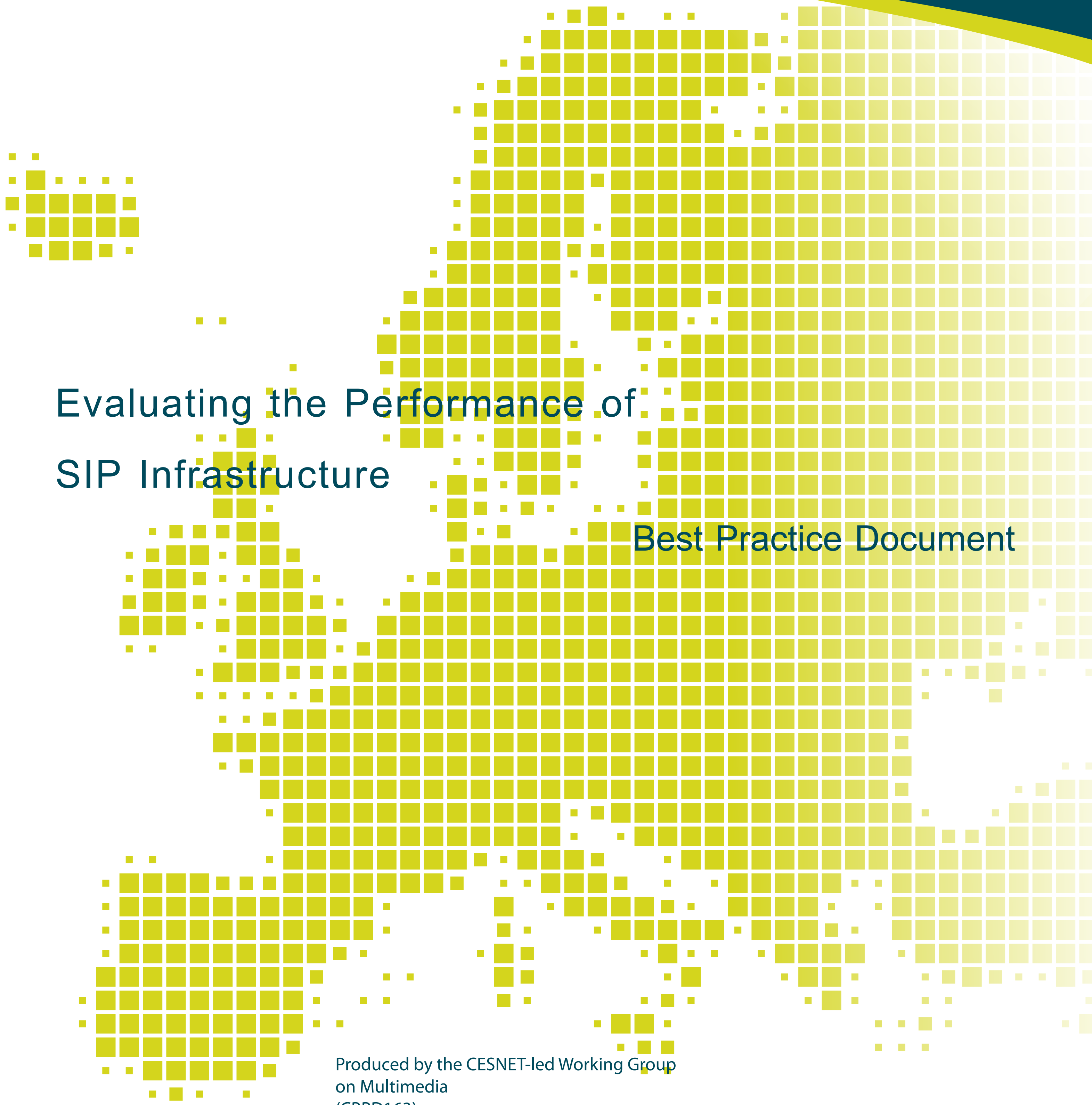
Evaluating the Performance of SIP Infrastructure

Best Practice Document



Produced by the CESNET-led Working Group on Multimedia
(CBPD163)

Miroslav VOZNAK
September 2011

Document No:	GN3-NA3-T4-CBPD163
Version / date:	September 30, 2011
Original language :	EN
Original title:	“Evaluating the Performance of SIP infrastructure”
Original version / date:	V1.1 of September 30, 2011
Contact:	voznak@cesnet.cz

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 238875, relating to the project 'Multi-Gigabit European Research and Education Network and Associated Services (GN3)'.

Table of Contents

Summary

The report deals with performance testing of a SIP infrastructure. While a working methodology for measuring the performance and effectiveness of the SIP B2BUA and SIP Proxy has recently been introduced, best practice lacks a more complex application of this methodology. This performance-testing method fills this gap and improves the methodology so that it fits into the planned modular design of the testing platform, which is being designed by the Czech NREN (National Research and Education Network).

Today, measurement of the performance and effectiveness of SIP infrastructure relies on the administrator's skills and experience with telephony infrastructure, but this experience, however good it is, could fall short, due to an increasing demand for the implementation of special services like queues, IVRs, transcoding, and conferences. The increasing utilisation of SIP-based VoIP technologies in the rapidly growing development of converged networks are the main reasons for creating a methodology that would allow administrators to measure the performance of SIP servers precisely, and for comparing this new methodology to other software and hardware platforms. There are two ways to address this issue – to rely on special, proprietary solutions, or to try to utilise some of the tools provided by the open-source software community to develop a new methodology. Our work embraces the second approach. It is based on the knowledge provided by the IT company, Transnexus, which created some useful test scenarios using the capabilities of the SIP testing tool called SIPp, and enhances it by the methodology described in RFC drafts, thus making it a fully open-source solution. This solution is then used to measure the performance of both variants of a SIP server implementation – SIP Proxy and B2BUA. Moreover, this work compares the results taken, both when transcoding was in use and when it was not, and it provides the means for an independent comparison of B2BUAs platforms. By separating registrations from calls, we were able to measure both cases without the need for extensive post-processing of data to ensure that the data in one case was not affected by the data from another case. Furthermore, the security-vulnerability of the SIP protocol has been harnessed to facilitate the measurement of the performance of the software that performs both registrations and calls together but in individual processes. This facility provides the basis for the modular design of the platform.

In this report, the results from analyses of separate registration stress tests are presented as examples of applications of this methodology including samples of collected data, in the easily readable form of charts. The author presents results achieved by the CESNET-led working group on Multimedia.

1 Introduction

We performed several series of tests on multiple, different platforms using the proposed methodology for testing and benchmarking SIP infrastructure. From these tests, we realised that it would be very beneficial to modify the existing testing platform to allow us to perform separate test scenarios on each of the important SIP dialogues, initiating a movement towards development of a modular design. At the beginning of 2011, the new RFC 6076 was adopted, finally standardising the most essential measurement parameters.

With these standardised parameters, we developed the most important testing scenarios – the registration test scenario and the call test scenario, both rooted in previously used scenarios for complex performance measurement [voz133], [voz222] and [roz].

Each of those scenarios offers a different perspective for defining the limits of the SIP server and can be run separately, either to test special environments or events or to simulate the behaviour of real VoIP clients. The latter presented a big challenge, because the testing software does not inherently allow the running of multiple scenarios simultanously. However, exploiting the vulnerability of SIP security, which allows a client from another address to register, provided a work-around and the basis for the creation of a module-based testing platform.

In this report, we present the results of the testing of two different versions of the most commonly used VoIP PBX, Asterisk, and focus on its ability to handle multiple, simultaneous registrations coming in several consecutive bursts. This example is particularly useful in determining how the SIP server react to network failure and the consequent restoration of full connectivity, when all the clients try to register at once. In these examples, the way that the SIP server responds to bursts with high loads can be determined and all the conclusions can be made according to information obtained from measurements made on the client side. Measurements on the server side are often impossible because of the restrictions of the provider(s).

Implementations of this protocol are now commonplace, because of the integration of the SIP protocol in next generation networks. This results in an increasing need for a methodology and tools to determine whether the selected hardware has enough performance capability to provide for all the needs of the given environment, (with some margin for future expansion). However, at present, this demand can be only be satisfied by proprietary solutions, which, on one hand require special hardware, and on the other, do not provide results that would be comparable with the results achieved by other competitive solutions [roz].

On the other hand, open-source solutions do not provide the methodology for performing such tests and simply pass the definition of procedures and parameters for the particular test to the user. Though this restriction may look like a disadvantage, these solutions can be used to perform tests quite easily, under clearly defined conditions, during which precisely defined parameters can be measured. These conditions and parameters need to be defined and then implemented and both of these phases have already been partially solved. The definition of the parameters, conditions of the test and the whole methodology have been done in RFC 6076 and IETF drafts [por1], [por2] and these drafts will probably form the basis of the future IETF standards. Therefore, it is propitious to use the knowledge contained in them.

Most of the work on implementation has been done by the American IT company, Transnexus. This firm created a complex scenario for testing both of the most common variants of the SIP server [tran1], [tran2]. Their implementation uses the capabilities of the open-source testing tool program call SIPp. However, the implementation has severe limitations. First, it is designed primarily to test the environment, the basis of which is the proprietary server of this company. Second, the measured parameters and the overall methodology cannot recognise which of the measured time intervals belong to the same call. Thus, the output results are inconsistent and only raw conclusions can be drawn. The last deficiency of the Transnexus' test topology is the vast complexity of the design; many hardware components are required and too many variants of a call setup are tested. The complicated hardware requirements and the complexity of implementation of the software make the Transnexus design unsuitable for practical use.

Our work focuses on combining the best parts of the two named sources – IETF drafts [por1], [por2], recently issued as IETF RFC 6076 [mal] and the Transnexus design [tran1], [tran2]. The result of this combination is modified, so that the final solution is simple enough for the practical use and comprehensive enough to provide all the needed data to analyse the performance of the SIP server. This modification also includes the means for testing the B2BUA platform independently by harnessing a media flow. This report presents the complete methodology, as well as the design of a test scenario. Furthermore, the output results are included in the easily readable form of charts, and these charts are explained so that everyone who reads this report can easily understand the results of the tests of SIP server performance.

2 State of the Art

Currently, an administrator has two ways to carry out an analysis of the performance of VoIP infrastructure, He can rely on a proprietary solution (software or hardware), or he can use open-source testing tools.

The first choice offers good possibilities, because proprietary solutions offer comprehensible test scenarios and also come with easily readable, automatic output results, so the whole testing process is very user friendly. However, this choice also has also disadvantages. Proprietary solutions usually require the use of the specially design hardware, which uses special components, such as fast, digital signalling processors. Because of the need for these components, this hardware is not cost-effective. In addition, each producer uses his own methodology; therefore the output results from proprietary solutions from two different producers are incompatible and thus, also incomparable.

On the other hand, the second choice offers significant independence for the user, who is free to choose what will be measured and how. This freedom of choice can result in the same problematic incompatibility of results as the proprietary solutions. However, this freedom can easily be used in the testing scenario to reflect the parameters of any standard. Open-source testing tools differ from proprietary solutions, because these commercial companies will never be as flexible as the open-source community. A standard can be implemented to avoid incompatibility of the results. There are two drafts that focus on the terminology and methodology of performance testing [por1], [por2] and a recently issued standard, RFC 6076, of performance metrics [mal]. This reflects the growing importance of performance testing and attests that the bulk of the work on the standardisation of basic terms and methodologies has been already done. The results of these drafts should be implemented to ensure consistency in the future.

From the point of view of methodology, useful insight can be obtained from the testing scenario presented in the white papers of the IT company, Transnexus [tran1], [tran2]. Although these papers are somewhat outdated, the methodology they present can be useful in current work.

3 Methodology

In their testing platform, Transnexus used the open-source testing tool, SIPp, which enables the generation of a high SIP load and the measurement of the key parameters of individual SIP calls [sipp]. This makes SIPp the ideal option for testing SIP performance. Although Transnexus' benchmarking model served as an inspiration in the early phase of the development of our methodology, it lacks the potential for standardisation. The Transnexus tool measures the interval between transmission and reception of some key messages (e.g., Invite, 100 Trying, 180 Ringing). However, the approach does not look at these messages as a part of the SIP transaction. The user is unable to read the more complex attributes of the system from these results. To be more specific, with this tool one can learn how quickly the SIP server is able to respond to a message, but one cannot learn how quickly it can process and resend the message to the destination. Our approach makes this possible, so it is not the issue to recognize the "real world" parameters of the SIP server such as the time it takes to provision a call (later described as SRD).

