TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Maksim Lind 153608IAPM

CLUSTERED SIP SESSION BORDER CONTROLLER WITH TOPOLOGY HIDING USING OPEN-SOURCE TECHNOLOGIES

Master's Thesis

Supervisors: Margarita Spitšakova
PhD
Tanel Vakker

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Maksim Lind 153608IAPM

TOPOLOOGIAT PEITEV KLASTERDATUD SIP SESSIOONIPIIRIKONTROLLER KASUTADES AVATUD LÄHTEKOODIGA TEHNOLOOGIAID

Magistritöö

Juhendajad: Margarita Spitšakova
PhD
Tanel Vakker

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Maksim Lind

04.01.2018

Abstract

SIP is a widely used signaling protocol in VoIP and instant messaging applications and services. A SIP Session Border Controller is an element in the SIP network that is placed on the border of two networks, providing security and quality of service features, as well as exerting control over signaling and media. There are complex proprietary SBC solutions that offer service with minimal downtime thanks to their clustered architecture. However, using proprietary solutions implies maintenance and license costs as well as risks associated with support policy at the end of product lifecycle. As an alternative, a highly available SBC is developed in the scope of the current thesis relying on open-source technologies. This thesis delivers system configuration files and the deployment script that can be used to install the system in test or production environment. All the scripts and configuration files are well commented, explained and addressed in current thesis. Furthermore, a critical bug has been discovered in the open-source server software used in the system setup. It was reported to the maintainers along with a suggestion for a fix. Finally, test cases validating the system's conformity to SBC and cluster requirements are specified.

The result of the current thesis is available from https://github.com/maksiml2014/hasbc-master-thesis.

The current thesis is written in English language and is 57 pages long, including 5 chapters, 21 figures and 2 tables.

Annotatsioon

Topoloogiat peitev klasterdatud SIP sessioonipiirikontroller kasutades avatud lähtekoodiga tehnoloogiaid

SIP on signaaliprotokoll, mida kasutatakse laialdaselt internetikõnedes ja kiirsuhtluses. SIP sessioonipiirikontroller on SIP võrgu element, mis asub kahe võrgu piiril ning tagab turvalisust ja kvaliteeti ja saab mõjutada SIP kontrollsignaale ja ka saadetavat meediat. Saadaval on keerulisi kommertslahendusi mis pakkuvad sessioonipiirikontrolleri teenuseid minimaalse seisuajaga. Kasutades kommertslahendusi suurenevad aga serverite ülalpidamis- ja litsentsikulud ning kaasneb kasutajatoe lõppemise risk. Käesoleva töö raames arendati vabavaralistel tehnoloogiatel põhinev sessioonipiirikontroller töökindlusklastris. Töö tulemusteks on konfiguratsioonifailid ja paigaldusskript, mida saab kasutada testimis- või päris töökeskkonnas. Antud väitekiri kirjeldab ja kommenteerib kõiki paigalduseks vajalikke seadeid ja faile. Samuti leiti kasutatud vabavaralise serveri tarkvaras kriitiline viga. Vea kirjeldus ja parandamisettepanek esitati projekti arendajatele. Töös kirjeldatud katsed näitavad tehtud süsteemi vastavust sessioonipiirikontrolleri ja töökindlusklastri nõuetele.

Töö tulemus on kättesaadav aadressilt https://github.com/maksiml2014/hasbc-master-thesis.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 57 leheküljel, 5 peatükki, 21 joonist, 2 tabelit.

List of abbreviations and terms

SIP	Session Initiation Protocol
IP	Internet Protocol
UA	User Agent
UAC	User Agent Client
DoS	Denial-of-Service
VRRP	Virtual Router Redundancy Protocol
HA	High Availability
LAN	Local Area Network
SBC	Session Border Controller
QoS	Quality of Service
VIP or VIPA	Virtual IP Address
PSTN	Public Switched Telephone Network
UAS	User Agent Server
VoIP	Voice over IP
SDP	Session Description Protocol
NIC	Network Interface Controller

Table of contents

List of figures

List of tables

Introduction

Session Initialization Protocol (SIP) is a signaling protocol used to create, control and end multimedia sessions in Internet Protocol (IP) based networks. SIP is widely used in Voice Over IP (VoIP) applications and services. The media session, e.g. a voice call or instant messaging session, is not part of SIP routing and is handled in Session Description Protocol (SDP).

