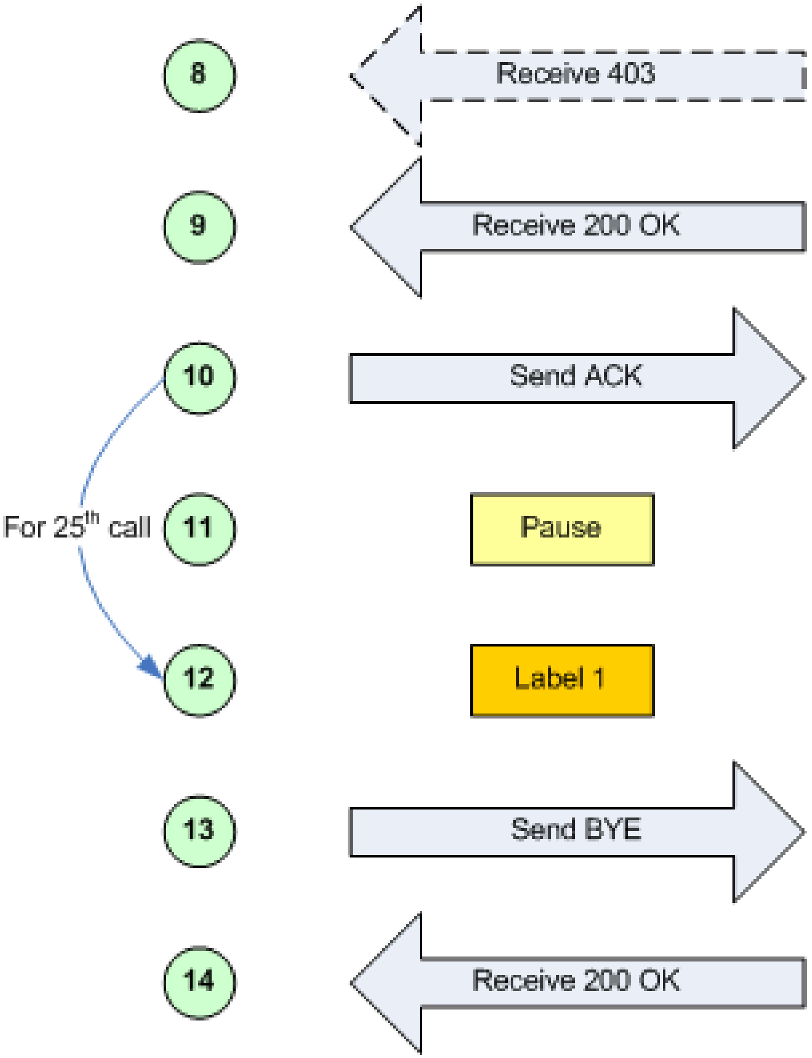
Warning:

If you add special cases at the end, don’t forget to put a label at the real end and jump to it at the end of the normal flow.

Example:

The following example corresponds to the embedded 'branchc' (client side) scenario. It has to run against the embedded 'branchs' (server side) scenario.


Branch_client
Branch_server
1
Send REGISTER
2
Receive 200 OK
Always skip
2nd 200 OK
3
Receive 200 OK
4
Label 5
5
Send INVITE
6
Receive 100 Trying
7
Receive 180 Ringing



3.7.5.2. Randomness in conditional branching

To have SIPp behave somewhat more like a "normal" SIP client being used by a human, it is possible to use "statistical branching". Wherever you can have a conditional branch on a variable being set (test="4"), you can also branch based on a statistical decision using the attribute "chance" (e.g. chance="0.90"). Chance can have a value between 0 (never) and 1 (always). "test" and "chance" can be combined, i.e. only branching when the test succeeds and the chance is good.

With this, you can have a variable reaction in a given scenario (e.g.. answer the call or reject with busy), or run around in a loop (e.g. registrations) and break out of it after some random number of iterations.

3.7.6. SIP authentication

SIPp supports SIP authentication. Two authentication algorithm are supported: Digest/MD5 ("algorithm="MD5"") and Digest/AKA ("algorithm="AKAv1-MD5"", as specified by 3GPP for IMS).

Warning:

To enable authentication support, SIPp must be compiled in a special way. See SIPp installation for details

Enabling authentication is simple. When receiving a 401 (Unauthorized) or a 407 (Proxy Authentication Required), you must add auth="true" in the <recv> command to take the challenge into account. Then, the authorization header can be re-injected in the next message by using [authentication] keyword.