From a practical point of view, the Transnexus model is rather too complex. As the commercial company, Transnexus has focused on creating a model that uses some of their commercial products, such as their management and billing platform, which requires two additional, separate computers. Moreover, the testing scenarios they created use several different end-locations for the simulation of call rejection, and do not take such contingencies as route issues or device problems into account. Again, this increases the complexity of the test platform due to the need for more physical machines. These restrictions clearly indicate that this model is unsuitable for practical use. From our point of view, it is beneficial to create a testing platform that is as simple as possible, and which can be easily deployed in any environment. Therefore, we decided not to use any specific hardware but simulate the end destination of calls by using the UASs. This is made possible by our principle aim to evaluate the ability of the SIP server to successfully connect the calling and called party [voz179], [voz222].

3.1 Measured Parameters

As mentioned in the Introduction, we used the parameters defined in RFC 6076 for all our measurements [mal]. However, we also use the parameters specified for the hardware. These groups of parameters were measured at the following locations:

- The first group was measured at the UAC and included the call statistics, such as the number of (un)successful calls and the duration of the message exchanges within particular transactions. RTP samples for analysis were also captured here;

- The second group – the hardware utilisation parameters –were measured directly on the SIP server. At this point, CPU and memory utilisation and network traffic were measured.

The complete list of all measured parameters includes:

- CPU utilisation;
- Memory utilisation;
- Number of (un)successful calls;
- Registration Request Delay – the time between first Register method and its related 200 OK response [mal];
- Session Request Delay (SRD) - the time between first Invite method and related 180 Ringing message [mal];
- Mean Jitter at Maximum RTP Packet Delay.

Figure. 1 shows the meaning of the RRD and SRD delays in more detail.

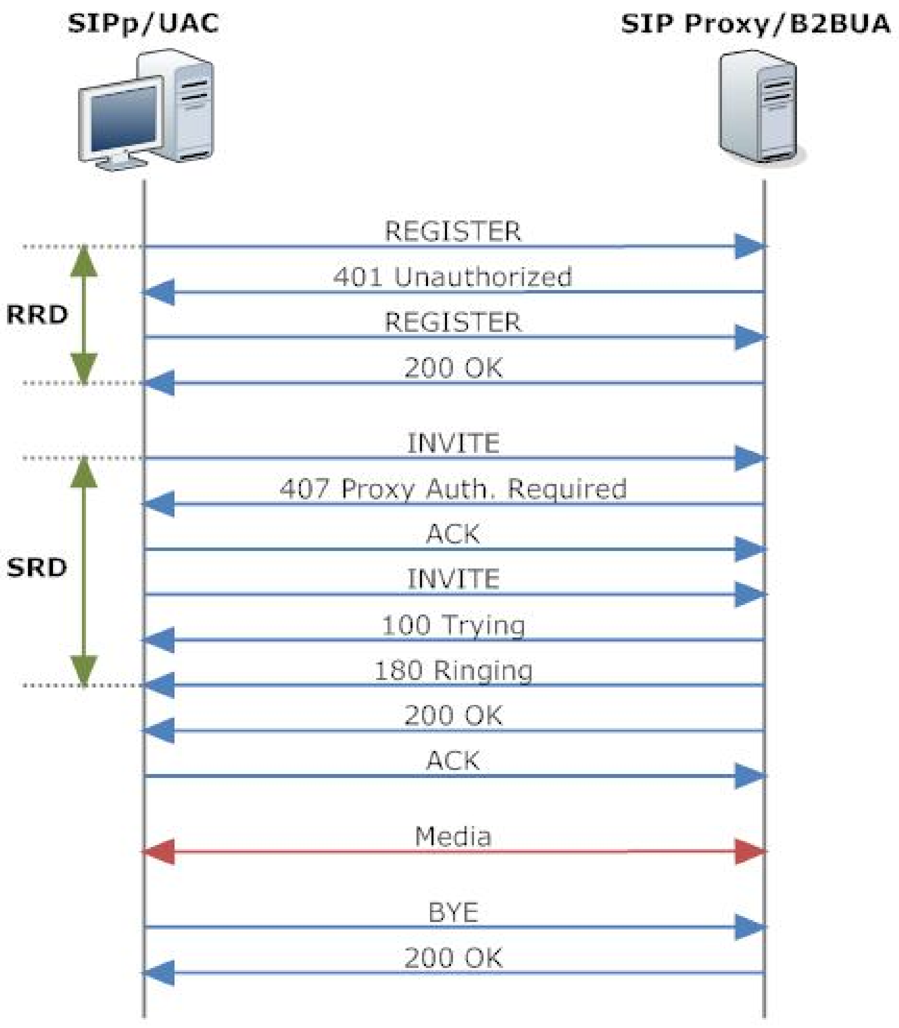


Figure 1: Registration Request Delay (RRD) and Session Request Delay in the context of a SIP dialogue

3.2 Limits to the Definition of the Parameters in the Analysis of the Results

The previously defined parameters do not suffice to assess the SIP server’s performance. To be able to determine the SIP server’s performance from the collected data, we need to define the values of the limits for each category of the measured parameters. This definition must come from the features of the SIP protocol and the generally recognised conventions of IP and classic telephony.

From the characteristics of the hardware, CPU use plays the main role in performance analysis of the SIP server. This conclusion is logical because of the importance of CPU in the architecture of the computer and the CPU-oriented operations of the general architecture of the SIP server.

In general, the characteristic of CPU utilisation is limited by maximal CPU performance, which is 100%, but this boundary can rarely be reached. To be more specific, due to the time intervals between particular measurements of CPU utilisation (as explained later), a short peak in the CPU-utilisation characteristic may not be registered. However, during this peak, delays and call-quality impairments can occur. To compensate for this imperfection of our methodology, a performance boundary of below 100% was anticipated. However, the actual value of the CPU performance boundary may vary. Therefore, we search the CPU-utilisation characteristic for the first point where maximum CPU utilisation is reached. This point is the maximum number of calls that the SIP server can handle, from the point of view of the performance of the hardware.

The definition of the limit for the SIP-delay characteristics, RRD and SRD, comes from the nature of the SIP protocol. When the call is set up, the delays between messages should not exceed several hundreds of milliseconds. Although these limitations are dependent upon the travel-time of the SIP message from one end of the call to the other, they can also be used for our purposes, because of the need to set up a call quickly enough that it does not bother the user with noticeable delays.

From this, we can estimate that the boundary the quality of RRD and SRD is somewhere around 300 milliseconds. However, this value may vary according to the need of each particular user. Generally, we can say that limit for SIP transactions is reached when the SRD and RRD characteristics start increasing rapidly. This boundary will give us a slight margin for a potential reserve.

The quality of speech is vulnerable to long delays between consecutive RTP packets. It is affected by jitter as well, but the jitter-issue can be eliminated by a sufficient jitter-buffer on the receiving side. Therefore, maximum packet delay is the key characteristic for analysis of the RTP stream. From the theory of IP telephony, the delays between packets should be in tens of milliseconds. Therefore, and because of the similar reasons mentioned with SRD and RRD, we decided to set this boundary to approximately 90 milliseconds.

All delay characteristics use a similar analogy to the theoretical values for end-to-end delays. This is why their definition cannot be exact and these parameters may vary in different environments. To eliminate different interpretations of the same results and to simplify analysis of the delays, we used the point, where the particular characteristic changes its “almost constant” trend, to rapidly increasing as the quality-boundary for all the delay-characteristics This approach gave us accurate results, which were tested experimentally. The resulting methodology of the analysis was also much simpler.

3.3 SIP Proxy Testing

In the basic configuration of the SIP Proxy, we are able to measure only the SIP and its utilisation-parameters. The RTP stream does not flow through SIP Proxy and thus it does not represent the load for it. This is why we did not have to think about the call-length, because no matter how long the call is, the utilisation of the hardware is the same, so the only appropriate metric for measuring SIP Proxy is the number of calls generated per second (or any other time interval).

Each measurement of the SIP Proxy consists of several steps. Every single step takes about siixteen minutes. This means that for ten minutes, ten-second calls were generated at a user-defined call rate. Then, there was a ten-second period when the unfinished calls were terminated. This was repeated for every single step of the call-rate. Every call consisted of a standard SIP dialogue and pause, omitting the media over RTP. Because the load was not constant, but increased slowly at the beginning of the test (the first ten seconds) and decreased at the end of it (the last ten seconds), the results taken after this starting period and before the ending period were the only ones that were considered to be valid. To allow for additional changes in setting the time interval in the scenario, and to strengthen the consistency of the method, we used the data collected during the middle ten minutes of each step. All the parameters named in the previous subsection were measured except those related to the RTP stream.

The ten-second time interval that has been mentioned several times came from a compromise between a reasonable call-length and the need for generating as many calls per second as possible. It allows for acceptable performance and does not require a huge database of subscribers. This interval can be changed but cannot exceed 2.5 minutes, allowing for the collection valid data.

We measured SRD, even though this scenario cannot be considered to be end-to-end (this condition is defined in RFC 6076). We decided to measure it because the load on the UASs is minimal, even for high call-rates. This makes the delays created by the UASs both minimal and almost constant. Therefore, we can use this parameter to assess the performance of the SIP Proxy. Because the delays it creates are the only variable, the collected data is useful. This is the only deviation in our method from RFC 6076.

3.4 B2BUA Testing

Unlike SIP Proxy, the RTP stream presents the highest load on a B2BUA server and therefore, the number of simultaneous calls must be used as the metric. This is the main difference between the B2BUA and the SIP Proxy testing-scenarios. The second, and not so important difference (from the point of view of methodology), is that in this configuration, we were measuring the effectiveness of the transcoding. In this scenario, the performance of the B2BUA was affected, not only by its setting, but also by the UAC and UAS configurations. The test routine was repeated for each codec setting.

However, the method of the test was almost the same; the only issue we faced was the new metric, together with the need for revising the time interval for a single call. The new metric can be an issue when the SIP traffic generator cannot be ordered to create a specific number of simultaneous calls. In this case, it was necessary to calculate the number of calls generated per second, using this this equation:

$$C_R = C_S \cdot T \tag{3.1}$$

C_R is the desired Call Rate, C_S is the number of simultaneous Calls we wanted to generate and T is Time interval defining how long the call (media) should be. The Time interval used for B2BUA in our measurements was set to 60 seconds because most calls have this length, but again, this parameter can be changed.

To perform the testing of RTP streams, we used a special computer, which allowed us to use more sophisticated tools for capturing the network traffic. This prevented the RTP and SIP parts of the tests from influencing each other.

Because we focused on the effectiveness of the testing and the speed of transcoding, at this point we were able to determine the maximum load that the SIP server can handle from the point of view of SIP or RTP. However, these results would only be valid for a single machine/platform and that is why we added one more step to the data analysis. The same procedure of testing was performed on a machine configured so that it allowed only media to pass through the SIP server. The results taken during this test serve as the basis to which we related all the other results.

$$P_{RF} = \frac{P_{CT}}{P} \cdot 100 \tag{3.2}$$

This step allowed us to compare the results from the hardware and the platform independently.

4 Implementation

This methodology can be implemented in several ways. We can design hardware that would have enough power and capability to perform the testing. This hardware would also include the high performance processors and other components required for signal processing, because of the need for quick dispatch of messages and media. This hardware would be expensive, single purpose and only suitable for difficult open-source solutions, which makes it cost-ineffective.

Our approach uses the hardware that is present in any environment. We find computers in all VoIP installations and these computers can be instructed to perform the test. This greatly increases the effectiveness of the design of the test platform, because no special hardware is required. The program SIPp was already mentioned as a SIP traffic generator and data collector. We decided to use the System Activity Reporter (SAR) as the monitoring tool for hardware utilisation.

