Hochschule Ostfalia
Fakultät Elektrotechnik
Labor für Kommunikationssysteme

Ostfalia
Hochschule für angewandte Wissenschaften

AskoziaPBX-Performancetests

Written by

Mark Stephan

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

23.07.2010

Inhaltsverzeichnis

Listings

1 Introduction

This subproject of AskoziaPBX was developed for executing performance- and stress-tests to different Askozia-boxes. It was written by Mark Stephan (mark.stephan@askozia.com) during a job as a student assistant for the Askozia-founder Michael Iedema (michael@askozia.com) in Spring/Summer 2010.

It is very important to understand the differences between the three test-types (two-way-calls, conference calls with fixed number of calls and conference calls with fixed number of rooms). It is explained detailed in the first paragraphs of chapters 3, 4 and 5.

1.1 Problem

The AskoziaPBX software can be downloaded as firmware image for embedded systems and as 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. So we had to develop an algorithm to find out how tough the current Askozia box is.

1.2 Features

The current testsuite supports the following features:

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

  two-way calls The testsuite establishes a variable number of A-to-B (or end-to-end) calls between the AskoziaPBX and the testsystem. This test simulates normal telephone calls between two persons.

  conference-tests with fixed number of calls The testsuite calls a variable number of different conference rooms with a fixed number of users in each room.

  conference-tests with fixed number of rooms The testsuite calls a fix number of conference rooms with a variable number of users in each rooms.
- monitoring of the CPU-load of the AskoziaPBX caused by the testcalls
- download of the recorded CPU-load data
- interpretation and creation of graphs of the testresults

1.3 Usage

The script can be called in the command line, e.g.:

Listing 1: Script call

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

`--local-ip=<IP>`
: The IP-adress of the testcomputer that executes the testscript. <IP> stands for the address of the device connected to the AskoziaPBX.
Standard: undefined (**must** be defined)
Example: `--local-ip=192.168.0.2`

`--askozia-ip=<IP>`
: The IP-address of the Askozia box that should be tested.
Standard: `10.10.10.1`
Example: `--askozia-ip=192.168.0.1`

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

`--askozia-confpage=<string>`
: Name of the php-page for down-/uploading the XML-configfile
Standard: `system_backup.php`
Example: `--askozia-confpage=system_backup.php`

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

`--askozia-user=<string>`
: Name of the root-user of the AskoziaPBX (necessary for executing commands using the webserver).
Standard: `admin`
Example: `--askozia-user=admin`

`--askozia-pw=<string>`
: Password for the root-user of the AskoziaPBX.
Standard: `askozia`
Example: `--askozia-pw=askozia`

`--save-sipp-log=<string>`

The output generated by the testprogram `sipp` can be saved in a file for debugging. The path to the file where the output should be saved can be specified in the passed string and may be absolute or relative to the subdirectory. `./results/<testname>/`. If not specified, the output is ignored.
Standard: undefined (output ignored)
Example: `--save-sipp-log=../sipp.log` or `--save-sipp-log=/tmp/sipp.log`

`--debug`

Activates debug messages. Have a look at the option `--save-debug`.
Standard: undefined (no debug output)
Example: `--debug`

`--save-debug=<string>`

Because of the many debug messages, thus can be saved in a file. The string is the relative or absolute path to the file where the messages should be saved. The cwd for this file is `./results/<testname>/`, this means, if `debug.txt` is entered, the path to this file is `./results/<testname>/debug.txt`.
Standard: undefined (no saving)
Example: `--save-debug=../debug.txt` or `--save-debug=/tmp/debug.txt`

`--2way-calls=<number>`

This parameter defines how many two-leg calls should be established. During the test, the testsuite will increase the current number of calls from one till the number of this parameter. For more information, please have a look at chapter 3.
Standard: undefined (no two-way tests)
Example: `--2way-calls=30`

`--2way-pause=<number>`

This parameter defines the time to wait between increasing the number of users in the test. This means, every x seconds is a new user added to the test. For more information, please have a look at 3.
Standard: undefined (**must** be defined for a two-way test)
Example: `--2way-pause=10`