Computing the authorization header is done through the usage of the "[authentication]" keyword. Depending on the algorithm ("MD5" or "AKAv1-MD5"), different parameters must be passed next to the authentication keyword:

- Digest/MD5 (example: [authentication username=joe password=schmo])
  - **username**: username: if no username is specified, the username is taken from the '-s' (service) command line parameter
  - **password**: password: if no password is specified, the password is taken from the '-ap' (authentication password) command line parameter
- Digest/AKA: (example: [authentication username=HappyFeet aka_OP=0xCDC202D5123E20F62B6D676AC72CB318 aka_K=0x465B5CE8B199B49FAA5F0A2EE238A6BC aka_AMF=0xB9B9])
  - **username**: username: if no username is specified, the username is taken from the '-s' (service) command line parameter
  - **aka_K**: Permanent secret key. If no aka_K is provided, the "password" attributed is used as aka_K.
  - **aka_OP**: OPerator variant key
  - **aka_AMF**: Authentication Management Field (indicates the algorithm and key in use)

In case you want to use authentication with a different username/password or aka_K for each call, you can do this:

- Make a CSV like this:

```
SEQUENTIAL
User0001;[authentication username=joe password=schmo]
User0002;[authentication username=john password=smith]
```

```
User0003;[authentication username=betty password=boop]
```

- And an XML like this (the [field1] will be substituted with the full auth string, which is the processed as a new keyword):

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

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port]
      To: <sip:[field0]@sip.com:[remote_port]>
      From: <sip:[field0]@[remote_ip]:[remote_port]>
      Contact: <sip:[field0]@[local_ip]:[local_port]>;transport=[transport]
      [field1]
      Expires: 300
      Call-ID: [call_id]
      CSeq: 2 REGISTER
      Content-Length: 0

    ]]>
  </send>
```

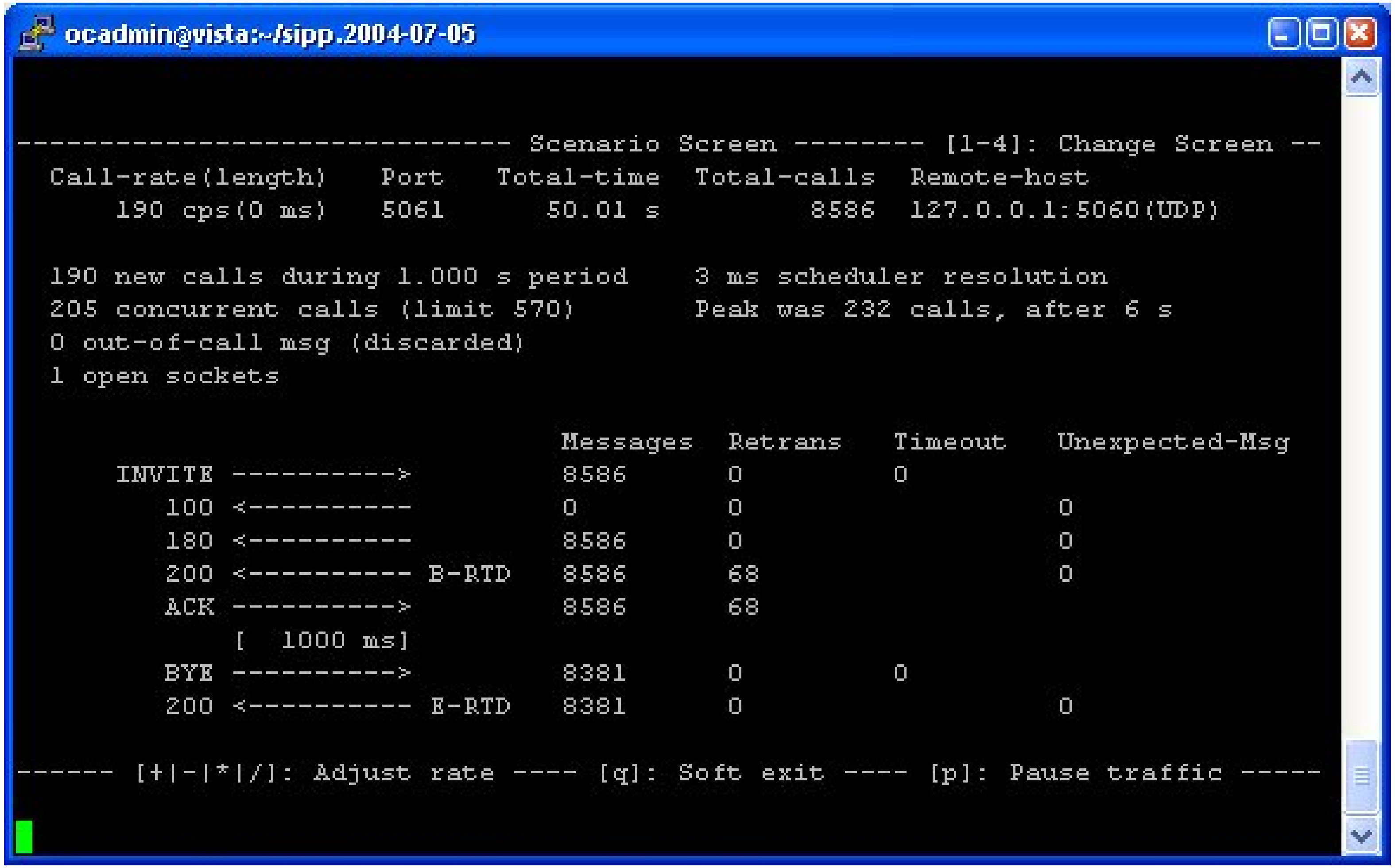
Example:

```
  <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 IP[local_ip_type] [local_ip]
      s=-
      t=0 0
      c=IN IP[media_ip_type] [media_ip]
      m=audio [media_port] RTP/AVP 0
      a=rtpmap:0 PCMU/8000