A SIP network usually consists of multiple components. Each component has a set of rules that apply to the passing traffic to ensure that network packets reach their destination and do not contain sensitive data. A User Agent Client (UAC) is a component that initiates the signaling and a User Agent Server (UAS) is a component that responds to this signaling. An IP phone or any other SIP terminal acts as UAC and UAS interchangeably.

Besides UAC and UAS, there are three more SIP components: proxy server, redirect server and registrar server. These can be separate physical servers as well as logical units of a server software. When a server combines multiple functions, it is called a SIP server.

A proxy server mediates the messages between UAC and UAS and can perform accounting and security functions. A redirect server informs the caller that the callee has changed its location. A registrar server or a location server receives, stores and provides information about UAs' SIP addresses.

This work is organized as follows: the current chapter contains the problem statement and motivation for this work. Chapter 1 reviews the related work and explains the technologies used. Chapter 2 describes the design of the implemented system. System validation process is described in Chapter 3. In Chapter 4, a summary is given, and future work is discussed.

Motivation

The motivation behind this work is to provide an open-source alternative for complex proprietary solutions like Acme Packet Net-Net OS [1]. Independence from proprietary

hardware and software in communication solutions was proposed by Almeida and Cruz to significantly reduce the maintenance and licensing costs [2]. Moreover, open-source solution enables higher level of customization by providing access to the source code. Another advantage of open-source software is that if the original maintainer deprecates or abandons the product, it can be forked into a separate project and its development can be taken over.

Problem Statement

The current work aims to describe a configuration of a SIP proxy server that would satisfy the following requirements.

1. The proxy should have redundant backup elements. The elements should form a cluster and exchange information in real time. When one of the cluster nodes fails or is under maintenance, the other nodes should take over its works seamlessly for the client.

2. Neither of the call sides should be aware of the clustered architecture of the proxy. The UACs should only be aware of the public IP address of the clustered proxy. The packets leaving the proxy cluster should not contain any information about cluster topology.

3. The proxy should be able to route traffic to multiple SIP servers based on their load.

4. The proxy should protect the network and other devices from Denial-of-Service (DoS) attacks.

5. The proxy should accept UAC registration requests and subsequently provide the means to locate it for peers on the network. The proxy should also lower the rate of the registration traffic when redirecting it to the main SIP registration proxy.

The transmission of media streams is not in the scope of this solution.

Solution Design

The following steps will be performed to achieve the goals.

- OpenSIPS [3] server software will be installed on several machines.
- OpenSIPS clusterer [4] module will be used to enable multiple SIP proxies to exchange real-time data.
- Keepalived [5] service will be used to implement Virtual Router Redundancy Protocol (VRRP) to provide a single IP address known to the UAs and assign it to the most suitable cluster node. OpenSIPS topology_hiding module [6] will obfuscate the outgoing SIP packets.
- OpenSIPS load_balancer module [7] will route the traffic to SIP servers based on their load.
- OpenSIPS pike module [8] will protect the network from DoS attacks.

Design Validation

Each requirement of the solution will be tested followingly:

- Redundancy and the capability for the call failover will be tested by disconnecting the master node of the cluster in the middle of a SIP dialog. Subsequent requests should be routed through one of the backup cluster nodes and reach the original recipient.
- Inspecting the SIP packet headers on the UAC side will reveal if the topology hiding and registration traffic rate decrease is working.
- SIPp [9] test tool can generate SIP traffic to test the load balancing and DoS protection capabilities of the proxy server. Moreover, Stanek and Kencl [10] have presented a modified version, capable of simulating a distributed DoS attack.

1 Related Work

1.1 Session Initiation Protocol

SIP's purpose is to establish and end multimedia sessions between UAs. To do this, five features are communicated.

- User location: allows locating UAs in the network.
- User parameters: determines parameters of the multimedia session, for example the audio codec to use.
- User availability: determines if the desirable user is available for session initiation.
- Call establishment: exchanges the parameters between the end points and informs them of the current call state, e.g. ringing, busy or OK.
- Call management: pauses, transfers or ends the session.

