Hochschule Ostfalia
Fakultät Elektrotechnik
Labor für Kommunikationssysteme

Ostfalia
Hochschule für angewandte Wissenschaften

AskoziaPBX
Performance Tests

Written by

Mark Stephan

Head: Prof. Dr.-Ing. D. Wermser
Supervisor: M. Iedema, B.Sc.

23.07.2010

Contents

Listings

1 Introduction

This subproject of AskoziaPBX was developed for executing performance and stress tests on different Askozia installations. It was written by Mark Stephan (mark.stephan@askozia.com) during a job as a student assistant in Spring/Summer 2010.

It was designed to execute three different test types in only one script call. The three existing test types are "two-party-tests", "conference-participants-test" and "conference-rooms-test". Of course, it is also possible to execute only one or two of these tests. They are described in detail in chapters 3, 4 and 5.

1.1 Problem

The AskoziaPBX software can be downloaded as a firmware image for embedded systems and as a live cd. The live cd can be run on every normal computer, so the underlying hardware may have very different performance (e.g., the same software can handle three, 30 or 300 parallel two-way-calls, depending on the computer performance). For this reasion, we had to develop an algorithm to find out how capable the current Askozia box is.

1.2 Features

The current testsuite supports the following features:

- completely automated testing of one AskoziaPBX
- automatic configuration of the AskoziaPBX installations with the needed settings
- three different types of tests:

  two-participants tests The testsuite establishes a variable number of A-to-B (or end-to-end) calls between the AskoziaPBX installation and the system performing the test. This test simulates a variable number of normal telephone calls between two parties.

  conference rooms tests The testsuite establishes a variable number of conference rooms on the AskoziaPBX installation. Each conference room has three participants. This test simulates a variable number of three-way conferences on the AskoziaPBX installation.

  conference participants tests The testsuite establishes one conference with a variable number of participants. This test simulates a conference with many many participants.

- monitoring of the CPU load of the AskoziaPBX caused by the testcalls
- downloading the recorded CPU load data
- interpretation and creation of graphs of the testresults

1.3 Usage

The script can be called from the command line as described below:

Listing 1: Script usage

```
./PERF_TEST <options>
perl PERF_TEST <options>

./PERF_TEST --local-ip=192.168.0.2
    --askozia-ip=192.168.0.1
    --max-two-party-test=30
    (two-participants test with maximal 30 users)

./PERF_TEST --local-ip=192.168.100.20
    --askozia-ip=192.168.100.200
    --max-conference-rooms-test=15
    (conference rooms test with maximal 15 rooms)

./PERF_TEST --local-ip=10.10.10.10
    --askozia-ip=10.10.10.5
    --max-conference-participants-test=40
    (conference participants test with maximal 40 users)

./PERF_TEST --local-ip=192.168.2.100
    --askozia-ip=192.168.2.1
    --max-two-party-test=30
    --max-conference-rooms-test=15
    --max-conference-participants-test=40
    (executes all three different tests sequentially)
```

The script's parameters can be classified in three sections: "Necessary", "Optional" and "Developers". The first two groups are described below, the "Developers" parameters are listed in the appendix. You have to be root to execute this script because `sipp` reserves port for its connection to Askozia.

1.3.1 Necessary Parameters

`--local-ip=<IP>`
: The IP-adress of the testcomputer that executes the testscript. <IP> stands for the address of the network interface connected to the AskoziaPBX.
Default: undefined
Example: `--local-ip=192.168.0.2`

At least one of these three following parameters is necessary, too:

`--max-two-party-test=<number>` Defines the maximum number of two-party calls that should be established. The two-party test begins with one call, increases this number of calls step-by-step and finishes with the number of calls specified by this parameter.
Default: undefined (no two-way tests)
Example: `--max-two-party-test=30`

`--max-conference-rooms-test=<number>` Defines the maximum number of three-way conferences that should be established. The conference-rooms test begins with one three-way conference, adds another one after a specific time and finishes with the number of three-way conferences set by this parameter.
Default: undefined (no conference rooms test)
Example: `--max-conference-rooms-test=15`

`--max-conference-participants-test` Defines the maximum number of participants in the conference-participants test. There is only one conference room which has only one participant at beginning. Step-by-step, the number of participants is increased up to the value of this parameter.
Default: undefined (no conference participants tests)
Example: `--max-conference-participants-test=40`

1.3.2 Optional Parameters

`--askozia-ip=<IP>`
The IP-address of the AskoziaPBX installation that is to be tested.
Default: `10.10.10.1`
Example: `--askozia-ip=192.168.0.1`

`--askozia-port=<number>`
The number of the webport of the AskoziaPBX.
Default: `80`
Example: `--askozia-port=80`

`--askozia-user=<string>`
Name of the administrator user of the AskoziaPBX webinterface.
Default: `admin`
Example: `--askozia-user=admin`

`--askozia-pw=<string>`
Password for the administrator user of the AskoziaPBX webinterface.
Default: `askozia`
Example: `--askozia-pw=askozia`

`--testname=<string>`
This parameter helps to keep your results directory uncluddered. All files of the current script call (all tests, debug files etc.) are saved in the subdir `./results/<testname>_<timestamp>/`. Furthermore, this testname is used for the appliance title in the created graphs.
Default: undefined
E.g. `--testname=Hugo`
(saving of results in subdir `./results/Hugo_2010-01-01_1030/`)

`--debug`
: Activates debug messages. Activates automatic saving of debug messages in file `./results/<testname>_<timestamp>/debug.log`, too. Testname is specified by using the `--testname` parameter.
Default: undefined (no debug output)
Example: `--debug`

`--save-sipp-log`
: The output generated by the testprogram `sipp` can be saved in a file for debugging.
The path to the file where the output is saved is `./results/<testname>_<timestamp>/sipp.log`.
Testname is specified by using the `--testname` parameter.
Default: undefined (output ignored)
Example: `--save-sipp-log`

`--help`
: Displays a short help for using the testscript and exits immediatly.
Default: undefined (no help shown)
Example: `--help`

1.4 Dependencies

This script was developed under Linux Mint 8 Helena (`http://www.linuxmint.com`). It is not possible to execute this script on Windows because there were many Linux specific system commands (like `kill`, `killall`, `which`, `date`, `id` and `ping`) used.

The script has the following dependencies:

- Perl v5.10.0 (`http://www.perl.org`)
- gnuplot 4.2 patchlevel 5 (`http://www.gnuplot.info`)

2 Automated Configuration of the AskoziaPBX Installation

This paragraph is about how the AskoziaPBX installations are automatically configured. There is a complete configuration file in the appendix. First, the used configuration file from Askozia is downloaded because the user should not have to reconfigure the whole box including all accounts and the dialplan after every test. The target is to deliver the Askozia box just like it was issued. So, there are three necessary steps which are described in the next chapters.