    ]]>
  </send>
```

3.8. Screens

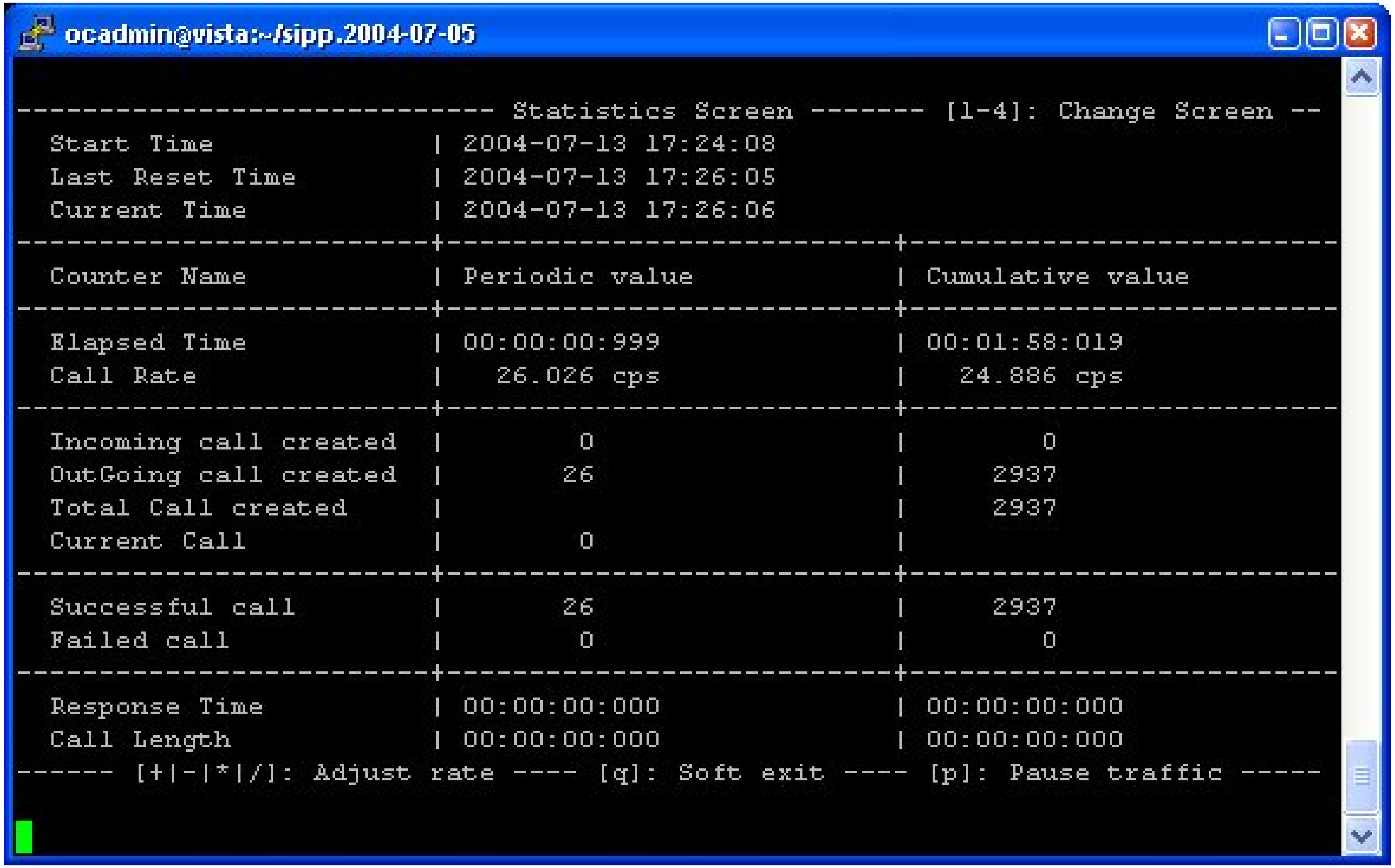
Several screens are available to monitor SIP traffic. You can change the screen view by pressing 1 to 9 keys on the keyboard.

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



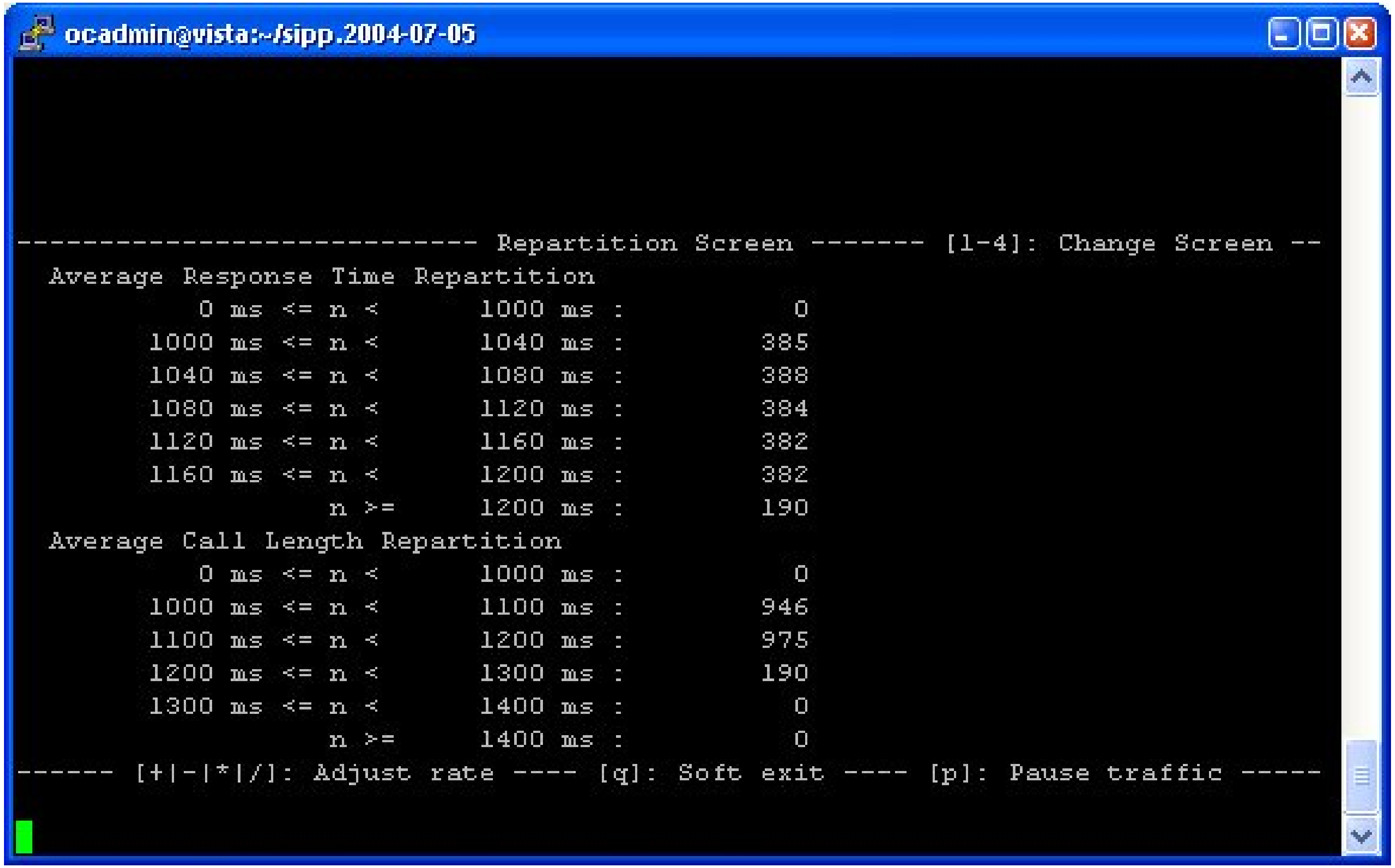
```
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
      190 cps(0 ms)   5061      50.01 s         8586  127.0.0.1:5060(UDP)

