A Proposal for A High Availability Architecture for VoIP Telephone Systems based on Open Source Software

Alejandro Martin[1]
School of Computing
Central University of Venezuela
Caracas, Venezuela

Eric Gamess[2]
MCIS Department
Jacksonville State University
Jacksonville, AL, USA

Dedaniel Urribarri[3], Jesús Gómez[4]
School of Computing
Central University of Venezuela
Caracas, Venezuela

***Abstract*—The inherent needs of organizations to improve and amplify their technological platform entail large expenses with the goal to enhance their performance. Hence, they have to contemplate mechanisms of optimization and the improvement of their operational infrastructure. In this direction arises the need to guarantee the correct operation and non-degradation of the services provided by the platform during the periods with a significant load of work. This type of scenario is perfectly applicable to the field of VoIP technologies, where users generate elevated loads of work on critical points of the infrastructure, during the process of interaction with their peers. In this research work, we propose a solution for high availability, with the goal of maintaining the continuity of the operation of communication environments based on the SIP protocol in high load. We validate our proposal through numerous experiments. Also, we compare our solution with other classical VoIP scenarios and show the advantages of a high availability and fault tolerance architecture for organizations.**

***Keywords*—Cluster; high availability; load balancer; VoIP; SIP; kamailio; corosync; asterisk; SIPp**

I. Introduction

The rise of new technologies in the field of communications through the usage of computer networks has driven the growth of organizations. Motivated by their interest in an efficient application of resources, these organizations invest on innovative communication mechanisms, to establish connections between devices for the exchange of information, allowing the communication of well-identified entities. Usually, these communication technologies follow well-regulated operating schemes, which clearly define intermediate points, states, processes, and possible behaviors of communication, also known as communication protocols.

Extrapolating these concepts, organizations interested in optimally diversifying their communication capacities are inclined to use these state-of-the-art technologies and protocols of communication. However, there are cases in which the magnitude of some organizations requires network infrastructures that prioritize the performance and stability of the communication platform, and it is at this point that the following topics of interest arise:

1) Optimization of the network infrastructure at the level of performance

2) Scalability and longevity of the network infrastructure

3) Adequate performance of the communication system in stressful conditions

4) Backup plan to tackle faults and malfunction events.

In this work, we propose a Voice over IP (VoIP) solution based on the Session Initiation Protocol [1][2][3] (SIP), and the usage of tools that allow the establishment of a network infrastructure of high availability and good performance, such as Kamailio [4], Corosync [5], Asterisk [6][7], among others. To validate our work, we test our suggested architecture under high stress and report the obtained results.

The following document is organized as follows. Section II presents the communication problems faced by large organizations when it comes to scalability and fault tolerance. Section III introduces some general solutions for high availability. The related works are discussed in Section IV. In Section V, we propose our high availability architecture for VoIP based on open source software. The test scenarios and definitions are presented in Section VI and Section VII, respectively. In Section VIII, we analyze the results of our tests to validate the proposed architecture. Finally, Section IX concludes the paper and gives some directions for future work.

II. Organizational Problems

From a technological perspective, there are some limitations in existing IP telephony architectures aimed at the provision of a high-quality telephony service, particularly in high-volume conditions where a significant workload must be supported, such as organizational environments, in which users rely on services in a sustained and repetitive manner. These limitations can lead to a point in which an organization, for the sake of the improvement of the quality of service and non-interruption of the business, opts for proprietary solutions that in many cases represent a significant monetary investment. The main problems faced by organizations that use open-source software solutions for telephony are those that prevent the continuity of the service in environments of high concurrency and high demand.

Many of these problems are due to the fact that some organizations ignore the expected requirement parameters in the environments in which the service is provided within their networks. Therefore, these organizations do not evaluate certain topological and architectural solutions with open-source software tools to face their limitations, which could be solved by offering proactive mechanisms of high availability and contingency, in case of malfunction of their systems. In addition, if the possible growth in the requirements of the services managed by the organization is anticipated, a good approach in the management and configuration of the services to be offered within the architecture, plus load balancing solutions, would allow greater scalability in the service.

The concept of high availability and continuity of the operation in computer networks is a solution for environments of high workload. However, the high cost associated with implementing proprietary solutions to improve the quality of the communication service makes them unfeasible for some organizations. Therefore, our interest in proposing an architecture of high availability through the usage of open-source software tools. Our solution considers that the particular limitations of each piece of software can be overcome by the coupling of multiple tools synchronized with each other to offer a standout service, that is, by using the famous saying “Unity is Strength.”

III. High Availability Solutions

In any type of communication solution, faults are highly probable. Therefore, contingency mechanisms are required to resolve these faults automatically, with the smallest possible response time, and when possible, in a transparent way for users. To achieve this goal, a high availability solution must have the following aspects:

- high availability mechanisms
- load balancing mechanisms
- division of services
- monitoring and management of services and load.

A. High Availability Mechanisms

From the computational point of view, the term “availability” refers to the period of time a service is available. When seen as an expression, “availability” can be either the response time of the service or whether it is accessible to be consumed by the users [8]. Hence, “high availability” refers to “... a system that is continuously operational and available for the use of the services provided to end-users” [9]. Providing high availability to a system or service is not an easy task, since not every service manages mechanisms to support it. Therefore, offering high availability in a system requires a specific design and implementation, mitigating all the points of failure that a system may have.

Redundancy is important in a system that offers high availability. Redundancy generally consists of backup components that automatically “kick in” if one component fails. It can be achieved at the level of the nodes that provide the service, i.e., the servers that are managing the service. The two most commonly used high availability cluster configurations are active-passive and active-active.

1) Active-Passive: In an active-passive scheme, one of the nodes is processing the requests received by the critical service (thus considered as the active node), while the other node is monitoring the active node and ready to take over as soon as the active node gets disconnected or is unable to serve [10].

2) Active-Active: In an active-active scheme, both nodes are actively providing critical services simultaneously. One of the main purposes of an active-active cluster is to achieve load balancing, that is, distribute the workload. In case of a failure of either node, the other node will be responsible for the provision of all the critical services [10].

B. Load Balancing Mechanisms

Let consider two servers or nodes with similar characteristics that are setup as telephone exchanges, one active and one passive. The basic idea is that the passive server will take over when the active server faces a failure. Now, if the active server were to fail due to very high service requirements (numerous controls for calls managed simultaneously: user registration, start of calls, end of calls, etc), it is very likely that the same will happen to the other server after the failover, since they are servers with similar characteristics. To solve this kind of problems, it is advisable to distribute the load among multiple servers (load balancing).

