DIPLOMA THESIS

Study programme:
Information Systems

Martin POHANČENÍK

Design and implementation of a system
to interconnect VoIP services
and CERN’s telephony networks
Supervisor: Rodrigo Sierra Moral
Tutor: Ing. Pavel Segeč PhD.
Registration number: 117/2012 April 2013

GENEVA, 2013
ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY DIPLOMOVÉJ PRÁCE.
Študijný program: Informačné systémy
Zameranie: Informačno-komunikačné siete

Meno a priezvisko | Osobné číslo
-----------------|----------------
Martin Pohančenik | 553515

Názov práce v slovenskom aj anglickom jazyku
Návrh a implementácia systému prepojenia VoIP služieb a telefónnej siete CERN-u
Design and implementation of a system to interconnect VoIP services and CERN’s telephony network

Zadanie úlohy, ciele, pokyny pre vypracovanie (Ak je mésto miesta, použite opačnú stranu)

Cieľ diplomovej práce:
Táto práca má byť zameraná na vytvorenie systému, ktorý bude umožňovať používateľom služby založené na SIP protokole a služby SKYPE telefonovať na pevné a mobilné číslo v telefónnej sieti CERN-u. The aim of this thesis is to create a system that will allow users of SIP-based services and SKYPE service to call to fixed and mobile numbers of CERN’s telephony network.

Obsah:
Dôraz sa kladie na:

- vyriešenie systému autorizácie volajúcich používateľov pri uskutočnení hovoru,
- bezpečnosť systému (dynamický firewall, ochrana proti útokom),
- vysoká dostupnosť systému (redundancia serverov, load balancing),
- udržateľnosť, monitoring a podpora.

Témy z predmetov študijného zamerania
Mená a pracovisko vedúceho DP: 
Mená a pracovisko tátora DP: 

24.10.2012

vedúci DP (dátum a podpis) 
tátór (dátum a podpis) 
vedúci katedry (dátum a podpis) 
garant (dátum a podpis)

Zadanie zaregistrované dňa 18. 10. 2012 pod číslom 117/2012 podpis
Emphasis on:

- means of authorizing for users trying to establish a call,
- security of the system (dynamic firewall, protection from attacks),
- high availability (redundancy of servers, load balancing),
- maintainability, monitoring and support.
Declaration

I hereby declare that this Diploma thesis is my own work, based on my knowledge and skills. This thesis was written under guidance of my supervisor, Mr. Rodrigo Sierra Moral and my tutor, Ing. Pavel Segeč PhD. I thank them for their advices and support throughout working on the project.

........................................

GENEVA, 19th April 2013                                                                 Martin Pohančeník
ABSTRACT


The purpose of this diploma thesis was to design and implement a system that would provide an interface of CERN’s telephony network for Voice over IP services (SIP-based services specifically). This system thus serves as an entry point for calls originating from outside of CERN’s telephony network and enables users of these services to call CERN’s fixed and mobile phone numbers. The theoretical part of the thesis talks in detail about the project specification and describes the goals that were trying to be achieved. It also describes the topic of VoIP telephony and current trends, alongside with analysis of the current telephone network of CERN. The practical part is targeted to explain the design of the solution and deployment of the system at CERN. The final part of the thesis shows testing scenarios and results and states several extension capabilities.

Key words: Voice over IP (VoIP). Session Initiation Protocol (SIP). CERN. SIP proxy. Media server. Open source.
# Table of contents

1. Introduction ........................................................................................................... 1

2. VoIP telephony ......................................................................................................... 2
   2.1 VoIP in corporate environment ........................................................................... 3
   2.2 Commonly used VoIP protocol - SIP ................................................................. 4
      2.2.1 Types of SIP servers .................................................................................... 5

3. The scope of the project ............................................................................................ 8
   3.1 Usage cases ......................................................................................................... 9
      3.1.1 Unauthorized calls ........................................................................................ 10
      3.1.2 Authorized calls ......................................................................................... 10
      3.1.3 Explicitly forbidden calls ............................................................................ 11
      3.1.4 Emergency calls ......................................................................................... 11
   3.2 Users .................................................................................................................. 11
      3.2.1 CERN users ................................................................................................ 11
      3.2.2 Non-CERN users ....................................................................................... 12

4. Project analysis ........................................................................................................ 12
   4.1 Logical topology ................................................................................................. 12
   4.2 Physical topology ............................................................................................... 13
   4.3 Desired call flow ................................................................................................ 14
   4.4 Elements of the system ..................................................................................... 16
      4.4.1 SIP proxy server ........................................................................................... 16
      4.4.2 Media server ................................................................................................ 17
      4.4.3 Database connectivity software .................................................................... 17
      4.4.4 Computer clustering software ..................................................................... 18
      4.4.5 Monitoring software .................................................................................... 19
      4.4.6 Administrator script .................................................................................... 19
   4.5 Software selection ............................................................................................... 20
      4.5.1 Proxy server .................................................................................................. 22
      4.5.2 Media server ................................................................................................ 25
      4.5.3 Other elements ............................................................................................. 26
   4.6 Summary of used technologies .......................................................................... 27

5. Configuration and deployment .................................................................................. 28
   5.1 DNS configuration .............................................................................................. 28
   5.2 Server preparation ............................................................................................... 28
   5.3 Database connectivity ......................................................................................... 30
      5.3.1 UnixODBC configuration ............................................................................. 30
      5.3.2 Database schema .......................................................................................... 32
   5.4 Kamailio installation ........................................................................................... 32
      5.4.1 Kamailio installation from GIT ................................................................. 33
      5.4.2 Kamailio installation from RPMs ............................................................... 34
   5.5 Kamailio configuration ...................................................................................... 34
      5.5.1 Core parameters configuration ................................................................. 36
      5.5.2 UnixODBC module configuration ............................................................. 37
      5.5.3 Load balancing configuration ...................................................................... 37
      5.5.4 Dialog module configuration ...................................................................... 40
      5.5.5 TLS module configuration ......................................................................... 41
      5.5.6 SNMP module configuration .................................................................... 41
      5.5.7 Permissions module configuration .......................................................... 42

6. Description of CERN’s telephony network ................................................................ 6

7. Commonly used VoIP protocol - SIP ....................................................................... 4
   7.1 Types of SIP servers .......................................................................................... 5

8. Software selection ..................................................................................................... 20
   8.1 Desired call flow ................................................................................................. 14

9. Elements of the system ............................................................................................ 16
   9.1 SIP proxy server ............................................................................................... 16

10. Configuration and deployment ............................................................................... 28
    10.1 Kamailio installation ...................................................................................... 32
    10.2 Kamailio configuration ................................................................................... 34
    10.3 Core parameters configuration ...................................................................... 36
    10.4 UnixODBC module configuration ................................................................ 37
    10.5 Load balancing configuration ...................................................................... 37
    10.6 Dialog module configuration ...................................................................... 40
    10.7 TLS module configuration ........................................................................... 41
    10.8 SNMP module configuration .................................................................... 41
    10.9 Permissions module configuration .......................................................... 42
5.5.8 Attack protection configuration .......................................................... 45
5.5.9 LDAP configuration ........................................................................... 47
5.5.10 Configuration of miscellaneous parameters ....................................... 48
5.6 FreeSwitch installation .......................................................................... 50
  5.6.1 FreeSwitch installation from GIT ...................................................... 50
  5.6.2 FreeSwitch installation from RPMs .................................................. 51
5.7 FreeSwitch configuration ........................................................................ 52
  5.7.1 FreeSwitch Lua scripts ....................................................................... 56
5.8 Clustering software ................................................................................ 58
  5.8.1 Cluster configuration ......................................................................... 59
  5.8.2 Cluster information base configuration ............................................. 60
5.9 Firewall .................................................................................................. 64
  5.9.1 Traffic characteristics ....................................................................... 64
  5.9.2 Configuration .................................................................................... 65
5.10 Complete installation procedure ............................................................ 66
5.11 Client side requirements ........................................................................ 68
6 Testing ........................................................................................................ 68
  6.1 Calls ..................................................................................................... 68
  6.2 IP address banning ............................................................................... 69
  6.3 High availability of the Proxy server .................................................... 70
  6.4 Media servers load balancing and monitoring ...................................... 70
7 Future development .................................................................................... 70
  7.1 LDAP improvements ........................................................................... 71
  7.2 WebRTC and Websockets ................................................................. 71
  7.3 Other improvements ............................................................................ 72
8 Conclusion .................................................................................................. 73
9 Abbreviations and references ..................................................................... 74

List of figures
  Figure 1: VoIP Call ....................................................................................... 2
  Figure 2: CERN network ............................................................................ 7
  Figure 3: Logical placement ....................................................................... 12
  Figure 4: System composition ................................................................... 12
  Figure 5: High availability composition .................................................... 13
  Figure 6: Physical servers configured in cluster ....................................... 14
  Figure 7: Call flow diagram of valid call ................................................ 15
  Figure 8: Call flow diagram of failed authentication ............................... 15
  Figure 9: Call flow diagram of unauthorized call ................................... 16
1 Introduction

The European Organization for Nuclear Research (CERN) [1] is an international organization created in 1954 with target to further the human knowledge in the field of high energy particle physics. The organization is supported by twenty European member states (including Slovak republic) and is located near Geneva, at the French-Swiss border. CERN operates the largest particle accelerator, the “Large Hadron Collider” (LHC) and several particle detectors, such as the largest ones named CMS, ATLAS, ALICE and LHCb.

With around two thousand and five hundred employees and over ten thousand participating scientists from all over the world, the communications inside the organization became essential for collaboration.

Over the years CERN has deployed its own telephone network as well as its own mobile network with collaboration with a Swiss mobile operator. The organization is providing its employees and other members with phone numbers in order to use the telecommunication services.

Cost-free usage of mobile numbers is geographically limited to Switzerland, within Sunrise network span. If people want to minimize communication expenses when they are not physically present at CERN (or Switzerland) they have to search for other means of communication. “Voice Over Internet Protocol” (VoIP) is a very commonly selected option. Existing public services require people to create accounts and to keep separate contact lists of their colleagues, who are also forced to do the same.

VoIP has become common in corporate deployments and CERN started to study the possibility to interconnect VoIP with its telephone network. Such system would allow people communicate directly with their colleagues by dialling the same phone numbers as they would dial on their office or cell phones. No separate contact list would be necessary.

Furthermore the gateway can offer connectivity for users of publicly available VoIP services (such as Skype) or users of VoIP networks of universities and other organizations.

Deploying such system is therefore targeted to provide wider communication options for CERN users in order to improve collaboration experience with their colleagues across the world.
2 VoIP telephony

The classical telephone networks used circuit switched architecture to transport calls. This meant that whenever a call was being established, the circuit was reserved and only a single call was placed on the line for its entire duration. This ensured a significant call quality but on the other and utilized the bandwidth quite ineffectively.

With the introduction of packet-switched networks and voice digitization the new possibility of transporting calls emerged. The development of the IP networks enabled data to be split into packets, which are transported over the network independently. Voice was recognized only as another form of data and as a result, VoIP standards came into existence.

VoIP is a term used for naming a set of techniques that describe the way how real-time communication data (such as voice or video) are transmitted. It is a package of features and services commonly referred to as IP telephony (or Internet Telephony). The purpose of VoIP is to utilize an existing infrastructure of the IP network rather than infrastructure of telecommunication operators, also known as “Public Switched Telephone Network” (PSTN). The term incorporates a set of protocols (that can be used to carry call signalization and data), a set of techniques describing digitization and packetization of voice (or video) streams and a package of features that are available to the end-user.

Realization of a VoIP call consists of a few elementary steps. These steps are “call signalling and media format negotiation” over the Internet, followed by “encoding of analogue input into digital form”. Analogue input is obtained using caller's input devices (such as web camera or microphone). The encoded digital stream is then split into segments and transmitted as IP (Internet Protocol) packets. The other side has to receive these packets and perform digital-to-analogue decoding according to parameters negotiated in the call setup. Decoded analogue stream is then played to the callee using his output device.

![Figure 1: VoIP Call](image-url)
The first software IP phone was introduced in 1995 by Vocaltec, Inc [2]. Since then the potential of VoIP has been widely recognized and organizations like IEEE, IETF, ITU-T and 3GPP have put a lot of effort into developing standards for evolving from PSTN systems into new packet-switched based telephone networks.

The early VoIP services imitated features and nature of classical telephone networks. These systems were usually offered and sold as proprietary solutions. Later on, new kind of services began to take part in the VoIP world. The example of such service is Skype, which enabled people to call each other, but only within the closed network of Skype users (later extended to Skype to PSTN functionality). Current trends in VoIP indicate services that are able to interconnect a call between users regardless of the VoIP service provider. This means a call can be established using standard protocols amongst servers of VoIP providers themselves. Examples of such services are “sip2sip.info” [3] or “iptel.org” [4]. Furthermore, gateways between the VoIP domains and standard, legacy telephony systems were developed to interconnect the IP networks with the existing infrastructure to maintain compatibility.

2.1 VoIP in corporate environment

Thanks to the low bandwidth consumption and the ability to take advantage of an existing IP network infrastructure, many companies are deploying VoIP solutions instead of a separate telephone network. The main reasons why this is happening are the desire to lower communication costs and to merge voice and data networks [5]. VoIP solutions are often part of Unified communications platforms.

Since VoIP runs on packet-switched networks many concurrent calls can be placed on the line at the same time. Call routing can take different paths than media, and therefore the network itself is optimally utilized. There are authentication and encryption mechanisms available for VoIP, thus making it a secure and trusted service.

Utilizing VoIP in a corporate environment has other benefits as well. Many VoIP solutions natively support more advanced services “out of the box”, such as conferencing among multiple callers or voice mail. Although these services are available in legacy solutions as well, the customer must pay for licences in order to use them. Furthermore, the phone is no longer fixed to a position (or cabling) and the user is free to move across the company, while staying connected for example via Wi-Fi or changing his Ethernet plug. Software phones can be used on mobile devices as well, making the user fully flexible with
his mobile internet connection. Integrating VoIP with other information systems enables
administrators and developers to build and maintain fully featured communication
platform, including contact lists, emails, IM and presence, voice and video services. These
platforms are commonly referred to as Unified Communications Platforms.

2.2 Commonly used VoIP protocol - SIP

If a modern VoIP service wants to coexist in a competing environment it must be
built on widely available standards for call signalization and media types. Most common
examples of protocols used for call signalization are H.323 and “Session Initiation
Protocol” (SIP). Recently, SIP is being adopted by vast majority of VoIP providers and is
replacing H.323 in deployments.

This thesis utilizes SIP protocol for call signalization and uses its features to achieve
desired system characteristics. The protocol was standardized by the IETF in “Request For
Comments” 3261 (RFC) document [6]. The latest specification was done in 2002. Since
then, SIP has received many extensions and event definitions. It is described as a protocol
intended for any kind of session establishment, although it is used most commonly for call
signalization. SIP is a text-based protocol with message design similar to “Hyper Text
Transfer Protocol” (HTTP). The difference from HTTP is that SIP is a stateful protocol
with concepts of dialog and transaction. A transaction is a set of requests and responses
related to a single event. Dialog is a set of transactions describing one session. SIP requests
[7] and responses [8] are defined by numerous RFC documents.

The standard port for SIP protocol is 5060 (both TCP and UDP). The encrypted
version of SIP (or SIP secure – SIPS) uses port 5061.

The SIP message consists of several headers and their value. Each header has
different meaning and helps the clients/servers to decide what action should be taken.
These headers create the “header” section of the message. Header section is followed by
message body which can carry information necessary for negotiation of session parameters
by end-devices. In SIP the “Session Description Protocol” (SDP) protocol is used for this
purpose.

The media are carried by the “Real-time Transport Protocol” (RTP) or his encrypted
version “Secure RTP” (SRTP). The protocol was standardized in 2005 by RFC 3550 [9].
There protocols are used alongside with “RTP Control Protocol” (RTCP) which carries
statistics, information for stream synchronization and QoS reporting.
2.2.1 Types of SIP servers

SIP is an application layer protocol of the OSI model. Therefore SIP servers are not to be seen as hardware bound appliances. They are simply applications capable of serving requests from clients. In fact, many SIP servers can physically reside on one physical machine, each with different purpose.