`--conf-calls-room=<number>`

Pronounced, this parameter is called "number of conference calls for conference tests with fixed number of rooms". It is the maximum number of users per conference room in a test where the number of existing conference rooms is fixed. So - if the number of conference rooms is fixed in this test, logically the number of users has to increase. The testsuite starts with one user and ends with this passed value. For more information, please have a look at chapter 5.
Standard: undefined (no conference-tests with fixed number of rooms)
Example: `--conf-calls-room=10`

`--conf-rooms-room=<number>`

Pronounced, this parameter is called "number of conference room for conference tests with fixed number of rooms". This means the number of conference rooms existing during a conference test with fixed rooms - it is constant for this complete test. For a more understable explanation, please have a look at chapter 5.
Standard: `1`
Example: `--conf-rooms-room=1`

`--conf-pause-room=<number>`

This parameter defines the time in seconds between adding new users in a conference test with fixed number of conference rooms.
Standard: undefined (**must** be defined for a conf room test)
Example: `--conf-pause-room=10`

`--conf-calls-call=<number>`

Pronounced, this parameter is called "number of conference calls for conference tests with fixed number of calls" - so it is constant during the whole conf call test. It means the maximum number of users exisiting in each conference room in this test. For more information, please have a look at chapter 4.
Standard: undefined (no conference-tests with fixed number of calls)
Example: `--conf-calls-call=3`

`--conf-rooms-call=<number>`

Pronounced, this parameter is called "number of conference rooms for conference tests with fixed number of conference rooms". It means the number of currently existing conference rooms during a conf room test. For more information, please have a look at chapter 4.
Standard: `1`
Example: `--conf-rooms-call=10`

`--conf-pause-call=<number>`

This parameter defines the time in seconds between adding new users in a conference test with fixed number of conference calls per conference room.
Standard: undefined (**must** be defined for a conf call test)
Example: `--conf-pause-call=10`

`--sipp-exe=<string>`

This parameter specifies the sipp-executable that should be used to execute the performance-tests. It is recommended to use the supplied version because it is self compiled and does not crash if its parameter `-aa` is used contrary to the repositories-version. String is the path to the sipp-exe that should be used; it may be relative or absolute.
Standard: `./PERF_TESTS/sipp` or, if not existing, the result of `which sipp`
E.g. `--sipp-exe=../sipp` or `--sipp-exe=/tmp/sipp`

`--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.
Standard: result of `which gnuplot` or, if not existing, undefined
E.g. `--gnuplot-exe=../gnuplot` or `--gnuplot-exe=/tmp/gnuplot`

`--users-2way-file=<string>`

For executing two-way 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.
Standard: `./results/<testname>/Users_two-way.csv`
E.g. `--users-2way-file=../Users_two-way.csv`
or `--users-2way-file=/tmp/Users_two-way.csv`

`--users-conf-room-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.
Standard: `./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`

`--users-conf-call-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.
Standard: `./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-way tests only. For more information, please have a look at chapter 3.
Standard: `./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-way tests only. For more information, please have a look at chapter 3.
Standard: `./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-way tests only. For more information, please have a look at chapter 3.
Standard: `./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).
Standard: `./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-way 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.
Standard: `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-way 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-way tests only.
Standard: `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-way 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.
Standard: `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-way tests only, so it not needed for conference testing.
Standard: `6030`
E.g. `--rtp-dst-port=6030`

`--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>/`. If undefined, the files will be saved in the results directory directly, so it will be messy soon.
Standard: undefined (direct saving of results in subdir `./results`)
E.g. `--testname=2010-01-01_1030`
(saving of results in subdir `./results/2010-01-01_1030/`)

`--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.

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

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

For <options>, there are many possibilities. Some are optional (like the name says), but some them you have to specify. It is necessary to specify your own IP (`--local-ip`) and one of the parameters `--2way-calls`, `--conf-calls-call` or `--conf-calls-room` because these three parameters define which test should be executed. If none is specified, the script does not know what to test. Furthermore, the pause time of the tests that should be executed has to be specified.

You have to be root to execute this script because `sipp` reserves port for its connection to Askozia.

