SIP End to End Performance Metrics

Miroslav Voznak and Jan Rozhon

***Abstract*—**The paper deals with a SIP performance testing methodology. The main contribution to the field of performance testing of SIP infrastructure consists in the possibility to perform the standardized stress tests with the developed SIP TesterApp without a deeper knowledge in the area of SIP communication. The developed tool exploits several of open-source applications such as jQuery, Python, JSON and the cornerstone SIP generator SIPp, the result is highly modifiable and the application is able to carry out benchmarking in accordance with RFC 6076. The main advantage is high performance, meanwhile has been tested up to tens thousands simultaneous connections, and scalability.

***Keywords*—**SIP Benchmarking, SIPp, RFC 6076, RRD, SRD.

I. Introduction

THE end-to-end performance metrics enable to determine performance characteristics of SIP servers or User Agents and are crucial in design of modern SIP communication infrastrucure. With proposed methodology for testing and benchmarking SIP infrastructure, we had the opportunity to perform several series of tests on multiple different platforms. From these tests we realized, that it would be very beneficial to modify the existing testing platform to allow us for performing separate test scenarios on each of the important SIP dialogs. This way the movement towards the modular design started. During this work at the beginning of 2011 the new RFC 6076 was adopted finally standardizing most essential measured parameters [1].

With the parameters standardized we have developed the most important testing scenarios – the registration test scenario and the call test scenario, both having its roots in the previously used scenario for complex performance measuring [2],[3]. Each of those scenarios offers a different perspective when defining the SIP server limits and can be run either separately to test some special environments or occasions or simultaneously to simulate the real VoIP client behavior. The latter presented a big challenge, because the testing software does not allow running multiple scenarios at once inherently. However this problem was walked around by exploiting SIP security vulnerability, which allows a client from one address register another. This way the basis of module based testing platform has been created.

In this article we present the example of results gained by testing different versions of most commonly used VoIP PBX Asterisk focusing on its ability to handle multiple simultaneous registrations coming in several consequent bursts. This example is particularly useful to determine how the SIP server reacts in the case of network failure and consequent restoration of full connectivity, when all the clients try to register at once. In the given examples the way how the SIP server responds to bursts with high loads can be determined and all the conclusions are made according to information obtained by the measurements on the client side exclusively, because the measurements on the server side are often impossible with regard to the provider restrictions.

II. State of the Art

Due to the integration of SIP protocol to the concept of next generation networks the implementations of this protocol are now commonplace. This results in the increasing need for the methodology and tools that would allow for determination whether selected hardware has enough performance to cover all the needs of the given environment with some spare for the future expansion. However, in these days this demand can be satisfied just by the proprietary solutions, which on one hand require special hardware and on the other do not provide the results that would be comparable with the results taken by some competitive solution [4].

The open-source solutions, on the other hand, 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 feature may look like a disadvantage, it can easily be utilized to perform test 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 these phases have already been partially solved. The definition of parameters, conditions of the test and the whole methodology has been done in RFC 6076 and IETF drafts [5], [6] and these drafts will probably form the basis of the future IETF standards that is why it is useful to utilize the knowledge contained within them.

As for the the implementation the great work has been done by the American IT company Transnexus, which created a complex scenario for testing both most common variants of SIP Server [4]. This implementation utilizes the capabilities of open-source testing tool program call SIPp. This implementation has severe deficiencies though. First, it is primarily designed to test the environment, the base part of

Manuscript received March 8, 2012. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 218086.

Miroslav Voznak is an associate professor with the Department of Telecommunications, VSB –Technical University of Ostrava, Ostrava, Czech Republic (phone: +420 597321699; e-mail: miroslav.voznak@vsb.cz).

Jan Rozhon is a PhD. student with the Department of Telecommunications, VSB –Technical University of Ostrava, Ostrava, Czech Republic (phone: +420 597321641; e-mail: jan.rozhon@vsb.cz).

which is the proprietary server of this company. Second, the measured parameters and the overall methodology is unsuitable for recognition what of the measured time intervals belong to the same call, which results in inconsistency of the output results, from which only raw conclusion can be made. The last deficiency of the Transnexus' test topology that should be mentioned is the huge complexity of the design, because many hardware components are required and because too many variants of a call setup are tested. The named attributes of the Transnexus' design result in complicated hardware and software implementation of the design, which makes it unsuitable for the practical use.

Our work primarily focuses on combining the best parts of the two named sources – IETF drafts [5], [6], recently issued IETF RFC 6076 [1] and Transnexus' design [4]. The result of this combination is then modified, so that the final solution is simple enough for the practical use and comprehensive enough to provide all the needed data for the performance analysis of the SIP Server. This modifications also include the means for testing the B2BUA platform independently due to the harnessing of a media flow through it [7], [8]. In this article the complete methodology as well as the test scenario design are presented. Furthermore, the output results are included in a human readable form of the charts and these charts are commented so that everyone who reads this article can easily come to the final performance results.