By the IETF standard, there are three main categories of SIP servers:

- **SIP proxy server**
  - this type of server acts as a router of SIP messages originated by clients,
  - during the processing of the message the proxy server examines the contents of SIP request headers and based on information gathered from them it decides what action to take,
  - the outcome of the routing process can be message forwarding (with optional or necessary modification) to another proxy server or directly to end-device, forking the message to multiple destinations or rejecting the message,
  - usually implements several security features such as user authentication or call signalling encryption,
  - it is capable of handling large amounts of messages,
  - can behave statefully or statelessly, depending whether it keeps track about timers and retransmissions,

- **SIP redirect server**
  - does not perform any message routing decisions,
  - instead it replies with 3xx messages, redirecting the user to other destination,
  - many possible redirect destinations can be included within the reply,

- **SIP registrar server**
  - handles REGISTER requests,
  - the user has to notify the registrar server of his location in order to be reachable in the future,
  - registrar server saves the data into the location database (this database can be accessed by the proxy,
  - registration involves user authentication based on credentials,
  - keeps track of every “Address of Record” (AOR) per user account.

The above mentioned types of application servers were strictly defined to handle only call signalization. However, many features were added into the basic definitions over
years as SIP was being adopted by service providers. New types of servers emerged and many of them incorporate the ability to handle media streams. Their classification can be done based on what features they provide. Most frequently used terms to distinguish them are:

- **Media server**
  - this type of server acts as a SIP endpoint (meaning it is capable of originating, answering or hanging up the call and generating and receiving media streams),
  - is it commonly used for “Interactive Voice Response” (IVR) applications,

- **Back2back user agent (B2B UA)**
  - typically implements many features of a SIP proxy server,
  - modifies SIP messages in a way that it seems the server is always the second participant of a call (both for the caller and the callee),
  - this causes the media streams to always traverse the server, which can alter the streams, for example by transcoding the media from one codec into another,
  - processing media streams requires a significant amount of resources (CPU and memory) resulting in this type of server not being so powerful in handling large number of simultaneous call and routing of SIP messages,

- **Gateway**
  - gateways are similar to back2back user agent servers, but the main difference is that they bridge the SIP environment to other environments, such as PSTN,

- **“Session Border Controller” (SBC)**
  - placed at the edge of the private network where it meets the internet,
  - it is a type of B2B UA which implements various protection mechanisms.

### 2.3 Description of CERN’s telephony network

There are several IP networks deployed at CERN, each with a different purpose. The “General Purpose Network” (GPN) offers intranet services and Internet connection for its users. There is a general CERN firewall system between this network and the public Internet.

The “Technical Network” (TN) is used for connecting industrial systems and accelerator control devices. Another level of protection (by a separate, stricter firewall system) is deployed between this network and the GPN.
A group of IP Private Branch Exchange (PBX) devices, produced by Alcatel, is connected to the GPN and TN. These devices have separate CPUs that handle call signalization. These CPUs are capable of handling SIP protocol, enabling CERN to promote IP phones as well as analogue phones (which are still necessary, for instance for emergency systems inside the accelerator tunnels). The PBX devices are connected in a redundant topology. Around the LHC ring, at the eight surface access points, several smaller PBX setups are deployed to connect surface and underground telephones with the main PBX devices. This also ensures higher stability of the telephone network (by relieving the main PBX nodes of the load by not passing the media through them if not necessary).

Throughout CERN sites, there are fourteen main telecom rooms where PBX devices or distributor patch panels are placed. Each building has its own telephone patch panel, connected to PBX through the distributor patch panels. Within a building the cables from the patch panel goes through floor patch panels to office phone plugs. This applies for analogue connections. The interconnection of the PBX devices is IP based and the connections to the telecom operators (Sunrise, Swisscom, Cablecom, Colt and French Orange) are made through fiber optics lines connected in the CERN Computer Centre [10].

The mobile network spans across all CERN sites and it also covers the whole underground Accelerator complex (coverage is provided by leaky feeder cables).
The position of the system (VoIP gateway) is displayed on the picture above. The intention is to connect the system to GPN in front PBX devices in order to provide security features like topology hiding and attack protection techniques.

2.3.1 Numbering scheme and services

Phone numbers can be dialled in three forms in order to reach CERN. The form depends, respectively, on the location of the caller:

- **“CERN format”**
  - available to callers on CERN telephone network (generally only on CERN sites),
  - 6xxxx and 7xxxx for fixed phones,
  - 16xxxx for mobile phones.

- **Switzerland regional format**
  - reachable from within Switzerland,
  - 022766xxxx and 022767xxxx for fixed phones,
  - 076487xxxx for mobile phones.

- **International format**
  - globally reachable,
  - +4122766xxxx and +4122767xxxx for fixed phones,
  - +4176487xxxx for mobile phones.

CERN provides many services related to the fixed and mobile networks. Amongst the basic ones are worldwide calls with least cost routing (calls are divided into private and professional calls), “Short Message Service” (SMS) and data connections for mobile devices (with roaming options when the user is outside CERN). There is also a mail-to-sms gateway available for usage.

The more advanced services provided by CERN are voicemail, numerous conferencing services (such as Vidyo [11] or MyTeamWork [12]), call centres, “Interactive Voice Response” applications (IVR) and a complex internal charging application.

3 The scope of the project

The purpose of designing this system was to provide an entry point for SIP (port 5060) and SIPS (port 5061) protocols to “@cern.ch” domain. Interconnection with
CERN’s PBX provides users with the ability to make cost-free VoIP calls to CERN’s fixed and mobile phone numbers.

The specification of the project required the system to provide:

- high-availability
  - featuring load balancing and failover of the service,
- securing the system against undesired attacks
  - such as Denial of service or password cracking,
- way of proving the call is valid
  - meaning if it is established by human,
- monitoring features and maintainability scripts and tools.

There are many conditions the system had to meet in order to be deployed in production environment. This thesis provides overview of system components and their purpose. It also talks about design and functions usable by the end user. Software selection and configuration guidelines are stated in the next chapters (4.5 Software selection and 5 Configuration and deployment).

### 3.1 Usage cases

The system was required to provide the ability to make cost-free audio-only calls to CERN’s numbers (both fixed and mobile) for users utilizing VoIP. This specifically includes users of Skype service and any SIP-based services. Dialled numbers were expected to be in all three forms, as described in the chapter 2.3.1 Numbering scheme and services.

These numbers were followed by “@cern.ch”, when calling the CERN domain. Optionally the system could have offered the ability to call people dialling their standard CERN e-mail addresses.

There were numbers which were not desired to be reachable from outside CERN. They represent specific services that were not supposed to be opened to general public (see chapter 5.5.7 Permissions module configuration).

The system was intended to be designed to enable Skype users to call to CERN’s numbers. Skype operates a service called “Skype Connect” [13], which is oriented for business customers. It enables companies to integrate Skype into their VoIP environment, thus enabling public to call the company using Skype. This service is not cost-free and the company has to pay monthly fees for every registered SIP profile and a number of
channels (meaning number of available concurrent calls). Skype call signalling messages are converted to SIP protocol and forwarded to customer’s VoIP server.

This approach was suitable for CERN as it was considered sufficient to register only one SIP profile and pay for a small number of channels (supported by observation of number of concurrent calls traversing the PBX at any given time). Number of channels can be increased if necessary. Skype users would call to this registered profile number and once the call is established with the system at CERN the Skype user is prompted to dial the desired number he wishes to reach inside CERN’s telephony network.

When a user from any SIP-based service is establishing a call, he can directly dial the CERN number he wants to reach (followed by “@cern.ch”). The number itself can be in any of the formats mentioned above, for example “166468@cern.ch” as shortened version, or “martin.pohancenik@cern.ch” as e-mail form.

3.1.1 Unauthorized calls

Call is unauthorized if the caller is not recognized by the system or when calls to the destination number are not permitted.

If a call is not authorized for the caller, call is dropped right away and appropriate SIP response is sent to the caller (such as 403 Forbidden, or 403 with situation-specific text description).

3.1.2 Authorized calls

If a call is authorized and caller is from Skype network, he will use an IVR application to dial CERN number he wants to reach. If he dials a permitted number he is bridged immediately. Therefore, the call is authorized for every Skype user provided he dials the correct number using “Dual-Tone Multi-Frequency” (DTMF). If not, call is dropped.

If the incoming call is not from Skype Connect servers (verified by source IP address and/or server certificate if using TLS) it must be treated differently. Digest-MD5 mechanism [14] (way of proving that user and server both have the same user credentials) must be used to verify the legitimacy of the caller, if he is a local subscriber. This mechanism originates from HTTP protocol.

The digest-MD5 authentication might not be possible because a user does not have to be a local subscriber, which means there are no user credentials stored within the system for the particular caller. In this case the system needs a way how to prove the “Uniform
Resource Identifier” (URI) can be trusted. Once the system decides that the caller ID is authorized, it must have a way to authenticate the user to make sure there is a human on the other side of the channel.

This may not seem in order, because in most cases authentication comes before authorization. However, in this particular use case, “authentication” should be seen more as a way of proving that the call is being made by human.

3.1.3 Explicitly forbidden calls

The system must not provide a way of establishing other kinds of calls and it must strictly forbid them (for example calls between SIP subscribers or calls to forbidden dialled numbers). It must accept only incoming calls, therefore not to act as an outbound sip proxy server.

3.1.4 Emergency calls

This system does not provide a way to call 112 and other emergency numbers in Switzerland (or France) and must not be considered as a replacement for such system.

3.2 Users

Various types of users must have been distinguished within the system. General classification was done over the condition whether the user is a CERN member (has CERN computing account) or not (does not have such account).

3.2.1 CERN users

This group includes people with CERN computing accounts. These accounts were considered to be treated as a local subscriber accounts. However, this option depended on what client software does the person choose to use and if it can be trusted (because of the sensitivity of CERN's password).

Since this system was intended to serve only as inbound VoIP gateway into the telephony network of CERN, providing an account that overlaps with the main computing account is far beyond the scope of this project, as the system does not offer the ability to receive calls.

In the end, the option to map the CERN computing account to the SIP account was discarded because of the above-mentioned reasons. Also, there was no way to integrate
CERN’s centralized authentication (Radius, Active Directory, Kerberos, etc.) services with the system.

3.2.2 Non-CERN users

The other types of users are those that do not own a CERN computing account. They can be further classified as:

- users with Skype account,
- users with any other SIP-based service account.

The system does not store user credentials for neither of these groups. It required accepting call signalization incoming from other SIP proxies, but because of security precautions it must have incorporated ways to verify the legitimacy of the caller.

There was a possibility to provide a person with a local SIP account, but this is indeed not necessary because of the fact that the system servers only for inbound calls. Therefore no registrations are necessary for user to originate call. It is an extra feature, intended for future improvements and development.

4 Project analysis

4.1 Logical topology

The purpose of building the system is to put some logic between CERN’s telephony network and the public Internet infrastructure. This system must serve as an entry point for VoIP, thus allowing users to establish their calls using SIP.

![Figure 3: Logical placement](image)

The system logically consists of several components. Based on facts stated about SIP server types (described in chapter 2.2.1 - Types of SIP servers) we were able to outline the basic design of the system.

![Figure 4: System composition](image)
The Media server is in fact the core of the system, as it provides functionality for user. These servers are prone to be highly resource-demanding when running on standard PC hardware, as all media processing is done by software. They tend to overload the whole system and degrade its performance. Therefore the Media server had to be protected by signalling-only Proxy server.

Because of the high-availability requirement, every element had to be at least doubled (it was not limited) to prevent single points of failure in the topology.

![Figure 5: High availability composition](image)

This enabled us to deploy load-balancing and failover features. Load-balancing was done between the two Media servers. As the dialog information was saved into the database, the Proxy always knew how to route in-dialog messages (meaning they were not misrouted because of load-balancing).

With this setup a call can take up to four paths, depending on which Proxy server is active. If the Media server stops responding to keep-alive messages he is declared inactive and call is transferred to another Media server instance. This is a transparent process and the end-user is not aware of it.

Each Media server was configured to handle only a limited number of calls while refusing every call above the limit. The load-balancing feature also enabled us to distribute channels among multiple physical servers (thirty channels per Media server instance). This setup provides availability of channels even if one of Media server instances brakes. The number of Media server instances is not limited and the only configuration necessary for the Proxy server is adding the IP address of new Media server to the database.

### 4.2 Physical topology

Applications were installed on Linux servers configured in a cluster and thus acting as a single device, managing its resources and being reachable at one point of the network (one service IP address).
Because of using Linux, the system is not hardware specific. It can run on many servers from various vendors. For the system the probability of losing the power supply can be lowered by deploying the machines at different locations.

![Physical servers configured in cluster](image)

The machines required only one Ethernet port to connect to CERN’s General Purpose Network. This network is IP based, offers Internet connectivity and is behind the main firewall. In addition, the machines run their own firewall, allowing only defined connections to be established. This is described later in the configuration section.

### 4.3 Desired call flow

When a user is communicating with the system he needs to direct the signalling to the Proxy server. When the proxy server replies, the IP address of the Media server handling the call will be included in the response.

Call consists of several phases. The first phase is the call establishment. Here the user negotiates the call parameters with the Media server by sending the INVITE message and receiving a 183 reply. By receiving the 183 reply the call enters the early media session phase. In this phase the user is prompted to enter information that would prove he is a person.

If the authentication is positive, the Media server creates another call leg with PBX and bridges both legs. The call thus enters an active phase. It ends when one of the legs hangs up.
The authentication of the user is done by inputting information using DTMF. The codecs are negotiated between the caller and the Media server on the first call leg. The codec negotiation is done separately in the second call leg. After the bridge, the Media server does the codec transcoding if necessary.

Dialog is only recognized in the Proxy server once the authentication has been accepted by the Media server.

If the authentication fails during the early media phase, the call is ended.
If the call is not permitted, the messages are not relayed to Media server. Instead an error message is sent to the caller and the transaction ends in the first phase. This means the Media server is not contacted at all.

![Call flow diagram of unauthorized call](image)

**Figure 9: Call flow diagram of unauthorized call**

### 4.4 Elements of the system

Based on the requirements that had to be met according to the specifications described in the previous chapters, the solution had to provide necessary interface to the end-user alongside being autonomous in stand-alone operations. It also had to provide several management interfaces for the administrator. Therefore it is possible to see the system as a set of several functional elements.

The elements that provide interface for the end-user are:

- SIP proxy server,
- Media server.

There are also elements that take care of running the user-interface modules and provide resources for their operation. They are:

- database connectivity software,
- computer clustering software,
- monitoring software,
- various scripts.

All the listed elements cooperate together and in the end are seen as a complete package and service.

#### 4.4.1 SIP proxy server

Proxy server must be a contact point for “cern.ch” domain for SIP protocol. It must be able to accept connections on standard ports for both TCP and UDP transport protocols.

It must not serve as an outbound proxy server for the above mentioned domain and must accept only calls destined for the CERN’s PBX. It must not act as a proxy server between SIP users (local or foreign).
In a process of a call establishment it must keep track about ongoing progress and when the call is answered, it must save dialog information into persistent data storage. In event of call hang-up, the information must be erased.

The software must have a way to take call establishment permissions into account when evaluating whether the SIP message can be forwarded to media server. These rules must be easily manageable and kept in place outside the system.

The proxy application must protect the Media server from unnecessary load by implementing several flooding detection/protection mechanisms. The source of attack must be detected as soon as possible and banned for a period of time. If this is done on IP address basis, then strict parameters has to be considered and they must be easily customizable to adapt to the needs of production environment (such as not to deny users behind the same “Network address Translation” (NAT).

If the SIP request is considered valid (passes all checks), it must be forwarded to the Media server (only).

### 4.4.2 Media server

Media server application will have to reserve a channel for new incoming call. Next step is to perform authentication of the user, as mentioned before. The number of channels must be limited to sixty by CERN’s requirements. This number was arbitrary as it was just a security precaution.

If the user authentication succeeds, the Media server must create another call leg towards the PBX and bridge the call. Since the system needs to act as an SBC, a completely new dialog is created with a new call leg. The ringing tone can be bridged during the early media session as well. If the extension is dialled via DTMF, it has to be verified against the same rules the proxy server uses, prior to second call leg establishment. Upon hang-up of one of the call legs, the other call leg must be ended as well and channels freed.

If the user authentication fails, the call must be dropped and channel freed.

### 4.4.3 Database connectivity software

Several database system options were available for the system:

- **MySQL database** [15]
  - can run locally within the system,
would need to be included in the cluster configuration with data replication to maintain the database if one of the machines fail.