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 needs the following packages including their own dependencies:

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

2 Configuration of the AskoziaPBX

This paragraph is about the configuration of the AskoziaPBX. There is a complete configuration file in the appendix. First, the used configfile 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.

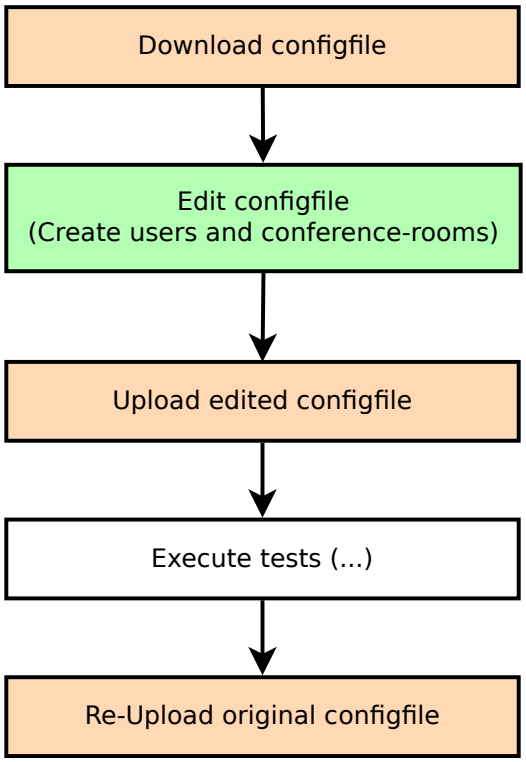

Abbildung 2.1: Process of editing the Askozia-configfile

2.1 Down-/Upload Configfile

For downloading the configfile, it is necessary to send a HTML POST request to Askozia. Or, to be more precise, it has to be sent to the qstat-page of the Askozia box. The useragent has to be authenticated as root and the content type must be "multipart/form-data". With this POST request, Askozia sends the output of the qstat page. For downloading the configfile, the parameter "Download" has to be set to any desired value – but it 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 configfile

```
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 has to be expected:

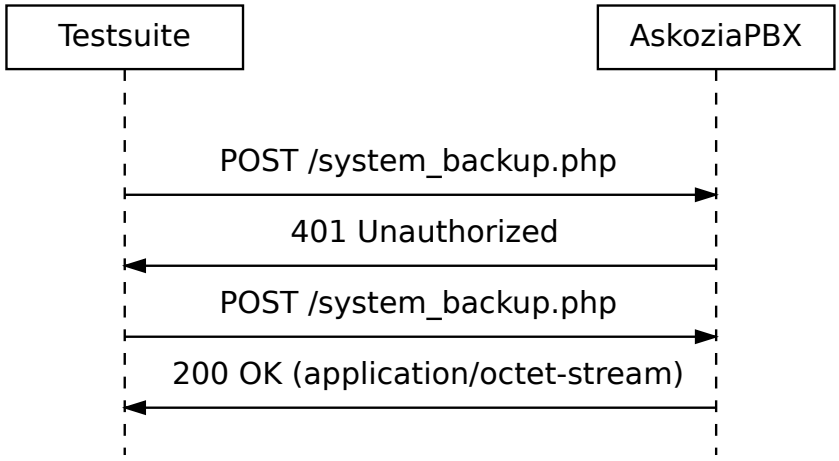

Abbildung 2.2: Dataflow of configfile download

The `200 OK` message sent by Askozia includes the configfile 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 configfile is saved with the original named followed by an `_edited` string. This edited configfile is now uploaded to the AskoziaPBX as follows:

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 configfile