1.2 Session Border Controller

A Session Border Controller (SBC) is a type of a SIP proxy that is installed on the border of SIP networks, e.g. between service provider and its private or enterprise users. Its main purpose is to provide a secure and reliable connection between networks. Security is enforced for example by protecting network segments from DoS attacks and hiding information about network topology in the transmitted packets. An SBC is often used to collect statistics and billing information, normalize the SIP message flow and perform load balancing between multiple destinations.

1.3 Acme Packet Net-Net OS

Acme Packet is a company that produces network communication server solutions. One of its products is a software platform called Acme Packet Net-Net OS. It operates on a series of hardware platforms and offers an industry-leading SBC solution. According to a fiscal year report of Acme Packet in 2008, their products have been purchased by ca 600 clients consisting of service providers and enterprises in 92 countries [11].

In 2013, Acme Packet was acquired by Oracle Corporation [12]. This lead to deprecation of old services and now Oracle Communications Session Border Controller [13] is the successor of Acme Packet Net-Net solutions. For existing Acme Packet clients this meant that the support and updates for their servers were dropped and they were encouraged to update to the Oracle SBC solution with new licensing and management fees. An alternative to that is using an open-source SBC.

1.4 Open-source in Communication Solutions

Segec and Kovacikova [14] analyzed the existing protocols, technologies and services and proposed an open SIP-based communication platform, which however is more complex and converged than the one described in the current work. The current work focuses on the signaling and gateway sections of such platform. One of the protocols they suggest for improving service availability is Virtual Router Redundancy Protocol (VRRP). They also state that real-time data synchronization between servers is an open question and can be solved by an entity that would replicate SIP messages. For SIP servers, two successors of OpenSER project are recommended: Kamailio [15] and OpenSIPS.

Thompson et al. [16] designed and implemented a distributed telecommunication system that connects VoIP devices over a Public Switched Telephone Network (PSTN). The solution uses open-source SIP and focuses on de-centralizing the communication systems. Kamailio is used as a SIP router implementation, operating in a stateless mode. While allowing for a better performance, it also makes it impossible to perform topology hiding. In their setup, the topology hiding is implemented by a FreeSWITCH [17] SBC on the border of IP network and PSTN. They suggest introducing redundant components to such systems to improve reliability.

1.5 High Availability and Clustering

In the context of computer systems, the availability metric shows the probability that a system is ready to be used at any given time. A highly available system provides its service even when a failure occurs within a component of this system.

A computer cluster is a set of computers that are connected to each other and appear to its user as a single system. The separate computers in the cluster are called cluster nodes. The nodes are redundant components – they do not have separate responsibilities, but all perform the same task.

A high-availability cluster has active nodes that provide a service and backup nodes that are activated when a failure occurs in an active node. Such event is called a failover. For a failover to occur seamlessly for the user, the backup node must receive constant updates relevant to the service provided. Another way to activate the backup node is to load the relevant data from a shared database during the failover, which is slower than the first method.

The process of restoring the system to the state before the failover is called failback. In the process of a failback the failed node recovers and goes back to active state, while the node that became active in the process of failover goes back to backup state.

1.5.1 Virtual Router Redundancy Protocol

VRRP is a protocol that enables connecting several IP hosts into a HA cluster by creating a virtual router. Instead of configuring the machines that use the cluster to connect to backup servers in case of the master server failure, the VRRP requires that the users connect to a static virtual IP address. The master node assigns this IP address to itself and process the traffic flow. The backup nodes monitor the state of the master node and assign the virtual IP to the newly elected master node upon its failure. When the first server recovers from the failure, it goes back to being the master node. Refer to Figure 1 for VRRP illustration.

Yang et al. [18] analyze the VRRP and its applicability to SIP servers. They highlight a problem of a SIP proxy requiring real-time synchronization to be a node in a cluster while stateful (i.e. unable to relay a SIP message only based on the information contained). They argue that synchronizing the SIP server state using a shared database is too time-consuming in the context of a service with thousands or millions of users, and propose a solution based on an in-memory database that replicates its state over the network using Remote Procedure Protocol. This functionality has been implemented in OpenSIPS clusterer module in 2016 [19] [20].