- **CERN Oracle service**
  - CERN offers several possibilities to deploy the databases,
    - devdb11 – database dedicated for development purposes,
    - cerndb1 – database dedicated for production purposes; offers transparent failover, meaning no reconfiguration is necessary for clients (but there is approx. ten minutes of unavailability in case of failure).
  - **Combination of previous two**
    - possibility to use both database systems at the same time with options of parallel access or on try-fail basis (meaning if the Oracle is unavailable, use MySQL),
    - this would require keeping data replication from one DB system to other,
    - Oracle offers its GoldenGate [16] service which is intended for this purpose,
      - GoldenGate is a successor of Oracle Streams service,
    - note: for the time being CERN does not support GoldenGate as a separate service,
      - this feature would be used only for testing purposes.

4.4.4 **Computer clustering software**

All devices running the service must be included in a Linux Cluster as cluster nodes. A computer cluster is a group of computers that act as a single system. Clusters are used for many tasks, such as parallel computing or providing a highly available service.

The management of a cluster consists of two elements. The first element is a messaging layer that takes care of mutual communication between the cluster nodes. This layer has to provide ways how to announce events happening within the cluster (for example node failure, node becoming alive, administrator’s intervention or command interpretation). The second element in service providing clusters is resource management. Usually there are several resources configured and shared between the nodes. Basic types of resources are unique resources (meaning the resource runs only on one node at any given moment and the other nodes are in stand-by mode) or cloned resources (meaning the resource runs at multiple nodes at the same time). The management within a Linux cluster is usually done using “Open Cluster Framework” (OCF) [17] scripts. These scripts incorporate every action that has to take place once a specific event within the cluster occurs.
4.4.5 Monitoring software

The system must be monitored using the “Simple Network Management Protocol” (SNMP). This was one of the CERN’s requirements. In general, the system must incorporate an application that will be able to respond to SNMP queries and send SNMP traps to the trusted manager.

4.4.6 Administrator script

A shell script was developed for administrative use, named “SIPuser”. This script can be used directly from the shell as a regular command. It enables the administrator to interact with the system, its database and perform basic administrative tasks.

SIPuser add account_name account_password

When the script is used as shown above, it adds a user account into the database. This was useful if the system was acting also as a SIP registrar server.

SIPuser rm account_name

By this command the administrator was able to remove the user account.

SIPuser db show table_name [N]

The output of this usage of the script was the content of the database table specified as a parameter. The optional parameter N was used to control the number of characters displayed per column.

SIPuser db “sql_query” [N]

This variant of the script usage enabled the administrator to execute basic operations (select, insert, update, delete). Parameter N was available for usage if the command outputted contents of database tables.

SIPuser dispatcher [loop]

This command displayed the real-time state of the available Media servers. If the loop parameter was used the command displayed the state while refreshing every second. This command was available only on cluster node where the Proxy server was currently running.

SIPuser permissions {rm|add} {allow|deny} ‘rule’

It was possible to use the script to add or remove the rule specified by the syntax above. The apostrophe sings were necessary to hold the whole rule (as it might have contained the quotation marks).

SIPuser permissions show
The output of this command was the formatted content of the database stored permissions rules.

*SIPuser permissions upload*

This command parsed the rules stored in the local files line by line and uploaded them to the database, while completely overwriting the old content. This command was advised to be used only on the cluster node where the Proxy server was running.

*SIPuser ipban*

This showed the currently banned IP addresses, both at Proxy and at firewall.

The full SIPuser script can be found in the attachment [2 SIPuser administrator script].

### 4.5 Software selection

Several OpenSource products available under general public licences were considered. These included:

- **Kamailio** [18]
  - original fork project of OpenSER with the recent source code merging,
  - very stable SIP proxy server capable of handling thousands of call setups per second,
  - reliable and secure application that supports all transport protocols,
  - modular design offers easy extendibility and configuration,
  - configurable with very powerful routing logic,
  - up-to-date implementation of standards with the parallel development of new ones during their standardization process,
  - very commonly used in SIP-to-PSTN deployments,

- **OpenSIPS** [19]
  - fork project of OpenSER, developed separately since 2008,
  - mature SIP proxy server that is extensible to provide back2back functionalities,
  - modular design provides great extendibility options,

- **OverSIP** [20]
  - fairly new project intended to be a SIP proxy server deployed at the edge of a network,
  - provides the ability to build the SIP routing logic in Ruby language,

- **FreeSwitch** [21]
o cross-platform multi-protocol software switch,
  o powerful media server and media bridge, frequently used to build voice mail
    and conference servers,
  o natively supports most recent standards in media encryption, such as ZRTP,
  o supports many codecs, SIP, Jingle extension XMPP, H.323,
  o enables the developer to build voice-applications using JavaScript, Lua, Perl or
    Python languages,

• Asterisk [22]
  o very popular IP PBX solution,
  o used in many deployments by service providers,
  o includes SIP proxy and registrar server,
  o can act as a Back2back user agent,
  o offers hardware support in form of network interface cards from Digium, which
    is a company sponsoring the Asterisk project,
  o Digium cards allows the provider to connect to E1/T1 lines to Asterisk server
    and thus interconnect Asterisk with classical PBX systems,

• SipXecs [23]
  o a complete SIP IP PBX focused on ease of use,
  o can be extended by modules to provide a full unified-communication platform,
  o if offers a very powerful web-based management interface,
  o supports voice, video, IM and presence services and several advanced services,
    such as conferencing or voice mail,
  o offers the interfaces to interconnect with other protocols, for example XMPP,
    enabling the service to interconnect with Google Talk,

• MjServer [24]
  o java-based implementation of SIP stack,
  o comes with a core package that implements SIP features,

• MySIPSwitch [25]
  o SIP proxy server combining tools to manage networks of multiple SIP
    providers, with focus on guaranteeing least cost routing,

• OpenSBC [26]
  o all-in-one session border controller with support of registrations and (NAT)
    traversal,
Partysip [27]
- incorporates functionalities of SIP registrar, proxy and redirect,
- runs on many platforms (including Linux, Win32 and BSD),
- currently supports only UDP, while TCP support is in development,

Siproxd [28]
- is a masquerading SIP daemon most valuable in environments behind a firewall or NAT,
- it is a SIP proxy server with integrated registrar functionalities,
- currently supports only UDP.

It was also possible to develop a simple SIP server from available open source SIP stack libraries. Using this solution would lack the support of the community as is available for solutions mentioned above. Also this would significantly extend the time of the development process.

Options that met the requirements for the above mentioned tasks of the proxy server were Kamailio, OpenSIPS and OverSIP. FreeSwitch and Asterisk fulfilled the requirements for Media server. Their consideration and reasons for final decision are described in the next chapters.

4.5.1 Proxy server

Kamailio and OpenSIPS are similar in design and configuration, as they both were created as fork projects of SER (SIP Express Router) [29]. The original fork was established in 2005 and was named OpenSER. Later in 2008 it was renamed to Kamailio due to trademark conflicts. However, OpenSER still remained developed under the same name. In the same year, OpenSIPS became a separate project, resulting from a conflict between the developers [30].

Both application servers have a modular design, which means the core application is extensible by modules, based on functionality requirements. The configuration of the modules tend to be very straight-forward in many cases. It consists of setting module parameters based on what is the desired functional outcome. After setting the parameters, the developer can make use of functions the module provides. Kamailio and OpenSIPS have a very powerful way of customization of routing logic - a script composed of functions and variables gained by parsing the SIP request.

This script uses a very simple syntax to build a routing logic for a SIP message, making use of functions provided by loaded modules. This enables the developer to write a
very scalable policy that can be specific in behaviour to every distinct SIP request. The
script writer can add headers to SIP message, alter its parameters or forward, reply or drop
it.

Kamailio and OpenSIPS can also track dialog information about ongoing calls and keep them in the data storage, such as database. In case of application failure another instance of application can take over these calls. This is a feature that fulfils the failover requirement.

Overall, both Kamailio and OpenSIPS enable the administrator to scale the routing process to every detail, and that is what we needed to build a flexible, while extremely stable SIP proxy.

OverSIP is a fairly new project (started August 2012). It’s being developed by authors of websocket standard draft (draft-ietf-sip-core-sip-websocket-06) [31]. Websockets allow web pages to create a network sockets and thus be able to communicate with foreign servers (this is planned for future web-based communications and gaming experience). Since this application server is new, it doesn’t support as many features as Kamailio or OpenSIPS. The configuration is written in Ruby language, which is a bit more complicated then Kamailio’s or OpenSIPS’ configuration. Also the documentation of functionality is much scarce, while on the other hand Kamailio’s and OpenSIPS’ modules are documented quite precisely.

Because of this fact, the option to use OverSIP was discarded.

Major versions of Kamailio are released approximately every ten months [30]. Version 4.0, released in March 2013 is the most recent stable release, that has been heavily tested [32]. The version number was changed to next major release (from 3 to 4) due to source code merging of Kamailio and OpenSER, which is considered a milestone in the development process.

OpenSIPS sticks to version numbering inherited from the original SER project. New versions are released with more or less similar schedule as the one of Kamailio [30]. Most recent version of OpenSIPS is version 1.9, released in February 2013.

Kamailio offers the following modules to meet the requirements of the system:

- **dialog** - SIP dialog tracking information module with ability to write the data into persistent storage, ensuring their accessibility of the server fails,
- **dispatcher** - load-balancer and failover module, capable of autonomous monitoring of availability of other SIP servers,
• **permissions** - module providing checks of several SIP message header, determining whether the call can/can't be established based on rules stored in external configuration files,

• **pike** - security module; performs checks for IP addresses originating the SIP requests and decides whether the IP address is an originator of flooding based on configured thresholds,

• **htable** – provides hash table data structures access from routing logic to enhance several features, such as security checks,

• **ratelimit/pipelimit** - security modules, implementing traffic limits based on request type basis (ratelimit) and on user/IP/request basis (pipelimit),

• **sanity** - performs verification of SIP message by many types of checks from which several are required by standard,

• **ldap** – “Lightweight Directory Access Protocol” (LDAP) connectivity module,

• **db_unixodbc** - provides “Open Database Connectivity” (ODBC) interface,

• **db_oracle** - provides Oracle database connectivity,

• **db_mysql** - MySQL database connectivity module,

• **db_cluster** - combines many database connections into one “cluster connection”, while letting the developer to choose how the databases are accessed (for example on round robin basis),

• **websocket, outbound** - new modules for version 4.0 which provide the ability to use the websocket transport protocol (accepting connections from web-browsers). This enables clients behind NAT to keep the TCP connections opened using the outbound mechanism.

OpenSIPS provides a smaller range of modules. Their description and naming is basically the same as for Kamailio. Only differences are stated:

• **ratelimit** - implements security policies based on message type limiting,
  
  ◦ OpenSIPS does not provide pipelimit module functionality,

• **db_virtual** - similar to Kamailio’s db_cluster.

OpenSIPS does not offer any module to verify the correctness of the SIP message. Surely, this process can be written into the routing logic by adding many header checks etc., but it would lack the performance of lower level programming language. Compared to Kamailio, OpenSIPS is also less equipped when it comes to security implementation. Kamailio offers wider options with its htable module. Other great advantages of Kamailio
are its *websocket* and *outbound* modules, which are missing for OpenSIPS. These modules are important for future improvements, for development of a web-based phone application (see chapter 7 Future development).

Considering the variety of modules and all facts stated about them, Kamailio was the best candidate for Proxy server in the system. Its modules provide greater configuration options for ensuring security features which, when utilized, can produce a solid, stable and high-performance proxy server. Kamailio takes care of call routing decisions (based on information read from the SIP messages) before routing the call to media server.

### 4.5.2 Media server

Media server chosen for this system must be able to act as a Back2Back User Agent and must provide a way to build large, secure and scalable voice applications. These conditions were met by Asterisk and FreeSwitch.

Both of them can be used to achieve the required behaviour and both can be seen also as a proxy server. But because of the fact, that they handle media as well, they are rarely used in stand-alone setups. It is a good practice to put SIP-only proxy server in front of them. By having the proxy capable of handling much bigger number of calls and making routing decisions before forwarding the call to media handling server, the service becomes more solid.

FreeSwitch developers have chosen a different approach in releasing their software. By releasing multiple minor versions of FreeSwitch (each available as a frequent update of a source code) they target to fix occasional bugs as soon as possible, while developing new features. Current development version is FreeSwitch 1.3. The last stable version, also available in form of RPM packages, is 1.2.8 [33].

Asterisk developers support several “Long Term Support” (LTS) releases as well as several standard ones, all with frozen pack of features. The current stable release of Asterisk is version 11, released in October 2012 [34].

FreeSwitch was founded by one of the Asterisk developers, Anthony Minessale, who disliked the way things were approached at Asterisk’s development process. In the end he decided to start his own product [35].

The main difference between Asterisk and FreeSwitch is that Asterisk is branded as an IP PBX, while FreeSwitch tends to be named as a VoIP soft switch. Asterisk is therefore targeted to run as all-in-one voice service provider deployed internally in the company, while on the other hand FreeSwitch soft switch provides an interconnection
between one network and the other (can also be understood as a bridge between different signalization protocols) while still having the ability to act as a IP PBX. This fact speaks in favor of FreeSwitch.

The configuration in Asterisk is done using a large number of files, where various different parameters for the server are set. It can be quite difficult to know your way around within these files and the developer often finds himself stranded among the files, stuck with reading lengthy documentation.

FreeSwitch is configured by number of XML-formatted files (“eXtensible Markup Language”) whose structure is quite transparent and every parameter is precisely linked to some element within the XML tree.

What is required from Media servers is in fact a very basic functionality. They need to be able to accept calls only from the Proxy server and run a script that will control the call and perform user and number validation.

The user validation was not required only on SIP level but also an input from user was necessary. This signified running an external script, capable of handling the call (meaning the media server had to provide and “Application Programming Interface” (API) for the script). Many languages were available by both servers, such as Perl, Python or Lua. Considering the APIs and level of their documentation, FreeSwitch server was selected for usage. The API provided for Lua language is significantly richer than the one of Asterisk. It enables the designer to handle real-time call-related parameters, make database queries and manage call legs (regarding their call state).

On the other hand, Asterisk was not completely discarded and some testing regarding the web-based client applications was done with it. Its progress seemed to be a bit advanced in this field. For more information see chapter 7.2 WebRTC and Websockets.

4.5.3 Other elements

Database connectivity software

After evaluation of database deployment possibilities, CERN’s Oracle service was selected to provide persistent data storage. Database cerndbl was used to store the database schema and the user was created through the service interface. This decision strips the system of disk and CPU load associated with running a local database system. It also ensures a single DB entry point for a single node because the database was accessed using the standard UnixODBC connector and Oracle driver provided by CERN’s repositories.
**Computer clustering software**

As described in the previous chapters, the clustering software consists of two elements. To stick with CERN practices, Corosync [36] was selected for the messaging layer and Pacemaker [37] for resource management. This combination is used in other systems of CERN, such as the DNS system.

**Monitoring software**

SNMPd was selected as a standard SNMP daemon for Linux based systems. It provides AgentX support for applications within the node, enabling them to run SNMP subagents and communicate with the SNMP manager.

### 4.6 Summary of used technologies

The main pillar of the system is SIP protocol, which is used for call signalling between the end-user and the system, system and the PBX.

SIP protocol was also used when communicating with the Skype Connect servers. SIP operates at application layer of ISO OSI model.

On the lower layers of the OSI model, all stands on IP protocol (layer 3) and TCP/UDP protocols (layer 4), including TLS over TCP. I was a plus to prepare the system for websocket and websocket-secure sub-protocols over TCP, as they are essential for future development. There was an opportunity to facilitate other transport layer protocols, such as UDP-lite [38] or SCTP [39]. However, because of insufficient support on client side, it was not a useful feature to include.

There are several scripts on the server side that take care of operation-critical tasks. These scripts are written using python and bash and are called by cron daemon. Cron is a Linux scheduler program capable of executing commands on periodical basis, as defined by the configuration. It can handle short time periods as well as long ones.

The database connectivity was provided by ODBC drivers. These drivers provide standard ODBC interface for applications running on local machine. For a database system itself, CERN’s centralized Oracle service was used.

The clustering software was autonomous in operation and generally does not require any administration actions. To stick with CERN practices, Corosync was selected for the messaging layer and Pacemaker for resource management. Pacemaker utilizes the OCF scripts for its functionality.
The configuration files for proxy, media servers and cluster nodes are described in the next chapter.

5 Configuration and deployment

A detailed configuration of the system elements is described in following chapters, with emphasis on highlighting important facts and relations between the configuration parameters.

5.1 DNS configuration

The DNS servers of CERN required adding a few lines in order to enable resolving for SIP protocol. This enabled clients to automatically resolve how to contact “cern.ch”.