III. Methodology

Transnexus used in their testing platform the open-source testing tool SIPp, which allows for generation of high SIP load and measurement of key parameters of individual SIP calls. This makes the SIPp the ideal option for SIP performance testing. Although Transnexus' benchmarking model served as an inspiration in the early phase of the development of our methodology it lacks the effort for standardization. They measure times between transmission and reception of some key messages (e.g. Invite, 100 Trying, 180 Ringing), however their approach does not look at these messages as the part of the SIP transaction. This results in outputs from which the user is unable to read more complex attributes of the system. To be more specific, you can learn how quickly the SIP server is able to respond to your message, but you cannot learn how quickly it can process and resend to the destination. Our approach on the other hand makes this possible, so it is not the issue to recognize the "real world" parameters of the SIP server such as Call Setup Length (later described as SRD).

From the practical point of view Transnexus' model is rather too complex. As the commercial subject, Transnexus has focused on creating the model that would utilize some of their commercial products, which led them to use their management and billing platform, which required two more separate computers. Moreover, the testing scenarios they created utilize several different end locations for the simulation of call rejection, no route issue, no device problem and so on. This again increases the complexity of the test platform due to the need of more physical machines. From mentioned it is clear that this model is unsuitable for practice. From our point of view it is beneficial to create the testing platform that would be as simple as possible, which would make it easier to deploy in any practical environment. This is why we decided not to use any other special hardware and to simulate the end location for calls just by the listening UASs, which is made possible by the fact that we want to evaluate the ability of the SIP server to successfully connect calling and called party.

In order to perform SIP testing, we simulate 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 always involving SIP Registrar and SIP Proxy or B2BUA (Back to Back User Agent). The latter is the most used solution in enterprise environment, for both SMEs (Small and Medium sized Enterprise) and LEs (Large Enterprise). Fig. 1 depicts test hardware configuration for testing the SIP Proxy and B2BUA.

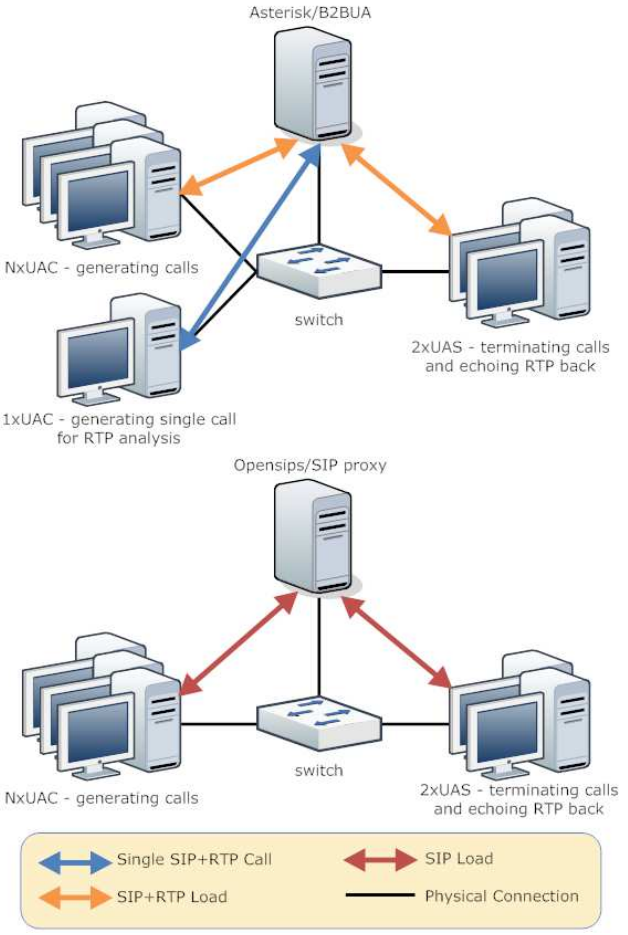


Fig. 1. Test Bed Diagram for B2BUA and SIP Proxy.

This is a general configuration which does not reflect all the aspects of test platform used for our measurements. Firstly, we used both physical and virtual computers to simulate SIP traffic. The results with both configurations were almost identical allowing future user of this methodology to decide for topology that would be best for him according to available hardware.

The only condition required for testing SIP server successfully and comparably is the interconnecting device (or

system). Basically, this can be any device or network capable of routing of SIP messages among SIP traffic generators, SIP server and SIP traffic recipients, but to make the results of measurements comparable with those taken in different network, we would be required to use the exact same topology, which may be the issue. This is why it is advantageous to use as simple topology as possible to reduce additional costs and work caused by the need of some special topology. So, the most flexible variant is to use the single switch, which is undoubtedly a commonplace in all modern SIP installations.

Secondly, the number of devices used for the testing may vary due to the performance of the SIP server. The more the SIP server is efficient 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) no matter what the hardware configuration of the PC running SIPp instance is. Therefore we need to estimate the SIP server performance to determine the number of computers (physical or virtual) needed for test, which makes the virtualization the more viable option. Number of UASs is not affected by the SIP server's performance that much, however it is necessary to force the SIP server to decide between different paths to UAS, therefore there have to be at least two computers in UAS mode in the test topology.