  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 -----
```

- 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).



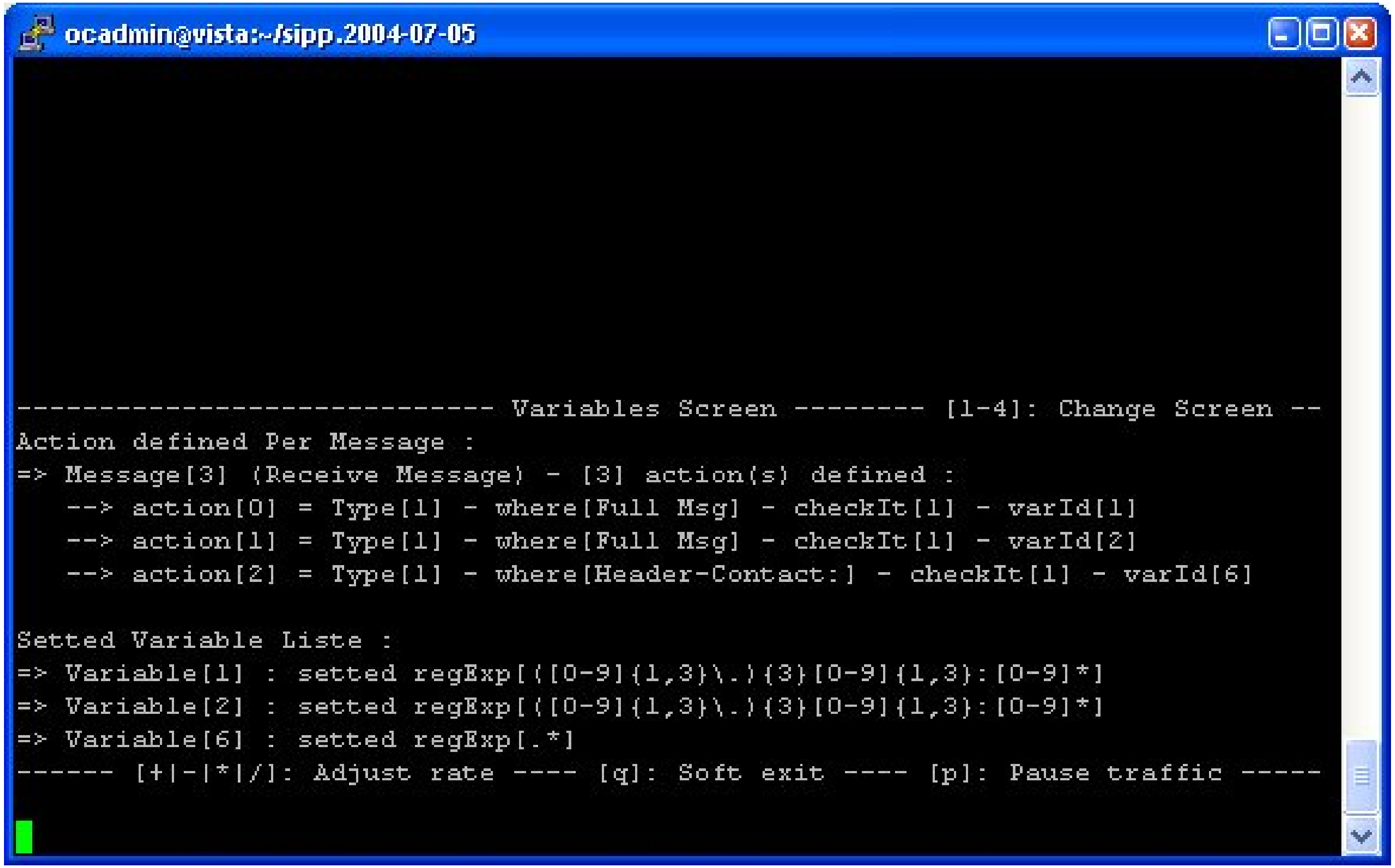
```
----------------------------- Statistics Screen ------- [1-4]: Change Screen --
  Start Time                | 2004-07-13 17:24:08
  Last Reset Time           | 2004-07-13 17:26:05
  Current Time              | 2004-07-13 17:26:06
-------------------------+---------------------------+--------------------------
  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 -----
```

- 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 -----
```