4.1 Test Topology

The keystone of the testing platform is the SIP testing tool SIPp. This tool's features and limitations influence the design of the final testing platform significantly. In order to perform SIP testing, we simulated both ends of the SIP dialogue to test the main part of the SIP infrastructure, the SIP server. The SIP server represents a set of servers that always include a SIP Registrar and a SIP Proxy or B2BUA (Back to Back User Agent). The latter is the most commonly used solution in an enterprise environment, for both SMEs (Small- and Medium-sized Enterprises) and LEs (Large Enterprises). Figure 2 depicts the configuration of the test hardware for testing the SIP Proxy and B2BUA.

This is a general configuration that does not include all of the aspects of test platform that we used for our measurements. First, we used both physical and virtual computers to simulate SIP traffic. The results for both configurations were almost identical so future users of this methodology can choose which topology would be best for him, based on available hardware.

The only condition required for testing a SIP server successfully is the interconnecting device (or system). Basically, this can be any device or network capable of routing of SIP messages among SIP traffic generators, a SIP server, and recipients of the SIP traffic, but to compare the results of the measurements with those taken in different network, we would be required to use the exact same topology. This is why it is advantageous to use the simplest possible topology in order to reduce additional costs and work caused by the requirements of some special topology. The most flexible variant for this topology is to use the single switch, which is commonplace in all modern SIP installations.

The number of devices used for the testing may also vary due to the performance of the SIP server. The more efficient the SIP server, the more devices are needed to test its performance, especially on the UAC-side. Due to the software-limitations of the SIP traffic generator (SIPp), one computer in UAC-mode is capable of creating 200 simultaneous calls with media (for testing B2BUA) and about 220 calls per second without media (for testing SIP Proxy), regardless of the hardware configuration of the PC running SIPp. Therefore, we needed to estimate the performance of the SIP server in order to determine the number of computers (physical or virtual) needed for the test. This makes the virtualisation the more viable option. The performance of SIP servers is not affected significantly by the number of UASs. However, it is necessary to force the SIP server to decide between different paths to UAS; therefore, there had to be at least two computers in UAS-mode in the test topology.

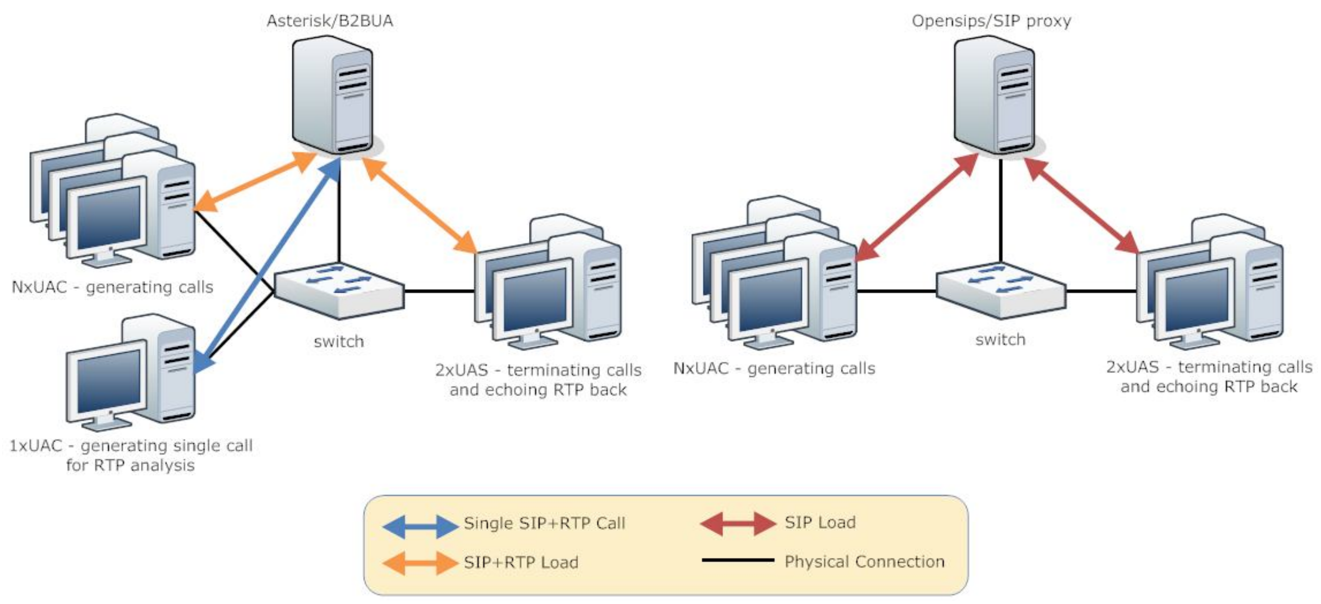


Figure 2: Test-bed diagram for configuration of B2BUA and SIP Proxy

As well as the topology, the test scenario should also be as simple as possible, mainly to reduce the complexity of the test, but also because it is not possible to test the SIP Proxy (and B2BUA as well) in all possible configurations. Thus, it is useful to focus on a basic, default configuration and perform the tests with it. The output results then represent the information about the “best case scenario”, from which we can assess the SIP server’s performance and compare it with its rivals.

4.2 Management

We knew that we needed to use multiple computers in the test platform due to the limitations of the testing tool, SIPp. The effectiveness and precision of a test depends on the synchronisation of the time and processes of the computers. To achieve this synchronisation, one of the computers was appointed to the role of a controller. We were aware of the management computer model presented on the SIPp IMS Bench website [ims], where a single computer was dedicated to the role of supervisor over the test, but we decided to go another way. We integrated the controller mechanisms to the one of the UAC computers. This computer was then called the

“Main UAC”, and in addition to supervision of the test, it also processed and stored the results. This integration was made possible through the use of a quite simple test algorithm, which is depicted in Figure 3.

As shown in Figure 3, a collision of two processes happened only once. The data measurement occurred when call generation was underway. However, this collision was not a problem, because data measurement on all UACs was provided by the SIPp, itself, and this was instructed to perform the measurements when it was started. On the other hand, SAR provided the data measurement on the SIP server, but only after the initiating time interval had expired. Therefore, the excess load on the main UAC was created during the test itself. This would have influenced the results, unless counter measures were taken. This problem was quite simple to solve. The data measurement on the SIP server was invoked during the start-up process of the test, with the instruction to delay the execution of a command just long enough to overcome the initiating time interval, where the results would be inaccurate. It was clear that there was no actual need for the supervisor function to be implemented on a separate computer, thus simplifying the test topology and increasing of the effectiveness of computer usage.

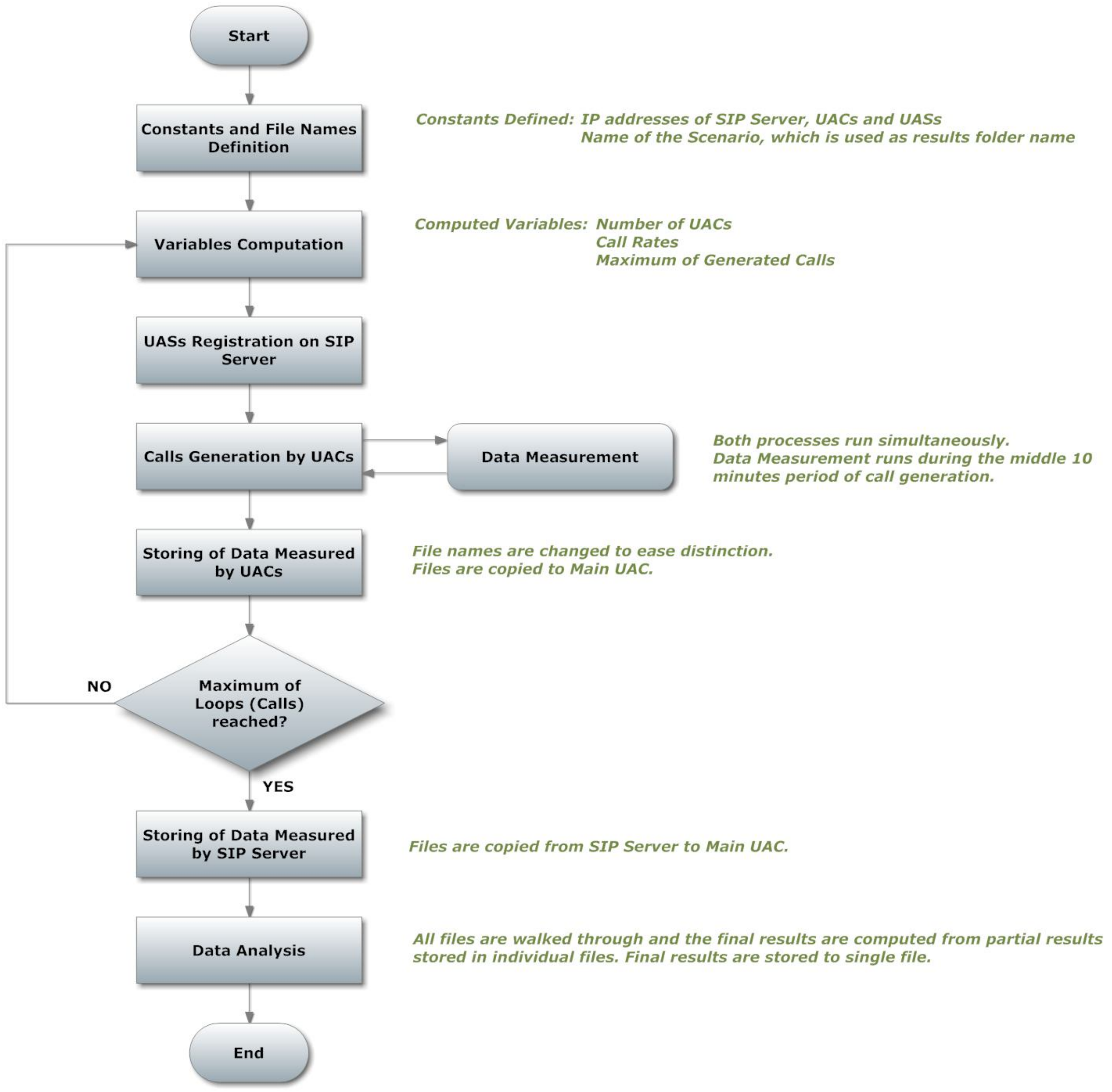

Start
Constants and File Names Definition
Constants Defined: IP addresses of SIP Server, UACs and UASs
Name of the Scenario, which is used as results folder name
Variables Computation
Computed Variables: Number of UACs
Call Rates
Maximum of Generated Calls
UASs Registration on SIP Server
Calls Generation by UACs
Data Measurement
Both processes run simultaneously.
Data Measurement runs during the middle 10 minutes period of call generation.
Storing of Data Measured by UACs
File names are changed to ease distinction.
Files are copied to Main UAC.
NO
Maximum of Loops (Calls) reached?
YES
Storing of Data Measured by SIP Server
Files are copied from SIP Server to Main UAC.
Data Analysis
All files are walked through and the final results are computed from partial results stored in individual files. Final results are stored to single file.
End

Figure 3: Diagram of the test algorithm

The entire algorithm was implemented as a bash script, which performed all the required computation services and passed the commands from the main UAC to each of the computers in the topology. To be able to do this,, the algorithm needed the SSH service to be installed on all the computers. For a successful SSH connection, all the computers except the main UAC were required to run a SSH server daemon, while the main UAC needed to use a SSH client service. Due to need for the establishment of an automatic SSH connection, all of the computers had to be in the "known hosts" database of the main UAC and the connection had to use the public key authentication method. If either of these two conditions were not satisfied, ,user interaction would have been required and the time synchronisation would have been lost. The test algorithm, in the form of consecutive SSH connections, is depicted in the Figure 4.

