SIP Overload Control Testbed

Roly, Antoine; Schumacher, Laurent

Published in:
2011 IEEE Consumer Communications and Networking Conference

DOI:
10.1109/CCNC.2011.5766460

Publication date:
2011

Document Version
Early version, also known as pre-print

SIP Overload Control Testbed: Design, Building and Validation Tests

Antoine Roly
University of Namur
Faculty of Computer Science
Namur, Belgium
Email: aro@info.fundp.ac.be

Laurent Schumacher
University of Namur
Faculty of Computer Science
Namur, Belgium
Email: lsc@info.fundp.ac.be

***Abstract*—This paper[1] presents the first steps of our work related to overload control techniques. These techniques are currently under development in the SIP Overload Control (SOC) Internet Engineering Task Force (IETF) Working Group (WG). A preliminary step to this work is to build a realistic test environment.**

I. Introduction

Our work focuses on overload control in SIP networks. Overload occurs if a SIP server does not have sufficient resources to process all incoming messages. These resources may include CPU, memory, bandwidth, or disk space. Despite the fact that these situations are relatively common, and can have important economic impacts on service providers, the inter-domain congestion problem is not well handled yet.

To face this problematic, the IETF created in 2009 a Working Group to work on the overload problem: the SIP Overload Control WG [1]. The objective of the SOC WG is to develop functional solutions for SIP overload control. These overload situations can have various causes:

- Holy days, when calls are more numerous than usual, with calls coming from the entire network, without specific destination;
- Natural disasters, with a lot of calls to a specific area, and the issue of emergency calls' priority. In this case, for each message arriving at a server, it must be decided whether the message must absolutely be processed (even if another call has to be stopped to free resources) or if it can safely be dropped.
- TV game shows with participation of the audience, and some fairness issues when people must all have a fair chance to be selected.
- An electrical failure or any other problem that causes a server to reboot. Once power is restored, SIP agents will re-register at the same time with their server. All of a sudden, the latter will be flooded with messages and would not be able to process them. In the case of an Instant Messaging server, if all buddies try to reconnect after a failure, the same problem can occur and cause the server to collapse.

Our goal is to develop and test overload control techniques. To allow us to realize realistic tests, a virtual SIP test environment has been built. It must include SIP callers and callees able to simulate various types of traffic, SIP proxies/registrars with real routing capabilities and a tool allowing reproducible test, result analysis and generation of complete reports.

The environment can be used to validate overload control techniques specified by the SOC WG. Currently, WG participants seem to agree about a technique based on the use of new parameters for the *Via* header of SIP messages ('oc', 'oc-seq' and 'oc-validity' parameters). The 'oc' parameter "*when inserted by a SIP entity in a request going downstream, indicates that the SIP entity supports overload control. When the downstream SIP server sends a response, the downstream SIP server will add a value to the parameter that indicates a loss rate (in percentage) by which the requests arriving at the downstream SIP server should be reduced.*" [2] To validate this technique, a SIP environment including several SIP proxies is mandatory. Our environment meets this requirement, and behavior of SIP servers can be easily defined, modified and analyzed thanks to the use of appropriate softwares (more details below). Moreover, this environment can also be used for mathematical studies of congestion control techniques chosen within the body of knowledge in this matter, for example related to the Internet Research Task Force (IRTF) work described in [3], or the European Telecommunications Standards Institute (ETSI) Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) Next Generation Network (NGN) Specifications [4].

II. Building of the test environment

The creation, configuration and deployment of a test environment is usually very time-consuming. To achieve tests as close as possible to reality, a set of virtual machines generating SIP traffic were built. These virtual machines use free software in most cases, both for the simulation of traffic and the SIP servers deployment.

The server is a double quad-core, Intel Xeon @2.13GHz, with 12GB of RAM. The main host and virtual machines run Linux distributions, respectively Debian 5.0.5 (Debian testing, kernel 2.6.32-3) and CentOs 5.4 (kernel 2.6.18-194). Virtual environment is managed by qemu-kvm 0.12.5. The advantage of using virtual machines is obvious for several reasons. The cost (a single 3 kEUR server hosts more than 15 virtual machines) and ease of configuration are the main

[1]This paper was student-peer reviewed for CCNC 2011

incentives. A virtual network has been deployed to represent a basic real network. This network is made of four DNS domains, including DNS SRV and A records pointing to each domain's proxy. This configuration allows hosts to do some "real" routing activities.

An important aspect was the deployment of different platforms, in order to investigate potential compatibility issues. For the SIP part, these platforms are Sailfin on the one hand, and a commercial IMS platform on the other hand. The common feature of these two platforms is to relieve the developer from low-level aspects of programming, such as receiving or sending messages, dealing with potential retransmissions, management of error, etc. Hence, the developer can focus on the logic of his/her application.