In the concept of load balancing, multiple servers are offering the same service and do the same tasks. Usually, an appliance (specialized physical device) or software solution is in charge of the load balancing management. It receives all the requests that are directed to the servers and is in charge of redirecting the traffic to one of the servers that it considers appropriate, through different techniques and mechanisms to balance the workload.

Load balancers can be divided into stateless and stateful load balancers:

- **Stateless Load Balancer:** Stateless load balancers typically use hashing algorithms to transform part of the data taken from the requests or packets into a low hash value, for the selection of a server from the farm [11]. This method can permit a certain level of persistence; For example, if the data to be hashed is the source IP address of the client, all requests made by a single client will be sent to a specific server. There are multiple hashing methods: hash based on IP source, hash buckets, etc.
- **Stateful Load Balancer:** Stateless load balancers have certain limitations in a fair distribution of workloads. Stateful load balancers maintain the state of the sessions. To accomplish this, the balancer must be able to determine when a session starts and when it ends. When a session is started, the target server is determined using load distribution mechanisms. Once determined, all subsequent packets that belong to the same session will be sent to the same destination server, until the end of the session [11].

There are many methods for load distribution, such as:

- **Round-Robin:** The Round-Robin method is one of the most popular and simple methods of load distribution.

Its distribution is based on granting the incoming requests to a different server, until all servers have received one. After reaching this point, the process is repeated with the subsequent requests. Some of the advantages of Round-Robin are its simplicity and the few resources needed by the balancer. Also, the method is very fast, even in large architectures with numerous servers in the farm. However, it has the disadvantage of providing a rather poor mechanism of load distribution, because it has no mechanisms to evaluate the load of each element of the farm.

- **Choice by Load:** In this load balancing technique, each server of the farm is monitored by a program known as a server-side agent. The server-side agent provides real-time information about the load of the server where it is running, such as: CPU load, memory usage, disk I/O, etc. Hence, the load balancer obtains information about the load conditions of each server in the farm with a very high level of detail [11], before it determines which server is the most suitable to take the next request.

Many parameters can be used to assess the load of the servers in the farm. The parameters to choose will depend on the application. Some of the most common load parameters are: CPU utilization, RAM utilization, quantity of connections, and quantity of transactions or active calls (very common in solutions based on SIP). In many cases, parameters are combined for a better load evaluation.

C. Division of Services

When there are multiple tools with the ability to offer the same functionality, it is important to consider which one can provide it in the best way. Throughout this paper, the advantages of the division of functionalities of certain services will be described, with the corresponding evidence to support the decisions.

D. Monitoring and Management of Services and Load

The establishment of monitoring and management service mechanisms within a high availability solution allows the identification of failure points, as well as the efficient determination and management of the components involved in its operation. The monitoring mechanisms permit to have an instant state of the situation, which facilitates the decision to be taken on the exposed services.

IV. Related Works

Some studies and solutions related to providing mechanisms of contingency and scalability in IP telephony systems currently exist. Throughout this section, some related researches will be described and analyzed, highlighting their advantages and weaknesses.

A. High Availability for SIP: Solutions and Real-Time Measurement Performance Evaluation

This research proposes the creation of a transparent and practical failover solution for proxy servers, for both SIP and RTP [12]. The objective is to increase the availability, stability, and scalability of multimedia systems based on SIP by using active-passive failover mechanisms with floating IP addresses, for both SIP and RTP.

1) SIP redundancy architecture: In order to have redundancy through SIP, it is essential to have two or more SIP servers. Also, each SIP server must be aware of all the SIP transaction made [12]. This is achieved by replicating all messages received by the active server to all the backup servers, even if these last ones are not available to the public. To accomplish this, the authors proposed the creation of a daemon called High Availability Daemon (HAD), which is run in each SIP server, including a SIP proxy.

2) RTP redundancy architecture: The approach of the RTP architecture is very similar to the one for SIP. However, the RTP proxy allocates two port numbers for each forwarding relation. Data which are received on one port are forwarded to the other peer through the other port number. This prior knowledge by the RTP proxy is done through a mapping list that is shared by HAD.

To begin the RTP message exchange of a SIP session, the RTP proxy opens the ports and virtually interconnects with the peer. At this point, it does not know what are the IP addresses and ports of the peer and learns them when receiving the first RTP message of the peer (the mapping list is updated) [12]. This mapping list must be exchanged with the other RTP proxy. HAD is in charge for this, thus storing the mapping list and replicating it with the other HAD of the other RTP server. HAD receives mapping requests from the SIP proxy and forwards them to the local RTP proxy. Additionally, HAD intercepts the local RTP proxy responses and forwards the chosen port to the backup system. Figure 1 and Figure 2 [12] refer to the operation mechanism of the HAD daemon, for SIP and RTP, respectively.

3) Load Balancing scheme: In this work, a DNS load balancing mechanism based on SRV registers was used. The load balancing module is responsible for requesting in static time interval the number of SIP servers available to the DNS server (DNS records) and resolves its Fully Qualified Domain Name (FQDN). This means that for each SIP domain, the DNS server has multiple DNS records for the DNS proxies added by it.

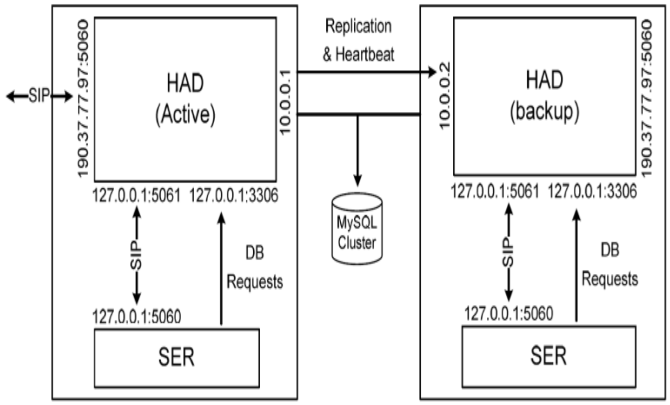

Fig. 1. SIP Replication Architecture.

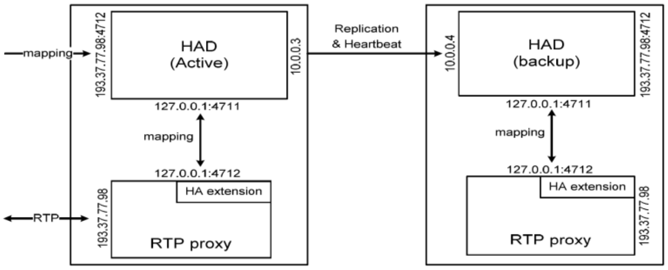

Fig. 2. RTP Relay Replication.