- 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 -----
```

3.9. Transport modes

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

3.9.1. UDP mono socket

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

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

3.9.2. UDP multi socket

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

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

3.9.3. UDP with one socket per IP address

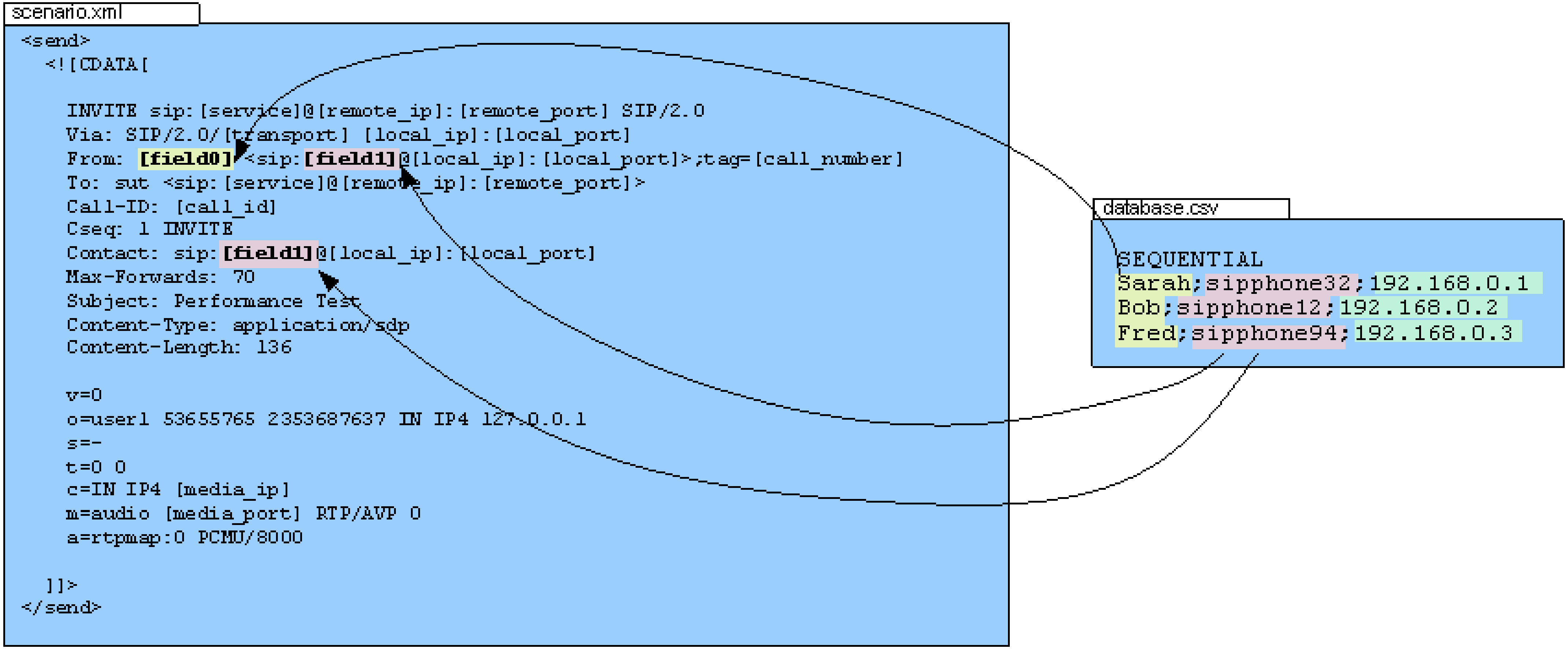
In UDP with one socket per IP address mode (`-t ui` command line parameter), one IP/UDP socket is opened for each IP address given in the inf file.

In addition to the "-t ui" command line parameter, one must indicate which field in the inf file is to be used as local IP address for this given call. Use "-ip_field <nb>" to provide the field number.

There are two distinct cases to use this feature:

- Client side: when using -t ui for a client, SIPp will originate each call with a different IP address, as provided in the inf file. In this case, when your IP addresses are in field X of the inject file, then you have to use [fieldX] instead of [local_ip] in your UAC XML scenario file.
- Server side: when using -t ui for a server, SIPp will bind itself to all the IP addresses listed in the inf file instead of using 0.0.0.0. This will have the effect SIPp will answer the request on the same IP on which it received the request. In order to have proper Contact and Via fields, a keyword [server_ip] can be used and provides the IP address on which a request was received. So when using this, you have to replace the [local_ip] in your UAS XML scenario file by [server_ip].

In the following diagram, the command line for a client scenario will look like: `./sipp -sf myscenario.xml -t ui -inf database.csv -ip_field 2 192.168.1.1`
By doing so, each new call will come sequentially from IP 192.168.0.1, 192.168.0.2, 192.168.0.3, 192.168.0.1, ...