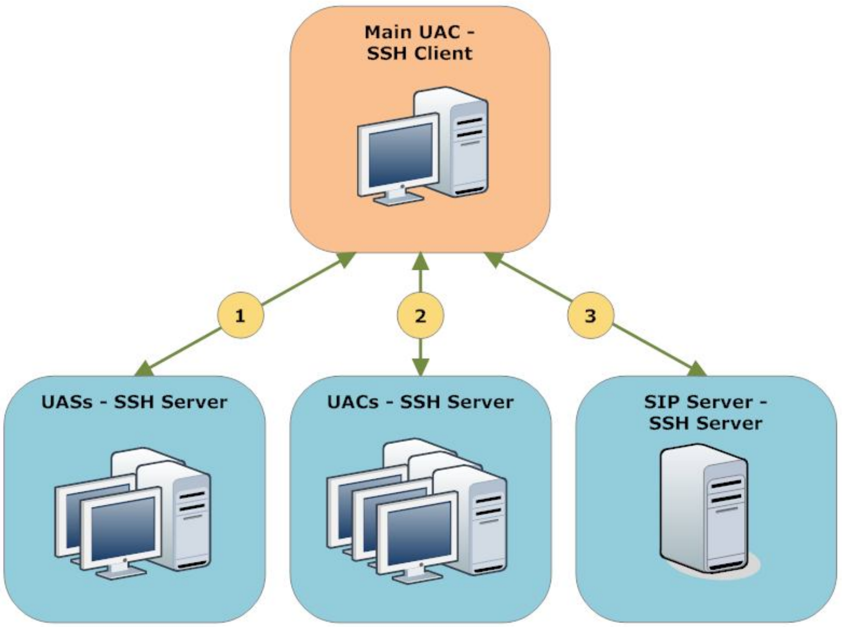
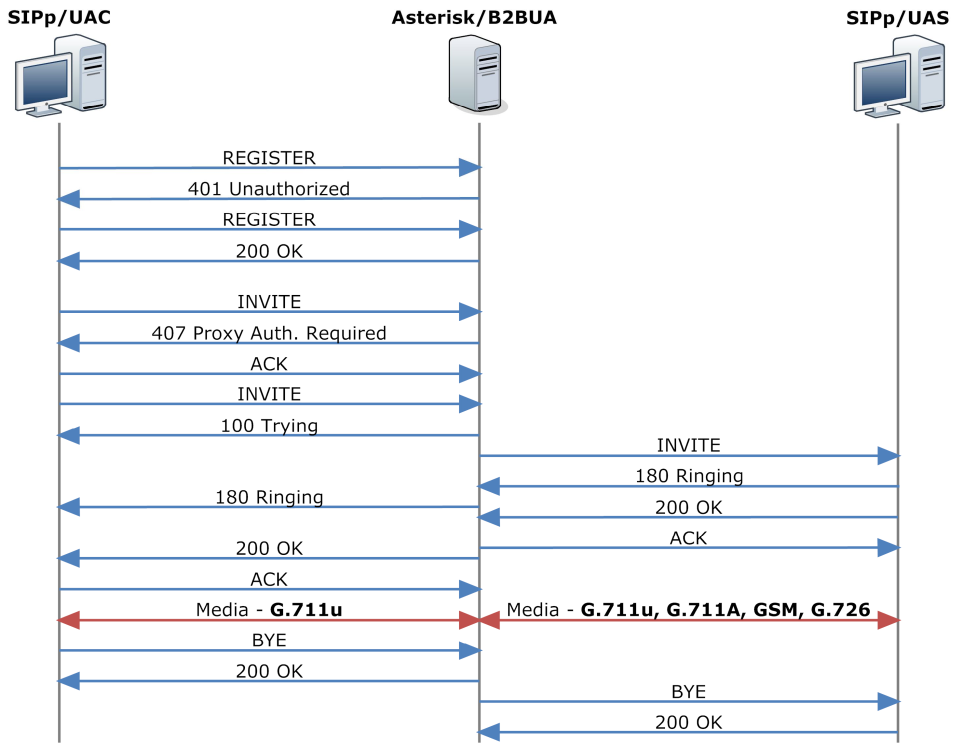


Figure 4: Consecutive SSH connections during the test scenario

4.3 Call Generation

For call generation, the SIP testing tool called SIPp was used in our implementation. A closer look at the mechanisms of call generation. as it is performed by SIPp, reveals that the whole SIP dialogue is generated by the SIPp and that it is user-definable through XML scenarios. These scenarios define exactly what the SIP dialogue should look like, as well as the structure of the individual SIP messages. The scenarios used for both B2BUA and SIP Proxy testing are depicted in Figure 5, in a reader-friendly format. In addition to this scenario, SIPp needed to be told what information about the SIP account it should use. For this, we used a database file in csv format, as this was the only way to pass this large amount of information to the SIPp. This file consists of the account name, username, and password, which are used for the registration and authentication of the server, and the account number of the called party. It is different on each computer, and because it was read sequentially, it allowed SIPp to connect to a different physical (or virtual) computer with each call. In other words, each UAC was connected to all UASs, forcing the SIP server to perform the routing process with each call.

When SIPp, in the role of UAC, generated a call, the information from csv file was read and the REGISTER request was sent to the IP address of the SIP server. This IP address is defined as one of the parameters from the command line. This REGISTER request was defined in the XML scenario and SIPp dynamically inputted the data from the csv file to the user-defined locations, so that a valid, non-repeating REGISTER request was sent. After this, the server required an authentication, which was performed the same way. SIPp formed the REGISTER request again, now with the authentication data from the csv file included. The time interval required for this exchange was measured.



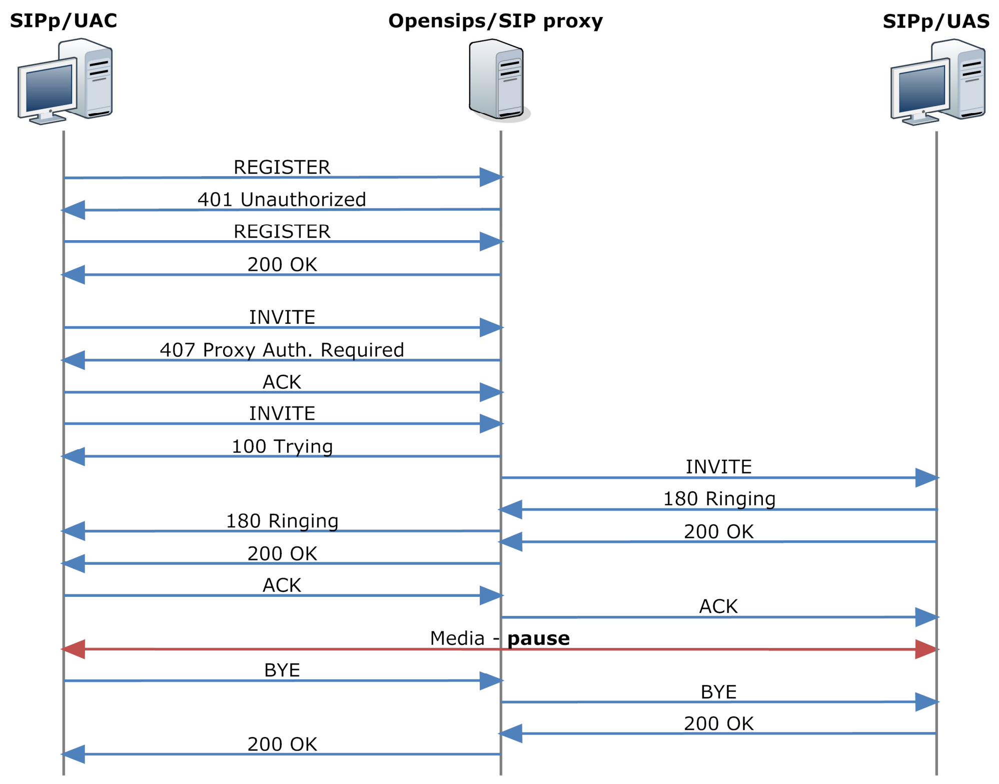


Figure 5: SIP messages and dialogues as defined in the XML scenarios for B2BUA and SIP Proxy testing

Consequently, a new call was created with the INVITE message. This message was processed in a similar way to the REGISTER request. The SIP server then routed the message to the recipient that had responded with the appropriate messages, and the call was set up. After successful initialisation of the call, the media was sent by the UAC. Because SIPp cannot generate the media flow on its own, a recorded pcap file is required. On the side of UACs, the pcap file with the recorded audio with a length of 60s in PCM u-law format was used. This media was sent to the SIP server (B2BUA to be more specific), which then searched the database and determined what media format is supported by the recipient. If the media formats are not identical, transcoding takes place. In addition to the PCM u-law format, codecs also use formats,such as PCM A-law, G.726 and GSM. The receiving UASs accepted this media flow and echoed it back, unchanged thus creating the second media flow in the call. This media flow occurred only in B2BUA scenarios. When SIP proxy was used in place of a SIP server, no media flow was generated and its place was taken by a ten-second pause interval.

After the media or pause interval ended, the standard call termination using the BYE method took place. This procedure repeated for every single call. When load on the SIP server is low, no problems appear, but with higher loads, an increasing number of calls end unfinished, which prevents SIPp from ending. SIPp has a mechanism to prevent this behaviour, but this procedure also influences the results. Therefore, to prevent the next testing step from being influenced by the previous one, a management script terminated all the hanging (unfinished) SIPp instances.

4.4 Data Gathering

The data gathering operation in our implementation included three elements – SAR, SIPp and Tshark. The data about utilisation of the hardware (the SIP server) was collected by a System Activity Reporter, which was invoked by a bash command and then ran in the background as a daemon service, collecting the data at regular time intervals. SAR was run after the initiating time period had passed, and collected the data only during the middle ten minutes of the test. Therefore, its resulting output was not affected by variability in the number of concurrent calls. More information about the time intervals in the test can be found in the next subsection. SAR checked and stored the information about utilsation of the hardware every ten seconds and mades a total of 60 measurements. Utilisation of the CPU, memory and network traffic can be found in the data measured by SAR. SAR stored this data in a binary SAR format on the local hard drive of the SIP server. These data were downloaded and decoded by the main UAC at the very end of the test algorithm.

In addition to traffic generation, SIPp also measured and stored information about the length of the SIP message exchanges. Therefore it was used to measure the defined parameters RRD and SRD. SIPp also measured information about the total number of created calls and the number of successful ones. These data were stored on every UAC computer in text format, as well as on the main UAC, which downloaded and renamed these files at the end of each step (loop) in the test scenario.

To obtain information about the RTP stream, Tshark was used in our implementation. During the each test step (loop) of the B2BUA testing scenario, one call was generated on one dedicated computer and forwarded to the B2BUA. It handled the call as any other. The media stream was then echoed back by the UAS, and after it had passed the B2BUA again, it was forwarded back to its source. Network capture was run on a dedicated host and it stored both the original media flow and the one that was echoed. A great deal of information can be obtained from the differences between these two captured media streams, such as round trip time, packet loss, packet delay and jitter. For our purposes, only last two of these parameters were evaluated, because they contain the main information.

4.5 Data Analysis

Due to the imperfection of the testing software, inaccurate and flawed results were stored among the correct ones. These flawed data were mainly generated during the starting and ending time-period of a test step (loop). This problem is caused by the SIPp, which does not support call generation that is based on a fixed number of simultaneous calls. Instead, it only allows a user to set the rate at which new calls are to be generated. This is why the load on the SIP server was not constant at the beginning and at the end of the test step. The length of these two time intervals was roughly equivalent to the length of the media during the call, because the number of simultaneous calls only becomes constant after the first call has been terminated. After this, the number of newly created calls and the number of terminated calls is equal and the load on the SIP server is constant. The characteristics of the load on the server, together with the problematic time intervals, are shown in the Figure 6.

To assure that only accurate data were included in the final logs, two steps were implemented. First, SAR measured only in the middle time interval (the green zone) so that only correct results were stored. Second, a data analysis script was designed. This script walked through the collected data after the end of the test (not just the test step), and according to the timestamps stored together with this data, it deleted the values that were stored in the problematic time period. After this deletion, it processed the rest of the values and created only three files, containing the final results. Moreover, it created charts from these final results so that the user can examine the performance of the SIP server immediately after the test. This script was written in Python.