As well as the topology the test scenario should be as simple as possible mainly to reduce the complexity of the test and except of that also because it is not possible to test the SIP Proxy (and B2BUA as well) in all the possible configurations. Thus it is useful to focus on basic default configuration and perform the tests with it. The output results then carry the information about the "best case scenario" according to which we can decide about the SIP server's performance and compare it with its rivals.

A. *Measured Parameters*

As mentioned in the Introduction we use the parameters defined in IETF draft for all our measurements [1],[4],[5]. But except of them we use the hardware utilization parameters as well. Let's now take a look at the locations, where these groups of parameters are measured.

First group is measured at UAC and includes the call statistics such as number of (un)successful calls and durations of the message exchanges. RTP samples for analysis are captured here as well.

Second group – the hardware utilization parameters – is measured directly on the SIP server. At this place CPU and memory utilization and network traffic is measured. The complete list of all measured parameters includes:

- CPU utilization.
- Memory utilization.
- Number of (un)successful calls.
- Registration Request Delay – time between first Register method and its related 200 OK response [3].
- Session Request Delay (SRD), the time between first Invite method and related 180 Ringing message [2],[3].
- Mean Jitter a Maximum RTP Packet Delay.

Fig. 2 shows the meaning of the RRD and SRD delays in more detail.

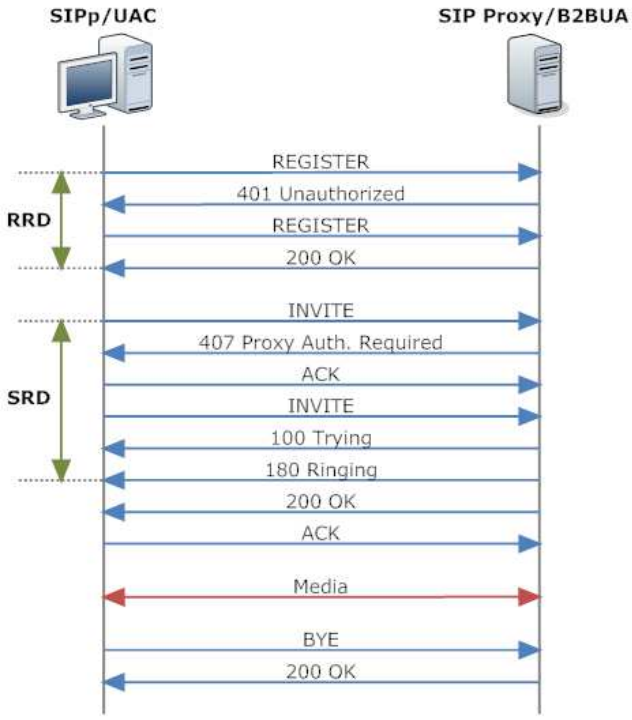


Fig. 2. RRD and SRD in SIP Dialog.

B. *Limit Definition in Results Analysis*

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 limit values for each category of the measured parameters. This definition must come out from the features of the SIP protocol and generally recognized convention from IP and classic telephony.

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

In general, the CPU utilization characteristic is limited by the maximal CPU performance, which is 100%, but this boundary can be reached rarely. To be more specific, due to the time intervals between particular measurements of the CPU utilization can cause that short peak in CPU utilization characteristic is not registered. However, during this peak delays and call quality impairments can occur. To reflect this imperfection of our methodology, performance boundary under 100% should be anticipated. Actual value of the CPU performance boundary may vary, though. Therefore we search the CPU utilization characteristic for the first point where maximum CPU utilization is reached. This point is then the

maximum number of calls, which the SIP server can handle from the hardware performance point of view.

The limit definition for the SIP delay characteristics RRD and SRD comes from the nature of the SIP protocol [9], [10]. When the call is set up the delays between messages should not exceed several hundreds of milliseconds and although these limitations are tied up with the travel of the SIP message from one end of call to another, it can be used for our purposes as well, because of the similarities that come from the need to set up a call quickly enough not to bother the user with noticeable delays.

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

The quality of speech is vulnerable to great delays between consecutive RTP packets. It is affected by the jitter as well, but the jitter issue can be eliminated by the sufficient Jitter buffer on the receiving side, therefore maximum packet delay is the key characteristic in RTP stream analysis [11], [12]. From the theory of IP telephony the delays between packets should be in the 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 the delay characteristics use similar analogy with the theoretical values for end-to-end delays, that is why their definition could not be exact and these parameters may vary in different environments. To eliminate different interpretation of the same results and to simplify the delays analysis, we use as the quality boundary for all the delay characteristics the point, where the particular characteristic change its “almost constant” trend to rapid increase. This approach gives us correct results, which was tested experimentally, and the methodology of the analysis is much simplier.

C. SIP Proxy Testing

In basic configuration of the SIP Proxy we are able to measure just the SIP and utilization parameters. RTP stream does not flow through SIP Proxy and thus it does not represent the load for it. This is why we do not have to think about the call length because no matter how long the call is the hardware utilization 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 on SIP Proxy consists of several steps. Every single step takes about 16 minutes, this means that for 15 minutes, 10-second long calls are to be generated at a user-defined call rate. Then there is a 10-second period when the unfinished calls are terminated. This repeats for every single step of the call rate. Every call consists of a standard SIP dialogue and pause instead of media. Because the load is not constant but increases slowly at the beginning of the test (first 10 seconds) and decreases at the end of it (last 10 seconds), the results taken after this starting period and before the ending one are the only ones which are going to be considered valid. To allow additional changes in time interval setting in the scenario and to strengthen the consistency of the method we decided to use the data collected during the middle 10 minutes of each step. All the parameters named in the previous subsection are measured except those related with RTP stream.