- cern.ch.
  IN NAPTR 0 0 "s" "SIPS+D2T" "" _sips._tcp
- cern.ch.
  IN NAPTR 1 1 "s" "SIP+D2T" "" _sip._tcp
- cern.ch.
  IN NAPTR 2 2 "s" "SIP+D2U" "" _sip._udp
- _sips._tcp
  IN SRV 0 1 5061 <<#HA_Hostname#>>
- _sip._tcp
  IN SRV 1 1 5060 <<#HA_Hostname#>>
- _sip._udp
  IN SRV 2 1 5060 <<#HA_Hostname#>>
- <<#HA_Hostname#>>
  IN A <<#SERVICE_IP#>>

This information lists “Name Authority Pointer” (NAPTR) records with preference (lower is more preferred). Therefore the system prefers to be contacted by encrypted messages. Next section consists of “Service record” (SRV) records which specify on which ports is the service reachable. The last section is an “Address record” (A) resolving to an IP address of the service.

See attachment [3 DNS zone file] for an example file with a complete DNS configuration necessary for the zone.

5.2 Server preparation

Majority of systems running Linux-based services in CERN run Linux distribution named “Scientific Linux CERN” (SLC) [40]. This distribution is based on RedHat Enterprise Linux and is maintained by CERN’s IT department. This is a standard operating system, which incorporates scientific software (widely used by physicists at CERN), various server packages and CERN-specific configurations. CERN also maintains its own RPM repositories for this OS. SLC is also widely used for deploying Linux-based systems (such as DNS).
This system was designed to run on SLC version 6, 32-bit operating system, based on RedHat Enterprise Linux 6.

Unless stated otherwise, every command or configuration file edit had to be made on both cluster nodes. Configuration tasks or actions required to be made only on one of the machines are explicitly stated throughout the text.

After a clean, minimal installation with no additional packages selected during setup there was only the root user created. All authentication options were disabled, except for local authentication (using the shadow file). Several parameters had to be changed within the OS.

By enabling a Linux kernel module “nf_conntrack_sip”, the server was able to keep track of ongoing calls and temporarily open ports in firewall according to information gained from the call signalling. The module offered several configurable parameters. We needed to disable parameter “sip_direct_media” in order to tell the module to expect media to traverse the server (otherwise it would have expected only signalization messages). Also local SIP and SIPS ports used in the system could have been specified as a comma-separated list in the “ports” parameter. This is necessary only if non-standard port numbers are used. These numbers tell the module where to expect SIP traffic.

```
#echo "modprobe nf_conntrack_sip" >> /etc/rc.local
#echo "options nf_conntrack_sip sip_direct_media=0" >> /etc/modprobe.d/nf_conntrack_sip.conf
#echo "options nf_conntrack_sip ports=<#SIP_PORT#,<#SIPS_PORT#>>" >> /etc/modprobe.d/nf_conntrack_sip.conf
```

Verification whether the module has been loaded after reboot of the system could be done using the command “lsmod”.

The output of this command must include the module name. The module parameters could have been verified by reading the content of files in the “/sys/modules/nf_conntrack_sip/parameters/” directory.

Since the service consisted of several computers connected in a cluster and it was reachable only through a single IP address (service IP), the operating system must allowed the applications to bind to non-local IP address on the Ethernet interface.

```
#echo "net.ipv4.ip_nonlocal_bind = 1" >> /etc/sysctl.conf
```

In the end we disabled unnecessary services that run on SLC6 by default. These services were rpcbind, cups, sendmail, avahi-daemon and portreserve.
5.3 Database connectivity

The central Oracle database service was selected as the main data storage for the system. Therefore, the system must have incorporated a database client. Since many applications within the system needed to access the database, the effort of keeping the configuration centralized became essential. UnixODBC was capable of handling the job as it offered a standard ODBC interface for applications.

The installation was straight-forward using the RPM packages obtained from yum repositories. Furthermore, CERN repositories provide packages that take care of automatic updating of Oracle aliases. That means if the information how to access the database changes (for example the IP address or port change), the changes are distributed through update of the package. These data are stored within the “/etc/tnsnames.ora” file. Update to this file is announced some time before it takes place, by the mailing list of Oracle users. This means that even if the automatic updates were turned off, the administrators would be notified about the changes.

We installed the UnixODBC software and the oracle drivers using the “rpm –i” command. The list of installed packages was:

- unixodbc-2.2.14 + development libraries,
- oracle-instantclient-11.2.0.3.0, packages basic, odbc, devel and CERN package containing tnsnames,
- their dependencies libaio-0.3.107 and libodbc-3.52.7.

Once the packages were successfully installed we advanced to the configuration part.

5.3.1 UnixODBC configuration

Every application in Linux runs under a certain user, whether it is a system user or a standard account for human. This user needs to have several environment variables exported in order to use the oracle driver correctly. Instead of configuring these variables separately for every user we changed the content of the “/etc/profile” file by adding

```
LD_LIBRARY_PATH="\$LD_LIBRARY_PATH:/usr/lib/oracle/11.2.0.3.0/client/lib"
export LD_LIBRARY_PATH
TNS_ADMIN="/usr/lib/oracle/11.2.0.3.0/client/bin"
export TNS_ADMIN
ORACLE_HOME="/usr/lib/oracle/11.2.0.3.0/client/bin"
export ORACLE_HOME
```
at the end of the file. This ensured that these variables will be copied and exported for every user created in the system henceforth.

ODBC drivers use “Data Source Names” (DSNs) to differ between various database connections and drivers. Prior to creating our DSN we needed to configure the UnixODBC with path to the Oracles driver. Within the “/etc/odbcinst.ini” file we created a driver called “Oracle”.

```
[Oracle]
Description     = ODBC for Oracle
Driver          = /usr/lib/oracle/11.2.0.3.0/client/lib/libsqora.so
...
```

This driver could now be used by DSNs to connect to the database servers. In the “/etc/odbc.ini” file we created a DSN named “OracleSIPGatewayProduction” with several parameters specific for the driver. Amongst them we specified that the tns name for the connection is “cerndb1”. This alias name is automatically searched for in the above-mentioned “/etc/tnsnames.ora” file for resolve the IP address of the database server and other connection parameters.

```
[OracleSIPGatewayProduction]
...[output omitted]...
Driver = Oracle
DSN = OracleSIPGatewayProduction
...[output omitted]...
ServerName = cerndb1
...[output omitted]...
NUM = NLS
```

As can be seen from the contents of the file, there was no database user or password specified for this connection. Applications themselves must specify the credentials when using this DSN. This is a security precaution.

Database account was created first on the devdb11 database server, which is a database used for development purposes. For the production setup the cerndb1 database is used, as included in the configuration above. Cerndb1 is a redundant database setup that provides high availability and failover to its users.

Files “odbc.ini” and “odbcinst.ini” can be found in the attachment [4 UnixODBC configuration files].

The periodical download of “/etc/tnsnames.ora” file requires a configuration script in the “/etc/sysconfig/” folder. This file holds information where the tnsnames.ora file can be downloaded from. The download process itself is made by a
“/etc/cron.hourly/tnsnames-update.cron” file which is installed automatically with the RPM package. The complete configuration file for the cron script can be found in the attachment [5 Cron script for updating the TNS oracle aliases].

5.3.2 Database schema

Kamailio required several database tables for saving data. There was a script shipped with Kamailio installation for every major database system, including Oracle. From this script only tables necessary for the system were extracted.

These tables were:

- **version** - necessary for Kamailio in order to verify table versions (correct tables must be loaded for the used version of Kamailio),
- **subscriber** - table with information about local user accounts,
- **dialog** and **dialog_vars** - these two tables hold information about ongoing calls,
- **dispatcher** - holds information about configured Media server,
- **location** - holds information about registered users.

There were also several other tables created. Their purpose is described later throughout the text. These tables include:

- **subscriber_allowed**,  
- **permissions_rules**,  
- **pinauth**.

The SQL script that creates the whole database schema can be found in the attachment [6 SQL script for schema creation].

Several functions were included in the script, such as “auto_id()” function that performs auto-incrementation of row IDs.

5.4 Kamailio installation

Kamailio could have been installed in two ways. The first one was to build Kamailio from source code downloaded from its GIT repository. The second one was to install RPM packages. The difference between them is time it takes (RPMs takes less) and availability of modules. With building of Kamailio from GIT, we have full control of what is going to be installed. RPM packages not support the full module list (although it is still sufficient for production). The configuration process after installation differs in mainly in files paths.
5.4.1 Kamailio installation from GIT

GIT is a source code management system capable of tracking changes. Kamailio’s GIT repository is available at “git://git.sip-router.org/sip-router”. We needed to install several packages in order to be able to build Kamailio.

```bash
# yum install git git-core gcc gcc-c++ autoconf automake flex bison \
> which libmysqlclient-dev make openssl libcurl libxml2 \
> openldap-devel libxml2-devel python-devel net-snmp net-snmp-libs net-snmp-utils net-snmp-devel pcres-devel mysql-devel \
> libstdc++-devel
```

Kamailio was a fork project of OpenSER and even with current source code merging of these two applications, we still had to specify that we wanted to pull Kamailio’s sources from the server.

```bash
# git clone --depth 1 git://git.sip-router.org/sip-router kamailio
# cd kamailio
# git checkout -b 4.0 origin/4.0
```

The build configuration was made by executing the following command.

```bash
# make PREFIX="/usr/local/kamailio-devel" cfg
```

This command created the “modules.lst” file which included the variable “include_modules” holding the list of modules we wanted to build. The list was empty by default and thus we needed to specify the modules by editing it. Modules included were the ones described in software selection chapter, subchapter 4.5.1 Proxy server.

The build and installation processes were then started by executing these two commands.

```bash
# make all && make install
```

As specified earlier by the “prefix” parameter of make script, everything was installed within the folder “/usr/local/kamailio-devel”.

After the installation we created a symbolic link for the “kamcmd” command to be available from shell. This command is used by FreeSwitch, as explained later in chapter 5.7.1 FreeSwitch Lua scripts.

We created the user and a group under which Kamailio was supposed to run. A directory “/var/run/kamailio/” for holding the process id file for Kamailio needed to be
created as well, as it wasn’t part of the installation. This folder had to be owned by the user under who Kamailio was supposed to run. The same applied for the Kamailio root folder.

The startup script was placed in the “/etc/init.d” folder. The default script was modified to export several environment variables before starting Kamailio.

```
export NLS_DATE_FORMAT="YYYY-MM-DD HH24:MI:SS"
export LANG=en_US.UTF-8
```

These variables were needed to set parameters for the UnixODBC connection, namely time format and character encoding used in queries.

Other functions were added into this script and the changes are described throughout the Kamailio configuration chapters. The complete file can be found in the attachment [7 Kamailio init.d script for GIT installation].

## 5.4.2 Kamailio installation from RPMs

The RPM packages were downloaded from OpenSuse website. Their dependencies were assembled from YUM repositories using the “yumdownloader” utility.

Kamailio packages downloaded were kamailio-4.0.0, with module packages named kamailio-*-4.0.0 (where * is ldap, outbound, snmpstats, tls, unixodbc and utils).

For their dependencies the packages lm_sensors-libs-3.1.1, net-snmp-5.5, net-snmp-libs-5.5, net-snmp-utils-5.5, pyodbc-2.1.7 were downloaded as well.

Once all packages were in place, we installed them by “rpm” command.

```
# rpm -i kamailio-* lm_sensors-libs* net-snmp* pyodbc*
```

The *kamailio* user was created automatically during the installation process and the only thing that remained was to add the variable exporting into the “/etc/init.d/kamailio” script, same as described in the previous chapter. This file can be found in the attachment [8 Kamailio init.d script for RPM installation].

The configuration files were installed into the “/etc/kamailio/” folder. Binary files were placed into the “/usr/sbin/” folder and modules into the “/usr/lib/kamailio/modules/” folder.

### 5.5 Kamailio configuration

The following configuration remarks in this and the following chapters apply both to Kamailio installed from GIT and for Kamailio installed from RPMs. File paths to modules
and other configuration files for modules differ respectively as they were described in the previous two installation subchapters.

To allow Kamailio to run, the variable “RUN_KAMAILIO” in the file “/etc/default/kamailio” must have been set to value “yes”. Within this file there were several other variables configured, including the “USER” and “GROUP” variables (which must have been set to “kamailio”) and amount of memory allocated for running the process.

Kamailio is configured by a single configuration file (by default), although it can be split into several different files which are then referred to from the main one.

This file consists of several sections:

- **administrator defined values section**
  - here are the directives defined by the administrator to turn on/off features of the Proxy,

- **defined values section**
  - parameters and aliases are defined in this section,
  - these values can be later referred to from the routing logic,

- **global parameters section**
  - holds core proxy parameters, such as IP address to bind to, ports and transport protocols to listen on, logging facilities, etc.,

- **modules loading section**
  - directives telling Kamailio which modules to load during startup,

- **modules parameters section**
  - modules behaviour is modified here by setting their parameters,

- **routing logic section**
  - most important part of the configuration,
  - Kamailio’s functionality is configured in this section,
  - routing logic starts by receiving a SIP message, parsing its body and executing the main “request_route” function,
  - various customized routes can be executed from the main route, based on the information from SIP messages,
  - note: term “route” can be understood simply as a function (set of commands) within the routing logic (analogically to functions from any programming language).
The complete “kamailio.cfg” file can be found in the attachment [9 Main configuration file of Kamailio]. Complete set of configuration files (used in the following subchapters) for Kamailio can be found in the attachment [17 Complete set of configuration files for Kamailio build from GIT] for installation from GIT and in the attachment [18 Complete set of configuration files for Kamailio installed from RPMs] for installation from RPMs.

Next subchapters explain configuration of individual topics. Every change is made within the “kamailio.cfg” file, unless stated otherwise.

5.5.1 Core parameters configuration

Kamailio was configured to output its log messages to Linux logging facility called “LOG_LOCAL0”.

log_facility=LOG_LOCAL0

Linux log daemon (in our case “rsyslogd”) was configured to output messages from this facility to file “/var/log/kamailio.log”. The configuration was made within the “/etc/rsyslog.conf” file.

Server header to be sent in each Kamailio-generated SIP message was configured as follows.

server_header="Server: CERN_SIP_Proxy"

If sending this header was undesired, it could have been turned off by setting

server_signature=no

The alias parameter told Kamailio which domain to accept as its own. It is used when evaluating whether the SIP messages come from local subscribers.

alias="cern.ch"

Kamailio was configured to listen on TCP and UDP port for new, unencrypted connections. Another TCP port was configured for listening for new, encrypted connections. The bind address was configured to be the service IP address.

TLS protocol was enabled by setting the core parameters “tcp_enable=yes”, but further configuration was required (see subchapter 5.5.5 TLS module configuration).
5.5.2 UnixODBC module configuration

The key of Kamailio’s functionality was the database connectivity if we wanted to save dialog information (further in the configuration) or if we wanted to keep information about available Media servers. The database is also necessary if Kamailio acts as a registrar server.

The ODBC configuration can be tracked through the configuration file by watching changes enclosed by the “WITH_ODBC” directive, which was defined in the administrator defined values section.

#!define WITH_ODBC

In the defined values section, we specified the connection string, using unixodbc driver.

#!ifdef WITH_ODBC
  #!ifndef DBURL
  #!define DBURL
  "unixodbc://<<#DBUSER#>>:<<#DBPASSWD#>>@localhost/OracleSIPGatewayProduct ion"
  #!endif
  #!endif

Every piece of configuration relevant to usage of UnixODBC is enclosed within the “ifdef…endif” block. The “ifdef” directive checks if the string (in this case WITH_ODBC) is defined by the “define” directive. If it is, the configuration inside the block is taken into account. This applies for all directives used throughout the file.

We loaded the module, named “db_unixodbc”.

#!ifdef WITH_ODBC
  loadmodule "db_unixodbc.so"
  #!endif

That was all for UnixODBC database connectivity. Other modules then used the connection string DBURL to interact with the database.

5.5.3 Load balancing configuration

When balancing call load amongst multiple Media servers we needed to consider how to keep Kamailio updated about the availability of Media servers.

If eventually one or more media servers become unreachable (for example due to network or configuration error) Kamailio needs to know about this as soon as possible, in
order to stop routing calls to this failing destination. Availability needed to be checked periodically.

Basically there were two ways of doing the “keepalive” mechanism. The first was configuring the media servers to REGISTER to Kamailio instance and maintain the registration in a normal “registration expire” manner. Another approach was to “ping” the media servers from Kamailio using for example the OPTIONS request.

The configuration approach described in this chapter takes care of load balancing and failover providing for Media services. There is no load balancing deployed for Kamailio (that would require the second Service IP address) although the failover mechanism is ensured by the clustering software.