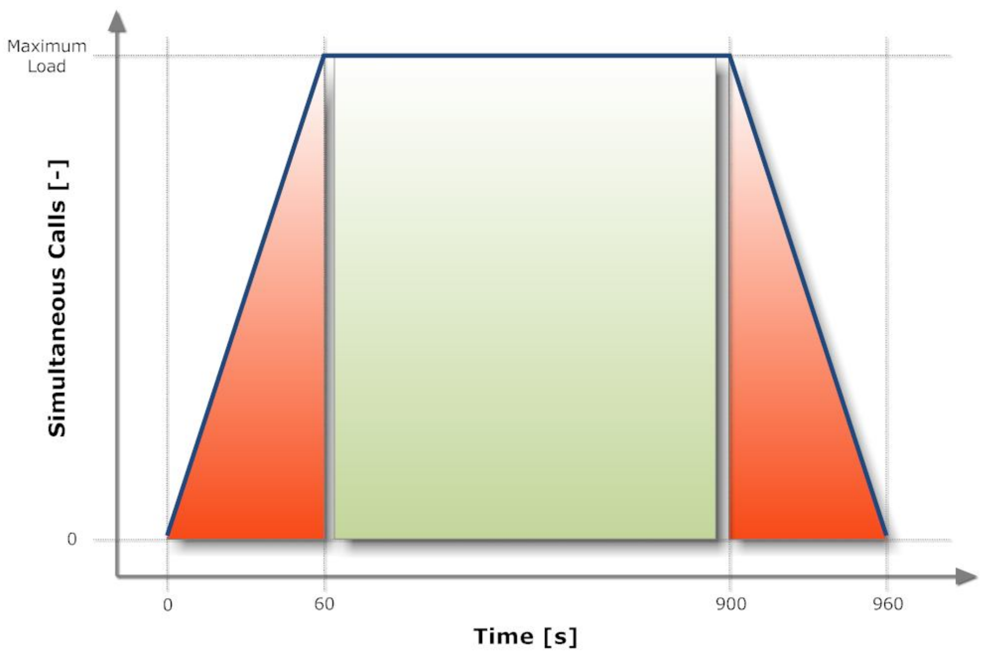


Figure 6: The characteristics of simultaneous calls and the highlighted time-intervals when results were inaccurate

In addition to problems with flawed data, inconsistent results can arise from problems in the network, or some error during the processing of calls in one or more SIPp instances. For example, if an error occurs on the SIPp so that it does not process the messages correctly for a period of one minute, during this period many of the calls will be marked as failed, and the load on the SIP server will be lower than expected. Therefore, each test should be repeated several times the average of the collected data should be calculated as the final result. Inconsistent results are replaced by the values obtained from the regression analysis and can be defined as values that does not reflect the increasing load on the SIP server during one or more tests. To be more specific, the increasing load should logically result in an increase in CPU utilisation, but during one test, a slight decrease in one value may be measured. After another test, the value has returned to an increasing trend, but another problem occurs somewhere else. This is a typical example of an inconsistent result. Fluctuations in the trend of a characteristic can occur repeatedly; this can be caused by the internal functionality of the SIP server and these fluctuations should be explored thoroughly. This is main reason why repetitive measurement is preferred over the replacement of the data by interpolated values. However, this approach cannot be applied to computed values. The assessment of these inconsistencies is described in more detail in the 'Results' section.

4.6 SIP Server

The SIP server is the most important part of the testing topologies that have been implemented to test the most commonly used variants of the SIP server. For the B2BUA, we used an Asterisk PBX, and for the SIP proxy, we decided to use Opensips. Because performance is influenced not only by the software architecture but also by the hardware used, the complete specifications are listed below.

- Software platform:
  - GNU Debian Linux 5.0.1 x86 (Lenny)
  - Asterisk PBX 1.6.2 and Opensips 1.6.0
- Hardware platform:
  - CPU AMD Athlon 64 X2 5200+
  - 4 GB DDR2 RAM 800 MHz, Chipset nVidia nForce 570-SLI

5 Results

Progress achieved by authors through this research has already been published [voz173], [voz179], [voz188], [voz191], [voz222] and [roz]. The following sub-chapters describe the individual results and their interpretation.

5.1 B2BUA

For each category, there are two charts. The first one shows the results for the case without transcoding and is coloured in blue. The second shows the normalised values of the cases with transcoding, which was acquired by inserting the collected data into the equation (3.2) and is coloured in three different colours.

5.1.1 Mean CPU Utilisation

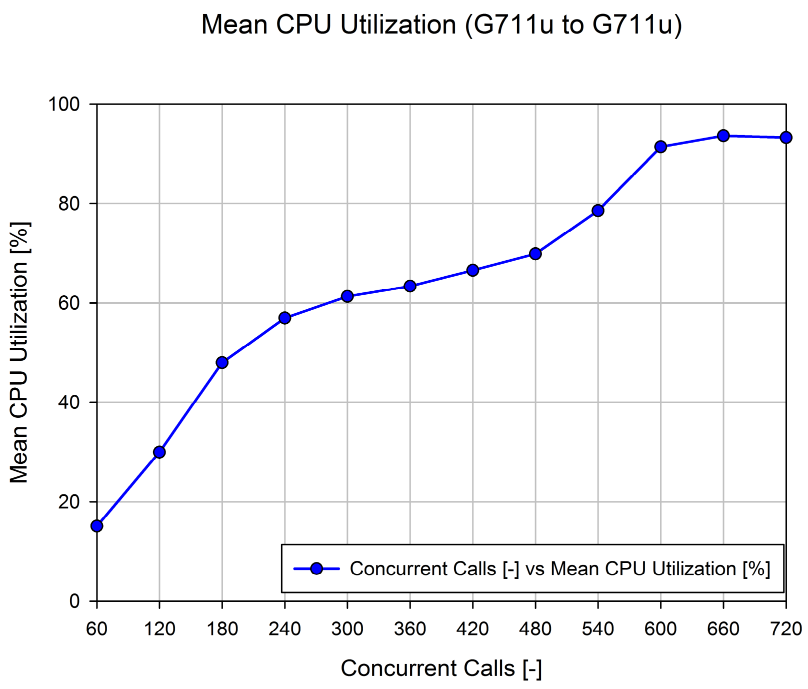


Figure 7: Mean CPU utilisation without transcoding

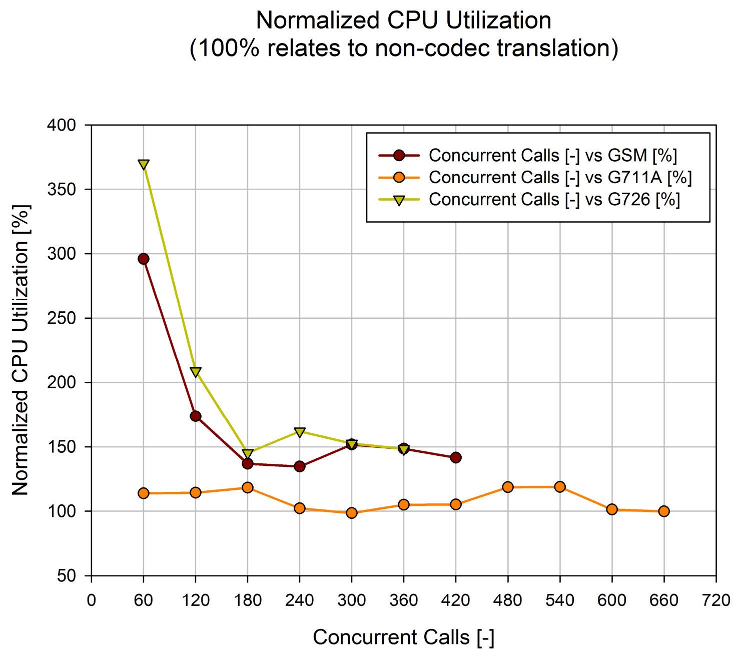


Figure 8: Normalised mean CPU utilisation for cases with transcoding

The first chart shows a simple relationship between the number of concurrent calls passing through the B2BUA and its CPU utilisation. The second chart shows that (as expected) transcoding from G711u to G711A consumes about 20% more CPU power than a simple G711u case without translation. On the other hand, the most demanding is the G726-32bit codec. The lowest load returns the most interesting information. With a load of 60 calls, the differences in CPU power consumption for GSM and G726 are the highest, compared to the load without transcoding. With higher loads, power consumption starts to decrease rapidly. The significant decrease of the values in the middle of the charts cannot be considered as an inconsistent result, because, as described in the previous section, these values were not measured, but were computed from the measured values. Therefore, the mutual relationship between the two trends can decrease even with higher loads. This applies to the normalised values of all of the characteristics.

5.1.2 RRD and SRD Delays

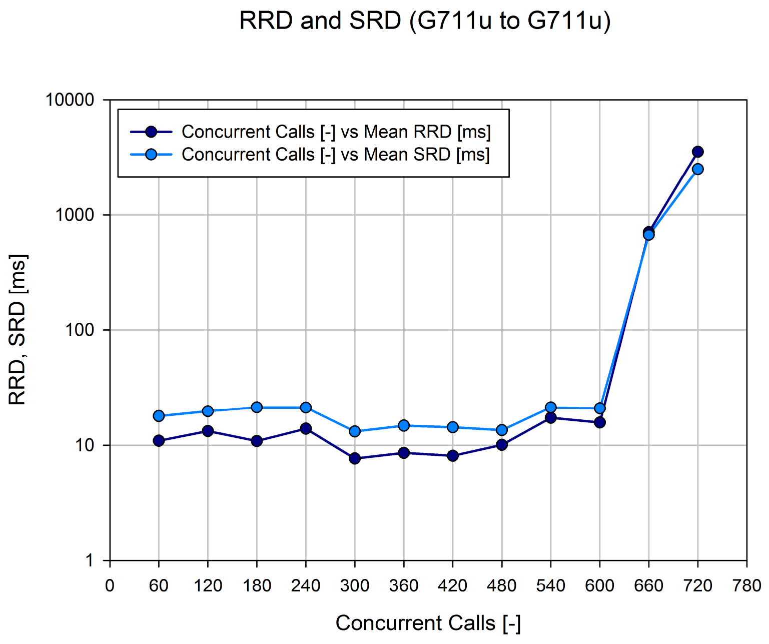


Figure 9: RRD and SRD without transcoding

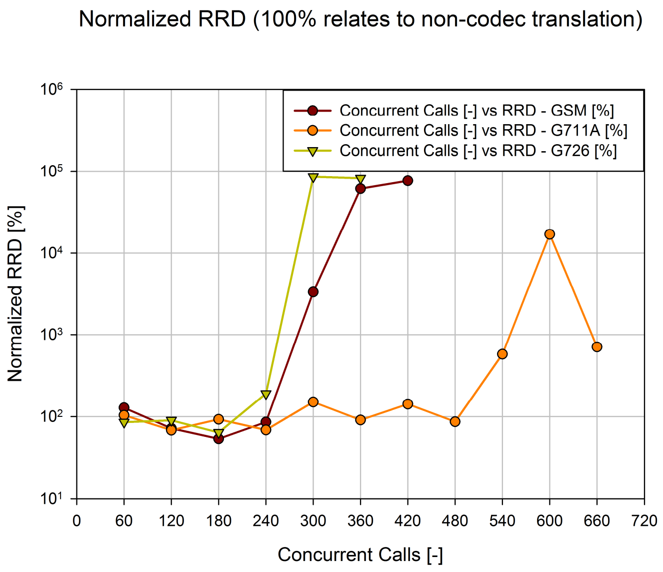


Figure 10: Normalised RRD for cases with transcoding

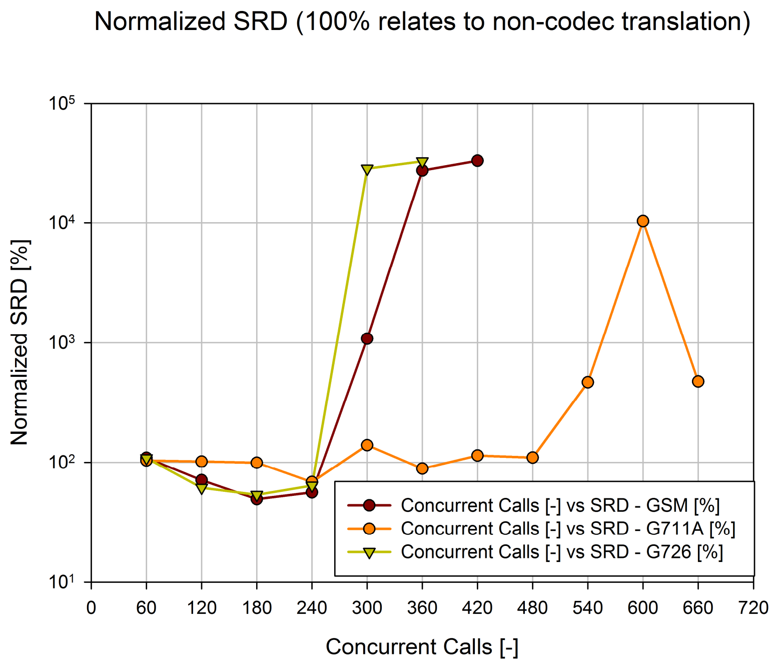


Figure 11: Normalised SRD for cases with transcoding

The charts in Figures 9, 10, and 11 clearly illustrate that the call is set up even more quickly when there is transcoding in use and the load is below 240 simultaneous calls. Then, as CPU utilisation increases, the delays get very long. The last G711A value for both charts is very low. This is due to a rapid increase in delays for G711u to G711u, when there are between 600-660 simultaneous calls. The fluctuations in charts with normalised values are caused by random events during the measurements, with and without transcoding. Because we related these values in a single equation, the variances get more distinctive. However, this does not affect the final decision about the performance of B2BUA from the point of view of SIP.