The balancer is responsible for querying the DNS and maintaining the DNS records. The SIP client, when making a SIP request, will first reach the balancer and, based on the information provided by the DNS and the weight of the different services, forward the request to the SIP server with more priority. This, however, is only required for certain SIP messages. Once a SIP client has established the connection (sending an INVITE and receiving an OK from the SIP server), it is not required to pass through the balancer again.

4) Differences with our proposed architecture: In contrast to our work, the authors wrote their own HAD daemon from scratch, and did not use a well-known and debugged software developed by the community. This solution is error-prone, since their software has only been exposed to a very small group of people. Finally, another clear difference is the use of a DNS-oriented balancer, which is stateless oriented, resulting in unfair balancing.

B. On The Reliability of Voice over IP (VoIP) Telephony

The architecture proposed by Pal, Gadde, and Latchman [13] consists of:

- Two or more virtual servers with Kamailio.
- Two or more virtual servers with FreeSWITCH [14][15].

Calls are routed from the clients to the Kamailio servers using a distributed DNS-based ENUM (E.164 Number to URI Mapping) system with priority settings [13]. If at any moment a particular Kamailio virtual server is down or running at its full capacity, the calls will be rerouted to another Kamailio server. If none of these servers is active, the call will be rerouted directly to any of the FreeSWITCH that are responsible for the voice messaging, music on hold, and automated attendant (see Figure 3 [13]).

To provide hardware redundancy, the authors propose Ultra Monkey [16], that uses Linux Virtual Server (LVS), for the creation of high availability in network services. Ultra Monkey is a framework that uses the Heartbeat protocol to monitor if two servers are operating properly or not (presence of periodic messages). In this proposed architecture, there are two balancers configured in an active-passive clustering scheme with Ultra Monkey. At any moment, if the active balancer stops responding requests, the balancer that was in passive mode will pass to active mode and will process the requests. Heartbeat uses a plugin called IPFail that helps determine, at layer 3 (via ICMP messages), whether the balancers are working properly or not.

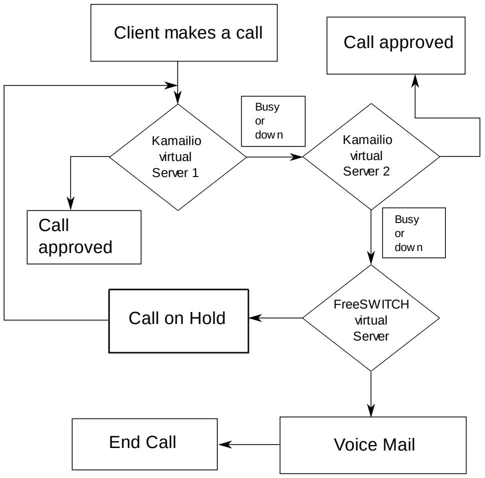

Fig. 3. Operating Scheme.

1) Differences with our proposed architecture: The use of Heartbeat as a communication and membership protocol differentiates this work from our proposed architecture. It is worth to remember that many developments in this field are migrating from Heartbeat to Pacemaker/Corosync, that is, Heartbeat can be considered as a project that is losing popularity. Also, the selected balancer, Ultra Monkey, does not maintain the state of SIP, resulting in a lower quality balancer than Kamailio.

C. Design and Implementation of a System to Interconnect VoIP Services and CERN's Telephony Networks

The scope of this work [17] is broad and is intended to provide an interface to the telephone networks of the European Organization for Nuclear Research (CERN), specifically services based on SIP. The implemented system serves as an entry point for calls originated outside the CERN's telephony network, allowing users using this service to communicate with the CERN telephone network (landline or mobile).

1) Logic topology and components: It consists of multiple components based primarily on common existing SIP servers. Among them are Media Servers and Proxy Servers, interconnected as can be seen in Figure 4 [17]. The Media Servers provide functionalities to SIP users. They tend to use many hardware resources (since they process multimedia information by software). Proxy Servers are used to protect them from the signaling part [17]. For high availability requirements, the topology consists of two Proxy Servers and two Media Servers (active-passive and active-active, respectively), thus preventing single fault points in the topology.

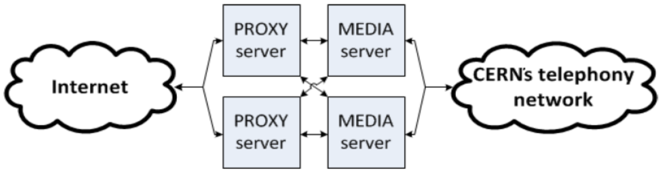

Fig. 4. Logical Topology.

The system has load balancing mechanisms served by Kamailio, including health checks or keepalive mechanisms to the different FreeSWITCH with failover capability, as well as active-active clustering techniques through Pacemaker and Corosync among Media Servers. Also, it makes use of active-passive clustering techniques between Proxy Servers.

Each Media Server is configured to manage a limited amount of channels while interrupting the others if the limit is already set. The load balancing mechanism allows the distribution of calls between the different Media Servers, thus increasing the scalability and the call limit [17].

2) Functions of Immersed components:

a) Proxy Server:

The Proxy Server must accept connections from both TCP and UDP. It must only accept calls destined for the CERN PBX; i.e., it should not act as Proxy Server among SIP users [17]. The call setup process keeps track of the progress of the call once answered. This is achieved by storing the dialog information using the Dialog module provided by Kamailio in a database [17].

b) Media Server:

The Media Server must reserve one channel for each incoming call and then proceed to authenticate the user of the incoming call. Users are not authenticated by using “Mod_directory” module, making it a Registrar Server, but by configuring an access list that only accepts incoming calls originated by the Proxy Server and, in addition, a custom mechanism that will be explained in the following paragraph. The SIP profiles were completely eliminated except for the internal profile in which incoming calls from Kamailio were assigned to this profile.

In addition, a method is used to perform an extra layer of users’ authentication. This functionality varies depending on how the call flow is performed. This mechanism is created by scripts written in Lua where a PIN is offered to the user who is calling through an IVR. The user must set the PIN correctly, using DTMF tones, for the authentication to be complete.

3) Differences with our Proposed Architecture: This architecture was designed with the purpose of allowing external entities to communicate through VoIP (using the SIP protocol for session establishment) with the CERN private segments, which are legacy telephone networks. In our proposed solution, the architecture is totally based on the SIP protocol.