The 10 second long time interval that was mentioned several times came from the compromise between reasonable call length and the need for generating as much of the calls per second as possible. It allows for decent performance and does not require huge database of subscribers. This interval can be changed but cannot exceed 2.5 minutes that allow for collecting the valid data.

SRD is measured although this scenario cannot be considered as end-to-end, this condition is defined in draft [6]. We decided to measure it because the load on the UASs is minimal even for high call rates, which makes the delays created by the UASs both minimal and almost constant. Therefore we can use this parameter to decide about the SIP Proxy’s performance, because the delays created by it are the only variable making the collected data useful. This is the only deviation of our method from the draft [6].

D. B2BUA Testing

Unlike SIP Proxy for this type of SIP server the RTP stream presents the highest load on the SIP server therefore the number of simultaneous calls must be used as a metric. This is the main difference between the B2BUA and SIP Proxy testing scenarios. Second not so important difference (from the methodology point of view) is that in this configuration we are to measure effectiveness of codec translation because in this scenario performance of the B2BUA is not affected only by its setting but also by UAC and UAS configurations. The test routine will then be repeated for each case of different codec setting.

The method of the test is however almost the same, the only issue we face is the new metric together with the need for revising the time interval for a single call. The new metric is an issue when the SIP traffic generator cannot be ordered to create certain number of simultaneous calls. In this case it is necessary to calculate the number of calls generated per second. This can be done by this equation:

$$C_R = C_S \cdot T \tag{1}$$

C_R is the desired Call Rate, C_S is the number of simultaneous Calls we want to generate and T is Time interval defining how long the call (media) should be. 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 use a special computer, which allows us to use more sophisticated tools for capturing the network traffic without the RTP and SIP parts of the tests influencing each other. Because we focus on testing effectiveness and speed of codec translation we were, at this point, able to determine the maximum load which the SIP server can handle from the SIP or RTP point of view. However, these results would only be

valid for a single machine/platform and that is why we add one more step to the data analysis. The same procedure of testing as mentioned above is performed on a machine configured to allow media to only pass through the SIP server. The results taken during this test serve as a basis to which we relate all the other results. The relation is expressed in (2) as a performance ratio. The performance rating factor P_{RF} is a ratio of any previously mentioned parameter measured in codec translation case (P_{CT}) with a certain number of simultaneous calls to the value of the same parameter (P) taken in case without codec translation and the same load.

$$P_{RF} = \frac{P_{CT}}{P} \cdot 100 \qquad (2)$$

This step allows us to compare the results from hardware and platform independently [17].

IV. Platform Description and its Enhancements

To successfully generate high loads of SIP traffic we have been using the open source traffic generator SIPp. This software allows for generating both simple and complex SIP dialogs with the emphasis on scenario modifiability. The scenarios are defined in the XML format which makes it possible to create and generate well-formed SIP messages as well as the malformed ones; therefore it can be used for security tests as well. Unlike other SIP traffic generation tools such as Seagull [13] or SIPSak [14], which are open-source as well, SIPp offers much greater variety of possible usage starting from optional messages in SIP dialog, through automatic call identification and dialog-oriented variables generation to RTP stream support.

Using the XML SIPp can create many calls and route them to the SIP server, however to successfully stress test the tested infrastructure we need to create huge number of calls (simultaneous or generated per time unit). And this is where SIPp's inherent limitation comes to scene preventing us to reach reasonable loads. This limitation comes from the SIPp's single threaded software design preventing it from using multiple processor cores to increase its call capacity. This could have been easily worked around by running multiple processes of the SIPp, if there was not a particular problem in the SIPp source codes. This problem appears when multiple virtual network interfaces are being used. SIPp ignores the command line arguments instructing it to use a specific network interface and automatically falls back to the primary network interface. This problem is connected with the media stream only; therefore it does not influence the SIP signaling messages. However, this problem prevents the user from generating most utilizing part of the call and thus diminishing the testing reasonability. Because of the stated problem we needed to use multiple computer platform design or virtualization techniques to spread the load evenly among the processor cores. By adopting this design we were able to create a powerful and stable testing platform which was on some hardware able to generate hundreds of simultaneous calls with media allowing us to stress test low to middle SIP servers. However the management and control over the platform was problematic and needed to be improved so it can be used by less experienced users and so its preparation is not so time consuming.