```
<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;192.168.0.1
Bob;sipphone12;192.168.0.2
Fred;sipphone94;192.168.0.3
```

This mode is generally used for emulating user agents, using on IP address per user agent and calling a SIP server.

3.9.4. TCP mono socket

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

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

3.9.5. TCP multi socket

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

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

3.9.6. TCP reconnections

SIPp handles TCP reconnections. In case the TCP socket is lost, SIPp will try to reconnect. The following parameters on the command line control this behaviour:

- **-max_reconnect**: Set the the maximum number of reconnection.
- **-reconnect_close true/false**: Should calls be closed on reconnect?
- **-reconnect_sleep int**: How long to sleep between the close and reconnect?

3.9.7. TLS mono socket

In TLS mono socket mode (`-t l1` command line parameter), one secured TLS (Transport Layer Security) socket is opened between SIPp and the remote. All calls are placed using this socket.

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

> **Warning:**
> When using TLS transport, SIPp will expect to have two files in the current directory: a certificate (cacert.pem) and a key (cakey.pem). If one is protected with a password, SIPp will ask for it.

SIPp supports X509's CRL (Certificate Revocation List). The CRL is read and used if `-tls_crl` command line specifies a CRL file to read.

3.9.8. TLS multi socket

In TLS multi socket mode (`-t ln` command line parameter), one secured TLS (Transport Layer Security) socket is opened for each new call between SIPp and the remote.

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

3.9.9. IPv6 support

SIPp includes IPv6 support. To use IPv6, just specify the local IP address (-i command line parameter) to be an IPv6 IP address.

The following example launches a UAS server listening on port 5063 and a UAC client sending IPv6 traffic to that port.