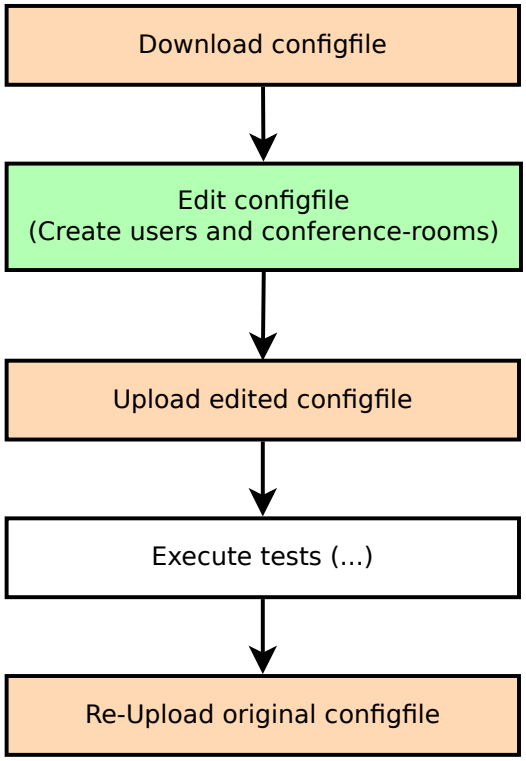

Figure 2.1: Process of editing the Askozia-configuration file

2.1 Downloading/Uploading Configuration File

For downloading the configuration file, it is necessary to send a HTML POST request to Askozia. The useragent has to be authenticated as root and the content type must be "multipart/form-data". For downloading the configuration file, the parameter "Download" has to be set.

In the performance test script, the following perl code is used to send this POST request:

Listing 2: POST request for downloading the configuration file

```
use HTTP::Request::Common;
use LWP;
my $ua = new LWP::UserAgent;
$ua->credentials ("$ask_ip:$ask_port",
        $ask_realm, "$ask_user" => "$ask_pw");
my $res = $ua->request (POST
        "http://$ask_ip:$ask_port/$ask_conf_page",
        "Content-Type" => "multipart/form-data",
        Content => [ Download => "1"]);
```

After executing this request, the following dataflow is to be expected:

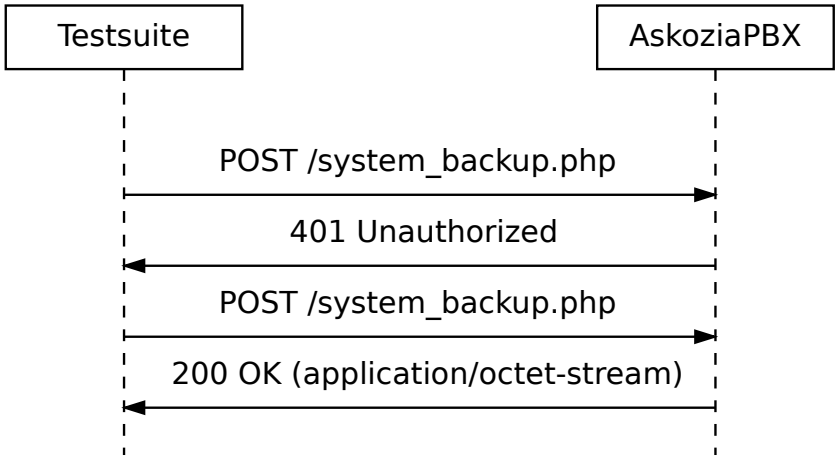

Figure 2.2: Dataflow of configuration file download

The `200 OK` message sent by Askozia includes the configuration file in XML format. The XML file is saved with its original name in the `./results/<testname>/` directory. Then, it is opened for reading to add the needed users and conference rooms (see the next sections). When finished, the edited configuration file is saved with the original named followed by an `_edited` string. This edited configuration file is now uploaded to the AskoziaPBX as follows:

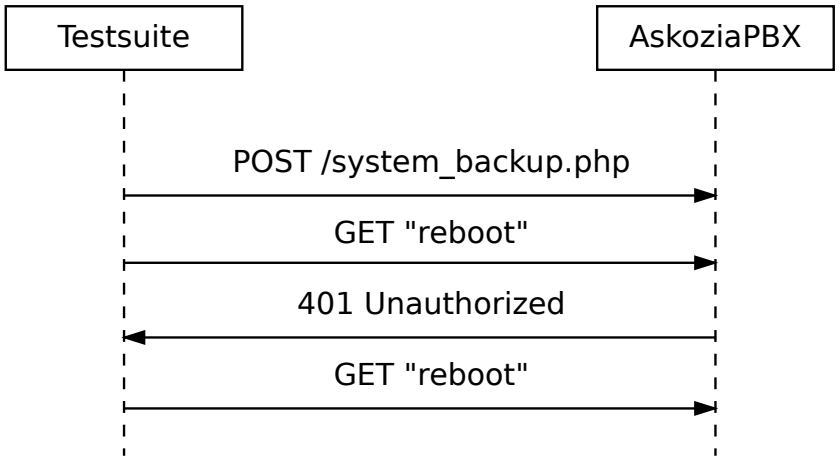

Figure 2.3: Dataflow of configuration file upload

This dataflow is created by the following perl listing (`$ua` and `$res` are the existing variables declared and defined above):

Listing 3: POST request for uploading configuration file

```
$res = $ua->request (POST
    "http://$ask_ip:$ask_port/$ask_conf_page",
        "Content-Type" => "multipart/form-data",
    Content => [ Restore => "1",
    conffile => [ $xml_configfile."_edited" ] ]);
```

After uploading the configuration file, the AskoziaPBX is rebooted. This happenes automatically, but in this case, it is forced by an extra command sent as a GET request by using the webbased shell feature. This manually forced reboot allows the script to ping Askozia and detect when it has finished rebooting. Furthermore, it is now possible to wait some time (specified by using the `--reboot-time` parameter) to make sure that the PBX has enough time to start all services etc. after the reboot.