5.1.3 Mean Jitter and Maximum Packet Delay

The expected outcome for the normalised values of mean jitter and maximum packet delay was confirmed, because the values related to a small load are very similar to the main values from the case without transcoding. Peaks in the area of the medium load were caused by the volatile nature of the parameters and had no effect on the final decision about the B2BUA's performance. A very rapid decrease in both normalised values for G711A was caused by the increase of the main values from the case without transcoding and by the significant number of unsuccessful calls in this scenario.

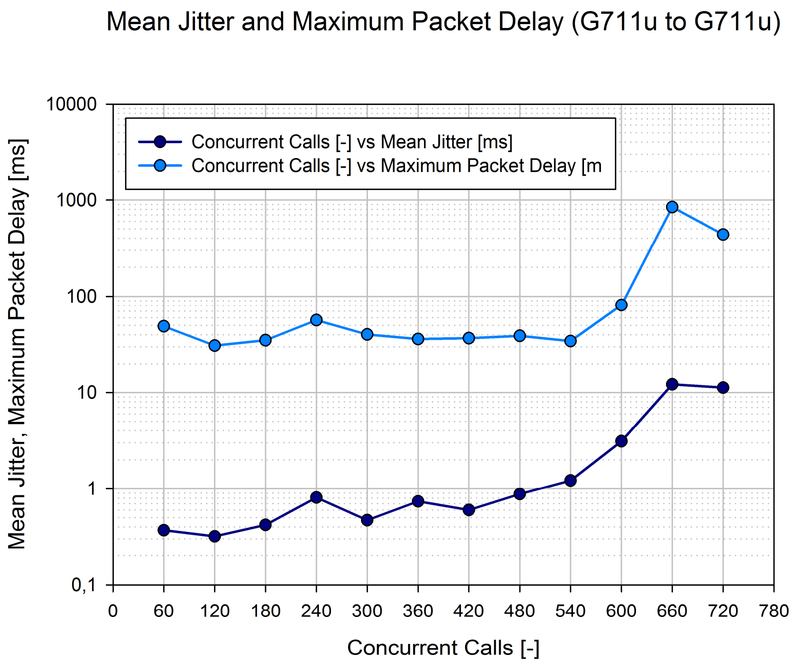


Figure 12: Mean jitter and maximum packet delay without transcoding

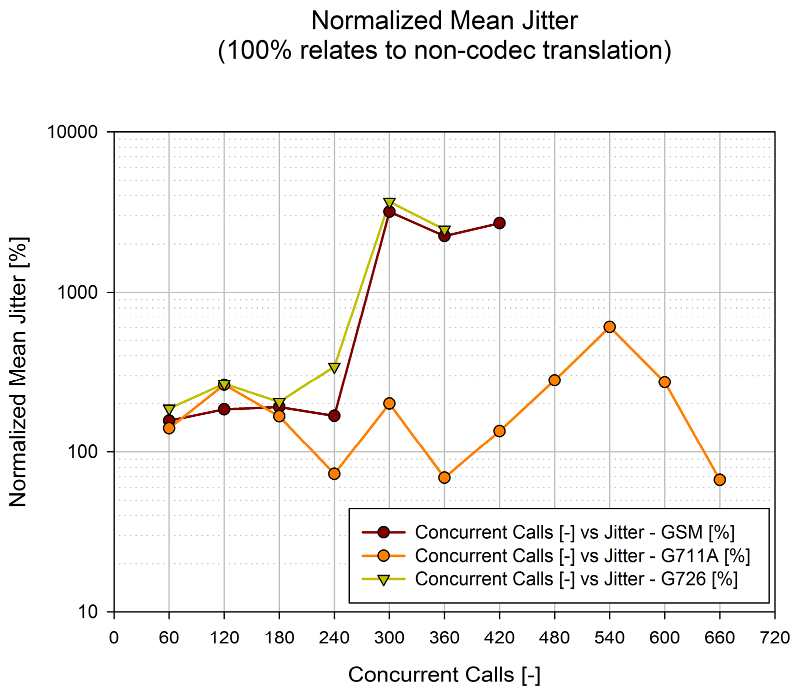


Figure 13: Normalised mean jitter for the cases with transcoding

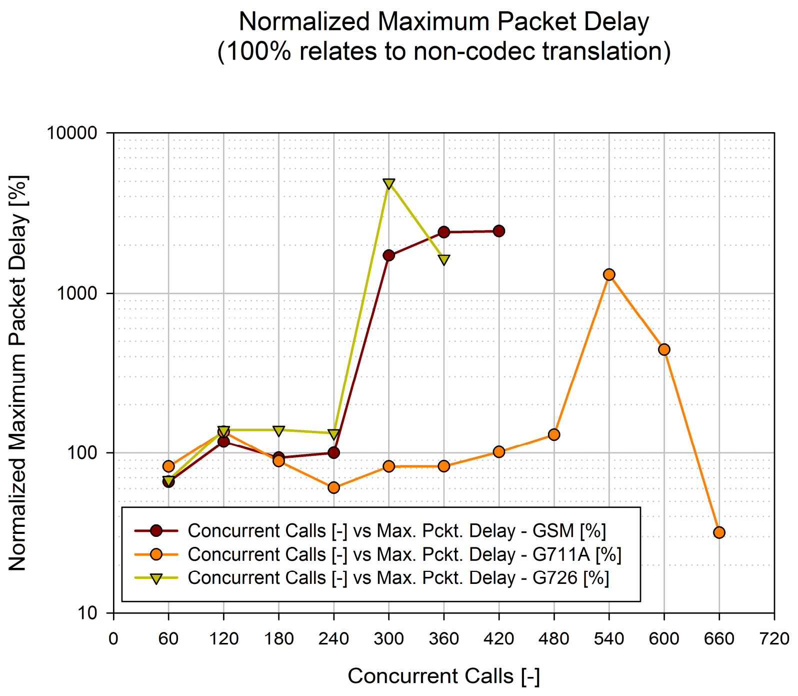


Figure 14: Normalised maximum packet delay for the cases with transcoding

5.2 SIP Proxy

The scenario with SIP Proxy is much simpler than the scenario with the B2BUA. The only output from our measurements is raw data describing the ability of the SIP Proxy to handle an increasing number of calls generated per second. However, all of the versions of SER architecture allow the user to set the number of listening sub processes and this number should (in theory) affect the performance of the SIP Proxy as well. Therefore, all of the measurements were performed with the number of UDP listeners set to values of four and sixteen.

5.2.1 Mean CPU Utilisation

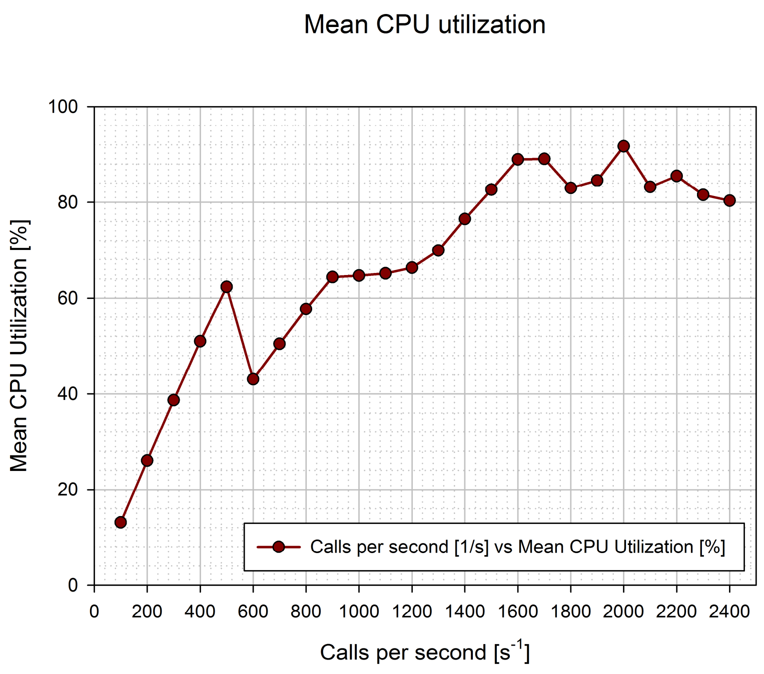


Figure 15: Mean CPU utilisation for SIP Proxy with four UDP listeners

The results are not surprising, except for the peak that appeared when 600 calls per second were generated. These data are not flawed, because a similar peak in the same area was measured many times. Therefore, it is much more likely to have been caused by the call handling mechanism of the SIP Proxy.

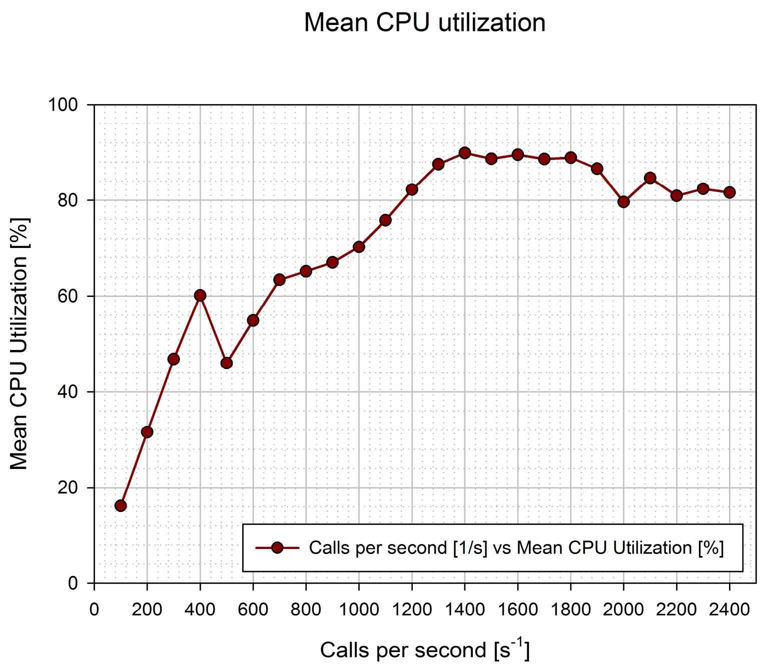


Figure 16: Mean CPU utilisation for SIP Proxy with 16 UDP listeners.

From both charts, it is obvious that increased number of UDP listeners did not have a positive effect on the SIP Proxy's performance. On the contrary, the CPU utilisation with sixteen listeners was comparable to the performance of the SIP Proxy, sub processed to four listeners, and with a load of about 150 more calls. This may have been caused by the inadequate performance of the CPU, which cannot handle an increased number of processes in real time, which can cause delays. The limiting factor for this measurement was the CPU utilisation, although increased performance could have been reached if another database, had been used, instead of MySQL. This is because this database, itself, consumed 17% of the CPU power. Unfortunately, when we performed the measurements, no alternative working database module for Opensips had been released, due to the transition between two major releases.

On both charts, we can see fluctuations in the values after the maximum load has been reached. These fluctuations appeared each time and were caused by the random number of failures when the highest reasonable load had been exceeded. A high number of repeated measurements can reduce these fluctuations, but the fluctuations do not affect the analysis of the final result, and therefore, the charts are presented in this raw form after two measurements.