```
./sipp -sn uas -i [fe80::204:75ff:fe4d:19d9] -p 5063
./sipp -sn uac -i [fe80::204:75ff:fe4d:19d9] [fe80::204:75ff:fe4d:19d9]:5063
```

3.9.10. Multi-socket limit

When using one of the "multi-socket" transports, the maximum number of sockets that can be opened (which corresponds to the number of simultaneous calls) will be determined by the system (see how to increase file descriptors section to modify those limits). You can also limit the number of socket used by using the `-max_socket` command line option. Once the maximum number of opened sockets is reached, the traffic will be distributed over the sockets already opened.

3.10. Handling media with SIPp

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

3.10.1. RTP echo

The "RTP echo" feature allows SIPp to listen to one or two 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.

RTP/UDP packets coming on this port + 2 are also echoed to their sender (used for sound and video echo).

3.10.2. PCAP Play

The PCAP play feature makes use of the PCAP library (http://www.tcpdump.org/pcap3_man.html) to replay pre-recorded RTP streams towards a destination. RTP streams can be recorded by tools like Wireshark (http://www.wireshark.org/) (formerly known as Ethereal) or tcpdump (http://www.tcpdump.org/) . This allows you to:

- Play any RTP stream (voice, video, voice+video, out of band DTMFs/RFC 2833, T38 fax, ...)
- Use any codec as the codec is not handled by SIPp
- Emulate precisely the behavior of any SIP equipment as the pcap play will try to replay the RTP stream as it was recorded (limited to the performances of the system).
- Reproduce exactly what has been captured using an IP sniffer like Wireshark (http://www.wireshark.org/) .

A good example is the UAC with media (uac_pcap) embedded scenario.

SIPp comes with a G711 alaw pre-recorded pcap file and out of band (RFC 2833) DTMFs in the pcap/ directory.

Warning:

The PCAP play feature uses pthread_setschedparam calls from pthread library. Depending on the system settings, you might need to be root to allow this. Please check "man 3 pthread_setschedparam" man page for details

More details on the possible PCAP play actions can be found in the action reference section.

The latest info on this feature, tips and tricks can be found on SIPp wiki (http://sipp.sourceforge.net/wiki/index.php/Pcapplay) .

3.11. 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
- 97: exit on internal command. Calls may have been processed
- 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.

3.12. Statistics

3.12.1. Response times

Response times can be gathered and reported. SIPp has 5 timers (the number is set at compile time) used to compute time between two SIPp commands (send, recv or nop). You can start a timer by using the start_rtd attribute and stop it using the rtd attribute.

You can view the value of those timers in the SIPp interface by pressing 3, 6, 7, 8 or 9. You can also save the values in a CSV file using the -trace_stat option (see below).

3.12.2. Available counters

The `-trace_stat` option dumps all statistics in the scenario_name_pid.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 with 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.

In addition, two other statistics are gathered:

- ResponseTime (see previous section)
- CallLength: this is the time of the duration of an entire call.

Both ResponseTime and CallLength statistics can be tuned using ResponseTimeRepartition and CallLengthRepartition commands in the scenario.

3.12.3. Importing statistics in spreadsheet applications

3.12.3.1. 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 (images/sipp-02.wmv)

3.13. Error handling

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

3.13.1. 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.

3.13.2. 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.

> **Note:**
>
> The retransmission mechanism follows RFC 3261, section 17.1.1.2. Retransmissions are differentiated between INVITE and non-INVITE methods.

`<send retrans="500">`: will initiate the T1 timer to 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.

3.13.3. 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 <name_of_the_scenario>_<pid>_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 <name_of_the_scenario>_<pid>_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`

3.14. 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 SIPp in background mode.

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

   -buff_size buff_size: Set the send and receive buffer size.

   -i local_ip       : Set the local IP address for 'Contact:',
                       'Via:', and 'From:' headers. Default is
                       primary host IP address.

   -bind_local       : Bind socket to local IP address, i.e. the local IP
                       address is used as the source IP address.
                       If SIPp runs in server mode it will only listen on the
                       local IP address instead of all IP addresses.

   -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.

  -rate_increase   : Specify the rate increase every -fd seconds
                     This allows you to increase the load for each
                     independent logging period
                     Example: -rate_increase 10 -fd 10
                       ==> increase calls by 10 every 10 seconds.

  -rate_max        : If -rate_increase is set, then quit after the rate
                     reaches this value.
                     Example: -rate_increase 10 -max_rate 100
                       ==> increase calls by 10 until 100 cps is hit.

  -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.

   -timer_resol     : Set the timer resolution in milliseconds.
                      This option has an impact on timers precision.
                      Small values allow more precise scheduling but
                      impacts CPU usage.
                      If the compression is on, the value is set to 50ms.
                      The default value is 200ms.

   -max_recv_loops  : Set the maximum number of messages received read per
                      cycle. Increase this value for high traffic level.
                      The default value is 1000.

   -up_nb           : Set the number of updates of the internal clock during
                      the reading of received messages.
                      Default value is 1.

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

   -lost            : Set the number of packets to lose by default (scenario
                      specifications override this value).

   -cid_str string  : Call ID string (default %u-%p@%s).
                      %u=call_number, %s=ip_address, %p=process_number,
                      %%=% (in any order).

   -auth_uri uri    : Force the value of the URI for authentication.
                      By default, the URI is composed of
                      remote_ip:remote_port.

   -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).
                        'uac_pcap' : Standard SipStone UAC with pcap
                                     play (RTP)
```

```
                       '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.
  -ip_field nr     : Set which field from the injection file contains the
                     IP address from which the client will send its
                     messages.
                     If this option is omitted and the '-t ui' option is
                     present, then field 0 is assumed.
                     Use this option together with '-t ui'

  -sd name         : Dumps a default scenario (embeded in
                     the sipp executable)

  -t [u1|un|ui|t1|tn|l1|ln] : Set the transport mode:

                       u1: UDP with one socket (default),
                       un: UDP with one socket per call,
                       ui: UDP with one socket per IP address
                           The IP addresses must be defined in the
                           injection file.
                       t1: TCP with one socket,
                       tn: TCP with one socket per call,
                       l1: TLS with one socket,
                       ln: TLS with one socket per call.

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

  -trace_screen    : Dump statistic screens in the
                     <scenario_name>_<pid>_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>_<pid>_timeout.log

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

   -stat_delimiter string : Set the delimiter for the statistics file

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

   -trace_logs      : Allow tracing of <log> actions in
                      <scenario file name>_<pid>_logs.log.

   -trace_rtt       : Allow tracing of all response times in
                      <scenario file name>_<pid>_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'

   -tls_crl name    : Set the name for Certificate Revocation List file.
                      If not specified, X509 CRL is not activated.

   -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.

  -rtp_echo        : Enable RTP echo. RTP/UDP packets received
                     on port defined by -mp 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).

  -mp media_port   : Set the local RTP echo port number. Default
                     is 6000.

  -mi local_rtp_ip : Set the local media IP address.

  -mb buf_size     : Set the RTP echo buffer size (default: 2048).

  -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.
                     SIPp will try to connect to this address:port to
                     send the twin command (This instance must be started
                     after all other 3PCC scenarii).
                     Example: 3PCC-C-A scenario.
                     - When the first twin command is 'recvCmd' then
                     this is the address of the local twin socket. SIPp
                     will open this address:port to listen for twin command.
                     Example: 3PCC-C-B scenario.

  -master          : 3pcc extended mode: indicates the name of the twin sipp
                     instance (if master)

  -slave           : 3pcc extended mode: indicates the name of the twin sipp
                     instance (if slave)

  -slave_cfg       : 3pcc extended mode: indicates the file where the master
                     and slave addresses are stored. This option
                     must be set in the command line before the -sf option
```

```
   -nr              : Disable retransmission in UDP mode.

   -max_retrans     : Maximum number of UDP retransmissions before call
                      ends on timeout.
                      Default is 5 for INVITE transactions and 7 for
                      others.

   -recv_timeout nb : Global receive timeout in milliseconds.
                      If the expected message is not received, the call
                      times out and is aborted

   -timeout nb      : Global timeout in seconds.
                      If this option is set, SIPp quits after nb seconds

   -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 receive timeout with no ontimeout attribute,
                      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

   -pause_msg_ign   : Ignore the messages received during a pause defined
                      in the scenario

   -rsa host[:port] : Set the remote sending address to host:port.
                      for sending the messages.

   -max_reconnect   : Set the the maximum number of reconnection.

   -reconnect_close true/false: Should calls be closed on reconnect?

   -reconnect_sleep int : How long to sleep between the close and reconnect?

   -aa              : Enable automatic 200 OK answer for INFO, UPDATE and NOTIFY
                      messages.

   -tdmmap map      : Generate and handle a table of TDM circuits.
                      A circuit must be available for the call to be placed.
                      Format: -tdmmap {0-3}{99}{5-8}{1-31}

   -key keyword value : Set the generic parameter named "keyword" to "value".
```

```
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>_<pid>_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
```