Listing 4: Executing Askozia reboot

```
my $ua = new LWP::UserAgent;
$ua->credentials ("$ask_ip:$ask_port",
    $ask_realm, "$ask_user" => "$ask_pw");
$res = $ua->request (GET "http://$ask_ip:$ask_port/
    cgi-bin/ajax.cgi?exec_shell=reboot");
```

2.2 Users

To execute the tests, it is necessary to add some test users to the AskoziaPBX installation. The number of users depends on the parameters that were passed to the script when launching. Here is a table of required users, the highest count will be added:

Test type	Required users
two-way	`= 2 * 2way-calls` (User A and B for each call)
conf room	`= conf-calls-room * conf-rooms-room`
	("conf-calls" users per "conf-rooms" conference rooms)
conf call	`= conf-calls-call * conf-rooms-call`
	("conf-calls" users per "conf-rooms" conference rooms)

The script downloads the complete Askozia configuration file and searches for the end of the `sip` paragraph Then, it adds its generated users. It is not checked whether the added phones already exist. The template for adding users looks as follows, where `_userno_` is replaced by a three-digit integer that is incremented with each user:

Listing 5: User template

```
<phone>
<extension>_userno_</extension>
<callerid>Testuser _userno_</callerid>
<manualattributes>cXVhbGlmeT1ubw==</manualattributes>
<codec>alaw</codec>
<secret>_userno_</secret>
<uniqid>SIP-PHONE-_userno_</uniqid>
<language>en-us</language>
<ringlength>indefinitely</ringlength>
<natmode>yes</natmode>
<dtmfmode>auto</dtmfmode>
</phone>
```

2.3 Conference Rooms

Just like the users, the needed conference rooms have to be added too. The script searches for the end of the `conferencing` paragraph and adds its generated rooms. The template looks as follows, where `_roomno_` is replaced by an integer that is incremented with each room:

Listing 6: Conference room template

```
<room>
<number>_roomno_</number>
<name>Default Conference</name>
<uniqid>CONFERENCE-ROOM-_roomno_</uniqid>
</room>
```

Conference rooms start with number 2000 (can be changed). So, for a conference test that needs ten conference rooms, rooms from 2000 to 2009 are created.

3 Two-Party Tests

Two-Party tests are the normal telephone calls between two participants. User A calls another user (perhaps User B) who has to be registered, makes a phone call and hangs up.

As a `sip` dialog, the scenario looks as follows: The original XML scenarios for sipp used

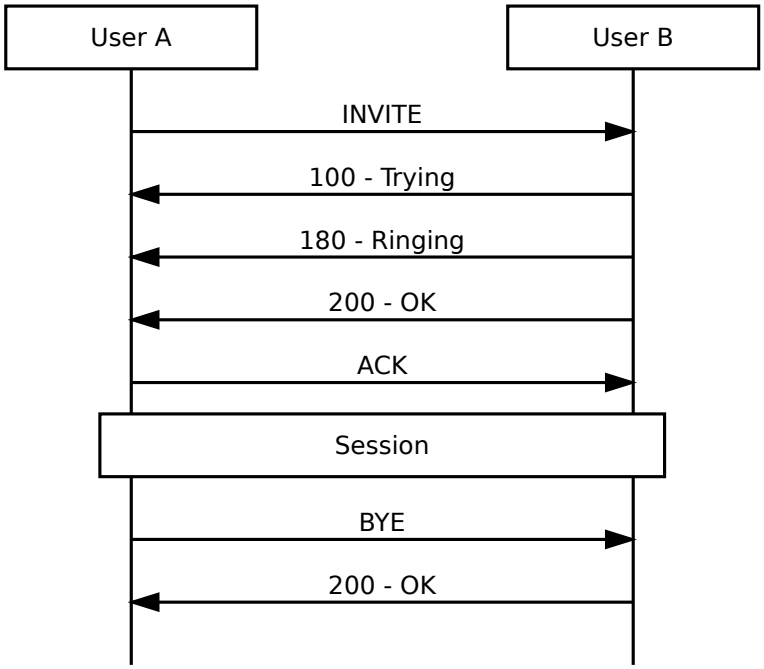

Figure 3.1: SIP dialog of a two-party call

to implement this are available in the appendix.