5.2.2 RRD and SRD Delays

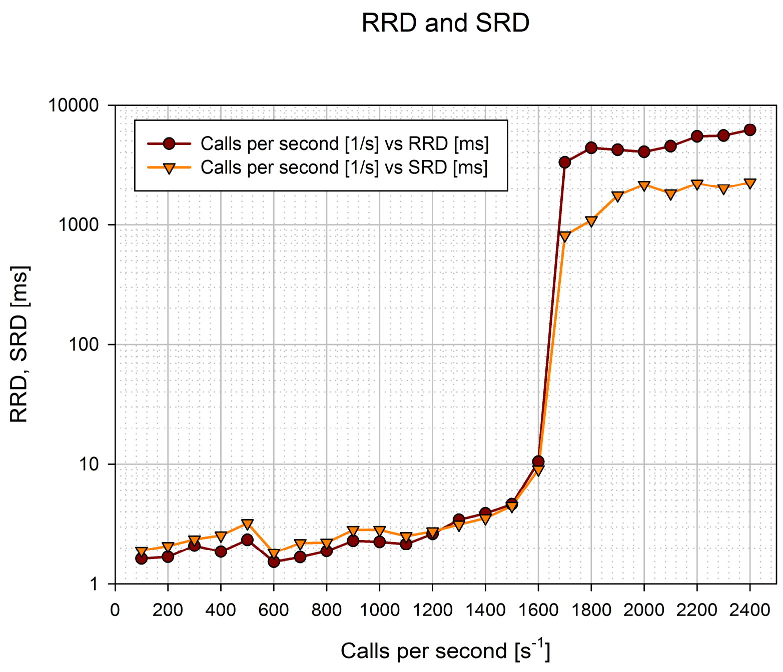


Figure 17: RRD and SRD delays for SIP Proxy with four UDP listeners

From the SIP perspective, the situation is similar to one that arose from statistics on CPU utilisation. Again, the number of sub processes has a negative influence on the overall performance of the SIP Proxy. RRD and SRD delays are about 2 to 3 times higher when the number of sub processes is set to sixteen. Moreover, the huge leap in both characteristics (RRD and SRD) appears about 200 calls earlier. From the perspective of the two parameters presented (CPU utilisation and delays), it is obvious that to achieve the best performance, a low cost processor should operate a small number of processes.

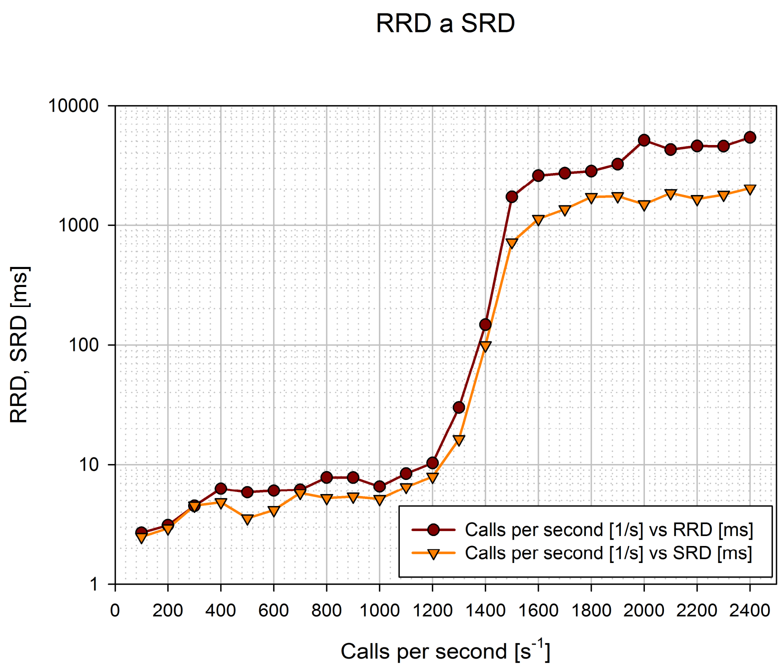


Figure 18: RRD and SRD delays for SIP Proxy with sixteen UDP listeners

6 SIP Registration Burst Load Test

For the registration test, the testing platform must differ slightly from the one presented in the complex methodology. The main reason for this is that the UAS part of the testing platform is not needed, because only end-to-end dialogues between the client and the SIP server will occur. Basically, all the computers will act as the initiators of the SIP registration dialogues, which is why they are called UACs (User Agent Clients). For the generation of SIP messages, the well-known testing tool, SIPp, was used, and to ensure that client computers did not run out of hardware resources, the generated load was spread among eight computers. From these assumptions, the basic test topology is depicted in Figure 19. Correct messages are coloured in green, while unexpected messages are orange and error messages are red. Dotted lines with arrows represent the hop after an error or unexpected message has been received, or when an error occurred during transmission of the message [roz], [voz222].

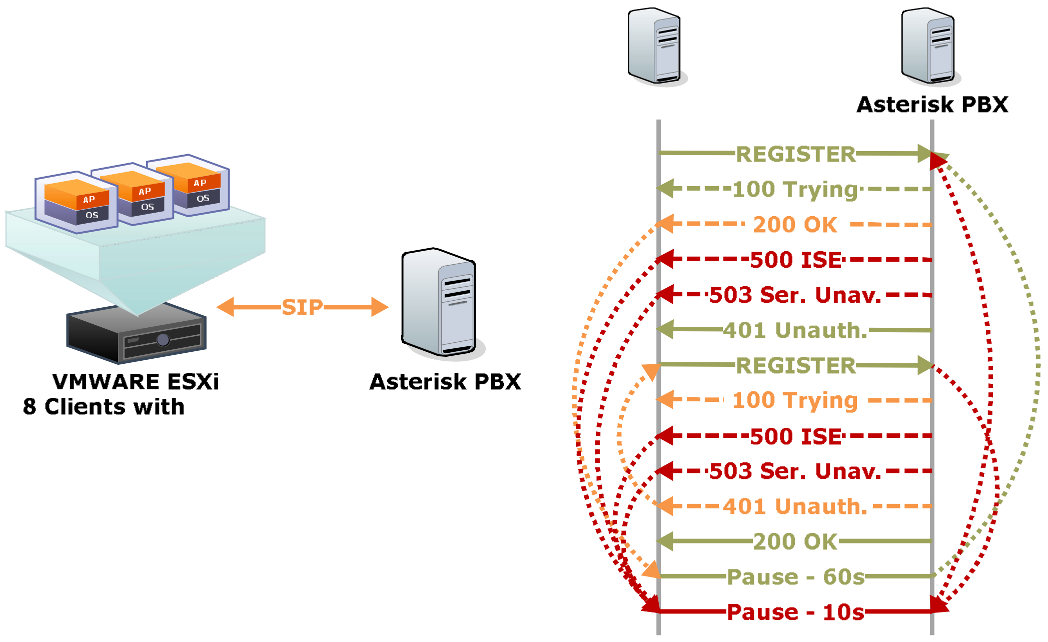


Figure 19: Test topology and the flow of SIP messages during the registration dialogue

The main configuration elements of the virtualised computers can be summarised as follows:

- Guest OS Ubuntu 10.10 x64 Server Edition;

- 1x Virtual x64 Processor Core @ 2.8 GHz;
- 2048 MB Virtual RAM;
- 8 GB HDD;
- UDP Buffer Size 131 071 B (default size);
- File Descriptor Limit 65535 (ulimit -n);
- Stack Size Limit unlimited (ulimit -s).

The key element of the platform – the SIP server - was realised on a separate hardware machine with this configuration:

- OS Ubuntu 10.04 x64 Server Edition;
- AMD Athlon X2;
- 4 GB DDR2 RAM;
- 500GB HDD;
- UDP Buffer Size 131 071 B (default size);
- File Descriptor Limit 100000 (ulimit -n);
- Stack Size Limit unlimited (ulimit -s);
- Asterisk 1.6.2.16.2;
- Asterisk 1.8.2.4;
- 100 000 SIP Peers.

Both network elements – the host virtualisation server and the SIP server - were interconnected via a gigabit switch to ensure minimal additional delays caused by the network infrastructure.

To reflect the practical situation, the measurement was performed only on client devices. When a SIP server was not accessible to perform measurements of any kind, these values were measured on the client devices:

- Number of individual SIP messages;
- Number of retransmissions;

- Number of Timeouts;
- Approximated RRD.

All these parameters were used to determine performance of the SIP server and the last one will be described in more detail and explained in the next section. The message flow of the registration dialogue is also depicted in Figure 19. The standard messages of the successful registration dialogue are coloured in green and are followed by a 60-second pause, after which the reregistration takes place. Additionally, out-of-sequence messages could be received from the SIP server, when the load exceeded the level SIP server can handle without significant delays. These messages are a valid part of the SIP dialogue and are coloured in orange. Each time such a message was received, the jump to the correct part of the dialogue was performed. When the load is even higher, the error messages or timeouts could occur. In this case, two 5XX messages were expected and when one of these messages was received or when one of the correct dialogue messages timed out, a second registration attempt was made after a 10 second delay.

6.1 Methodology and Limit Definition

The aim of the measurements was to determine how the SIP server Asterisk would respond to high burst loads of SIP registrations. These registrations were sent in five consecutive, one-second bursts with a given load and the client devices tried to hold these registrations for fifteen minutes by sending re-registration requests every 60 seconds. After fifteen minutes, all the measured parameters were logged and the process repeated with a higher load. If the registration attempt was not successful, no matter whether this was caused by an error message or a timeout, the client waited for ten seconds before he tried to register again. This way the SIP server was given the possibility to spread the load over a longer period and real-world, VoIP-client behaviour was preserved. Although the way the error messages and timeouts were treated was the same, the severity of these two kinds of errors is different. Although the receipt of an error message prevented the client from registering for about ten seconds (the error pause interval), after the message timeout, this period was extended to about 42 seconds, which was caused by the timeout mechanism of the SIP protocol. For this reason, timeouts have a greater impact on the evaluation of a SIP server's performance.

Due to the length of the test, the client attempted to send Register message fifteen times. This number did not include the retransmissions. From this number, the limit could be derived for determining whether the SIP server passed the test successfully or not. This result was interpreted to mean that the clients were not able to register successfully after 45 seconds if the number of timeouts exceeded 1/15 of the total number of unique Register requests sent. To ensure that SIP server had the capacity to recover from the burst, this limit was doubled. The same calculation can be made for the error messages, but with a lower weight because of the significantly shorter period when the client cannot register. Because no 5XX error message was received during the whole test, the analysis of this report's results will only work only with the number of 2/15 (~13%) timeouts as the limit for successfully passing the test.

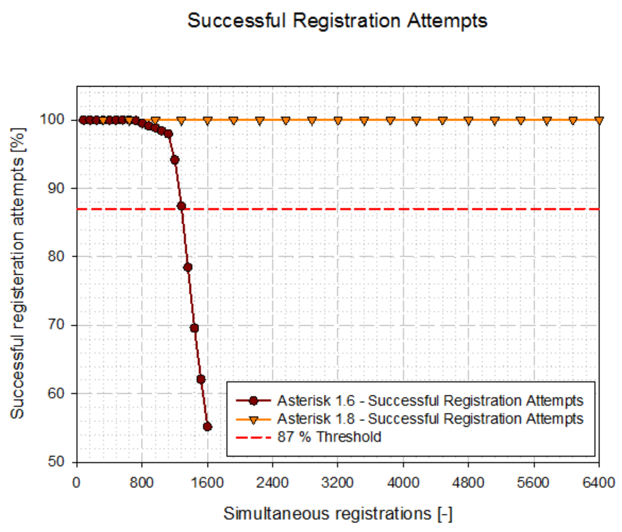
The approximated RRD, as defined in the previous section, was also measured. In previous work and in the RFC 6076, the RRD (Registration Request Delay) is defined as the time between sending the Register request and receiving the 200 OK response. Approximated RRD in this report was measured in exactly the same way, but due to the limitations of the SIPp in loop-scenarios, time measurement was not possible with the available timers and had to be performed by the logging mechanism of the SIPp. Therefore, no precise time could be written to the log. The most useful solution was to use the internal clock-ticks of the SIPp. One clock-tick is roughly comparable to one millisecond, but the precision may vary in higher loads Therefore, the measured parameter can be viewed only as a fair approximation of the RRD. Therefore, approximated RRD is not important for its absolute values but to indicate a trend while the load is being increased.