V. Proposed Architecture and Solution

Figure 5 depicts the conceptual architecture of the proposed solution. The subsystem that supports high availability telephony is in network A, on the left side (inside the green box), which the following elements:

- **Proxy/Registrar:** This is the main component in charge of the services provided by the SIP signaling protocol. It acts as a stateful SIP proxy to allow load balancing mechanisms towards the PBX entities. In addition, it functions as an RTP proxy to solve problems related to NAT Traversal.
- **PBX:** These components act as multimedia servers. They are used as secondary SIP traffic receivers (in case of call establishment requirements) and these servers are being balanced by the Proxy/Registrar servers.
- **DB/SAN:** This component acts as a shared storage for the two components defined above and maintains the information centralized.

Each instance of the different components, in Figure 5, indicates whether it is active (working perfectly and providing services), passive (working perfectly but not providing services), or failed (out of order). This information refers to the implementation of the clustering schemes that are used as contingency mechanisms within the architecture, consisting of multiple instances of the same component.

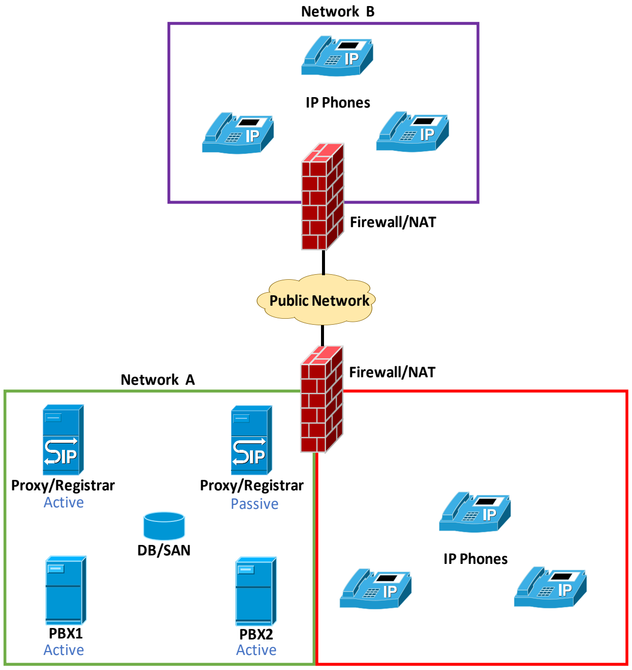

Fig. 5. General Architecture of the Proposed Solution.

The rest of the components that are part of the solution are the following:

- **Firewall/NAT:** These components act as delimiters between the internal networks and the public networks. Their main purpose is to achieve the simulation of the NAT Traversal phenomenon by having SIP users outside of network A, where the telephony architecture is implemented.
- **IP Phones:** These components act as the endpoints that require communication services using the implemented telephony architecture.

A. Open Source Components Used in our Architecture

As mentioned previously, our solution is based on open source software such as:

1) Kamailio: Component previously mentioned as Proxy/ Registrar provides:

- **Kamailio v4.3:** Tool that turns this component into a stateful SIP proxy, offering SIP transaction failover mechanisms and load balancing mechanisms. It also acts as a Registrar Server and a Location Server.
- **RTPProxy v2.0.0:** Tool used to convert the component into an RTP proxy if necessary (against NAT Traversal phenomenon).

2) Asterisk: The component mentioned as PBX is running Asterisk v13.1. It acts as a multimedia server with the ability to offer multiple functionalities such as: transcoding, voicemail service, IVR, call transfer, etc.

3) MySQL/SAN: Component mentioned as DB/SAN, that acts as a shared storage for the Proxy/Registrar and PBX servers. Since some information is stored in the database and other at the file system level, there are two tools to support these services:

- **MySQL Community Edition v5.7:** MySQL is used as a database management system to maintain shared information between the different instances of the different components that are part of the architecture.
- **Linux-IO Target (targetcli v2.1):** This tool is used to convert the SAN server into a server from which an iSCSI Target is configured to allow to different instances of the Asterisk servers to create iSCSI sessions against that Target and be able to manage this storage in a shared way. This is necessary because certain Asterisk information (some configuration files, multimedia files such as voicemail, music on hold, etc.) must be shared by both instances for the proper operation of the architecture.

B. Design of the Clustering Schemes

In this architecture, there are two clustering schemes of two nodes each, one for the different instances of the components that run Kamailio and RTPProxy, and another for the different instances of the components that execute Asterisk. As explained earlier, the reason behind these schemes is to provide contingency mechanisms in case of errors, manage shared storage, and monitor the status of the various relevant services that are part of the different schemes. These clustering schemes are formed by Pacemaker v1.1.13 [18] for the resources management and Corosync v2.3.4 [5] for the communication layer and the creation of the cluster infrastructure.

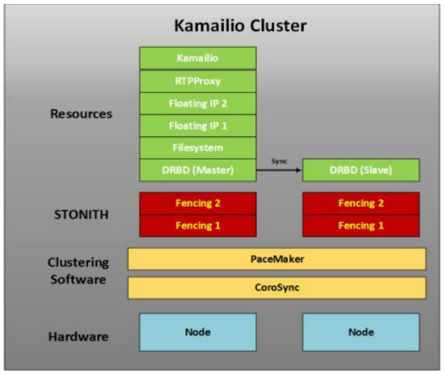

Fig. 6. Logical Organization for the Proxy/Registrar Clustering Scheme.

1) Clustering Scheme for the Proxy/Registrar Components: This is an active-passive clustering scheme, which means that one of the instances is the one that will be in charge of providing all the services, while the other instance will be monitoring and taking over when some error or problem occurs in the active instance. Figure 6 shows the logical organization of the components within the Proxy/Registrar cluster.

2) Clustering Scheme for the PBX Components: This is an active-active clustering scheme, which means that both instances are responsible for having all services active and for providing them. This scheme has the primary function of preventing data corruption of shared storage between both instances of the PBX. That is, the monitoring system is made such as that when a PBX fails while providing a service, the other PBX must terminate this service, to avoid inconsistencies in the data and a malfunction of the architecture. Figure 7 depicts the logical organization of the components within the PBX cluster.

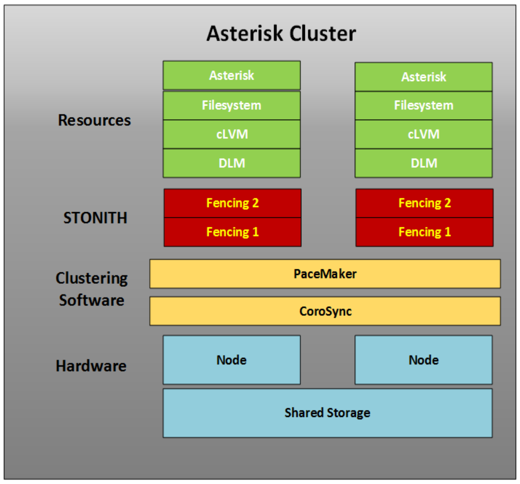

Fig. 7. Logical Organization for the PBX Clustering Scheme.