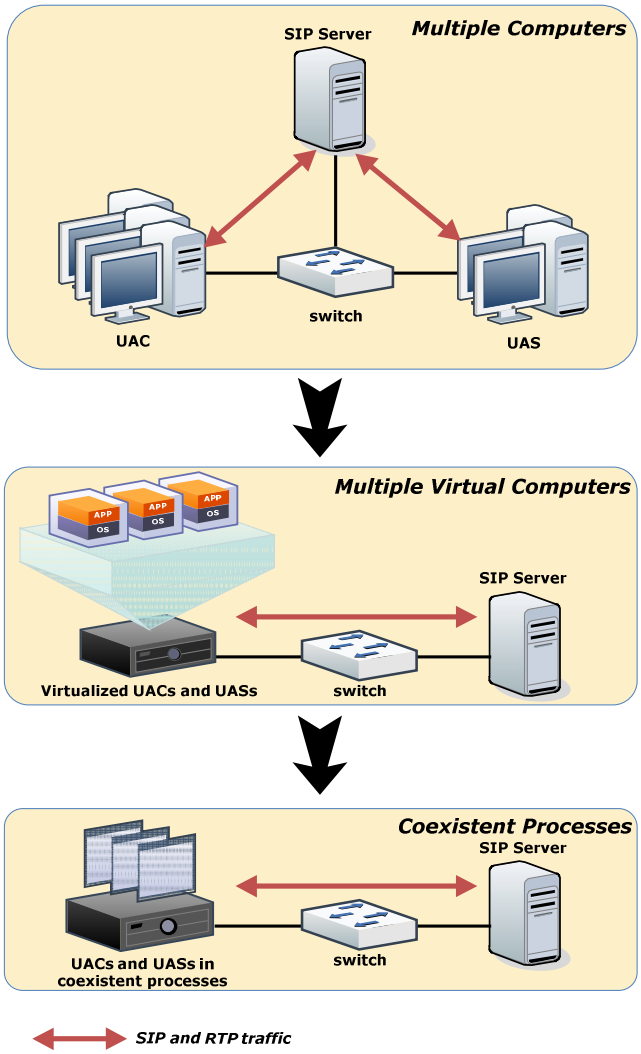


Fig. 3. Test platform development in time.

In this phase we focused on the right coding the SIPp's media capabilities, since it was the only feasible way to make it run on the single computer in multi-process mode. Through analysis of the source code we managed to find the problematic section of the code and fix it. This way we removed the biggest obstacle from our way to multi-process design. The whole platform synchronization moved from SSH protocol to internal scripting, which is much more efficient and convenient. The whole process of testing platform development is depicted on the Fig. 1, which illustrates the transition from multi computer design to design based on virtual computers and finally the design based on multi process approach, see Fig. 3.

Apart from code redesign we had to face the fact that multiple processes of SIPp caused nonlinear utilization of the hardware resources of the computer. Basically, we discovered that the distribution of the individual processes among processor cores is not performed well by the standard Linux kernel utilities – IRQbalance mostly. This utility tries to spread the interrupts generated in our case mainly by the network traffic. However in our case it causes the individual interrupts to be mapped to a core and then remapped again very frequently, which causes additional load mainly in the CPU cache memory. This had to be countered to allow our platform to reach highest possible load. Through analysis of the individual interrupt handling cases – without IRQbalance, with it and with manual distribution using SMP_affinity, we came to the conclusion that the manual distribution of interrupts serves our purpose best. The illustration of how the interrupts are handled in the 4-core system is depicted on the Fig. 4.

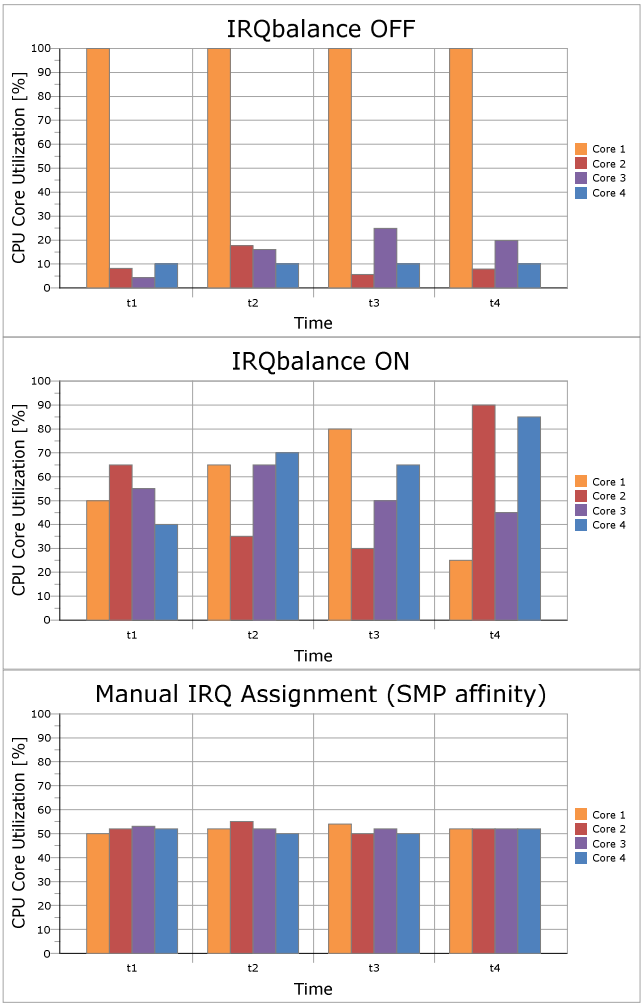


Fig. 4. Illustration of CPU Core Utilization with different kinds of Interrupt Handling.