6.2 Analysis of the Results

In this section, the results are reviewed. In two subsections, four charts are presented and on each of these charts, the Asterisk 1.6 is represented by dark brown, rounded symbols, while Asterisk 1.8 is depicted by orange triangles. All of the measured parameters are related to the number of simultaneous registrations and each dot in the chart represents a single fifteen-minute step in the test process.

6.2.1 Successful Registration Attempts and SIP Timeouts

Successful Registration Attempts indicate the ratio between attempted registrations and successful registrations with 200 OK responses. The best and optimal value is 100 %, but no architecture has been designed to handle a huge number of registrations at once. Therefore, mechanisms were implemented to spread the load over a longer interval if an unsuccessful registration occurred. As mentioned in the previous section, the threshold calculated from number of timeouts was designated as 13%, and because the timeouts were the only errors measured during the test, this threshold was used directly to determine the number of successful Registration attempts.



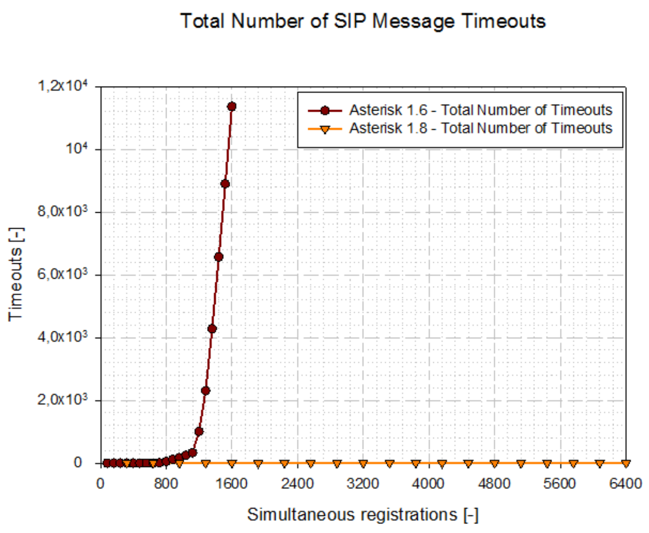


Figure 20: Successful registration attempts and the total number of SIP message timeouts

The charts in Figure 20 clearly indicate the difference between two architectures tested. While Asterisk 1.6 started having problems when a load of 1000 registrations was generated and fell under the designated threshold immediately after 1280 simultaneous registrations, Asterisk 1.8 holds the 100% ratio for the whole time of the test, displaying stable behaviour, even with a load of 6400 simultaneous registrations.

Similar behaviour can be seen on the chart depicting the number of SIP timeouts. For Asterisk 1.6, timeouts were the essential problem. On the other hand, no timeouts occurred while testing Asterisk 1.8.

From this information, it can be concluded that the new version of Asterisk handles the burst load of SIP registrations much better than the older one. It is noteworthy that all of the parameters were set to defaults on both Asterisks, and same number of SIP peers was used. Therefore, no external event could have influenced the results.

6.2.2 Retransmissions and Approximated RRD

From the information presented, a clear assumption can be made about which version is better in a situation of failure in network connectivity and subsequent recovery. The following explores whether the retransmissions and approximated RRD confirm this assumption.

Approximated RRD was explained in detail in the previous section, so no further explanation is presented. Retransmissions occur when there is long period between sending the message and receiving the appropriate answer. In the definition of a standard SIP timer, that the first retransmission takes place when the response is not received in 500 milliseconds after the request has been sent. Each subsequent retransmission is sent after a doubled interval, until four seconds are reached. When this occurs, all other retransmission are sent after four seconds, giving the following sequence of time periods between the neighbouring messages – 500ms, 1s, 2s, 4s, 4s… After a sequence of nine retransmissions, a timeout occurs. The number of retransmissions increases when the SIP server cannot successfully process all the messages. In the following charts, the total number of retransmissions means the sum of all retransmissions of all messages in the SIP registration dialogue.

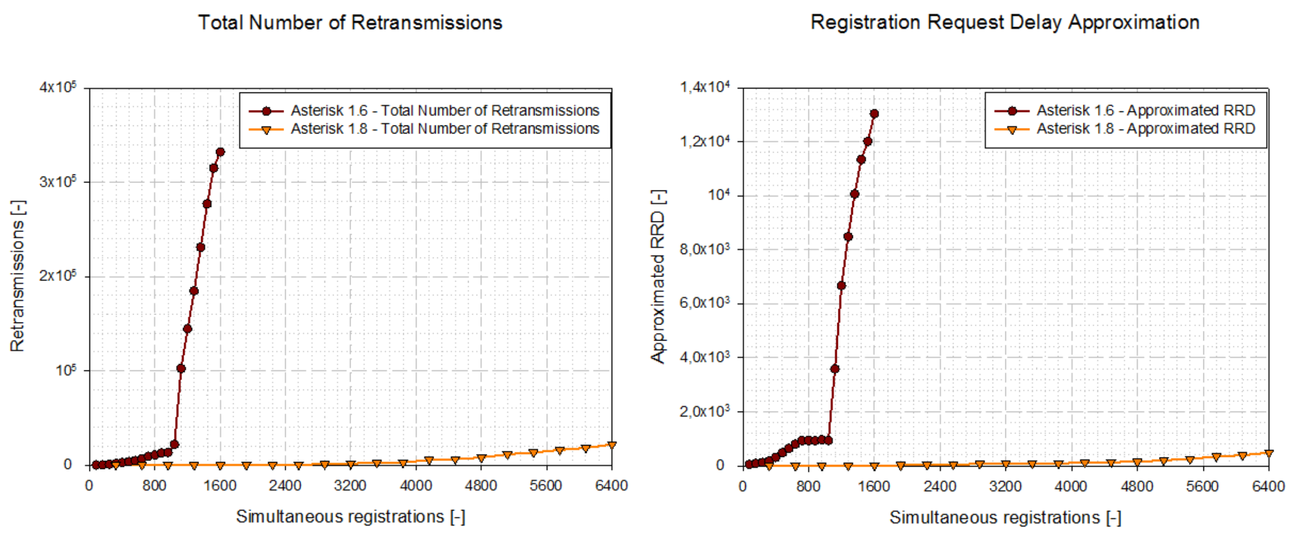


Figure 21: Total number of retransmissions and approximated Registration Request Delay

Both of the parameters that are depicted in Figure 21 – the total number of retransmissions and the approximated RRD – share the same trend and underscore the results described in the previous subsection. Asterisk 1.8 excelled, once again, achieving the minimal number of retransmissions even for very high loads, while for Asterisk 1.6, 1280 simultaneous registrations was the highest load it could process satisfactorily. If we use the close approximation of SIPp clock-ticks to milliseconds, we can see on the second chart in Figure 21, that for loads higher than 1280 registrations, the registration interval took up to fourteen seconds to complete, which is, of course, unacceptable. Asterisk 1.8 had no significant issues, even for loads that were five times greater.

7 Conclusion

The method the testing and benchmarking of SIP infrastructure presented in this report was designed for benchmarking SIP-based VoIP infrastructure. It enables users to determine the maximum load of the system, and reveals the dynamically changing characteristics of the system, such as response times and packet delay. The method is useful for deciding which system should be installed in a particular environment [roz], [voz222].

The method can be used to determine:

- The maximum number of simultaneous calls that B2BUA can handle in any particular configuration;
- The maximum number of calls per second that SIP Proxy can handle;
- B2BUA's effectiveness for transcoding.

Table 1 summarises some of the data collected about the SIP server and presents an example of output of measurements taken using our methodology.

	B2BUA (G.711u – G.711u)	**SIP Proxy (4 listeners)**
Criterion	*Max. simultaneous Calls [-]*	*Max. Calls per Second [s^{-1}]*
CPU Utilization	660	1600
RRD and SRD	600	1600
Jitter and MPD	600	-
Total	**600**	**1600**

Table 1: Example of a summary of the final results

This information can be acquired by examining the charts and looking for the optimum solution, which can be simply described as:

- The first point where maximum CPU utilisation is reached;
- Last point before a significant leap in any delay characteristic.

The results we have presented indicate that the new major version of Asterisk PBX offers a major leap forward in effectively handling burst loads of Register requests. This conclusion was reached, based on the data collected exclusively on client devices, but thanks to the possibility of collecting data on SIP servers we were able to determine that the only limiting factor was the Asterisk's capability to process the UDP segments from the UDP buffers, both successfully and quickly enough.

This example of measurements can also be combined with call tests, and there is also a possibility to test not only the bursts but also, to measure a slow, sequential increase in the number of simultaneous registrations. In other words, the possibilities of the newly redesigned platform are vast.

References

[ims]	IMS Bench SIPp – IMS/NGN Performance Benchmark, 2010. URL: http://sipp.sourceforge.net/ims_bench_sipp/index.html
[mal]	D. Malas, A. Morton, Basic Telephony SIP End-to-End Performance Metrics, IETF RFC 6076, 2011. URL: http://tools.ietf.org/html/rfc6076.
[por1]	Poretsky, S., Gurbani, V., Davids, C., Terminology for Benchmarking Session Initiation Protocol (SIP) Networking Devices, IETF draft, March 2011.
[por2]	Poretsky, S., Gurbani, V., Davids, C., Methodology for Benchmarking SIP Networking Devices, IETF draft, September 2011.
[roz]	J. Rozhon, M. Voznak, Registration Burst Load Test, Conference Proceedings ICDIPC 2011, *In SPRINGER Communications in Computer and Information Science* , (Part 2) 2011, Vol. 189 CCIS, pp. 329-336, July 2011, ISSN 1865-0929, ISBN 978-3-642-22409-6.
[sipp]	SIPp – performance testing tool for SIP protocol, 2010. URL: http://sipp.sourceforge.net
[tran1]	Transnexus, Performance Test of Asterisk V1.4 as a Back to Back User Agent (B2BUA), URL: http://www.transnexus.com/White%20Papers/Performance_Test_of_Asterisk_v1-4.htm, 2010.
[tran2]	Transnexus, Performance Results for OpenSER and SIP Express Router, 2010. URL: http://www.transnexus.com/White%20Papers/OpenSER-SER_Comparison.htm
[voz133]	Voznak,M.-Rozza,A.-Nappa,A.Performance comparision of secure and insecure VoIP environments.*TERENA Networking Conference 2008*, TERENA, Brugge, Belgium, May, 2008.
[voz173]	M. Voznak, J.Rozhon, SIP Infrastructure Performance Testing, p. 153-158, *In Proceedings of the 9th International Conference on TELECOMMUNICATIONS and INFORMATICS* , Catania, Italy, May 29-31, 2010, ISBN 978-954-92600-2-1, ISSN 1790-5117.

[voz179] M. Voznak, J. Rozhon, Performance testing and benchmarking of B2BUA and SIP Proxy, In conference Proceedings TSP 2010 , p.497-503, August 17th-20th 2010, Baden near Vienna, Austria.

[voz188] M. Voznak, J. Rozhon, SIP Back to Back User Benchmarking, *IEEE Computer Society: ICWMC 2010*, pp. 92-96, September 2010, University of Valencia , Valencia, Spain.

[voz191] M. Voznak, J. Rozhon, Methodology for SIP Infrastructure Performance Testing , *WSEAS Transaction on Computers*, Issue 1, Volume 9, September 2010, ISSN: 1109-2750, pp. 1012-1021.

[voz222] M. Voznak,J. Rozhon, Approach to stress tests in SIP environment based on marginal analysis, *In Journal SPRINGER: Telecommunication Systems* , 11p., DOI: 10.1007/s11235-011-9525-1, June 2011, ISSN 1018-4864 (Print), ISSN 1572-9451 (Online).

Glossary

B2BUA	Back-to-Back User Agent
CPU	Central Processing Unit
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IVR	Interactive Voice Response
LE	Large Enterprise
NGN	Next Generation Network
NREN	National Research and Education Network
PBX	Private Branch Exchange
PCM	Pulse-code Modulation
RFC	Request For Comment
RRD	Registration Request Delay
RTP	Real Time Protocol
SAR	System Activity Report
SER	SIP Express Router
SIP	Session Initiation Protocol
SME	Small- and Medium-sized Enterprise
SRD	Session Request Delay
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
VoIP	Voice over IP