Kamailio offers two modules that can do load balancing:
- PATH module,
- DISPATCHER module.

The path module used “Path” headers to force the SIP message to traverse the selected Media server. This module offered a very basic functionality and did not provide any means of keepalive mechanism for the Media servers. When using this module it was possible to use the “registration based keepalive” mechanism. Before the message was routed to the destination, Kamailio needed to check whether the destination is “online”. Also, a way of distinguishing the media servers would have needed to be deployed. For example, it would have been a good practice to register every media server under different SIP URI to avoid the parallel forking behaviour. If there was a need to prefer one media server over others, that server could have registered with the “q” value in the contact header, indicating it had a higher priority (this behaviour is defined by RFC 3261). The Media server selection could then happen with this value taken into account.

The dispatcher module had a more powerful implementation. It offered load balancing capabilities (with several destination selection algorithms), rerouting the request if the original destination failed and also a keepalive mechanism that was fully automatic. This module has been configured and used in the setup.

The configuration this module in enclosed in blocks bordered by directive “WITH_LOADBALANCE”.

The module “dispatcher.so” was loaded into configuration and several parameters were configured. The “flags” parameter was a two bit mask that influenced the behaviour of the module. Flag value equal to two configured the module to store all possible destinations in a list variable. If the selected destination failed, next one could have been
selected from the list. This, in fact, enabled failover. We configured the module to ping the Media servers every 20 seconds.

Probing mode indicated what happened in case the Media server failed to respond to the keepalive message. Value “1” meant that all destinations were being probed if they failed. Probing means keeping the keepalive mechanism ongoing but in a much more frequent scale, enabling the Proxy to see the gateway as soon as it becomes online again. By default, this was done only for destinations loaded from the database table with the probing flag set. Probing threshold meant how many keepalive messages could have failed before the destination was considered down.

Parameter “ds_ping_reply_codes” was very important for specifying how the Media server could have responded to the keepalive while not being considered down. For example if the server was overloaded, it could have replied with the “480 Temporarily unavailable” message. In this case the destination should have not been considered down. The proxy simply needed to try other destinations from the set, if there were any. We specified that the correct codes to reply were 404, 480 and the whole class 200 (from 200 to 299).

The routing logic was modified by disabling the default “route(location)”. This route normally takes care of looking up the IP address of the destination user. Since this behaviour was not desired and all calls were destined for Media server, this route was replaced by the new “route(loadbalance)”.

The new route itself was written as follows.

```c
#define WITH_LOADBALANCE
route[LOADBALANCE] {  # round robin dispatching on gateways group '0'
  prefix("pbx-");  
  if(!ds_select_dst("0", "4")
  {    
      send_reply("404", "No destination");    
      exit;
  }
  t_set_fr(0,2000);  
  t_on_failure("MANAGE_FAILURE");  
  return;
}
#endif
```

When a call was permitted to be established and was supposed to be relayed to FreeSwitch, the prefix “pbx-” was added to the request URI. This helped FreeSwitch to determine what should be made with the call.
The “ds_select_dst()” function selected a FreeSwitch from server set called “0”. The algorithm doing the load balancing was a Round-robin (number four) technique. This was the simplest one which required the lowest computing overhead. If no destination was available, error message was sent back to caller and script executing ended. If a destination was picked from the server set, the destination was expected to answer within two seconds. If this didn’t happen the failure route “MANAGE_FAILURE” was executed.

The failed FreeSwitch was marked inactive (i) and was put into the probing state (p). If new destination was selected, again the two seconds reply time was specified as well as the failure route (the same one, creating a loop until an available server was found). After that the request was immediately forwarded by calling the “route(RELAY)”. If all destination were inactive and even the last selected FreeSwitch failed to answer, an error message was send back to the caller and script exited.

Two FreeSwitch servers were added into the database. Kamailio load information about them upon startup. They were specified to be members of server set called “0”, as specified earlier in the configuration.

Their availability could have been verified by using the administrator script “SIPuser”.

### 5.5.4 Dialog module configuration

As mentioned in the previous chapters, we needed the information about ongoing dialogs saved into the database to provide failover by the Proxy server.

The “dialog” module was used for such purpose. The configuration can be tracked within the file by following the defined directive “WITH_DIALOG”.

A flag was defined that helped Kamailio core to track the messages belonging into the same dialog. Kamailio keeps information about ongoing transactions and when it sees this flag set within its inner data structures it knows that it is keeping track about the whole dialog and should treat the request respectively with the dialog state taken into account.

Kamailio was configured to load the module upon startup with several parameters to save the information about ongoing calls to the database (most importantly by parameter “db_mode” set to value “1”). Dialog information was configured to expire after thirty minutes (the same value as in FreeSwitch, as describer later in chapter 5.7 FreeSwitch configuration).
In the routing logic, in the main routing block, we called the function “dlg_manage()” which sets the dialog flag when processing a dialog-forming request and later maps any consequent request to the existing dialog.

5.5.5 TLS module configuration

TLS provides encrypted transport of SIP messages. The configuration of “tls” module was made within the file and can be tracked by following the directive “WITH_TLS”.

The TLS configuration is part of the default configuration file and no changes were necessary. The only thing that needed to be adjusted was the path to the TLS configuration file, which was placed in the “/etc/kamailio/tls.cfg”.

This file holds parameters telling Kamailio how the TLS connections need to be established.

We specified that we wanted to use TLS version 1. The server did not require a certificate from the client, but the client needed to verify the server certificate. Kamailio was required to present the certificate during TLS communication establishment. The complete “tls.cfg” file can be found in the attachment [10 TLS configuration file of Kamailio].

In order for clients to be able to verify Kamailio’s certificate, it needed to be signed by a trusted commercial certification authority. CERN is a certification authority, but not a trusted one outside CERN therefore a CERN certificate is not sufficient. This is similar for HTTP servers which are using certificates not signed by CERN.

5.5.6 SNMP module configuration

The “snmp” module of Kamailio is not a fully featured SNMP client. Instead it acts as subagent gathering information about the application. It requires the external snmp daemon to run on the Linux machine. Package “net-snmp” which was installed alongside with Kamailio provides such daemon. This application needs to act as so called AgentX which is a name for subagent interfaces, such as the one of Kamailio.

The default “/etc/init.d/snmpd” script used for starting the service was modified. The start options were extended by adding “-x localhost:705” in the end. The “-x” switch told snmpd to listen on the specified socket for AgentX connections.

The files “/etc/init.d/snmpd” can be found in the attachment [11 SNMPd init.d script].
The main configuration file of snmpd is file "/etc/snmpd/snmpd.conf". Several changes were required in this file, such as specifying the IP address of the CERN trap server, community string, SNMP protocol version and enabling the AgentX support.

The complete “snmpd.conf” file can be found in the attachment [12 SNMPd configuration files].

In the end the file “/etc/snmpd/snmpstats.conf” file was created and filled with the information where the AgentX socket was available (localhost socket, TCP port 705). This file can also be found in the same attachment.

In the kamailio.cfg file we defined a directive to enclose SNMP configuration, “WITH_SNMP”. We loaded the module and again, configured its parameters.

The SNMP community string was specified as well in order to allow Kamailio to properly send SNMP traps when the thresholds were exceeded. Threshold for number of active dialogs was configured to value fifty (minor alert) and to value fifty-five (major alert).

The MIB (Message Information Base) needed to be loaded into the monitoring software in order to interpret traps sent by Kamailio and based on the traps to define alarms.

5.5.7 Permissions module configuration

It was one of the main requirements to control the numbers which can be called from outside CERN’s telephone network by VoIP. Several destinations, such as conference services (Vydio and MyTeamWork), CERN Fire brigade or CERN Service desk were not supposed to be reachable.

It was essential to deploy a mechanism to control the numbering scheme and write down rules where users were/weren't allowed calling. This would enable the administrator to easily change the rules if CERN’s numbering scheme changes.

There is a module for Kamailio SIP server, called “permissions“, which enables Kamailio to check caller and callee info. Based on this info, Kamailio is able to decide if the call can be routed or not.

The configuration is enclosed within blocks bordered by “WITH_PERMISSIONS” directive.

We configured two module parameters telling the module where to look for files in which permissions rules were stored. The file path was “/etc/kamailio/rules.allow|deny”.
The example files are included in the attachment [13 Permission rules files of Kamailio] with description of the rules.

A route was created within the routing logic and called right after routing in-dialog request. This meant that any other dialog-forming request was evaluated against the permissions rules.

If the request was accepted, the routing continued. In other case, the routing process ended and error message was sent back to the caller.

The check was performed by the “allow_routing()” function provided by the permissions module. This function returned value “true” if the routing was allowed.

The rules are stored in files on hard drive. Their structure and syntax is similar to other access management files in Linux systems, such as “/etc/hosts” file.

The complete syntax of a rule (both allow and deny rule) was

```
from_uri_request [EXCEPT from_list] : request_uri_regex [EXCEPT request_list]
```

The “EXCEPT” parts were optional. The keyword “ALL” matched all URIs.

Since the system includes two proxy servers we needed to synchronize these rules across cluster nodes. This could have been done in several ways.

One option was to store the contents of the allow and deny files in the database. All that was required was a single table that would hold two rows (one for allow file and one for deny file). There are ways how to store text files in different database systems, like for example CLOB data type in Oracle. Once the file contents are in database, all that remains is to write a script that will pull the data and save them in the local files on HDD.

This could have been done in many ways. Kamailio is capable of calling Lua scripting language. This language has resources of handling DB query results and writing to local files (using modules “SQLops”, “App_Lua” and “Rtimer”). Using Lua would mean that Kamailio itself would be responsible for downloading the files periodically. If Kamailio fails, files are not refreshed. If we wanted to keep it out of Kamailio, there was a possibility to call Perl or Python scripts directly from Linux. Calling Lua script was not possible because there were no RPM packages available for UnixODBC driver for LuaSQL library (Lua database connectivity library).

The drawback of this solution was the fact that the whole content of the file would be stored as one row of the database table. If a rule was being edited, the whole row would need to be replaced by a new one.
From the structure of the files we could have seen that it would be better to store the rules in database on “1 row = 1 rule” basis. This would enable the administrator to add, remove or edit a single rule. The database can be accessed from many different systems, so creation of various interfaces for editing the rules is an easy task.

We created a database table “permissions_rules” for holding the rules. The “rule” column holds the body of the table and the column “type” tells whether the rule is of type “allow” or “deny”. Type can only have value “A” or “D”.

CREATE TABLE permissions_rules (
    type VARCHAR2(1) default '' CONSTRAINT check_type CHECK (type IN ('A','D')) NOT NULL,
    rule VARCHAR2(300) NOT NULL,
    last_change DATE NOT NULL
);

A script in Python language has been written to connect to the database using UnixODBC driver, download the rules and construct files on HDD. This script is called by a shell script that checks its return value. If the Python script exits successfully it checks whether Kamailio is currently running on local node. If it is the script restarts Kamailio in order for the new rules to take immediate effect.

Furthermore, Kamailio downloads the permission rules every time it starts, ensuring the most new version or permission rules are always in the running configuration.

For this, the “/etc/init.d/kamailio” script has been modified. The function “download_perm_rules()” has been added and is called right before the “start()” function is called.

The files “permissions_python.py” and “permissions_script.sh” can be found in the attachment [14 Scripts for periodic downloading of permission rules]. Both files need to belong to user kamailio.

The “rules.allow” and “rules.deny” files must exist on the disk before the python script creates their content. We created them using the “touch” command.

A cron task was created in order to periodically download the files from the database.

```
0 7,19 * * * /etc/kamailio/permissions_script.sh
```

This task calls the permissions_script.sh every day at 7:00 and 19:00. The idea was to download the rules and restart Kamailio during the period of lowest system utilization. The times can be easily changed by “crontab”.
5.5.8 Attack protection configuration

It was crucial to deploy attack protection mechanisms at Proxy level. The reasons were the security, denial of service protection and stability of Media servers by protecting them from unnecessary load.

The idea was to deploy an IP based attack protection. Message rate based protection was not desired as it would limit all users, with no respect if they are a source of flooding or not. Limiting traffic on per-user basis required a lot of computing overhead and a very delicate configuration. If it was deployed it might have caused users with for example presence enabled to be blocked. Presence related SIP messages are used for many events and if the system had taken them into account while utilizing very strict limits, there might not have been enough space for call-related messages.

Deploying an IP based attack protection enabled us to deploy time-based limits by using Kamailio’s "pike" module. This module constructs an inner data structure, supposedly a tree, where the nodes are bytes of the IP address. This enables the module to quickly search the tree and find out whether the IP address is already banned. The data structure itself could have been used in a form of a hash table, provided by the "htable" module. Using this module even increased the performance because pike was no longer required to search the whole IP address tree. The search was provided simply by looking into the hash table. With larger key usage there was a lower number of collisions.

The configuration was initialized by defining the "WITH_ANTIFLOOD" directive. This directive is even preconfigured in the default Kamailio’s routing logic. We loaded both modules and configured their parameters, based on the level of desired strictness of attack protection.

```c
#ifdef WITH_ANTIFLOOD
loadmodule "htable.so"
loadmodule "pike.so"

modparam("pike", "sampling_time_unit", 2)
modparam("pike", "reqs_density_per_unit", 16)
modparam("pike", "remove_latency", 4)

modparam("htable", "htable", "ipban=>size=8;autoexpire=300;")
modparam("htable", "htable", "authen=>size=10;autoexpire=920;")
#endif
```

We set pike to check the number of messages per IP as frequently as possible. The sampling period was configured to two seconds. Configuring time less than two seconds
was not recommended by Kamailio developers, as with higher traffic load there was respectively a higher computing resources demand.

For every sampling period there were only a specified number of messages allowed originating from one IP address (in this case sixteen). This meant that within a single second there were approximately eight messages allowed from a single IP. The message count required is four messages for one call establishment. This means if two calls per second are trying to be established, the IP address will get banned for 5 minutes. This time is a period in which the record about the IP address is kept in the hash table, named “ipban”. The second hash table “authen” was used for password cracking protection if the server was acting as a registrar server. This is explained in the last paragraph of this chapter.

In the very beginning of the routing logic script, there was a part added which called pike’s “pike_check_req()” function. This function verifies whether the IP address is already banned. If it was, the routing exited with no error response to the caller. The IP address was not banned, pike increased the message count for the source IP within the sampling period. After the period was over, the record was automatically erased.

If a new address was detected as a source of flooding, a record was saved into the hash table.

Furthermore if the flooding was detected for a single IP address 3 times in a row within the interval of twenty minutes, the IP address got banned for thirty minutes at firewall level. This feature was provided by installing a “fail2ban” application. Fail2ban monitors logging files within the Linux environment and if the specified pattern was matched several times within a configured period of time, it added a rule to iptables to block to source of the attack.

In the “/etc/fail2ban/jail.conf” file we, created a so called “jail” for Kamailio. We configured the jail to watch the logging file “/var/log/kamailioIPban.log”. If the jail was to be executed, all ports on all protocols were blocked.

The regular expression for pattern matching in the file was configured as “Message blocked from <HOST>”.

Both files “/etc/fail2ban/jail.conf” and “/etc/fail2ban/filter.d/kamailio.conf” can be found in the attachment [15 Configuration files for Fail2ban].

In the “kamailio.cfg” file we added a directive “WITH_ANTIFLOOD_WITH_IPTABLES”. Tracking this directive the only change to
the routing logic can be found, which was logging the IP address with the above-specified format into log facility “LOG_LOCAL1”. This was different than the default one.

This log facility was configured in the “/etc/rsyslog.conf” to print the messages into the “/var/log/kamailioIPban.log” file.

The password protection mechanism has been prepared to be used if Kamailio would act as a registrar server.

The mechanism was deployed by enabling the user to try to authenticate 3 times within a period of one minute. If this limit was exceeded, the user was considered as a source of password cracking attack. A record was saved into the “authen” hash table. It expired after 15 minutes and the user was again able to try to authenticate. All changes enabling this mechanism were made within the “route(AUTH)” routing block.

5.5.9 LDAP configuration

The LDAP service deployed at CERN enabled us to provide users the ability to dial CERN phones by names of people to whom they belong to. This is an advantage because the users do not need to remember phone numbers and format how they can be dialled. Instead the user can just dial the callee’s address, which is the same as his email address. Every CERN member of the personnel or associated member of the personnel has a phone number linked to his account (usually an office phone number, with ability to add a mobile phone number). These numbers are in most cases included in full form and (if available) also in CERN shortened form.

The names of LDAP attributes are:

- Mobile,
- otherMobile,
- TelephoneNumber,
- otherTelephone.