VI. TEST SCENARIOS

In order to validate our architecture, we developed two kinds of test scenarios: (1) scenarios to assess the contingency mechanisms and (2) scenarios to study the behavior of the proposed architecture under strong stress. It is worth mentioning that a PBX server has two possible states: (1) **active** when it is working properly and providing services and (2) **failed/passive** when it is out of order. A Proxy/Registrar component has three possible states: (1) **active** when it is working properly and providing services, (2) **passive** when it is working properly and not providing services, and (3) **failed** when it is out of order.

A. Scenarios to Assess the Contingency Mechanisms

1) Scenario 1 - Recovery from the Failure of a PBX Component:

Figure 8 depicts the situation when one of the Asterisk servers fails. In this case, all the load of the services must pass to the other Asterisk server.

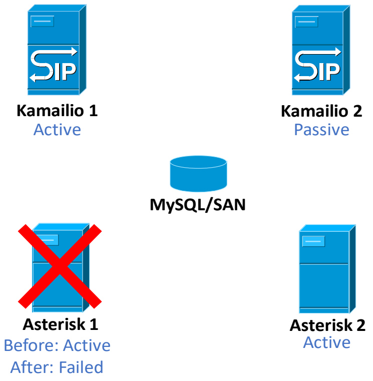

Fig. 8. Recovery from the Failure of a PBX Component.

2) Scenario 2 - Recovery from the Failure of a Proxy/ Registrar Component:

Figure 9 illustrates this scenario. Before the failure, Kamailio 1 was active and Kamailio 2 was passive. As soon as Kamailio 2 notices the failure of the other server, it switches from passive to active mode.

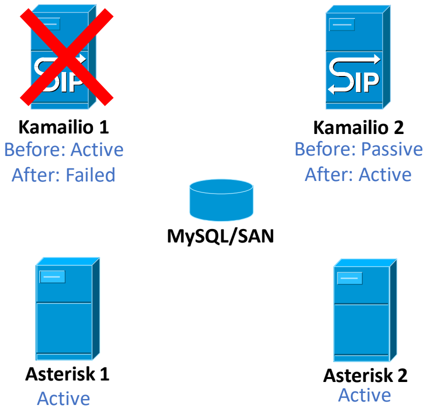

Fig. 9. Recovery from the Failure of a Proxy/Registrar Component.

3) Scenario 3 - Recovery from the Failure of a Proxy/Registrar Component and a PBX Component:

Figure 10 depicts this test scenario, where there are two failures: (1) failure of an instance that is providing the service of Kamailio and RTPProxy, and (2) failure of an instance that is offering the service of Asterisk.

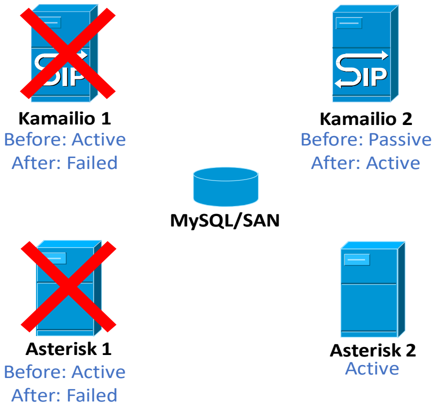

Fig. 10. Recovery from the Failure of a Proxy/Registrar Component and a PBX Component.

B. Scenarios to Assess the Architecture under Stress

To assess the behavior of our proposed architecture in scenarios of high concurrency of SIP services (INVITE and REGISTER transactions) and RTP traffic flows, the following scenarios were proposed:

1) Scenario 1 - System under Stress in a Conventional VoIP Implementation:

In this scenario, there is a single PBX component, as shown in Figure 11. This is a classical VoIP system, without redundancy. The idea is to study the scalability of a conventional VoIP implementation, by exposing it to a high volume of requests.

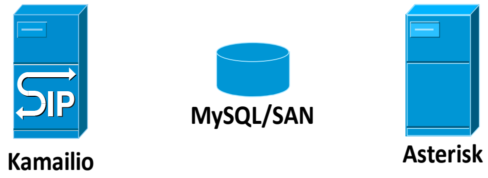

Fig. 11. Conventional VoIP Implementation.

2) Scenario 2 - System under Stress with the Proposed Architecture:

Figure 12 depicts our proposed architecture with redundancy at the level of the Proxy/Registrar servers, and load balancing at the level of the PBXs. The idea is to submit the proposed architecture under a high volume of requests, and compare its behavior with the one shown in Figure 11 (conventional VoIP implementation).

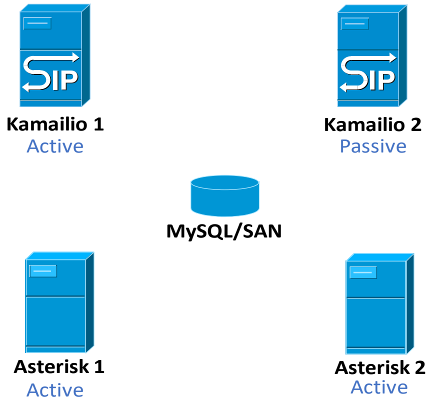

Fig. 12. Proposed Architecture when all the Components are Operating Properly.

VII. Definitions of Tests and Particular Situations

In this section, we define the performed assessments for the proposed scenarios.

A. Assessment of the Contingency Mechanisms

To assess the contingency mechanism, we defined three tests, that can be applied to any of the three scenarios, as specified below:

- **Test 1:** Send an INVITE transaction just after a failure and before the application of contingency mechanisms.
- **Test 2:** Send an INVITE transaction before a failure and the application of contingency mechanisms,

without the completion of the INVITE session (i.e., without multimedia flows).

- **Test 3:** Send an INVITE transaction before a failure and the application of contingency mechanisms, with the completion of the INVITE session (i.e., with multimedia flows).