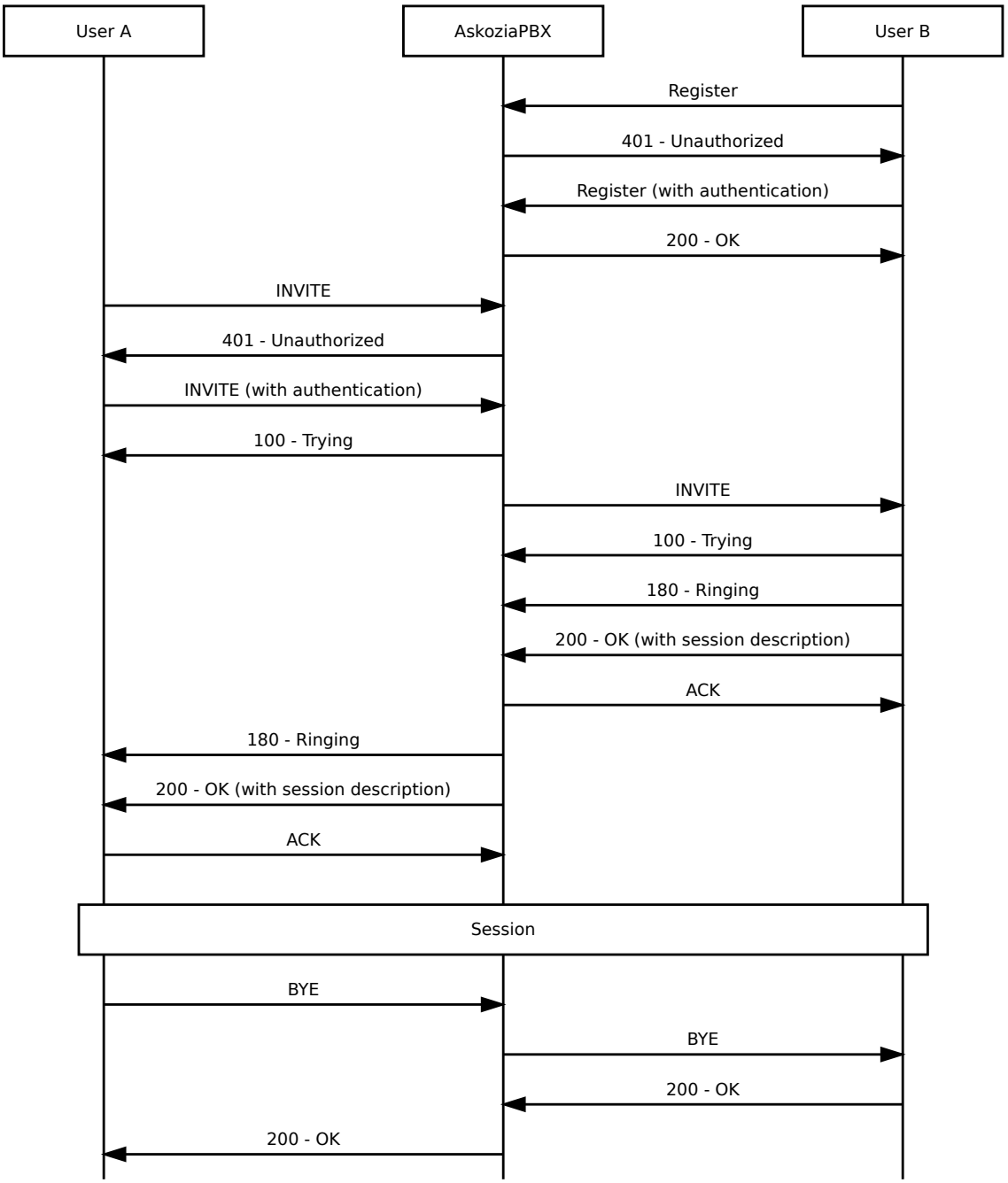

Figure 3.2: Dialog of a two-party call

The next diagramm shows the process of a complete two-party test:

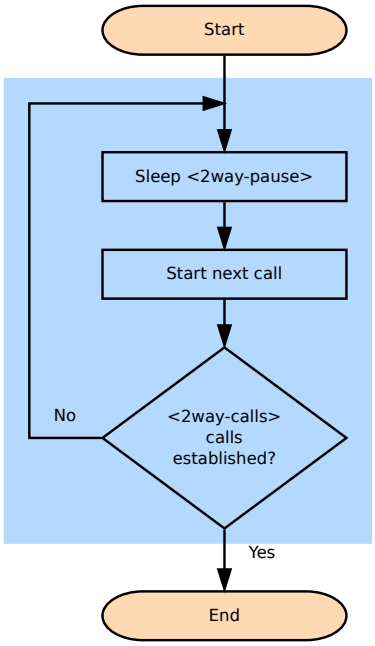

Figure 3.3: Complete two-party test

The number of calls is increased step-by-step. After every call, the script waits for the specified pause time to record the CPU load values. For executing two-party calls, the following sipp commands used. You can inform yourself about the used parameters by reading the sipp manpage (the appendix contains the sipp manpage). It is possible to change the used sip and rtp ports as well as the name of the csv files by using the developers parameters. For more information, please have a look at the appendix. In original call, the csv files are specified by their absolut path.

Listing 7: sipp command for inviting User B

```
REGISTER Command:
sipp -aa -inf 'Users_two-party.csv' -m $current_call -i $local_ip
  -p 5062 -mp 6030 -sf 'Register.xml' $ask_ip 2>&1

ACCEPT Command:
sipp -aa -inf 'Users_two-party.csv' -m $current_call -i $local_ip
  -p 5062 -mp 6030 -sf 'Accept.xml' -bg $ask_ip 2>&1 &

INVITE Command:
sipp -aa -inf 'Users_two-party.csv' -m $current_call -i $local_ip
  -p 5061 -mp 6020 -sf 'Invite.xml' $ask_ip 2>&1

De-REGISTER Command:
sipp -aa -inf 'Users_two-party.csv' -m $current_call -i $local_ip
  -p 5062 -mp 6030 -sf 'Deregister.xml' $ask_ip 2>&1
```

4 Maximum Three-Way Conference Rooms Tests

The conference rooms test was developed primarily for executing three-way conferences. Basically, there is a conference with three participants started. After that, there is another conference with three participants started until the maximum number of conferences is reached:

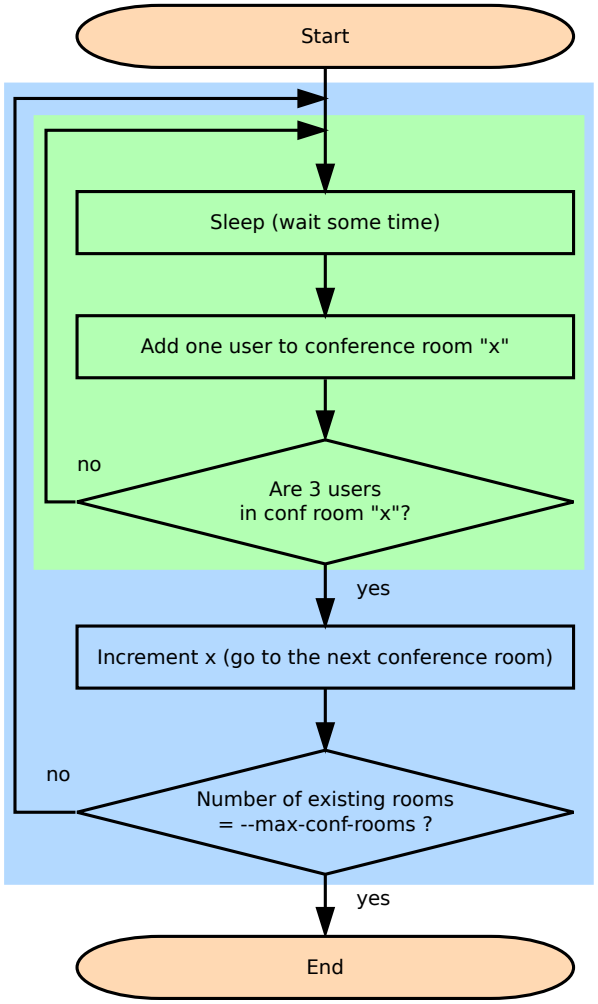

Figure 4.1: Process of conference rooms tests

For a conf call test with three participants and four conference rooms, the test would work like this:

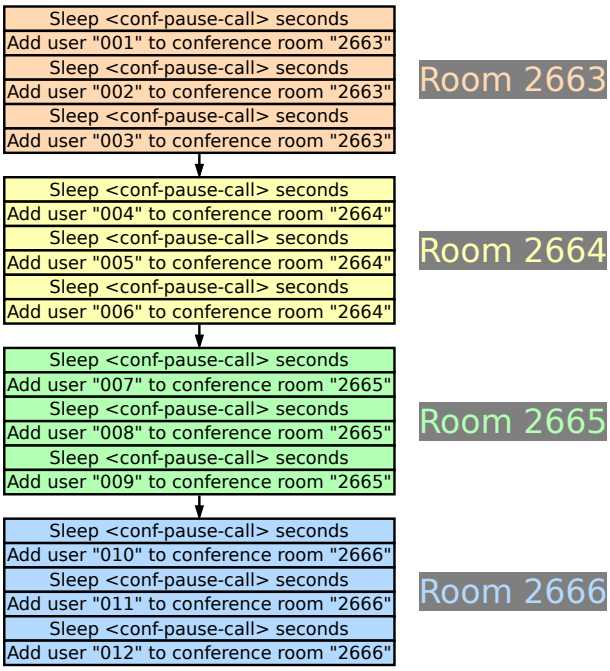

Figure 4.2: Conference rooms test example

This is implemented by using the sipp functionalities `call rates` (parameter -r) and `rate period` (parameter -rp):