```
$res = $ua->request (POST "http://$ask_ip:$ask_port/
   $ask_conf_page", "Content-Type" => "multipart/form-data",
```

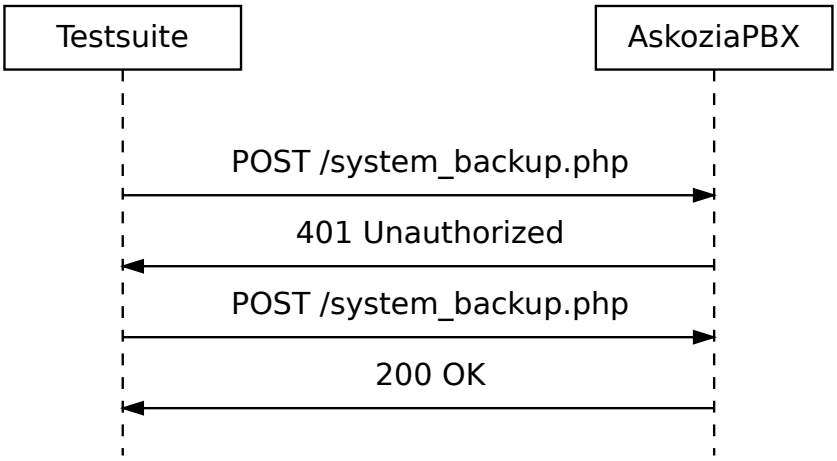

Abbildung 2.3: Dataflow of configfile upload

```
Content => [ Restore => "1", conffile => [
   $xml_configfile."_edited" ] ]);
```

2.2 Users

For executing the planned tests, it is necessary to add some testusers to the AskoziaPBX. The count of users depends on the parameters that were passed to the script when launching. Here is a table of needed users, the highest count will be added:

Testtype	Needed 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 begin of the `sipp` paragraph Then, it adds its information to the existing phones. It is not checked whether the added phones already exist. The template for adding users looks as follows, where `_userno_` is replaced by an integer that is incremented with each user:

Listing 4: User template

```
<phone>
<extension>User_userno_</extension>
<callerid>Performance-Test Testuser _userno_</callerid>
<codec>ulaw</codec>
```

```
<codec>alaw</codec>
<codec>gsm</codec>
<secret>0815</secret>
<uniqid>SIP-PHONE-1357012154bbefaaa79d96_userno_</uniqid>
<language>de-de</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. So the script searches for the begin of the `conferencing` paragraph and adds its needed rooms. The template looks as follows, where `_roomno_` is replaced by an integer that is incremented with each room:

Listing 5: Conference room template

```
<room>
<number>_roomno_</number>
<name>Default Conference</name>
<uniqid>CONFERENCE-ROOM-914902610465bd5b50d0c6_roomno_</
   uniqid>
</room>
```

Conference rooms start with number 2663. So, for a conference test that needs ten conference rooms, there are conference rooms 2663 till 2672 created.

3 Two-way calls

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

In `sip` diagram, the scenario looks as follows:

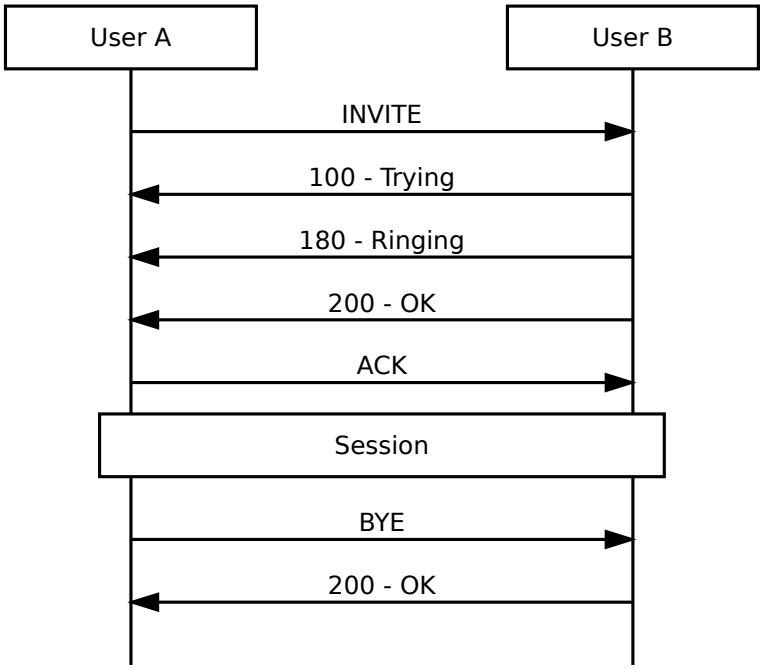

Abbildung 3.1: SIP diagram of a two-way call

This scenario is implemented like that, the original xml scenarios are deposed in the appendix:

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

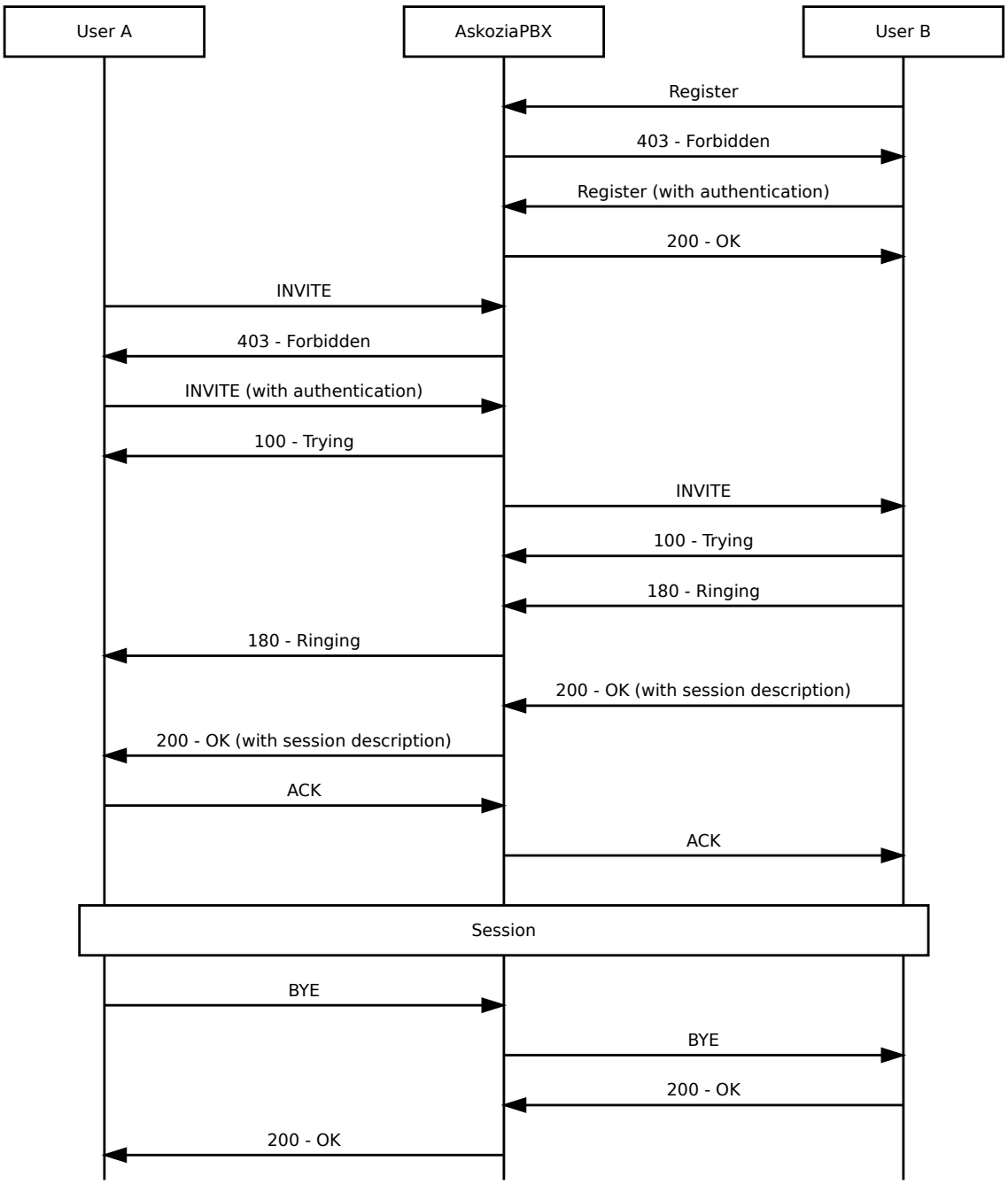

Abbildung 3.2: Scenario of a two-way call

The number of calls is increased step-by-step. After every call, the script waits for the with the script call passed parameters time to record the CPU load values. For executing two-way calls, there are the following commands used:

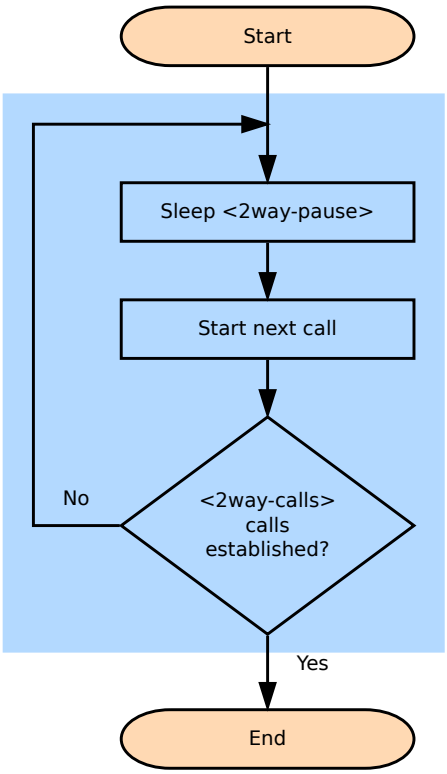

Abbildung 3.3: Complete two-way test

Listing 6: sipp command for inviting User B

```
my $constant = "'$sipp' -aa -inf '$users_twoway_file' -m
   $current_call -i $local_ip ";
my $reg_cmd = $constant . "-p $sip_dst_port -mp
   $rtp_dst_port -sf '$reg_scen' $ask_ip 2>&1";
my $acc_cmd = $constant . "-p $sip_dst_port -mp
   $rtp_dst_port -sf '$acc_scen' -bg $ask_ip 2>&1 &";
my $inv_cmd = $constant . "-p $sip_src_port -mp
   $rtp_src_port -sf '$inv_scen' $ask_ip 2>&1";
my $der_cmd = $constant . "-p $sip_dst_port -mp
   $rtp_dst_port -sf '$der_scen' $ask_ip 2>&1";
```

`$constant` is used because some settings of the current sipp call are needed in all other calls (register, accept, invite and deregister), too. So, it is not neccessary (and more comfortable) to write them down once and not four times. You can inform yourself about the used parameters by reading the sipp manpage (the appendix contains the sipp manpage).

4 Conference-Calls with fixed number of calls

Conference calls with fixed number of calls were developed chiefly for executing three-way conferences. Basically, there is a conference with y participants started. After that, there is another conference with y participants started until the maximum number of conferences is reached:

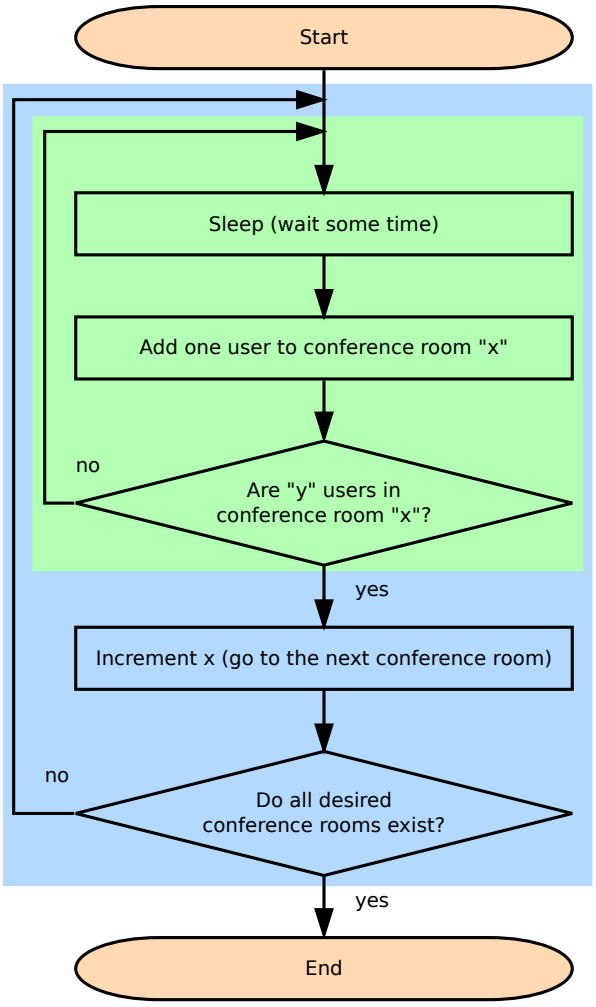

Abbildung 4.1: Process of conference call tests

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

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

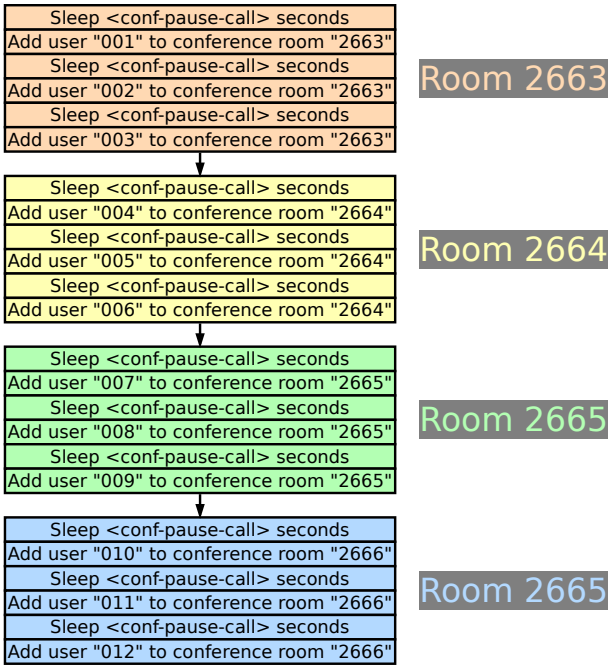

Abbildung 4.2: Conference call test example

Listing 7: sipp command for starting conf call tests

```
$sipp -aa -r 1 -i $local_ip
  -rp $conf_pause_call s
  -inf '$users_conf_call_file'
  -m $current_users
  -p $sip_src_port
  -mp $rtp_src_port
  -sf '$inv_scen'
  $ask_ip 2>&1
```

`-rp $conf_pause_call s` is the rate period in seconds; it has to be `-rp 3s` or `-rp 10s` for example. `-r 1 -rp $conf_pause_call` means that 1 user is added every `conf-pause-call` seconds. With this scenario, the following situation is simulated (remember: each `conf-room` has `conf-calls` calls – here, each of the 4 rooms has 3 calls):

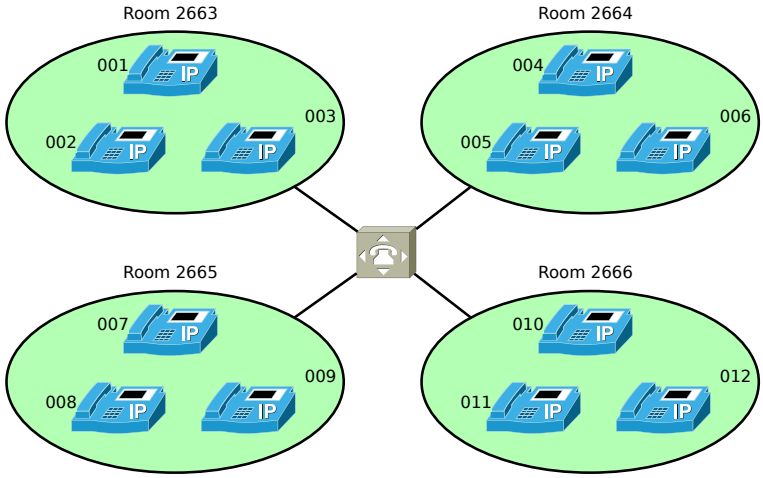

Abbildung 4.3: Conference call illustration with 3 calls, 4 rooms

5 Conference-Calls with fixed number of rooms

6 Watchdog-Feature

A Appendix

A.1 Example tests

A.2 XML scenarios

A.2.1 Register scenario

A.2.2 Deregister scenario

A.2.3 Invite scenario

A.2.4 Accept scenario

A.3 Example configuration file

A.4 SIPP manpage