Due to the ability of Kamailio to maintain the state of SIP transactions and make decisions based on the lack of response or error messages by the Asterisk nodes, we also defined the following two situations:

- **Situation 1:** Generation of an INVITE request that attempts to be transmitted to an asterisk node that has just failed, as explained in Test 1. If there is no response within 4 seconds, Kamailio will retransmit the INVITE request to the other active Asterisk node.
- **Situation 2:** It occurs after receiving the message “100 Trying”. If Kamailio does not receive any further message (“180 Ringing”) after 60 seconds (usually due to a crash of the Asterisk server), Kamailio will retransmit the INVITE transaction to the other active Asterisk node.

Tables I, II, and III represent the expected results (behavior and response time) for the different failover mechanisms.

TABLE I. EXPECTED RESULTS OF TRANSACTIONAL FAILOVER MECHANISMS

Scenario	Behavior	Response Time
1	**User Interaction:** The IP phone does not participate in the contingency **Contingency**: The active Kamailio retransmit the INVITE message after a timeout of the response	**Situation 1**: 4 seconds **Situation 2**: 60 seconds

TABLE II. EXPECTED RESULTS OF THE PROXY/REGISTRAR CONTINGENCY MECHANISMS

Scenario	Behavior	Response Time
2	**User Interaction:** The IP phone replicates the initial request in case of being in the establishment of a session **Contingency:** Service Reestablishment	**Fencing Application:** A few seconds **Services or Nodes Errors:** A few seconds
3	**User Interaction:** The IP phone replicates the initial request in case of being in the establishment of a session **Contingency:** Service Reestablishment	**Fencing Application:** A few seconds **Services or Nodes Errors:** A few seconds

TABLE III. EXPECTED RESULTS OF THE PBX CONTINGENCY MECHANISMS

Scenario	Behavior	Response Time
1	**User Interaction:** The IP phone replicates the initial request in case of being in the establishment of a session **Contingency 1:** In case of a first failure when monitoring the resources of an Asterisk node, Pacemaker restarts the service. **Contingency 2:** In any other cases, Pacemaker stops the service on the node where it fails.	**Fencing Application:** A few seconds **Service or Nodes Errors:** A few seconds
3	**User Interaction:** The IP phone replicates the initial request in case of being in the establishment of a session. **Contingency 1:** In case of a first failure when monitoring the resources of an Asterisk node, Pacemaker restarts the service. **Contingency 2:** In any other cases, Pacemaker stops the service on the node where it fails.	**Fencing Application:** A few seconds **Services or Nodes Errors:** A few seconds

B. Assessment of the Architecture under Stress

To assess the behavior of the proposed architecture when suffering stress, we defined the tests shown in Table IV and Table V, which were performed with SIPp [19], a free open source traffic generator for the SIP protocol. The idea was to flood the proposed architecture with SIP requests (INVITE and REGISTER) and study how well it can manage the stress and the load balancing among the Asterisk servers. Table IV is focused on INVITE messages, while the emphasis of Table V is on REGISTER messages. From the first to the last column, Table IV contains (1) the test that we are defining, (2) the scenario used for this test, (3) the desired calling rates (number of calls made by SIPp per second), (4) the total limit of active calls at any time, (5) the duration of each call, and (6) the duration of the test, respectively. The last four parameters are set through SIPp. It is worth to clarify that the total limit of active calls is the maximum number of calls that can be active at any time during the experiment. If SIPp reaches this limit, then it will not initiate another new call until one of the active calls is completed.

From the first to the last column, Table V contains (1) the test that we are defining, (2) the scenario used for this test, (3) the desired transaction rates (number of REGISTER messages sent by SIPp per second), (4) the total number of SIP users in the system, and (5) the duration of the test, respectively. The last three parameters are set through SIPp. Note that we did not define tests for Scenario 1, since the registration of users is done by the Proxy/Registrar servers (Kamailio), and just having one of them instead of an active-passive cluster will not change the observed behavior.

TABLE IV. PARAMETERS USED IN SIPP FOR INVITE MESSAGES

Test	Scenario	Calling Rate	Limit of Active Calls	Duration of Calls	Duration of Test
Test 1	1	175 cps	1000 calls	6 seconds	1 hour
Test 2		200 cps	1000 calls	5 seconds	1 hour
Test 3		200 cps	1200 calls	5 seconds	1 hour
Test 4		200 cps	1200 calls	6 seconds	1 hour
Test 1	2	300 cps	2000 calls	6 seconds	1 hour
Test 2		300 cps	2000 calls	7 seconds	1 hour
Test 3		350 cps	2000 calls	5 seconds	1 hour
Test 4		350 cps	2000 calls	6 seconds	1 hour

TABLE V. PARAMETERS USED IN SIPP FOR REGISTER MESSAGES

Test	Scenario	Transaction Rate	Number of SIP Users	Duration of Test
Test 1	2	500 tps	10000 users	1 hour
Test 2		750 tps	10000 users	1 hour
Test 3		1000 tps	10000 users	1 hour

VIII. Results And Analysis

To do the validation experiments, we used a server with Xen Project [20][21] for the management of virtualized environments. All elements that are part of the architecture, as well as the network topology of the architecture, were virtualized in this server.

Since the elements that are running in the proposed architecture have different requirements at the level of CPU resources, we did not use the default configuration provided by the Xen hypervisor with a single vCPU per device. We did multiple attempts of different hardware resource distributions, and selected the one shown in Table VI.

TABLE VI. Resource Assigned to Each Device of the Architecture Through the Xen Hypervisor

Machines	vCPUs Management	Weight Management (Schedule-Credit)
Domain-0	4 vCPUs, No affinity	256
DB/SAN	1 vCPU, No affinity	512
Asterisk 1	2 vCPUs, No affinity	1000
Asterisk 2	2 vCPUs, No affinity	1000
Kamailio 1	2 vCPUs, No affinity	1000
Kamailio 2	2 vCPUs, No affinity	1000

A. Tests for Contingency Mechanisms

For this case, the tests are divided according to the types of failover mechanisms, which are:

1) Tests for the Transactional Failover Mechanisms: Table VII represents the results obtained by our tests in Situation 1 and Situation 2, of Scenario 1. We repeated the experiments 200 times and the response time presented is an average. Our experiments showed an adequate response of the transactional failover mechanisms, with an average response time of 4.174 seconds in Situation 1 (as shown in Table I, it should be greater than 4 seconds) and 63.049 seconds in Situation 2 (as shown in Table I, it should be greater than 60 seconds).

TABLE VII. Results of the Transactional Failover Mechanisms Tests

Scenario	Test	Situation	User Interaction	Contingency	Response Time
1	1	1	None	Retransmission of the INVITE message after a timeout of the response	4.17 sec
	2	2	None	INVITE session retransmission after meeting response times	63.04 sec

2) Tests for the Failover Mechanisms at Services and Nodes Levels: Table VIII shows the results obtained by our tests in the case of failover mechanisms at the levels of services and nodes. It shows the average response time at the level of users and services (we repeated the experiments 200 times and the response time presented is an average). Our experiments showed that our architecture is working well with an adequate reaction of the failover mechanisms at the level of services and nodes, confirmed the expected results of Table II and Table III.