Listing 8: sipp command for starting conf call tests

```
sipp -aa -r 1
    -i $local_ip
    -rp 60s
    -inf 'Users_conf-rooms.csv'
    -m $current_users
    -p 5061
    -mp 6020
    -sf 'Invite.xml'
    $ask_ip 2>&1"
```

`-rp 60s` is the rate period in seconds; `-r 1 -rp 60s` means that 1 user is added every 60 seconds. With this scenario, the following situation is simulated:

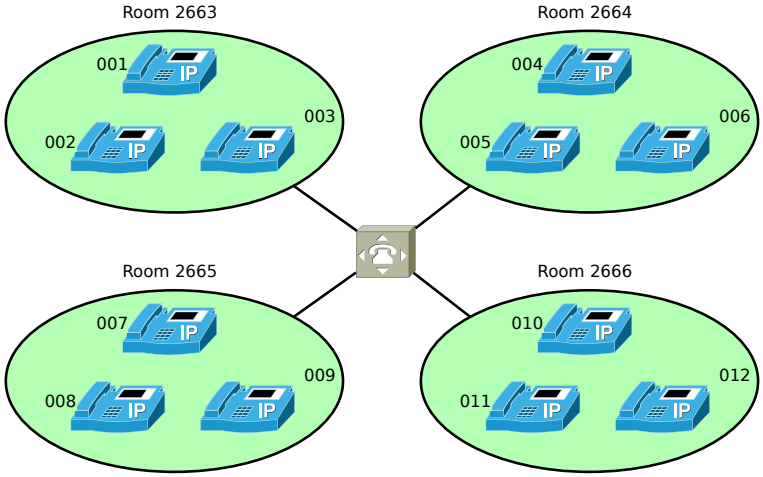

Figure 4.3: Conference rooms test illustration with 3 calls, 4 rooms

5 Maximum Participants in a Single Conference Room Test

The conference participants test was developed primarily for simulating one conference room with any number of participants. So, since the number of rooms is fixed, the number of calls has to be increased step-by-step:

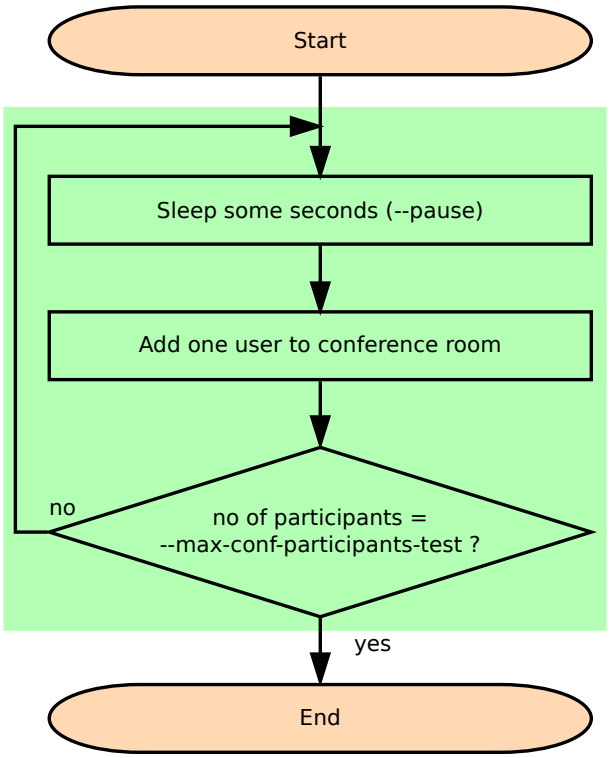

Figure 5.1: Process of conference participants test

For a conference participants test with five calls, the test would work like this:

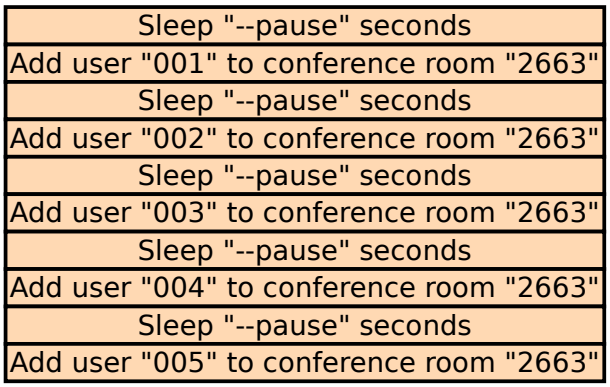

Figure 5.2: Conference participants test example

The command for executing `sipp` looks as follows:

Listing 9: sipp command for starting conference participants tests

```
sipp -r 1 -aa
  -rp 60s
  -inf 'Users_conf-participants.csv'
  -m $current_users
  -i $local_ip
  -p 5061
  -mp 6020
  -sf 'Invite.xml'
  $ask_ip 2>&1"
```

6 Watchdog Feature

The watchdog feature is a component that stops testing after a defined time. It is necessary because the AskoziaPBX does not respond any more if it reaches its limit. Is is implemented like that:

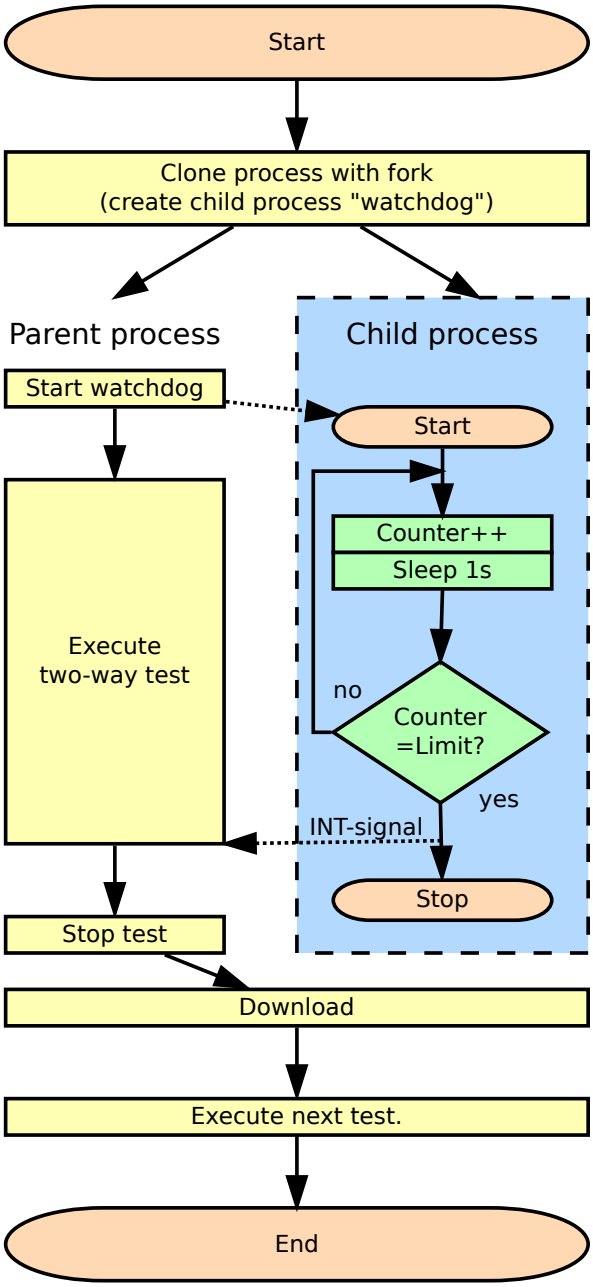

Figure 6.1: Basic watchdog process

Of course, it is not as easy as it is shown in figure 6.1. First of all, there is only one watchdog process that is forked before beginning the tests. This process is started and stopped multiple times (one time for every test type):

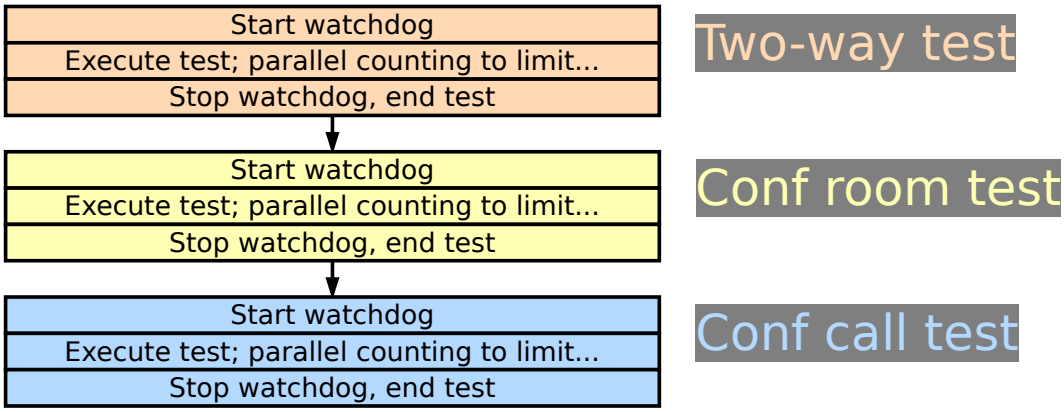

Figure 6.2: Starts and stops of watchdog

Let's sum up: Now there is a second process that is started before every test and stopped after every test. At the end of all tests, it is killed. But there is still one problem: Because of the variable parameters, the tests may have very different durations. So, it is necessary to tell the watchdog some settings of the test. For this reason, there is an IPC (Interprocess Communication) existing. After forking the main process, there is a pipe created to send messages from the parent (test) process to the child (watchdog). It can be treated like a normal print device in perl, so it is possible to send messages like this:

Listing 10: Send messages to watchdog

```
print $pipe $message;
```

It is possible to send to following commands to the watchdog. All commands have to end with a newline character for flushing the pipe:

Testtype	Needed users
`set pause '3'`	Set pause time to 3 seconds (necessary for call duration calculation)
`set users '5'`	Set number of users to 5 (necessary for call duration calculation)
`start watchdog`	Starts watchdog: Begin of incrementing counter per second
`stop watchdog`	Stops watchdog (incrementing counter)
`Tests finished.`	Terminates watchdog process.

After sending a command, the counter is reset to zero automatically (there is no provision for sending commands to the watchdog while running a test). Sometimes, there were problems if two commands are sent in quick succession, so it is recommended to sleep one second between sending two commands.

7 Recording CPU Load

Recording the CPU load is executed by a self-programmed tool of Michael Iedema (michael-laskozia.com). Its name is `qstat` and is available by pressing the `ESC` key somewhere on the Askozia webpage. Then, in the "Beta Features" tab, there is a link referencing to `debug_qstat.php`:

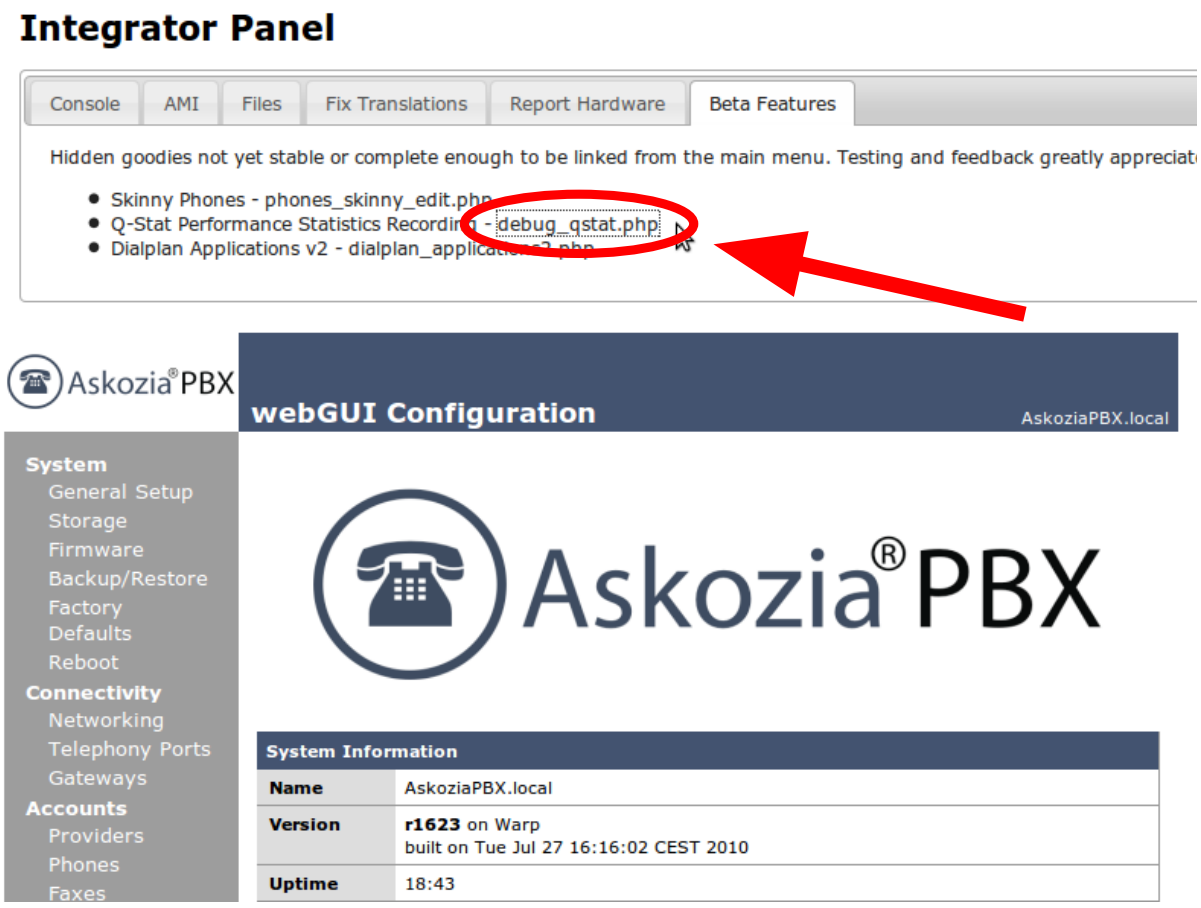

Figure 7.1: Starting qstat manually

On the debug qstat page, there is only one button labelled with `Start`. So, a click on this button starts CPU load recording by qstat. The button changes top `Stop` automatically and terminates CPU load recording by clicking on it. After this, there appears a downloadable file (... .qstat) on the webpage. It contains the recorded qstat data and looks like this:

The recorded data comprise multiple CPU load values. The script uses the CPU idle time by subtracting it from 100. They were verified by using `top` on the AskoziaPBX.

```
# qstat statistics file
#
# qstat version: v0.2.0
# timestamp: 19700101192334
#
# description:
#
# asterisk build info:
#   hostname:
#   kernel:   2.6.30.3-inside-t2-sandbox
#   machine:  ppc
#   os:       Linux
#   date:     2010-07-27 14:07:40 UTC
#   user:     root
#
# fields:
# seconds | processed calls | active calls | active channels | active ram (kB) | load avg (1 min)
#  | cpu user | cpu nice | cpu system | cpu idle | cpu iowait | cpu irq | cpu softirq | cpu steal
0 0 0 0 10480 0.080000 8.910892 0.000000 3.960396 85.148514 0.000000 0.000000 1.980198 0.000000
1 0 0 0 10484 0.080000 0.000000 0.000000 0.000000 100.000000 0.000000 0.000000 0.000000 0.000000
2 0 0 0 10484 0.080000 0.000000 0.000000 0.000000 100.000000 0.000000 0.000000 0.000000 0.000000
3 0 0 0 10484 0.080000 0.000000 0.000000 0.000000 100.000000 0.000000 0.000000 0.000000 0.000000
4 0 0 0 14464 0.150000 70.297035 0.000000 3.960396 24.752476 0.000000 0.000000 0.990099 0.000000
```

Figure 7.2: QStat results

A Appendix

A.1 Developers Parameters

`--askozia-confpage=<string>`
: Name of the PHP page for downloading/uploading the XML Configuration file
Default: `system_backup.php`
Example: `--askozia-confpage=system_backup.php`

`--askozia-realm=<string>`
: Name of the authentication-realm of the AskoziaPBX.
Default: `Web Server Authentication`
Example: `--askozia-realm='Web Server Authentication'`

`--sipp-path=<string>`
: This parameter allows one to override the sipp executable which is used to execute the performance tests. It is recommended to use the supplied version because it provides a standardized test base.
Default: `./PERF_TESTS/sipp` or, if not existing, the result of `which sipp`
E.g. `--sipp-path=../sipp` or `--sipp-path=/tmp/sipp`

`--two-party-user-file=<string>`
: For executing two-party tests with sipp, there has to be a so-called injection file. This is a csv file which is used by sipp for generating multiple calls automaticly. It is created by the script and contains all needed information (and not more) for sipp. It is possible to change the filepath and -name of this file by specifying this parameter and may be absolute or relative.
Default: `./results/<testname>/Users_two-party.csv`
E.g. `--users-2way-file=../Users_two-party.csv`
or `--users-2way-file=/tmp/Users_two-party.csv`

`--participants-user-file=<string>`
: Please have a look at the parameter `--users-2way-file`. This parameter is the same only for conference tests with fixed number of rooms.
Default: `./results/<testname>/Users_conf_room.csv`
E.g. `--users-conf-room-file=../Users_conf_room.csv`
or `--users-conf-room-file=/tmp/Users_conf_room.csv`

`--room-user-file=<string>`
: Please have a look at the parameter `--users-2way-file`. This parameter is the same only for conference tests with fixed number of calls.
Default: `./results/<testname>/Users_conf_call.csv`
E.g. `--users-conf-call-file=../Users_conf_call.csv`
or `--users-conf-call-file=/tmp/Users_conf_call.csv`

`--reg-scen=<string>`

This parameter specifies the path to the register scenario used by sipp. The register scenario is needed for two-party tests only. For more information, please have a look at chapter 3.

Default: `./PERF_TEST_FILES/Register.xml`

E.g. `--reg-scen=../Register.xml` or `--reg-scen=/tmp/Register.xml`

`--dereg-scen=<string>`

This parameter specifies the path to the deregister scenario used by sipp. The deregister scenario is needed for two-party tests only. For more information, please have a look at chapter 3.

Default: `./PERF_TEST_FILES/Deregister.xml`

E.g. `--dereg-scen=../Deregister.xml` or `--dereg-scen=/tmp/Deregister.xml`

`--acc-scen=<string>`

This parameter specifies the path to the accept scenario used by sipp. The accept scenario is needed for two-party tests only. For more information, please have a look at chapter 3.

Default: `./PERF_TEST_FILES/Accept.xml`

E.g. `--acc-scen=../Accept.xml` or `--acc-scen=/tmp/Accept.xml`

`--inv-scen=<string>`

This parameter specifies the path to the invite scneario used by sipp. The invite scenario is needed for every test type. For more information, please have a look at the description of the different testtypes (chapters 3, 4 and 5).

Default: `./PERF_TEST_Files/Invite.xml`

E.g. `--inv-scen=../Invite.xml` or `--inv-scen=/tmp/Invite.xml`

`--sip-src-port=<number>`

Port of the local computer (testcomputer) to communicate with Askozia. It is used for User A in two-party tests and for all users in conference tests. During the tests, there was a softphone running in background for communication in the office. Because of this, sipp was not able to reserve the usual sip port 5060.