A. Modification of SIPp Ims Bench

The implementation of a software useful for our work was another work item. The base used is the free software SIPp [5]. This software is a SIP traffic generator, relatively flexible and efficient. Unfortunately, it is relatively limited when one must generate "realistic" traffic. For example, the call rate is constant or increases linearly whereas the real traffic comes in bursts, and interactions between multiple instances of SIPp (one or more caller(s) and one or more callee(s)) are complicated or even impossible to implement.

To overcome these limitations, a modified version of SIPp has been developed by Intel: SIPp Ims Bench. These changes have been made to meet the requirements of the IMS/NGN Performance Benchmark document [6]. It defines the criteria for performance testing in the IMS[2] world. SIPp Ims Bench [7] therefore generates more realistic traffic patterns (with bursts, non-linear increase, etc.) and can execute a scenario with several callers. It also allows the generation of a complete report on performance of the different hosts used to generate traffic (CPU and memory use) and the performance of the system under test (CPU and memory use, call rate, error rate, response time, etc.). Plots are generated to give an overview of the results.

However, these changes do not yet fulfill our requirements. SIPp Ims Bench remains too limited in how the SIP traffic is routed between the callers and the callees. One of the requirements to achieve our tests is indeed having at least two routers between the ends of the network, to develop and test the sharing of the overload information between SIP servers, and the reaction of those servers. Another modification is related to the role of SIPp hosts. Currently, all SIPp Ims Bench hosts generate traffic. Modifications are currently done to allow the support of both roles, callers and callees. The network is depicted in Figure 1.

We have begun to look into the source code of SIPp Ims Bench to customize it. Some of the modifications have already been implemented, but the main changes are in progress. These modifications should allow us to generate traffic in a more flexible way, test solutions and produce detailed reports of the behavior of the servers (traffic rate accepted, rejected, errors, etc.). These reports will enable the evaluation of the congestion control mechanisms.

B. Building, configuration and deployment of SIP platform

As mentioned above, an important part of our work has been to design, configure and deploy a SIP virtual environment. This environment should be as close as possible of a real one, and use proper routing techniques. For the time being, the SIP generator is the free and open-source SIPp software, while the SIP platforms used to deploy proxies/registrars are Sailfin version 2[3] [8] and another IMS platform. When modifications to SIPp Ims Bench will be completed, it will be used instead of basic SIPp, bringing its advantages on the traffic generation.

Sailfin is based on the applications server Glassfish (version 2.1 is used) [9]. This open-source platform is designed to facilitate the deployment of Java applications. Applications can be easily launched, monitored, stopped, etc. via a control interface. As indicated above, it hides the low-level aspects to the developer and offers a congestion control mechanism which may be useful in testing.

The other platform is a commercial platform, much more complex than Sailfin, and using other technologies to manage the application life cycle (i.e. the OSGi framework [10]). Also, this platform has a web interface through which applications can be managed and offers a wide range of services to the developer. It manages a large number of protocols, and can be used to develop IMS applications using multiple protocols, e.g. SIP for Voice over IP, and HTTP for the Internet part, to build applications combining these two protocols. For example a chat window that opens if the callee is busy. This platform offers a local congestion control mechanism. Again, it may be useful to test its effectiveness at a local level, then combined with Sailfin congestion control.

Obviously, the goal is to define cooperative overload control mechanisms (for example between pairs of servers), but local mechanisms must be put in place as well, preferably to be used in last resort.

The current test environment consists of 9 machines acting as callers, 3 acting as callees. Added to these 12 hosts, 4 act as proxies using Sailfin and a small SIP proxy/registrar implementation. The environment is depicted in Figure 1.

Callers are on the left, callees on the right. Between them sit proxies. The target server (the system under test, which will be overloaded during the tests) is the one in the right frame. It receives traffic from callers via the three other servers, and routes it towards callees. Proxies on the left, in case of congestion of the tested server, are expected to reduce their outgoing traffic upon request.

The whole environment has been built to simulate a real network, but there are obviously some differences. The network (the data transport) is here reliable and instantaneous. This may affect the results, although delay and packets losses can be emulated, for instance with Network Emulator (netem) [11]. Another limitation is the absence of redundant servers. It is indeed common to configure two servers identically, so

[2]The IP Multimedia Subsystem is an architectural framework for delivering Internet Protocol (IP) multimedia services.

[3]Sailfin is the SipServlet implementation, initially developed by Ericsson.