TABLE VIII. Results of the Failover Mechanism Tests At Services and Nodes Levels

Scenario	Test	User Interaction	Contingency	Fencing	Response Time
2	1	None	Services are migrated from one node that is part of the clustering scheme to the other.	No	User - 4.50 sec Service – 3.49 sec
				Yes	User - 7.51 sec Service - 7.38 sec
	2	None	Services are migrated from one node that is part of the clustering scheme to the other.	No	User - 4.21 sec Service - 3.44 sec
				Yes	User - 7.51 sec Service - 7.41 sec
	3	Replicate call establishment	Services are migrated from one node that is part of the clustering scheme to the other.	No	User – Interaction Required Service - 3.48 sec
				Yes	User - It may or may not affect the service Service - 7.39 sec
3	1	None	Services are migrated from one node that is part of the clustering scheme to the other.	No	User – 7.47 sec Service - 3.43 sec
				Yes	User - 11.49 sec Service - 7.34 sec
	2	Replicate call establishment	Services are migrated from one node that is part of the clustering scheme to the other.	No	User - Interaction Required Service - 3.38 sec
				Yes	User - Interaction Required Service - 7.45 sec
	3	Replicate call establishment	Services are migrated from one node that is part of the clustering scheme to the other.	No	User - Interaction Required Service - 3.44 sec
				Yes	User - Interaction Required Service - 7.39 sec

B. Tests of Stress

Unlike the previous tests, the results of the tests of this section are highly dependent on the hardware resources and the topology. It is worth to mention that the SIPp tool has tests for both the SIP and RTP protocols. In the case of the RTP tests, SIPp sends media (RTP) traffic through RTP echo and RTP/pcap replay. We faced some difficulties with these RTP tests. The problem is based on the fact that SIPp has a single threaded architecture, where events are repeated through a loop with the same thread, which causes the .pcap files to be managed by a single core on the SIPp machine. Thus, in our case, when reaching a maximum of 800 active calls approximately, the core reached 100% of its capacity, resulting in a malfunction of SIPp for the RTP tests. Hence, we did not use the RTP tests to assess our architecture.

The objective of these tests is to study the ability of the proposed architecture to manage a big number of calls or registrations through multiple Asterisk servers, by doing load balancing.

1) Tests of Stress Performed with INVITE Messages: The results are shown in Table IX (conventional VoIP implementation) and Table X (our proposed architecture). For each test, we run the experiment twice. From the first to the last column of Table IX and Table X, we have (1) the test performed, (2) the actual duration of the test, (3) the average call rate performed by SIPp in one second, (4) the total number of calls during the experiment, (5) the average number of actives calls in the system at any moment, (6) the total number of successful calls during the experiment, (7) the total number of failed calls during the experiment, and (8) the total number of retransmissions made by SIP during the experiment, respectively.

TABLE IX. RESULTS OF STRESS TESTS PERFORMED BY INVITE MESSAGES IN SCENARIO 1 (CONVENTIONAL VOIP IMPLEMENTATION)

Test	Duration	Avg Call Rate	Generated Calls	Avg Active Calls	Successful Calls	Failed Calls	Retransmissions
1	1 hour	142.42	512733	998.33	512730	3	0
		142.42	512731	998.37	512726	5	0
2	1 hour	166.00	597613	997.79	512730	3	0
		165.97	597530	997.79	597522	8	0
3	1 hour	197.70	307064	509.27	98073	207796	23530
		199.10	717763	1197.62	717658	105	104
4	1 hour	170.86	615960	1199.04	615859	101	0
		170.85	615953	1198.83	615830	123	2

As we can see from Table IX and Table X, the proposed architecture significantly improves the VoIP system. The number of successful calls processed by our solution is almost the double of the one handled by a conventional VoIP architecture.

TABLE X. RESULTS OF STRESS TESTS PERFORMED BY INVITE MESSAGES IN SCENARIO 2 (OUR PROPOSED ARCHITECTURE)

Test	Duration	Avg Call Rate	Generated Calls	Avg Active Calls	Successful Calls	Failed Calls	Retransmissions
1	1 hour	284.75	1025107	1997.58	1025091	16	0
		284.80	1025314	1997.66	1024246	1068	34
2	1 hour	249.34	897640	1997.66	897626	14	13
		248.77	897573	1997.75	897562	11	58
3	20 min	345.76	418292	344.20	53614	364574	29308
		346.87	420175	268.00	59533	360638	33832
4	1 hour	284.87	1025566	1998.33	1025001	565	0
		284.76	1025162	1998.79	1025143	19	4

In Figure 13, we have more details on the evolution of the experiment that is highlighted in Table X. The x-axis shows the time during the realization of the experiment (0, 600, 1200, 1800, 2400, 3000, 3600, and 3607 seconds). For each sample, there are four bars representing (1) the number of active calls in PBX1, (2) the number of active calls in PBX2, (3) the total number of processed calls by PBX1, and (4) the total number of processed calls by PBX2. This experiment confirmed the correct operation done by Kamailio 1 (the active Proxy/ Registrar server), when doing load balancing between the two PBXs.

2) Tests of Stress Performed with REGISTER Messages: The results are shown in Table XI. For each test, we run the experiment twice. From the first to the last column of Table XI, we have (1) the test performed, (2) the actual duration of the test, (3) the average number of transactions/registrations performed by SIPp in one second, (4) the total number of transactions during the experiment, (5) the total number of successful transactions during the experiment, (6) the total number of failed transactions during the experiment, and (7) the total number of retransmissions made by SIP during the experiment, respectively.