This conclusion resulted in emergence of a new hardware requirement, since the interrupts can be manually assigned to processor core only when the network interface card supports the MSI-X technology. Since the load generated on our platform is composed from hundreds of separate low bandwidth connections the higher number of interrupt queues is beneficial. This led us to use NIC with Intel 82575/82576 chipsets.

Another problem we were facing when using the multiprocessing design was the high memory utilization especially in higher loads. This was caused by the buffering. Generally speaking, when UDP datagram is received it is firstly stored in the RAM until the process is given the opportunity to receive the datagram from the RAM. The total amount of memory one connection can use is precisely defined in the kernel as the UDP buffer size. In a case when huge number of UDP datagrams is received every second and the hardware is not performing enough to let the corresponding process handle this datagram stream is being stored in RAM until the maximum size of the buffer is reached, after that all new received datagrams are dropped. So in this case of high loads we face two opposing problems. First we need to set the UDP buffer size as large as possible to keep as many UDP datagrams as possible and second we need to reduce the UDP buffer size as much as possible to reduce the amount of memory which is then allocated to each connection. To gain the optimal size of the UDP buffers we performed a series of measurements which resulted in setting the default size to 256kB and the maximum possible size to 1024kB, which allowed us to reach high load without exhausting all our memory and to perform well enough not to interfere with the SIP and RTP sessions. The Fig. 5 shows illustratively the mentioned compromise.

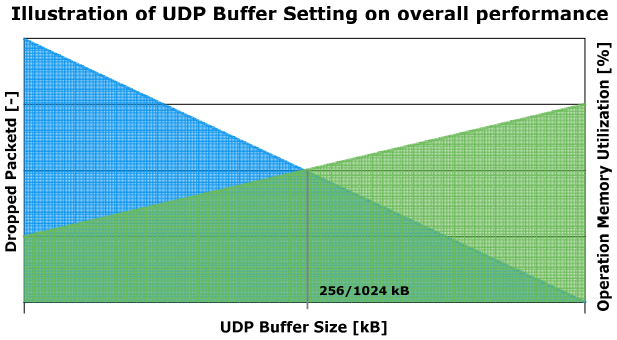


Fig. 5. Illustration of how to get the best UDP buffer size

By accomplishing the steps we have mentioned in this section we have created a powerful hardware optimized platform for generating high number of simultaneous calls reaching to somewhat about 6 000 simultaneous calls on 12-core machine allowing us to stress test the SIP elements of almost any performance [15], [16].

V. Platform Management and Control

Since the performance of the platform is not an issue now we moved our attention to the management and control mechanism so we could allow even not experienced user to perform a testing using our software tools and guides. To

achieve this we need to move the test control from the linux command line to a high layer graphical tool. Since the whole platform is meant to run on the high performance servers where display might not be present or available the web application was a logical step.

There are several languages to pick up from including the PHP, Perl, Python or Java. To allow for a rapid application development we decided to use a web framework that would provide us with enough predefined functionality to develop the application as quickly as possible but allowed to modify and control its function widely enough to fit our needs. Through series of attempts with different frameworks we focus on the Web2py framework which is written in Python and which provides both efficiency and user convenience.

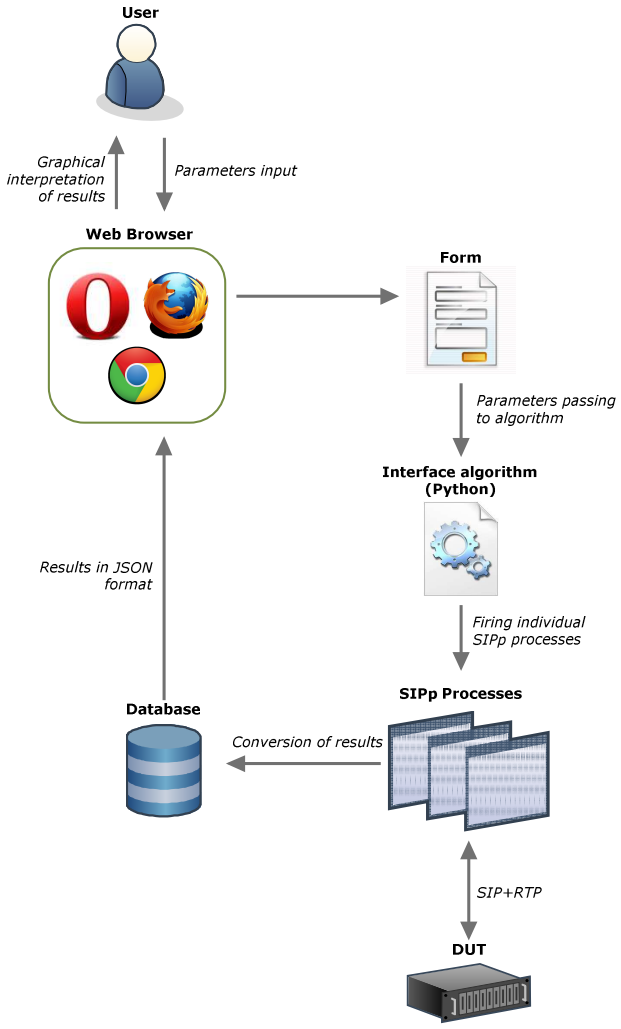


Fig. 6. Web interface application functionality scheme.