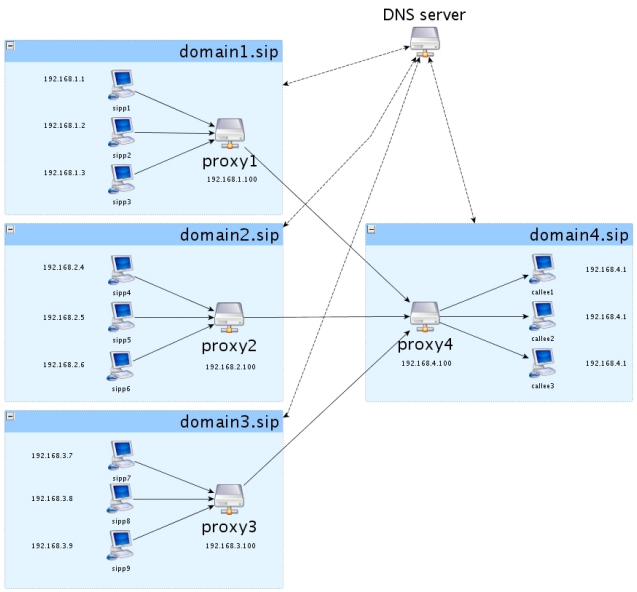

Fig. 1. Test environment

one can take over in the case of the failure of the first one. This kind of architecture is complex to implement (because of mechanisms of data replication, etc.) and has not been set up so far. Additionally, some problems were encountered when building up this testbed, probably due to the use of virtual machines. They were analyzed and quickly resolved.

The final aim is to generate enough traffic to overload the target proxy, and test solutions for preventing, managing or facing these situations in the best possible conditions.

III. Setting up and execution of preliminary tests

A series of preliminary tests have been implemented. They simply aimed at validating the configuration of the environment and the components. Indeed, as explained above, performing "realistic" tests requires the modification of SIPp Ims Bench, which is a work in progress. These tests have been performed with and without Sailfin's congestion mechanism to measure its effectiveness. Obtaining relevant results was relatively difficult, due to the limitations described above, e.g. the rate of outgoing calls increased linearly, and it was impossible to know precisely when the errors occur.

Nevertheless, some tests have been executed with the configuration described above (9 callers, 3 callees and 4 proxies) but did not produce really exploitable results.

Sailfin provides a limited Overload Protection Service. When enabled, the service monitors CPU and/or memory occupation, and sends 503 messages (Service Unavailable) back to the caller when a preconfigured threshold is reached. Unfortunately, the reaction of SIPp when receiving 503 error messages is not well defined. It is possible than the call rate is not really decreased. More precisely, SIPp does not wait several seconds to send more messages to the server, accordingly to RFC 3261 [12], but marks the call with the 503 as failed. The potential influence of the `Retry-After` header is not clear neither.

These tests showed that without the mechanism of Sailfin activated, errors occurs when the call rate is approximately 30 call attempts per second (CAPS) at the target proxy. With the congestion control enabled, the call rate increases to approximately 35 CAPS, thus a 15% increase. In extreme cases, when the traffic exceeds the rate Sailfin is able to process, CPU use increases to 100% and no more messages are sent. The platform is completely overloaded and tends to collapse. Both tests were made to serve as reference.

Once the modifications of SIPp Ims Bench completed, tests could be re-executed with the benefits of the software for the analysis and interpretation of the results.

IV. Conclusion and future work

In this paper, we have presented the very first steps of our work related to the SIP overload control techniques. The development and test of such techniques require the building of a complete SIP test environment, including benchmark tools as well as SIP hosts (traffic generator and proxies/registrars).

In the future, the work will be mainly focused on the modifications of SIPp Ims Bench. This step is a requirement to make tests and development of overload control techniques possible. The future modifications will be about roles of tests system, with the addition of callers and callees. Modifications related to the reaction when receiving 503 error messages are also planned.

References

[1] SIP Overload Control Working Group Charter. Last consulted on August 24, 2010. [Online]. Available: http://tools.ietf.org/wg/soc/charters

[2] V. Gurbani, V. Hilt, and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control," draft-gurbani-soc-overload-control-02, Aug. 2010, work in progress.

[3] D. Papadimitriou, M. Welzl, and B. Briscoe, "Open Research Issues in Internet Congestion Control," draft-irtf-iccrg-welzl-congestion-control-open-research-08, Sep. 2010, work in progress.

[4] European Telecommunications Standards Institute (ETSI) Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN). Last consulted on October 01, 2010. [Online]. Available: http://www.etsi.org/tispan

[5] SIPp. Last consulted on August 24, 2010. [Online]. Available: http://sipp.sourceforge.net/

[6] "Telecommunications and internet converged services and protocols for advanced networking (tispan); ims/ngn performance benchmark," ETSI TS 186 008, European Telecommunications Standards Institute, Oct. 2007.

[7] SIPp Ims Bench. Last consulted on August 24, 2010. [Online]. Available: http://sipp.sourceforge.net/ims_bench/index.html

[8] Sailfin. Last consulted on August 24, 2010. [Online]. Available: https://sailfin.dev.java.net/

[9] Glassfish Application Server. Last consulted on August 24, 2010. [Online]. Available: http://sipp.sourceforge.net/ims_bench/index.html

[10] The OSGi Alliance. Last consulted on August 24, 2010. [Online]. Available: http://www.osgi.org/Main/HomePage

[11] The Linux Foundation: Network Emulator (NETEM). Last consulted on August 24, 2010. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 2481, Jun. 2002.