TABLE XI. RESULTS OF STRESS TESTS PERFORMED BY REGISTER MESSAGES IN SCENARIO 2

Test	Duration	Avg Transaction Rate	Generated Transactions	Successful Transactions	Failed Transactions	Retransmissions
1	1 hour	498.81	1795733	1795733	0	72855
		498.86	1795932	1795932	0	71952
2	1 hour	748.15	2693403	2693385	18	232129
		748.29	2693882	2693882	0	174833
3	1 hour	996.05	3585832	3585831	1	301450
		995.23	3582895	3582887	8	275637

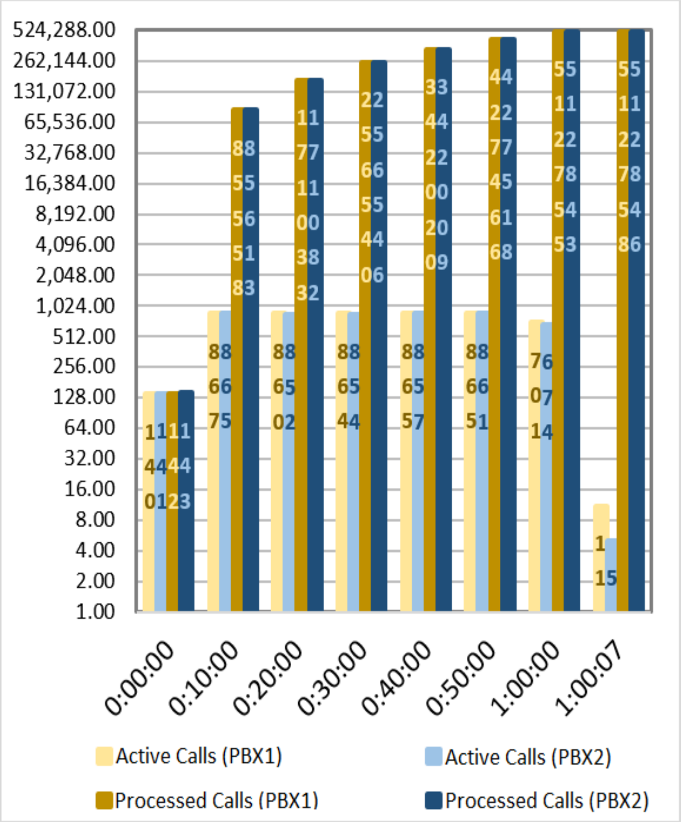

Fig. 13. Active and Processed Calls as the Time Passes - Confirmation of Load Balancing.

IX. CONCLUSIONS AND FUTURE WORK

VoIP is a technology that has achieved a great impact on the communication of digital networks. It takes advantage of existing networks and different standard protocols such as SIP, Inter-Asterisk eXchange [22] (IAX), RTP, and RTCP. On the one hand, many institutions have implemented VoIP solutions based on proprietary software, which tends to increase the implementation cost. On the other hand, other organizations have decided to go for open source solutions like Asterisk. Asterisk is focused on the inclusion and development of multiple services and the support of multiple signaling protocols. Hence, per see, it provides a high-quality service for telephony, but has weak aspects such as scalability and contingency mechanisms in case of failures.

In this work, we proposed a telephony architecture based on Asterisk and the SIP signaling protocol that covers those aspects not well supported by Asterisk per see, such as scalability and fault tolerance. Our solution is totally focused on open source software. In our architecture, we inserted different tools that allow high availability, and clustering concepts to offer contingency mechanisms, as well as load balancing technics to add greater scalability to the provided telephony service.

As future work, we are interested in developing other high availability VoIP telephony architectures based on the SIP protocol, using SIP servers such as OpenSIPS [23], in conjunction with clustering technology tools such as Pacemaker, Corosync, CMAN, etc. Another direction of research that we are planning to investigate is to construct cluster solutions based on different software, that is, instead of having several copies of the same server for redundancy, we are interested in using different implementations of software servers so that the system can be tolerant to bugs.

REFERENCES

[1] A. Johnston, SIP - Understanding the Session Initiation Protocol, Fourth Edition, Artech House, October 2015.

[2] R. Ranjan Roy, Handbook on Session Initiation Protocol: Networked Multimedia Communications for IP Telephony, First Edition, CRC Press, March 2016.

[3] N. Leonard, Understood SIP and IP Telephony in 3 Days: For the Beginner, Independently published, August 2017.

[4] Kamailio, https://www.kamailio.org.

[5] S. Dake, The Corosync High Performance Shared Memory IPC Reusable C Library, in Proceedings of the Linux Symposium, Montreal, Canada, July 2009.

[6] V. Stanislovaitis, How to Start a VoIP Business: A Six-Stage Guide to Becoming a VoIP Service Provider, February 2016.

[7] R. Bryant, Asterisk: The Definitive Guide: The Future of Telephony Is Now, O'Reilly, Fourth Edition, June 2013.

[8] P. Weygant, Cluster for High Availability: A Primer HP Solution, Second Edition, Prentice Hall, 2001.

[9] M. Resman, CentOS High Availability, Packt Publishing, April 2015.

[10] E. Marcus, Blueprints for High Availability, Second Edition, Wiley Publishing, September 2003.

[11] C. Kopparapu, Load Balancing, Servers, Firewalls and Caches, Wiley Computer Publishing, January 2002.

[12] G. Kambourakis, D. Geneiatakis, S. Gritzalis, C. Lambrinoudakis, T. Dagiuklas, S. Ehlert, and J. Fiedler, "High Availability for SIP: Solutions and Real-Time Measurement Performance Evaluation," International Journal of Disaster Recovery and Business Continuity, vol. 1, no. 1, pp. 11-29, February 2010.

[13] S. Pal, R. Gadde, and H. Latchman, On the Reliability of Voice Over IP (VoIP) Telephony, University of Florida, 2011.

[14] A. Minessale and G. Maruzzelli, FreeSWITCH 1.8 - VoIP and WebRTC with FreeSWITCH: The Definitive Source, Packt Publishing, July 2017.

[15] A. Minessale and G. Maruzzelli, Mastering FreeSWITCH, Packt Publishing, July 2016.

[16] Ultra Monkey, Load Balancing and Highly Available Solutions, http://www.ultramonkey.org.

[17] M. Pohančeník, "Design and Implementation of a System to Interconnect VoIP Services and CERN's Telephony Network," University of Žilina, Ginebra, 2013.

[18] Pacemaker, http://clusterlabs.org/pacemaker.

[19] R. Gayraud, O. Jacques, R. Day, C. P. Wright, et al., SIPp Reference Documentation, http://sipp.sourceforge.net/doc/reference.pdf.

[20] T. Mackey and J. K. Benedict, XenServer Administration Handbook: Practical Recipes for Successful Deployments, First Edition, O'Reilly Media, April 2016.

[21] M. Reed, Mastering Citrix XenServer, Packt Publishing, December 2014.

[22] M. Spencer, B. Capouch, E. Guy, F. Miller, and K. Shumard, IAX: Inter-Asterisk eXchange Version 2, RFC 5456, February 2010.

[23] F. Goncalves and B.-A. Iancu, Building Telephony Systems with OpenSIPS - Second Edition, Packt Publishing, January 2016.