The names beginning with “other” are the ones in full form. The order in which they are written above is also the order of their preference. This resulted from the assumption that if the CERN employee specified his mobile number in the CERN phonebook, he prefers to be reachable by mobile phone.

The configuration started by defining the “WITH_LDAP” directive and loading the module into configuration. The module parameter was configured with the value of file path, where the LDAP access configuration was stored.
Within the file “/etc/kamailio/ldap.ini” several parameters were set, such as LDAP URL and distinguished name and password for the connection. The complete “ldap.ini” file can be found in the attachment [16 LDAP configuration file for Kamailio].

In the routing logic we created a new route, named “route(ldap)”. Calling the route was placed right before dispatching the call to FreeSwitch.

The newly created route first performed the check whether the request URI contains the “.” sign. If not, the request URI was a number (ensured by permissions module) and the routing continued without any change.

If yes, LDAP search was performed, using the “ldapcern” alias specified in the “ldap.ini” file. If it was unsuccessful and a number was not found, Kamailio replied with 404 error and exited the routing script. If a number or numbers were found they were used as described above. The request URI was overwritten and the call was relayed to FreeSwitch.

5.5.10 Configuration of miscellaneous parameters

Kamailio was reporting an error every time it was shutting down. This error resulted from insufficient rights to access the “/tmp/kamailio_fifo” file. This file is created during Kamailio startup before the script switches to user kamailio. The file thus belonged to user root and Kamailio was not able to properly delete it. This was fixed by configuring the user parameter of “ctl” module, which forced Kamailio to create the file with ownership of the correct user.

Since the system was supposed to serve as an open proxy for internet connections, no user registrations are necessary. Kamailio does not need to keep track of reachability of the system users. Necessary dialog information (including IP addresses and port numbers) was saved real-time into the database. Once the call is over, the information is no longer necessary and is erased. Furthermore, if users from other domains are calling the system, Kamailio can reach their local SIP proxy servers (therefore it does not have any need to communicate with the client directly).

The ability to act as a registrar server is built-in into Kamailio’s configuration file and can be tracked by following the directive “WITH_USRLOCDB”. When enabled it save the location information into database. It is configured to store only one AOR per contact in order to prevent unnecessary information in the database. This was also very useful for mobile clients using websocket connections, which is explained in the following paragraphs (every time the webpage was refreshed, new AOR was created).
Websockets and “Web-Real-Time Communications” (WebRTC) are two new technologies intended to be used within the web browsers, enabling web pages to implement real-time communications. Websocket draft is a new protocol that runs over TCP, enabling the webpage to establish connection to remote server (other than web server). It generally takes care about signalling layer and enables SIP to be transported over it. WebRTC is a draft implemented by Google, which takes care about the media once the call is established.

NAT is a major problem for SIP as the client needs to be reachable by the server at anytime. There is a new mechanism, called Outbound that takes care of keeping the connection between the client and the server constantly opened, using a TCP keepalive mechanism. Outbound was inspired by similar mechanisms already used in other protocols, such as “eXtensible Messaging and Presence Protocol” (XMPP). The principle of Outbound is that the client is responsible of keeping the connection alive and therefore the server does not need to implement any kind of NAT pinging or other NAT traversal mechanisms.

Kamailio in its newest stable release already supports Websockets. The “outbound” module currently offers only functions for developers as an interface for other modules. Its functionality can, however, be replaced by the “nathelper” module which was extended to recognize websocket connections. The “websocket” module together with “nathelper” module was tested and the configuration file is prepared for future development implementing these new standards. The configuration has even been extended to support the web-based SIP client that would be developed in the future. The setup was tested with “jssip” JavaScript SIP library.

Kamailio was not able to integrate its authentication services with CERN’s mechanisms. However there was a workaround created for the web client. If the user authenticates with his CERN computing account when opening the webpage, the server-side script needs to write his authentication token (or Shibboleth authentication cookie) into the database (table “subscriber.allowed”). When registering, the cookie value is sent to Kamailio which then compares it with the value in the database. This way, Kamailio can be sure that the websocket connection is coming from the CERN webpage and can authenticate the messages without any other kind of password protection. The communication between the browser and Kamailio is encrypted by TLS, so no man in the middle attacks are possible if Kamailio is using a certificate issued by a trusted certification authority. This configuration can be tracked within the “kamailio.cfg” file by
the directives “WITH_WEBSOCKET”, “WITH_SQLOPS” and “WITH_NAT_SUPPORT”.

By the default setup in “kamailio.cfg” file, the routing logic can be exploited to completely bypass the authentication section by preloading a SIP request with set of “Route” headers and including a “to_tag” in the message. This causes Kamailio to treat the request as it was within an existing dialog, presuming the authentication already took place. This was fixed by using a function of the dialog module “is_known_dlg()” which checks whether the request belongs into a known dialog. If no, error was sent back to the user and the request was discarded.

5.6 FreeSwitch installation

Similar to Kamailio, there were two options how to install FreeSwitch. One was to get the recent source code of the stable version from GIT repository and building it ourselves. This option would take more time than the second one which is installing FreeSwitch from pre-compiled RPM packages. On the other hand it would offer us more flexibility in customizing the code if needed.

Both options are described in the two following chapters.

5.6.1 FreeSwitch installation from GIT

FreeSwitch’s GIT repository is available at “git://git.freeswitch.org”.

Prior to downloading the sources and compiling them we needed to install dependencies.

```bash
# yum gcc gcc-c++ make gettext-devel expat-devel curl-devel gawk zlib \
> zlib-devel openssl-devel bzip2 readline-devel libpcap-devel git \
> git-core autoconf automake libjpeg-turbo-devel libtool ncurses-devel \
> pkgconfig libogg-devel libvorbis-devel libtiff-devel which flex bison \
> lua-devel wget
```

After this we approached pulling the source code from GIT.

```bash
# git clone -b v1.2.stable git://git.freeswitch.org/freeswitch.git \
...[output omitted]... 
# cd freeswitch
```

This initialized a complete source tree within the newly (and automatically) created directory “freeswitch/”. After changing to this directory we fired a command to replace the
default database connectivity keepalive string. For the reason to do so see the last two paragraphs of chapter 5.7.1 FreeSwitch Lua scripts.

```
# sed -i 's|strcpy((char \*) sql, \"select 1\")| 
> strcpy((char \*) sql, \"select 1 from dual\")|' src/switch_odbc.c
```

After this we started a set of commands to build and install FreeSwitch.

```
# ./bootstrap.sh && ./configure --prefix=/usr/local/freeswitch
# make && make install
# make all cd-sounds-install cd-moh-install
```

Setting the prefix variable with the “configure” script resulted in FreeSwitch being installed in the specified directory. All binary files were therefore placed into this directory along with sound files and module libraries. Also configuration files must have been placed into this directory (subdirectory “conf/”) as well as script files (subdirectory “scripts/”).

After the installation was complete we created a system user “freeswitch” under which FreeSwitch would run. The prefix directory was set to be owned by this user. This user was supposed to belong to the “daemon” group.

As the compilation has not created the “/etc/init.d/” startup script, we had to create it on our own. This script is similar as the one used with RPM installation. The only difference is in the file paths, which are customized to match the prefix directory. The complete script can be found in the attachment [19 FreeSwitch init.d script for GIT installation].

### 5.6.2 FreeSwitch installation from RPMs

The RPM packages of FreeSwitch are hosted on the OpenSuse web server. We downloaded this list of packages using the “wget” command:

- core application RPM freeswitch-1.2.8-1.el6.i386.rpm,
- codec modules named as freeswitch-codec-*-1.2.8-1.el6.i386.rpm,
- application module packages named freeswitch-application-*-1.2.8-1.el6.i386.rpm, where “*” is db, expr, fifo, hash, limit, valet_parking and httpapi,
- sound packages of English language,
- other packages:
  - freeswitch-format-local-stream-1.2.8-1.el6.i386.rpm,
  - freeswitch-lang-en-1.2.8-1.el6.i386.rpm,
FreeSwitch-lua-1.2.8-1.el6.i386.rpm.

FreeSwitch requires “sox” utility to be installed on the PC before its installation. This command is used by FreeSwitch to generate sound files. List of sox packages and its dependencies is sox-14.2.0-6, gsm-1.0.13-4, libao-0.8.8-7 and libsamplerate-0.1.7-2.

These packages were also downloaded and installed from yum repositories. The installation was done by executing a command:

```
# rpm -i freeswitch-*
```

The configuration files were placed in the “/etc/freeswitch” directory, while sounds and scripts were placed into the “/usr/share/freeswitch” directory. Binary files were placed into the standard Linux directories containing majority of binaries, “/usr/bin/”.

The system user was in this case created automatically as well as the “/etc/init.d” script. Again however, this script needed to be modified a bit in order to be usable by the clustering software. Arguments “-nonat -ncwait” were added to be used when starting the FreeSwitch process.

These arguments tell FreeSwitch to disable automatic resolving of external (or public) IP address. As mentioned above, the external IP address was the same as the bind address, meaning this mechanism was not necessary. The second argument forced FreeSwitch not to exit the init.d script before it was fully started (see chapter 5.8.2 Cluster information base configuration, OCF scripts part for further explanation). Function to determine the status of FreeSwitch was also added to this script. For complete script see the attachment [20 FreeSwitch init.d script for RPM installation].

### 5.7 FreeSwitch configuration

FreeSwitch configuration is done by a set of XML-formatted configuration files. These files do not include any relative or absolute file path and therefore the following configuration could have been used both for FreeSwitch compiled from source code and FreeSwitch installed from RPM packages.

When FreeSwitch starts the running configuration is built as it is read from the tree of XML-files. The root of this tree is file “freeswitch.xml”. Every file that we wanted to use must have been included within this file of any other file that was included here. Including files was done by specifying “include” directives within the xml.

```
<X-PRE-PROCESS cmd="include" data="vars.xml"/>
```
File “vars.xml” was one of the most important configuration files for FreeSwitch. This file holds most of the parameters, such as domain name, codecs sets, IP address bindings, and most importantly the call profile used for incoming calls.

```xml
<X-PRE-PROCESS cmd="set" data="domain=cern.ch"/>
<X-PRE-PROCESS cmd="set" data="use_profile=internal"/>
<X-PRE-PROCESS cmd="set" data="bind_server_ip=<<#LOCALIP#>>"/>
<X-PRE-PROCESS cmd="set" data="external_rtp_ip=<<#LOCALIP#>>"/>
<X-PRE-PROCESS cmd="set" data="external_sip_ip=<<#LOCALIP#>>"/>
<X-PRE-PROCESS cmd="set" data="zrtp_secure_media=true"/>
```

With this configuration we specified, the domain name which is used in all outgoing messages generated by FreeSwitch. There is also a bind_server_IP parameter telling FreeSwitch on which interface to bind. The external_rtp_ip and external_sip_ip are significant when deploying FreeSwitch on a private network, while keeping it reachable from the public internet (FreeSwitch keeps NAT mapping opened at the router doing the NAT). In our case these two parameters were the same as the bind_server_ip parameter. The last parameter zrtp_secure_media enabled FreeSwitch to signal support for ZRTP which is a more user-agent friendly way of negotiating private keys for RTP encryption. When ZRTP is used, the keys are transmitted in clear text format in SDP within the SIP INVITE message. Therefore it is necessary to use SIP over TLS when using ZRTP.

The folder “autoload_configs” contains configuration files for modules loaded into FreeSwitch on startup. File “modules.conf.xml” contains a list of modules that are enabled. Mod_sofia is the SIP stack module for FreeSwitch. This module is in fact the reason why FreeSwitch is capable of speaking SIP protocol. We needed to enable several modules that were not included in the default configuration in order to be able to use advanced functions. Module mod_hash was used later in the dialplan to count the number of occupied channels and thus watching that the quota is not exceeded. Mod_lua is a language module that provides a call API for Lua language, enabling us to write fully customized voice applications. Two modules, mod_local_stream and mod_tone_stream, were enabled in order to extend FreeSwitch with the ability to stream audio files over RPT and generate and receive DTMF tones.

On the other hand, several modules were disabled, because we didn’t need their functionality, such as mod_conference or mod_voicemail. Mod_directory was also disabled because we didn’t want to use FreeSwitch as a registrar server. Directory normally holds user accounts and their parameters within FreeSwitch.
The important part of FreeSwitch configuration was to set an access list. This access list specifies that the only accepted incoming and dialog-forming SIP messages are those originating from Kamailio (specifically the service IP address). This access list was configured within the file “autoload_configs/acl.conf.xml”.

Configuring an access list enabled second-level protection of FreeSwitch from processing undesired SIP messages. The first level was at firewall level.

In the configuration file of SIP stack, “autoload_configs/sofia.conf.xml” we enabled only internal profile. Everything else was commented out.

```xml
<profiles>
  <X-PRE-PROCESS cmd="include" data="../sip_profiles/internal.xml"/>
</profiles>
```

In the file “autoload_configs/switch.conf.xml” we configured of local RTP port range available to values starting from 40000 to 41000.

The contents of the “sip-profiles/” directory was completely erased, except for the “internal.xml” profile file. All inbound call from Kamailio were assigned into this profile and processed within the configured “SBCwithKamailio” dialplan context, which was newly created.

The username and user-agent-string are values sent in the SIP and SDP messages. These parameters were changed for two reasons. The general public should not be aware of the software we are using and the SIP fingerprint scanners wouldn’t be able to use these values as a great help when determining the type of server. The port FreeSwitch will listen on for this profile binding was changed to 5080. This is the only port number reserved for FreeSwitch and the service port must have been different. The IP addresses where to bind the application were determined from the abovementioned configuration of “vars.xml” file.

The final element to configure was to create a “SBCwithKamailio” dialplan which was used for processing incoming calls within the internal profile. This dialplan was placed into a separate file “dialplan/SBCwithKamailio.xml”. Dialplans in general consist of extensions to which the call is fitting based on conditions.

```xml
<extension name="ip2pbx">
  <condition field="destination_number" expression="^pbx-.*$">
    <action application="set" data="call_timeout=50"/>
    <action application="set" data="continue_on_fail=false"/>
    <action application="set" data="hangup_after_bridge=true"/>
    <action application="set" data="sip_invite_domain=sip<<#PBX_IP#>>"/>
  </condition>
</extension>
```
<action application="export"
        data="sip_contact_user=${sip_from_user}"/>
<action application="set" data="sip_cid_type=none"/>
<action application="set" data="execute_on_answer=sched_hangup +1800
        allotted_timeout"/>
<action application="limit_execute"
data="hash max_call_num pbx 30 lua LUAIVR_M.lua
        ${sip_from_user}
        ${sip_from_host}
        ${destination_number}"/>
<action application="transfer" data="max-limit"/>
</condition>
</extension>

<extension name="max_limit">
  <condition>
    <action application="playback"
data="tone_stream://%(500,500,480,620)"/>
    <action application="playback"
data="tone_stream://%(500,500,480,620)"/>
    <action application="playback"
data="tone_stream://%(500,500,480,620)"/>
    <action application="playback"
data="tone_stream://%(500,500,480,620)"/>
    <action application="hangup"/>
  </condition>
</extension>

We created two extensions: ip2pbx and max_limit. The ip2pbx extension is fired for every call where destination number is in format of “pbx-*”, where sign “*” means any phone number. The “pbx-” prefix is added by Kamailio once the call is evaluated as allowed. By this prefix FreeSwitch knows that it can process the call. There are several parameters set on the leg A of the call (user ∞ FreeSwitch), telling FreeSwitch how to treat the call before the Lua script is fired, such as hanging up the call leg once the bridge has finished or setting the maximum call length to 30 minutes (1800 seconds), counting from point when the call is bridged.

The limit_execute application limits the number of concurrent calls. The limit is called max_call_num and the threshold is 30 calls. Once the Lua script is executed, it takes further care about both call legs, including hanging them up and existing the extension. If the Lua script fails to execute because of the concurrent call limit exceeding, the final application is executed from the extension, which transfers the call to the second extension, max_limit.

The second extension plays a busy tone back to the caller and hangs up the call.
This concluded the XML configuration part of FreeSwitch. The next thing to do was to write the voice applications that would have taken care of user authentication, creation of call leg B (FreeSwitch <-> PBX) and bridging it with call leg A. Complete set of configuration files for FreeSwitch can be found in the attachment [21 FreeSwitch configuration files]. There is a directory “conf/” which contains all xml files and directory “scripts/” which contains all Lua scripts described in the next chapter.