Using this framework we have developed a web interface application which provides functionality to run and monitor tests, view results and manage the hardware utilization. This application uses these technologies as its important and integral parts:

- Web2py framework for web page generation,
- JSON format to transfer results,
- jQuery to display the results graphically,
- Python to run the utilization distribution algorithms,
- SIPp for call signaling and media,
- SQLite database to store the results and test parameters.

Since all the mentioned technologies and applications are distributed freely under the GPL license or its derivatives the whole solution when finalized will also be distributed under this license.

The basic functionality of the web application is depicted on the Fig. 6 and shows us that the user enters basic parameters of the test to the form which is displayed as the web page on his browser.

Fig. 7. Web interface application with example of test results.

The data the user enters include IP addresses of individual UACs and UASs, address of the SIP server, scenario, which is to be used etc. These parameters are then passed to the python algorithm using the POST method. The algorithm then counts the best possible distribution of the load among the CPU cores and recounts the parameters so it is usable for individual SIPp processes, after which it runs them. The result files of the individual SIPp processes are periodically monitored and parsed and the data stored in them is inserted to the database from which the user can access it using its browser. The data between server and user's browser are encoded in JSON format and then interpreted using jQuery library Highcharts so that the user has the graphical overview of the ongoing test. The example of test results in graphical view is depicted on the Fig. 7. The data about the performed tests and its input parameters are stored for the future usage so that the user can

repeat a test as many times as it is needed without the need to repeatedly enter the parameters to the form.

This way the powerful web interface can improve the user experience with our platform and allow not experienced users to use it.

VI. Results

As an example of benchmarking, we analyse B2BUA in case with transcoding and without transcoding. The data collected during the whole test of B2BUA are in text format (binary data can be converted), so the data analysis can easily be done by any spreadsheet application, but for the correct interpretation of the data we have to perform a series of the same measurements to ensure that the effect of random events such as data packet scheduling techniques is marginal. The actual data then can be determined as the average of the collected data or the multitude of measurements can just serve to reveal the flawed data, which then can be replaced by the interpolated values.

Chart on Fig.8 clearly illustrate that the call is set up even quicker when there is a codec translation in use and the load is under 240 simultaneous calls, it is valid for Asterisk 1.6.2. Then, as the CPU utilization increases, the delays get very long. The fluctuations in charts with normalized values are caused by the random events during the measurements. Because we relate these values in a single equation (2), the variances get more distinctive, however this does not affect the final decision about the B2BUA performance from the SIP point of view. The Fig. 9 depicts a situation without codec translation where we can observe a rapid increase of delays between 600-660 simultaneous calls.

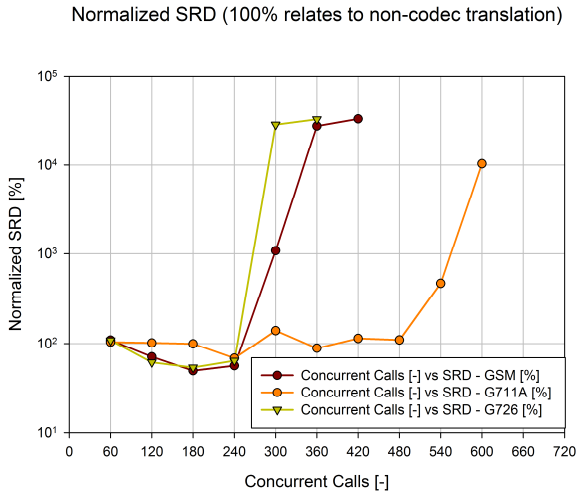


Fig. 8. B2BUA (Asterisk 1.6.2) with transcoding.

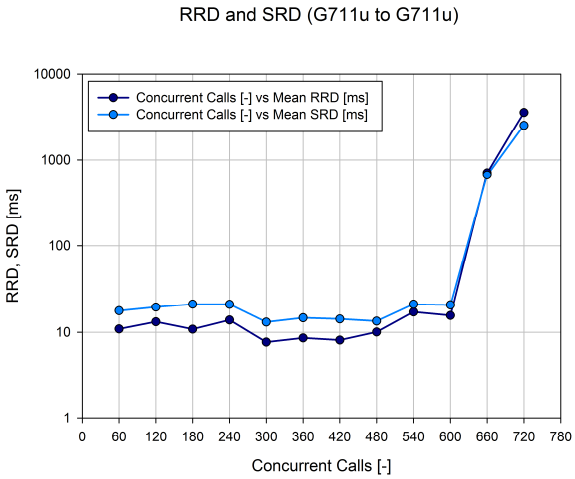


Fig. 9. B2BUA (Asterisk 1.6.2) without transcoding.