4. Performance testing with SIPp

4.1. Advices to run performance tests with SIPp

SIPp has been originally designed for SIP performance testing. Reaching high call rates and/or high number of simultaneous SIP calls is possible with SIPp, provided that you follow some guidelines:

- Use an HP-UX, Linux or other *ix system to reach high performances. The Windows port of SIPp (through CYGWIN) cannot handle high performances.
- Limit the traces to a minimum (usage of -trace_msg, -trace_logs should be limited to scenario debugging only)
- To reach a high number of simultaneous calls in multi-socket mode, you must increase the number of filedescriptors handled by your system. Check "Increasing File Descriptors Limit" section for more details.
- Understand internal SIPp's scheduling mechanism and use the -timer_resol, -max_recv_loops and -up_nb command line parameters to tune SIPp given the system it is running on.

Generally, running performance tests also implies measuring response times. You can use SIPp's timers (start_rtd, rtd in scenarios and -trace_rtt command line option) to measure those response times. The precision of those measures are entirely dependent on the timer_resol parameter (as described in "SIPp's internal scheduling" section). You might want to use another "objective" method if you want to measure those response times with a high precision (a tool like Wireshark (http://www.wireshark.org/) will allow you to do so).

4.2. SIPp's internal scheduling

Three parameters can be set to allow SIPp to benefit of the hardware it is running on. Tuning those parameters will also reduce the risk of unwanted retransmissions at high call rates.

Let's first describe SIPp's main scheduling loop:

```
+-->---+
|      |
|    Management of new calls (creation of new calls if needed ...):
|      | ->done every time
|      |
|    Management of ongoing calls (calculate wait, retransmissions ...):
|      | ->done every "timer_resol" ms at best
|      |
|    Management of received messages:
|      | ->done every time, "max_recv_loops" messages are read at the very most
|      |
|    Management of statistics:
|      | ->done every time
|      |
+--<---+
```

Several parameters can be specified on the command line to fine tune this scheduling.

- timer_resol: during the main loop, the management of calls (management of wait, retransmission ...) is done for all calls, every "timer_resol" ms at best. The delay of retransmission must be higher than "timer_resol". This parameter can be reduce to reduce retransmissions. If other treatments in SIPp are too long, "timer_resol" can not be respected. Reduce "max_recv_loops" to reduce retransmissions.
- max_recv_loops and up_nb: received messages are read and treated in batch. "max_recv_loops" is the maximum number of messages that can be read at one time.

During this treatment, internal clock ("clock_tick") is updated every "max_recv_loops/up_nb" read messages. For heavy call rate, reduce "max_recv_loops" and/or increase "up_nb" to limit the retransmissions. Be careful, those two parameters have a large influence on the CPU occupation of SIPp.

5. Useful tools aside SIPp

5.1. JEdit

JEdit (http://www.jedit.org/) is a GNU GPL text editor written in Java, and available on 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 (http://sipp.sourceforge.net/doc/sipp.dtd)) in the same directory as your XML scenario.

5.2. Wireshark/tshark

Wireshark (http://www.wireshark.org/) is a GNU GPL protocol analyzer. It was formerly known as Ethereal. It supports SIP/SDP/RTP.

5.3. SIP callflow

When tracing SIP calls, it is very useful to be able to get a call flow from an wireshark 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/

6. Getting support

You can likely get email-based support from the sipp users community. The mailing list address is sipp-users@lists.sourceforge.net (mailto: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

7. 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`) on the SIPp mailing list (http://lists.sourceforge.net/lists/listinfo/sipp-users) , so that we can review and possibly integrate it in SIPp.