Default: `5061`

E.g. `--sip-src-port=5061`

`--sip-dst-port=<number>`

Port of the local computer (testcomputer) to communicate with Askozia, but this time only for User B in two-party tests. The first sipp process (User A) blocks one port for communication with AskoziaPBX, so the second sipp process (User B) needs another one to talk to Askozia. This is necessary for two-party tests only.

Default: `5062`

E.g. `--sip-dst-port=5062`

`--rtp-src-port=<number>`

Port of the local computer for establishing RTP streams between the local testcomputer and AskoziaPBX. This one is used by User A of two-party tests and by all users of conference tests. Sipp was not able to use the standard port because of a softphone running on the testcomputer in background.
Default: `6020`
E.g. `--rtp-src-port=6020`

`--rtp-dst-port=<number>`

Port of the local computer for establishing RTP streams between the local testcomputer and AskoziaPBX. This one is used by User B of two-party tests only, so it not needed for conference testing.
Default: `6030`
E.g. `--rtp-dst-port=6030`

`--restore=<string>`

After the test, the AskoziaPBX is strongly reconfigured. There are possibly hundreds of testusers and some new conference rooms. To avoid cleaning up by hand, this parameter helps to reconfigure the box after the test. There are three possible values:

none The AskoziaPBX is not reconfigured.

old-config The AskoziaPBX is restored with the config existing before the test.

factory-defaults The AskoziaPBX is set to factory-defaults.

Default: `old-config`
E.g. `--restore=none` or `--restore=factory-defaults`

`--gnuplot-exe=<string>`

The path to the gnuplot executable for drawing graphs of the results at the end of the test. Has to be installed (`which gnuplot` determines path) or specified if the graphs should be drawed. If not installed and specified (undefined), there is no possibility to draw the graphs. The test can nevertheless be executed.
Default: result of `which gnuplot` or, if not existing, undefined
E.g. `--gnuplot-exe=../gnuplot` or `--gnuplot-exe=/tmp/gnuplot`

`--two-party-user-file=<string>`

For executing two-party tests with sipp, there has to be a so-called injection file. This is a csv file which is used by sipp for generating multiple calls automaticly. It is created by the script and contains all needed information (and not more) for sipp. It is possible to change the filepath and -name of this file by specifying this parameter and may be absolute or relative.
Default: `./results/<testname>/Users_two-party.csv`
E.g. `--users-2way-file=../Users_two-party.csv`
or `--users-2way-file=/tmp/Users_two-party.csv`

`--participants-user-file=<string>`
: Please have a look at the parameter `--users-2way-file`. This parameter is the same only for conference tests with fixed number of rooms.
Default: `./results/<testname>/Users_conf_room.csv`
E.g. `--users-conf-room-file=../Users_conf_room.csv`
or `--users-conf-room-file=/tmp/Users_conf_room.csv`

`--room-user-file=<string>`
: Please have a look at the parameter `--users-2way-file`. This parameter is the same only for conference tests with fixed number of calls.
Default: `./results/<testname>/Users_conf_call.csv`
E.g. `--users-conf-call-file=../Users_conf_call.csv`
or `--users-conf-call-file=/tmp/Users_conf_call.csv`

`--reg-scen=<string>`
: This parameter specifies the path to the register scenario used by sipp. The register scenario is needed for two-party tests only. For more information, please have a look at chapter 3.
Default: `./PERF_TEST_FILES/Register.xml`
E.g. `--reg-scen=../Register.xml` or `--reg-scen=/tmp/Register.xml`

`--dereg-scen=<string>`
: This parameter specifies the path to the deregister scenario used by sipp. The deregister scenario is needed for two-party tests only. For more information, please have a look at chapter 3.
Default: `./PERF_TEST_FILES/Deregister.xml`
E.g. `--dereg-scen=../Deregister.xml` or `--dereg-scen=/tmp/Deregister.xml`

`--acc-scen=<string>`
: This parameter specifies the path to the accept scenario used by sipp. The accept scenario is needed for two-party tests only. For more information, please have a look at chapter 3.
Default: `./PERF_TEST_FILES/Accept.xml`
E.g. `--acc-scen=../Accept.xml` or `--acc-scen=/tmp/Accept.xml`

`--inv-scen=<string>`
: This parameter specifies the path to the invite scneario used by sipp. The invite scenario is needed for every test type. For more information, please have a look at the description of the different testtypes (chapters 3, 4 and 5).
Default: `./PERF_TEST_Files/Invite.xml`
E.g. `--inv-scen=../Invite.xml` or `--inv-scen=/tmp/Invite.xml`

`--sip-src-port=<number>`
: Port of the local computer (testcomputer) to communicate with Askozia. It is used for User A in two-party tests and for all users in conference tests. During the tests, there

was a softphone running in background for communication in the office. Because of this, sipp was not able to reserve the usual sip port 5060.
Default: `5061`
E.g. `--sip-src-port=5061`

`--sip-dst-port=<number>`
Port of the local computer (testcomputer) to communicate with Askozia, but this time only for User B in two-party tests. The first sipp process (User A) blocks one port for communication with AskoziaPBX, so the second sipp process (User B) needs another one to talk to Askozia. This is necessary for two-party tests only.
Default: `5062`
E.g. `--sip-dst-port=5062`

`--rtp-src-port=<number>`
Port of the local computer for establishing RTP streams between the local testcomputer and AskoziaPBX. This one is used by User A of two-party tests and by all users of conference tests. Sipp was not able to use the standard port because of a softphone running on the testcomputer in background.
Default: `6020`
E.g. `--rtp-src-port=6020`

`--rtp-dst-port=<number>`
Port of the local computer for establishing RTP streams between the local testcomputer and AskoziaPBX. This one is used by User B of two-party tests only, so it not needed for conference testing.
Default: `6030`
E.g. `--rtp-dst-port=6030`

`--restore=<string>`
After the test, the AskoziaPBX is strongly reconfigured. There are possibly hundreds of testusers and some new conference rooms. To avoid cleaning up by hand, this parameter helps to reconfigure the box after the test. There are three possible values:

none The AskoziaPBX is not reconfigured.

old-config The AskoziaPBX is restored with the config existing before the test.

factory-defaults The AskoziaPBX is set to factory-defaults.

Default: `old-config`
E.g. `--restore=none` or `--restore=factory-defaults`

A.2 XML Scenarios

A.2.1 REGISTER Scenario

A.2.2 De-REGISTER Scenario

A.2.3 INVITE Scenario

A.2.4 ACCEPT Scenario

A.3 Example Configuration File

A.4 SIPP Manpage