VII. Conclusion

With the testing methodology standardized [1] we were able to concentrate more effort on a creation of the interface that would allow for running the vast diversity of tests based on the simplified user input form. The newly created high level management and control interface is primarily meant for the not experienced users that might have the need to perform a stress testing on their own VoIP infrastructure such as network administrators, telecommunication engineers and so on, who have no mean to perform this kind of testing, because as far as we know, there is not a similar solution, which can be used for stress testing and benchmarking of SIP infrastructure with high level control and management system based on web technologies. Using the technologies such as jQuery, Python, JSON and others together with the code change in the cornerstone application SIPp this was made possible and the final solution is now being tested for the possible faults and instability. The main contribution to the field of performance testing of SIP infrastructure lies in the possibility to perform the standardized stress tests with our developed SIP TesterApp without the deeper knowledge in the area of SIP communication. Our tool exploits several of open-source applications and the result is highly modifiable tool. Further step of the platform development is the easing of software distribution using one of the popular package management systems. This can be achieved till the end of the 2012.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 218086.

REFERENCES

[1] D. Malas, A. Morton, "Basic Telephony SIP End-to-End Performance Metrics," *Internet Engineering Task Force: RFC 6076*, 2011, ISSN 2070-1721.

[2] M. Voznak, J. Rozhon, "Approach to stress tests in SIP environment based on marginal analysis," In *SPRINGER Telecommunication Systems*, 11p., DOI: 10.1007/s11235-011-9525-1, June 2011.

[3] 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, July 2011, pp. 329-336.

[4] Transnexus, Performance Test of Asterisk V1.4 as a Back to Back User Agent [Online]. Available: http://www.transnexus.com/.

[5] S. Poretsky, V. Gurbani, C., Davids, "Terminology for Benchmarking Session Initiation Protocol (SIP) Networking Devices, " *IETF draft*, March 2011.

[6] S. Poretsky, V. Gurbani, C., Davids, "Methodology for Benchmarking SIP Networking Devices, " *IETF draft*, September 2011.

[7] S. Narayan S., S. Kolahi, R. Waiariki, M.Reid, "Performance Analysis of Network Operating Systems." Proceedings of the *2nd WSEAS International Conference on COMPUTER ENGINEERING and APPLICATIONS*, Acapulco, Mexico, 2008.

[8] C. Simian, V. Georgiev, "On Some Aspects Regarding Computer Networks' Performance Analysis." Proceedings of the *13th WSEAS International Conference on COMPUTERS*, Rhodos, Greece, 2009.

[9] I. Baronak and M. Halas, "Mathematical representation of VoIP connection delay," *RADIOENGINEERING* Volume: 16 Issue: 3, September 2007, pp. 77-85.

[10] F. Alvarez-Vaquero, J. Sanz-Gonzalez,"Network VoIP for corporative environment design," In Proc. *7th WSEAS International Conference on Telecommunications and Informatics*, May 2008, Istanbul, Turkey, pp. 194-198.

[11] A. Kovac, M. Halas, M. Orgon, M. Voznak, "E-model MOS Estimate Improvement through Jitter Buffer Packet Loss Modelling," In *Advances in Electrical and Electronic Engineering*, Volume 9, Number 5, December 2011, pp. 233-242.

[12] M. Kavacky, E. Chromy, L. Krulikovska and P. Pavlovic, "Quality of Service Issues for Multiservice IP Networks, " In Proc. *SIGMAP 2009 International Conference on Signal Processing and Multimedia Applications*, Milan, Italy, July 2009, pp. 185 – 188.

[13] Seagull development team, Seagull reference documentation [Online]. Available: http://gull.sourceforge.net/doc/index.html.

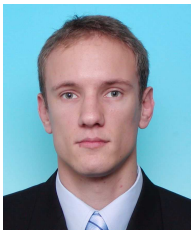
[14] SIPsak development team, SIPsak home page [Online]. Available: http://sipsak.org/.

[15] M. Voznak, J. Rozhon, "Methodology for SIP infrastructure performance testing," *WSEAS Transactions on Computers*, Volume 9, Issue 9, September 2010, pp. 1012-1021.

[16] M. Voznak, J. Rozhon, "SIP back to back user agent benchmarking," Proceedings - *6th International Conference on Wireless and Mobile Communications*, ICWMC 2010, Valencia, September 2010, pp.92-96.

[17] M. Voznak, J. Rozhon, "SIP infrastructure performance testing," 9th WSEAS International Conference on Telecommunications and Informatics, TELE-INFO '10, Catania, May 2010, pp. 153-158.

Miroslav Voznak holds position as an associate professor with Department of Telecommunications, Technical University of Ostrava. He received his M.S. and Ph.D. degrees in telecommunications, dissertation thesis "Voice traffic optimization with regard to speech quality in network with VoIP technology" from the Technical University of Ostrava, in 1995 and 2002, respectively. Topics of his research interests are Next Generation Networks, IP telephony, speech quality and network security.



Jan Rozhon received his M.S. degree in telecommunications from VSB – Technical University of Ostrava, Czech Republic, in 2010 and he continues in studying Ph.D. degree at the same university. His research is focused on performance testing of NGN and in this field he cooperates with CESNET association. He received rector's appreciation for scientific contribution of his diploma thesis in 2010.