5.7.1 FreeSwitch Lua scripts

Lua is a simple procedural language capable of being interpreted on many platforms. FreeSwitch provides a powerful call API to its mod_lua module, enabling the script author to perform various tasks he wouldn’t be able to do from dialplan. Examples of such tasks are random numbers generation and external scripts execution or DTMF tones buffering (which is exactly what we expected from the selected language). The mod_lua is a very lightweight module (all in all approximately 850 KB) when considering all its features. This language is highly embeddable, meaning if the system would migrate to other platform in the future, the script would remain unchanged.

Furthermore FreeSwitch is capable of passing parameters to Lua scripts, which is extremely essential to build voice application capable of handling user authentication.

As seen in the previous chapter, in the dialplan configuration part, there is a command that fires a Lua script if there is a channel available. When the script is fired, the channel is reserved.

There are two Lua scripts included in the attachment [21 FreeSwitch configuration files]:

- LUAIVR.lua
  - this script requires users to register their URIs before calling the system,
  - WEB application generates PIN code for the user, who is prompted to input it using DTMF during call establishment (3 retries),
  - this script reads PIN code from a database table and compares the data with data gained by DTMF,
  - if the PIN code matches, dialled extension is verified against the Kamailio’s permissions rules,
  - if the extension is permitted, call leg B is created with CERN PBX and both legs are bridged,
  - otherwise call is ended.
• LUAIVR_M.lua
  - different from the previous script by not prompting the user for PIN code,
  - instead, this script generates two numbers and prompts the user for result of
    their addition (only one try, which makes the script to end faster and free
    the channel),
  - if the result is correct, the procedure is the same as above.

In the situation that the called user number is busy, the Lua script plays busy tone
back to the caller and hangs up.

The verification of validity of dialled extension is done using Kamailio’s “kamcmd”
utility. Lua script calls this utility in form of an external shell command. The kamcmd
utility is instructed to connect to service IP address (ensuring the running proxy with the
most new version of permissions rules is always reached) and its output is trimmed so that
only the result in form of “Allowed” or “Denied” words is left behind.

```
local handle = io.popen("kamcmd -v -s tcp:<<#SERVICEIP#>>:2046
permissions.testUri /etc/kamailio/rules
sip:"..req_user.."@"..req_domain.."
sip:"..dial_num."@cern.ch
| head -n 1")
local resAllowed = handle:read("*all")
handle:close()
if resAllowed=="Allowed
and dial_num~="42" then
  ...[call allowed, bridge to PBX]...
else
  ...[call denied, hang up]...
```

A condition checks whether the running Proxy server approved the call and whether
the extension 42 was not dialled. If it was, call is hanged up.

Database accessibility in the first script was done using the “system handle” creation
mechanism and calling the SIPuser script. The problem with FreeSwitch with accessing the
database by itself when installed from RPM packages is that there is a hard-coded database
keepalive query specified as “select 1”. However this is not compliant with Oracle
database system which requires keepalive query to be set to “select 1 from dual”.

However, if FreeSwitch was compiled from source code, this string could have been
changed before the compilation process took place and therefore FreeSwitch was able to
use the ODBC driver without the necessity to call SIPuser script. Instead it used the DBH
object provided to Lua script by the API. The code to use the DBH object is commented
out in the LUAIVR.lua file and is ready to replace the “handle” mechanism if FreeSwitch is compiled.

5.8 Clustering software

Once we had the Kamailio and FreeSwitch applications installed on both nodes we could advance to installing the clustering software. As mentioned in the previous chapters, we needed software that would have taken care of the messaging (or communication) layer of the cluster and software that would have managed shared resources. Corosync and Pacekamer were selected for these tasks and therefore we began by assembling all RPM packages needed to run the software.

Several dependencies had to be met in order to install Corosync and Pacemaker.

- Package “corosync-1.4.3” was dependent on:
  - corosync-lib-1.4.3, libibverbs-1.1.6, librdmacm-1.0.17, lm_sensors-libs-3.1.1, net-snmp-libs-1.5.5,

- package “pacemaker 1.1.8” was dependent on:
  - corosync-1.4.3, pacemaker-cli-1.1.8, pacemaker-cluster-libs-1.1.8, pacemaker-libs-1.1.8,

- the “resource-agents-3.9.2” package dependencies were:
  - several binary libraries, cifs-utils-4.8.1, cluster-glue-libs-1.0.5, keyutils-3.0.12, nfs-utils-0.2.1, quota-1.3.17, rpcbind-0.2.0, samba-common 3.6.9, tcp-wrappers-7.6.

Because we wanted to use the “crm shell” interface (standard command line interface for cluster configuration also used in other systems at CERN) and this interface is discontinued for the new Pacemaker versions (above 1.1.8), we needed to download the rpm packages manually. Every other package mentioned above has been acquired from yum repositories with the “yumdownloader” command in form of RPM packages.

Package “crmsh-1.2.5” was dependant only on package “pssh-2.3.1”. Both were downloaded from the OpenSuse website hosting many packages (including RPMs for Kamailio and FreeSwitch) for various Linux distributions, including RedHat EL6.

When approaching the installation process itself, the Corosync had to be installed before Pacemaker.

```
#rpm -i corosync-1.4.3-26.2.i686.rpm corosync-lib-1.4.3-26.2.i686.rpm
```

After successful installation of Corosync we installed Pacemaker.
The package ‘resource-agents’ is a set of OCF scripts which are explained later in the configuration section.

Once Corosync and Pacemaker were set up, we installed the command line interface and its dependency, the parallel secure shell.

With everything installed we ensured the services Corosync and Pacemaker services are started at boot time, while automatic starting of Kamailio and FreeSwitch were turned off. After the configuration they were started and stopped as needed by Pacemaker.

Since Pacemaker is using Corosync to communicate between cluster nodes, Corosync service must start before Pacemaker. This is ensured by service starting priority specified in the initialization scripts located in ‘/etc/rc.d/rcX.d/’ directories where X means the level on which the service needs to be started. Lower number in the filename means higher priority. The range is between 1 and 99.

That concluded the installation process and we advanced to the configuration section.

5.8.1 Cluster configuration

The initial thing to do was to configure Corosync with IP addresses of cluster nodes and way how they are reachable. Within the ‘/etc/corosync/corosync.conf’ file we specified the IP addresses of the cluster nodes and that they are reachable by UDP unicast. Subnet IP of the local interface told corosync to which interface to bind to.

There are several other parameters that could have been configured within the file, such as logging settings (log file location, information to output etc.).

The contents of this file needed to be identical amongst all cluster nodes. For full file see attachment [22 Configuration files for Corosync].

For activating the pacemaker support Corosync requires the file ‘/etc/corosync/service.d/pcmk’ to be created with specifying the Pacemaker version.

This file is included in the same attachment.
5.8.2 Cluster information base configuration

The “Cluster Information Base” (CIB) is a set of configuration directives documenting the cluster resources, their sharing and parameters, their desired state and intervals for standard operations like monitoring of the resource or restarting it. The purpose of creating CIBs is to be able to have one main CIB for production setup and several “shadow” CIBs that can be used for testing or maintenance purposes. Information amongst CIBs can be freely copied. Only one CIB can be effective (running) at the given time.

Prior to creating the CIB we needed to create OCF scripts for Kamailio and FreeSwitch. These scripts are called resource agents and are not part of the standard resource-agents package we installed previously.

OCF scripts are basically Linux shell scripts called by pacemaker. They meet the requirements of Open Cluster Framework standard. An OCF script is required to return several pre-defined values on certain outcomes of their actions. By these standard values Pacemaker knows whether the resource is running, whether its start has been successful or what action to take if there is an unexpected error.

The OCF script generally consists of XML-formatted information section and several functions that are called by Pacemaker. These functions usually incorporate the start, stop, status, monitor and usage procedures.

The Kamailio OCF script can be found in the attachment [23 OCF script for Kamailio]. It interacts with the running Kamailio instance by calling the “/etc/init.d/kamailio” script. If any procedure succeeds then the OCF script returns the value OCF_SUCCESS. When Kamailio fails then the status functions returns OCF_NOT_RUNNING value and Pacemaker can attempt to restart the resource or move it to another cluster node if Kamailio fails multiple times.

The FreeSwitch OCF script was a bit more complicated to write because its “/etc/init.d/freeswitch” script did not offer any status function. We added the function ourselves by calling the “fs_cli” (FreeSwitch Command Line Interface) command and checking the status of the running profile. If the output didn’t contain the expected word count than FreeSwitch was considered not running and the OCF script again returned OCF_NOT_RUNNING to Pacemaker. Several other improvements needed to be made within the “/etc/init.d/freeswitch” script in order to wait for the FreeSwitch to block the procedure until it has fully started (otherwise it forked the process immediately and the
following monitor operation was due to fail upon the first call since FreeSwitch was not fully up yet). The FreeSwitch OCF script offered the same function set as the one of Kamailio and can be found in the attachment [24 OCF script for FreeSwitch].

Both new scripts were copied into “/usr/lib/ocf/resource.d/heartbeat/” folder. They must have been owned by user root and the rights to the files must have been set to 755.

```
chmod 755 /usr/lib/ocf/resource.d/heartbeat/kamailio
chown root:root /usr/lib/ocf/resource.d/heartbeat/kamailio
chmod 755 /usr/lib/ocf/resource.d/heartbeat/freeswitch
chown root:root /usr/lib/ocf/resource.d/heartbeat/freeswitch
```

Once the files were in place we continued by configuring the CIB. We entered the crm shell and configured the cluster nodes. The following configuration was only required on one of the cluster nodes and was then automatically copied to the other one.

```
#crm configure
crm(live)# node <<#FQDN_of_first_cluster_node#>>
crm(live)# node <<#FQDN_of_second_cluster_node#>>
```

The next step was to configure the cluster parameters. “Shoot The Other Node In The Head” (STONITH) is a mechanism that completely turns off the node once the resource fails to run on it. This is a security precaution to avoid the “split brain”. This is a situation where one or more cluster nodes are completely disconnected from the rest of the cluster while still thinking they should keep running the resource. With shared resources like IP addresses this can create a lot of confusion on the network and in the end can degrade or completely deny the service to the end user. However since the failure of the Proxy process didn’t mean that FreeSwitch failed as well, we wanted to avoid completely turning off the cluster node which failed to run the Proxy process. The proxy process was simply moved to another node.

When Pacemaker is deciding what to do with the failed resource the action which it takes needs to have the majority of votes amongst all cluster nodes. Since this cluster is only consisting of two nodes, there might have been situations when the nodes voted on different things and thus no action acquired the quorum. Therefore Pacemaker needed to be configured to ignore the quorum policy and simply do what is considered best at first place. There was only one designated node which made the decision. This decision was then populated to the other node, so no conflicts could have taken place.

```
crm(live)# property stonith-enabled=false
crm(live)# property no-quorum-policy=ignore
```
The last thing was to configure resources (also called primitives in the CIB configuration) and commit the changes so they were populated across the cluster.

The first resource configured was Kamailio. This was a shared resource amongst all cluster nodes and since it was reachable only by one service IP address, this address needed to be configured as a shared resource as well. In the end these two resources were configured into a resource group and several constraints needed to be met when running the resource group. The first constraint was never to run them separately, preventing the split brain situation from occurring. The second one was to always start the VIPv4 (Virtual IPv4) before Kamailio (so that the bind to virtual IP is successful).

```
primitive Kamailio ocf:heartbeat:kamailio 
params ip="<<#SERVICE_IP>>" port="<<#SERVICE_PORT>>" 
   op start interval="0" timeout="60" 
   op monitor interval="5" timeout="30" 
   op stop interval="0" timeout="60"
```

```
primitive VIPv4 ocf:heartbeat:IPaddr2 
params ip="<<#SERVICE_IP>>" cidr_netmask="16" 
   op start interval="0" timeout="60" 
   op monitor interval="5" timeout="30" 
   op stop interval="0" timeout="60"
```

```
group PROXYCLUSTER VIPv4 Kamailio 
meta target-role="Started"
```

```
location cli-prefer-PROXYCLUSTER PROXYCLUSTER 
rule $id="cli-prefer-rule-PROXYCLUSTER" inf: #uname eq <<#NODE1_FQDN>>
colocation PROXYCLUSTER_COLOCATION inf: VIPv4 Kamailio
order PROXYCLUSTER_START_ORDER inf: VIPv4 Kamailio
```

Next resource to configure was FreeSwitch. This resource was different than Kamailio because it was not a shared resource. Both cluster nodes were required to run FreeSwitch all the time and the management of every instance had to be autonomous. This was achieved using the resource cloning mechanism. Several types of cloned were available and the one that suited us best was the "Unique clone". This type of resource clone is completely independent of its original and its management is local on the node where it is running.

```
primitive FreeSwitch ocf:heartbeat:freeswitch 
   op start interval="0" timeout="60" 
   op monitor interval="5" timeout="30" 
   op stop interval="0" timeout="60"

clone FreeSwitchClone FreeSwitch 
meta globally-unique="true" target-role="Started"
```
The final primitive to be configured was the cluster monitor which was used to send SNMP traps to the trap server when Pacemaker took any start/stop/restart/move action. In this situation the ClusterMonitor resource called a shell script with several environment variables set to describe the action that has taken place. The shell script then sent the SNMP trap.

```
primitive ClusterMonitor ocf:pacemaker:ClusterMon \
params user="root" update="30" \
extra_options="-E /root/SNMPSCRIPT/script.sh -e <<#TRAP_SERVER_IP#>>" \
op monitor on-fail="restart" interval="10"
location cli-prefer-ClusterMonitor ClusterMonitor \
rule $id="cli-prefer-rule-ClusterMonitor" inf: #uname eq <<#NODE1_FQDN#>>
```

The file with the shell script was placed into the "/root/SNMPscript/" folder. The traps sent were compliant with the Pacemaker MIB available from

https://github.com/ClusterLabs/pacemaker/blob/master/extra/PCMK-MIB.txt

In the end the configuration was committed and saved into backup file CIBconfig.xml.

```
crm(live)# commit
crm(live)# save xml CIBbackup.xml
```

This xml file can be used to restore the CIB configuration in the future. For the complete file see attachment [25 Cluster Information Base XML file].

```
crm(live)# load xml replace CIBbackup.xml
```

Upon commit the cluster started to manage resources immediately. The status of cluster nodes and resources could have been verified by issuing the following command.

```
# crm status
```

The status of the resources could have been verified by issuing the command

```
# crm resource status
```

All configuration tasks were possible using the crm command which is well documented. Typing anything combined with the “help” word displays the usage for the command. List of available commands within the crm shell can be viewed by double-pressing the TAB key. Simple resource management tasks such as moving the resources amongst cluster nodes were possible by issuing the command.
The resources could be started/stopped/restarted by

```bash
# crm resource start|stop|restart <<#Resource_name#>>
```

The counters for each operation for each resource could have been reset by submitting the following command.

```bash
# crm resource cleanup <<#Resource_name#>>
```

This forced Pacemaker to clean all counters that could have been blocking starting the resource on one cluster node, for example because of high failure count in the past. If the error has been repaired, there is no need for the fail count to remain the same.

## 5.9 Firewall

### 5.9.1 Traffic characteristics

The system required only a few ports permanently opened for inbound traffic. These ports included SIP signalization, both on TCP and UDP. Packets needed to be accepted only if destination IP address was set to the service IP address. Call signalization sent from proxy to media servers occurred with destination port 5080 (UDP only). Furthermore, packets from Kamailio to Freeswitch needed to have the source IP address of the service and destination IP address of the node. No other packet must have been accepted on destination port 5080 (this was double ensured by using and access control list in Freeswitch itself).

Ports for RTP and SRTP should have been opened only when call signalization indicated that there was going to be traffic on the port soon. This setup is called pinhole firewall and was provided by the `nf_conntrack_sip` Linux kernel module.

The devices had to accept packets sent by Corosync from other node, specifically on port 5405 (UDP).

FreeSwitch Lua script used Kamailio’s “`kamcmd`” to verify the call destination number, using a communication over TCP socket with destination port number 2046. Packets with this destination port number had to contain source IP addresses of the nodes on destination IP address of the service.

Communication with the SSH management PC was enabled on port 22 with destination IP address of the cluster node.
SNMP manager uses port 161 to send SNMP polls. This port also needed to be opened permanently.

All other inbound connections were required to be denied.

No traffic forwarding was allowed and the system must drop by default any packets that were supposed to be forwarded.

5.9.2 Configuration

The above mentioned characteristics were reflected into **iptables** firewall rules. IPTables is a standard Linux firewall program which is highly customizable and offers a full control over firewall rules and packet manipulation. The rules were stored in the “/etc/sysconfig/iptables” file and restored from this file every time on boot. The list of all rules can be found in the attachment [26 Firewall rules].

The following example extracted from iptables configuration enables to accept connections on UDP port 5060 with destination IP address of the service. This rule is specified within the INPUT firewall chain, meaning it is used for inbound traffic.

```bash
# iptables -A INPUT -d <<#SERVICE_IP#>>/32 -p udp -m udp --dport 5060 \ 
> -j ACCEPT
```

There are general policies configured on the rule chains. These policies can be seen as default actions which are taken if no rule is matched for the given packet. The default policy on INPUT chain is to drop all unrecognized packets. The same policy is applied for the FORWARD chain.

IPtables command offers to see the output of the packet and data size counters for every rule specifically and for the whole rule chain in general. This output can be obtained by issuing the following command.

```bash
#iptables -vnL
...
1 601 ACCEPT udp -- * * 0.0.0.0/0 <<#SERVICE_IP#>> udp dpt:5060 
...
```

The example output shows there was 1 packet with size of 601 bytes that hit the rule described above and that this packet was accepted.
5.10 Complete installation procedure

Every above mentioned installation and configuration task has been automated in the form of shell script.

Within the directory "serviceUP" [attachment 1 ServiceUP installation script and sources] there are several files and subfolders. The subfolders are:

- cluster_cfg
  - contains RPMs and configuration files necessary to install and configure the clustering software,

- fail2ban_cfg
  - contains the configuration files for fail2ban application,

- kamailio_cfg
  - contains RPMs of Kamailio 4.0 and configuration files to replace the ones shipped with the installation,

- freeswitch_cfg
  - contains all RPMs of FreeSwitch 1.2.8 alongside with configuration files packed in a tar archive,

- system_cfg
  - contains RPMs for UnixODBC, and several other utilities,
  - also contains configuration files necessary to prepare the Linux system to run the service.

The purpose of this automated installation is the “INSTALL.sh” shell script that prompts the administrator for several system parameters, such as IP addresses of cluster nodes, virtual service IP address, port numbers, database credentials and domain name of the service. Every configuration file within all subfolders is preconfigured with <<#...#>> directives which are then replaced during the installation by real values specified by the administrator.

Kamailio and FreeSwitch are installed from RPM packages. The script has sections that are commented out, for compilation of both applications from source code. Once uncommented, they can be used instead of RPM installation.

The script takes care of configuring the system, setting the firewall rules, installing and configuring the applications and turning off unnecessary services within the system. It is designed to run on a clean installation of SLC6 and the administrator needs to run it as root. Care taking is advised when filling the prompted values.
The list of parameters prompted before installation is:

- **Cluster parameters:**
  - local IP address,
  - local IP subnet address,
  - FQDN of local cluster node,
  - remote cluster node IP address,
  - FQDN of remote cluster node,
  - service IP address,
  - FQDN of the service (how it’s resolvable by DNS).

- **SIP parameters:**
  - domain name,
  - SIP port,
  - SIPS port,
  - RTP port range,
  - PBX IP address.

- **Database connection parameters:**
  - database account username,
  - database account password.

- **LDAP connection parameters:**
  - service account username,
  - service account password.

- **SNMP parameters:**
  - IP address of the trap server,
  - SNMP community string.

- **System parameter:**
  - IP address of SSH manager.

When the script finishes and everything has been properly installed and configured, the administrator needs to restart the device and run script named “POSTINSTALL.sh”.

# cd serviceUP
# ./POSTINSTALL.sh
This will finish the configuration by setting up the cluster CIB and starting the service. At this point the system is ready for operation.

5.11 Client side requirements

The client application must have supported the following features in order to be able to establish calls through the system.

The application must have supported DNS (at least) SRV records resolving. If the client supports the full SIP parameters discovery, the client could have used the autoconfiguration of Proxy server. TLS transport is preferred by the system, TCP and UDP are available in case the client does not support usage of TLS.

In order for the media to be encrypted (SRTP) the client application must support the ZRTP or SDES key negotiation protocols. Usage of encrypted media was not configured to be mandatory but this fact may change with respect to CERN security team’s demands.

If the client is establishing a call from an account registered under another SIP service then the above-mentioned facts must have applied to the proxy server of the used service.

In order for the client to prove the call is valid the client software must have supported the DTMF transport defined in RFC 2833. Inband DTMF and DTMF transported within SIP INFO requests are not supported.

6 Testing

For testing we started our own DNS server, which was configured with information about domain “@cern-testing.ch” (records configured were similar as the ones described in chapter 5.1 DNS configuration). On other server we started a separate instance of Kamailio which was configured to be a SIP registrar and proxy server for this testing domain. Account “testacc” was configured as a local subscriber.

6.1 Calls

The system was configured to be only an inbound gateway. Therefore all calls were initiated by “testacc@cern-testing.ch”.

When the INVITE request arrived to Kamailio, it fired the routing logic and performed all checks. The most important check was whether the call was authorized, based on the permissions rules. If the call was permitted, dialog tracking began. The
request URI was changed if the “email-format” destination URI was originally dialled and if the record for that email was found in the LDAP directory. The output logs can be found in the attachment [27 Valid call documentation].

In first case the LUAIVR.lua script was configured to authenticate users originating calls. This script required the user to register his URI on a webpage. If the user called without registering and the Lua script did not find a database record for this user, the call was ended. This behaviour can be seen in the attachment [28 Failed user authentication by LUAIVR.lua script].

The source code for a testing webpage for registration is included in the attachment [29 SIP URI registration website with usage explanation] with usage explanation. It takes user’s URI and email address and generates the PIN code for the user (the record is saved into the database table “PINauth” – SQL script for creation included in attachment number [6 SQL script for schema creation]). If the user calls with a registered SIP URI he is prompted by FreeSwitch to input the PIN code. The script then compares the input with the value from database. If they mismatch, the call ends. If the PIN is ok, the call is bridged to PBX.

The second case was if the LUAIVR_M.lua script authenticated the caller. This option was more dynamic and more trustworthy because the generated numbers changed with every call. Similar authentication cases as described for the first script are included in the attachment [30 User authentication by LUAIVR_M.lua script].

### 6.2 IP address banning

For flooding the server with forged SIP requests to prove the IP address banning worked the “SIPp” testing tool was used. This tool uses xml-formatted testing scenarios describing the desired call flow. The message rate had to be higher than seven messages per second for the IP to become banned. This was proved by the “myUAC.xml” scenario. The tool even offered the PIN code DTMF dialling by streaming a “pcap” formatted file. The flooding was tested by a “flooding.xml” scenario file. Both files are found within the attachment [31 SIPp testing scenarios].

Furthermore, several other testing scenarios were developed to test the stability of the system. They are included within the same attachment, alongside with description and results.
If the message rate was too high, the pike module of Kamailio banned the IP. If this happened three times, according to configuration of fail2ban, the IP address was banned in iptables. This was verified by the “SIPuser ipban” command, which listed the banned IP addresses, both by pike and iptables in separate sections of the output.

6.3 High availability of the Proxy server

With the configuration described in the previous chapters we ensured that the service was reachable by a single IP address. The clustering software ensured that the Kamailio process was started on other node if the first one failed. The outputs of the “crm status” command for various situations are shown in the attachment [32 Cluster testing], alongside with SNMP traps examples sent to the trap server.

A failure of Kamailio was transparent to the end user as the resource moved to another cluster node.

A failure of FreeSwitch was noticeable only if the user was presently in call bridged on that particular FreeSwitch. This was not influenced by moving the service IP address to another node, because the RTP traffic was always flowing to the primary IP address of the node.

6.4 Media servers load balancing and monitoring

While Kamailio was running it was checking both FreeSwitches’ availability through the dispatcher module. The state could have been verified on the node where Kamailio was running by issuing the command “SIPuser dispatcher show”.

The flow chart for the load balancing feature is included in the attachment [33 Load balancing and failover of Media servers], alongside with more detailed description of what happened in case a FreeSwitch went down or up.

7 Future development

The system was designed with focus to fulfil the original requirements while several extra features were tested. These features are described throughout the next subchapters. They were explored in order to identify the possible objectives of further development and feature extendibility.
7.1 LDAP improvements

If the destination is dialled in the “email” format Kamailio resolves it to a phone number using the LDAP module. There are situations when the resolving process returns mobile and office numbers both. Kamailio’s routing logic can be improved to try both numbers on try-fail basis. This means, for example, that if the office number is not reachable or the user is not responding, the mobile number can be dialled next.

The reason why this was not included in the routing logic was that the current setup is only an extra feature that was not originally required. It was done as a test of integrating the system with the LDAP system of CERN. The original intention was to authenticate users based on their credentials stored in the active directory. However this option was discarded because the current “ldap” module of Kamailio can not pass the digest authentication string to the SIP subscriber. Fixing this feature would significantly prolong the development process and would extend the scope of the project.

7.2 WebRTC and Websockets

The WebRTC standard is in active development process and major changes are done on weekly basis. There are several JavaScript SIP stack libraries, such as “sipml5” (developed by Doubango Telecom [41]) or “jssip” (developed directly by creators of Websocket standard [42]) that are able to communicate via websockets with a proxy server.

This means the web-based clients are able to communicate with each other, while the signalization goes through the SIP proxy server and the media flows directly between the browsers (who implement the same state of WebRTC). However if the browser needs to communicate with a media server, such as FreeSwitch or Asterisk there are problems.

There are several attempts to make the WebRTC work with the current media servers, such as the “WebRTC2SIP” from Doubango Telecom. These workarounds are temporary solutions because of the fact that if something changes in the WebRTC specification there is no guarantee that the workaround will still work. This is the reason why none was deployed in the setup.

As mentioned in the 4.5.2 Media server chapter, Asterisk was being considered to replace FreeSwitch. The most significant advantage of Asterisk was that it partially supported the WebRTC standard. The web-based client was able to establish a call with Asterisk (although media flow was only one directional), however Asterisk must have been
configured to globally enable SRTP while completely disabling RTP. A way to support both encrypted and unencrypted protocols was not resolved. WebRTC supports only encrypted media.

The media profile offered in the SDP by the web-based client application was not currently compatible with FreeSwitch, which had a problem with indicating a “crypto” option while using an unencrypted profile. This is a result of ongoing standardization process of WebRTC and has to be resolved.

The client application which was tested with Asterisk was one which used the “jssip” SIP stack. The code was slightly modified to support the authentication by including the CERN authentication cookie value in the custom header of a REGISTER request.

If there will be a desire to support SIP web clients in the future when WebRTC becomes a standard in full form, the decision whether to replace the FreeSwitch with Asterisk will have to be re-evaluated. By the time both Asterisk and FreeSwitch should support the standard and tests will have to be performed.

7.3 Other improvements

The “dialog_ng” module [43] of Kamailio is currently in development process and is intended to replace the currently used “dialog” module. The new module has enhanced capabilities compared to the old one and is already available in the stable version of Kamailio with limited ability. It supports tracking initiation starting by early media sessions and tracking for forked calls simultaneously. The module currently does not support storing the dialog information in the database, which was essential for the system architecture. That is the reason why it was not deployed instead of the “dialog” module.

In order to provide a transparent failover of active calls, the FreeSwitch setup could be improved. They can be configured to store the dialog information in the database (similar to the proxy server). Upon FreeSwitch failure, another FreeSwitch can be automatically instructed to take over the ongoing calls. This would require the FreeSwitch servers to be bound to a virtual IP (load balancing features would have to be reconsidered). Call takeover logic can be included in the OCF cluster scripts. There is even a possibility to map the Kamailio’s dialog information with the FreeSwitch’s. It is possible that this mapping would require an intervention within the source code of both servers.
8 Conclusion

The purpose of this thesis was to develop a VoIP gateway into CERN’s fixed and mobile telephony infrastructure. Various options to do so were analyzed and the variant that best fulfilled the requirements was selected as the final solution.

Various configuration remarks were described throughout the text, with emphasis on important point of the requirements. The final platform was developed to run on a Linux cluster, while the installation was unified in a form of a shell script, with all the binary packages included.

The project was developed with the focus to stay compliant with the architecture of other services of CERN. Integration with the LDAP service enabled us to get numbers of members of the personnel by searching their active directory account by their email address. Oracle database service was used by the system to store session-related information and other operation-critical information, such as permission rules.

The High Availability requirement was met by deploying the system on two servers, while enabling us to deploy load balancing and failover for the media services. The configuration of the cluster is not limited to two machines. Cluster can be extended with no configuration necessary to the existing nodes, with respect to specifying the new node in the Corosync’s configuration file.

The security of the system was designed to be as strict as possible, although the parameters might change over time, as the process of finding the balance between limiting valid users and detecting malicious requests can be delicate.

Voice applications running in FreeSwitch provide a way to determine if a caller is a human being.

During my work on this thesis, many people at CERN have asked me whether they can already configure their devices (such as smartphones) to make cost-free calls to their co-workers. This demonstrates the fact that people are beginning to recognize the potential of VoIP in everyday life and are aware of its advantages and differences from classical telephony.

Although the system enables only to establish calls from the outside, it can be extended to provide a fully-featured SIP account for members of CERN. If mapped to the user’s main computing account it would augment its feature set. With the ongoing development done with the WebRTC to bring the VoIP to browsers (and mobile devices), this topic will soon be important to CERN.
# Abbreviations and references

## List of abbreviations in alphabetical order

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CERN</td>
<td>European Organization for Nuclear Research</td>
</tr>
<tr>
<td>CLOB</td>
<td>Character Large Object</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DTMF</td>
<td>Dual-tone multi-frequency</td>
</tr>
<tr>
<td>GPN</td>
<td>General Purpose Network</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IEEC</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union – Telecommunication standardization sector</td>
</tr>
<tr>
<td>IVR</td>
<td>Interactive Voice Response</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LHC</td>
<td>Large Hadron Collider</td>
</tr>
<tr>
<td>LTS</td>
<td>Long Term Support</td>
</tr>
<tr>
<td>NAPTR</td>
<td>Name Authority Pointer</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>OCF</td>
<td>Open Cluster Framework</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PBX</td>
<td>Private Branch eXchange</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-time Transport Control Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>SBC</td>
<td>Session Border Controller</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SIPS</td>
<td>Session Initiation Protocol Secure</td>
</tr>
<tr>
<td>SLC</td>
<td>Scientific Linux CERN</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SRTP</td>
<td>Secure Real-time Transport Protocol</td>
</tr>
<tr>
<td>SRV</td>
<td>Service Record</td>
</tr>
<tr>
<td>STONITH</td>
<td>Shoot The Other Node In The Head</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>TN</td>
<td>Technical Network</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Unified Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over Internet Protocol</td>
</tr>
<tr>
<td>WebRTC</td>
<td>Web Real-Time Communications</td>
</tr>
</tbody>
</table>
XML  – eXtensible Markup Language
XMPP  – eXtensible Messaging and Presence Protocol

List of references

[33] - FreeSwitch version history, http://jira.freeswitch.org/browse/FS#selectedTab=com.atlassian.jira.plugin.system.project%3Aversions-panel
[34] - Asterisk version history, https://wiki.asterisk.org/wiki/display/AST/Asterisk+Versions
[37] - Pacemaker project home website, http://clusterlabs.org/
[40] - CERN Linux main website, http://linux.web.cern.ch/linux
[41] - SIPml5 project home website, http://sipml5.org/
[42] - jSIP project home website, http://www.jssip.net/
Numbered list of attachments on the CD

1. ServiceUP installation script and sources
2. SIPuser administrator script
3. DNS zone file
4. UnixODBC configuration files
5. Cron script for updating the TNS oracle aliases
6. SQL script for schema creation
7. Kamailio init.d script for GIT installation
8. Kamailio init.d script for RPM installation
9. Main configuration file of Kamailio
10. TLS configuration file of Kamailio
11. SNMPd init.d script
12. SNMPd configuration files
13. Permission rules files of Kamailio
14. Scripts for periodic downloading of permission rules
15. Configuration files for Fail2ban
16. LDAP configuration file for Kamailio
17. Complete set of configuration files for Kamailio build from GIT
18. Complete set of configuration files for Kamailio installed from RPMs
19. FreeSwitch init.d script for GIT installation
20. FreeSwitch init.d script for RPM installation
21. FreeSwitch configuration files
22. Configuration files for Corosync
23. OCF script for Kamailio
24. OCF script for FreeSwitch
25. Cluster Information Base XML file
26. Firewall rules
27. Valid call documentation
28. Failed user authentication by LUAIVR.lua script
29. SIP URI registration website with usage explanation
30. User authentication by LUAIVR_M.lua script
31. SIPp testing scenarios
32. Cluster testing
33. Load balancing and failover of Media servers