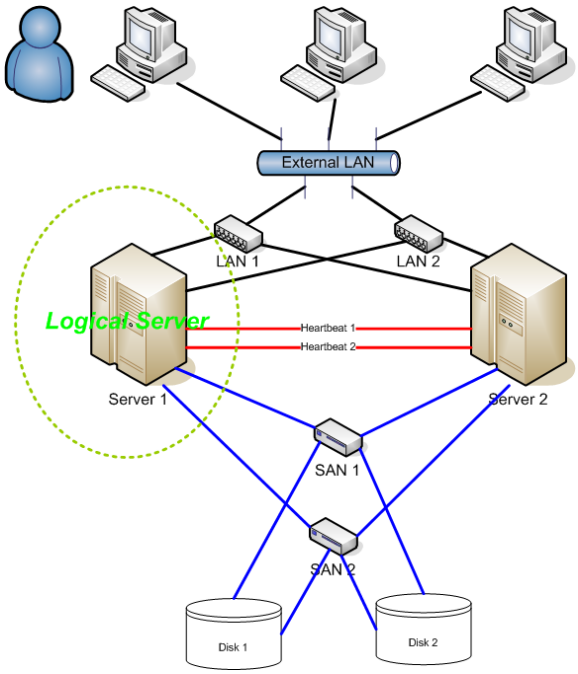

Figure 1. HA cluster using VRRP with one backup node

Servers 1 and 2 are nodes in a cluster. Users in the external LAN are configured to use a virtual IP address of the Logical Server. This address is assigned to the Server 1, which is currently active and provides a service to the users in external LAN. It keeps Server 2 up to date by transmitting relevant data over LAN1 and LAN2 networks. Some of the data is also written to the shared disks. In case of a failure in Server 1, Server 2 will be activated by virtual IP assignment, and all user requests will be routed to it [49].

1.5.2 Clustering OpenSIPS for High Availability

Smartvox Limited [21] is a company that focuses on designing and delivering OpenSIPS solutions for internet service providers. In 2010 they claim to have implemented a clustered SIP proxy using OpenSIPS [22]. In addition, they describe the cluster implementation using VRRP [23]. However, being a commercial software provider, they fail to publish the source code of the solution [24]. Moreover, the described solution does not implement topology hiding and operates only on stateless proxies.

In 2017, Smartvox Limited expanded their blogpost series on clustering OpenSIPS and described a way to improve failover times of clustered OpenSIPS proxies [25] by using Pacemaker [26] and Coro sync [27] resource management software. However, they admit that this setup may in some circumstances drop the established calls during a failover. To combat this, they configure binary replication provided by the clusterer module of newer versions of OpenSIPS [28]. Compared to the current work, their solution does not implement neither topology hiding, nor registration rate throttling.

1.6 Proxy Statefulness and Topology Hiding

A SIP server can be either stateful or stateless. A stateless proxy simply relays the messages it receives. For a stateless proxy to operate, the relayed message must contain all the necessary information about its destination. A cluster of stateless proxies is easy to configure because any cluster node at any given time will be able to deliver a SIP message.

Topology hiding is a security-enhancing functionality of a SIP server. It can hide sensitive client information and information about the internal topology of the SIP proxies network – e.g. IP addresses of cluster nodes. This is done by altering the relayed SIP messages. However, a response to such altered message cannot be routed to its originator by a stateless proxy since some of the information has been removed.

The removed or altered information is stored inside the proxy that performed the topology hiding. The same proxy can restore necessary information and route the response. Topology hiding can only be performed by a proxy that keeps track of ongoing dialogs – a stateful proxy.

For stateful proxies to operate as cluster nodes, the state of the master node must be replicated to backup nodes in real time. Without this, in the event of a failover new dialogs could still be established without problems – new dialog states would be written to the new master node. However, the messages of ongoing dialogs will not be relayed because the dialog states were saved on the failed master node.

1.6.1 SIP Messages

SIP components relay information to other components via SIP messages. A SIP message has a type, a header and a body. The type of the message indicates the function the receiving end should carry out. The header contains the information necessary to deliver the message to its destination. The body carries arguments or a payload necessary to carry out the action indicated by message type. Refer to Table 1 for most common SIP message types. A SIP response is in text format and contains a status code and a message, e.g. “200 OK”, “401 Unauthorized”.

Table 1. Most common SIP messages

SIP MESSAGE TYPE	DESCRIPTION
INVITE	Used to establish a session with a UA
BYE	Terminates an ongoing session
REGISTER	Registers user location in Registrar server
ACK	Acknowledge to INVITE

1.6.2 Sip Transaction and Dialog

A transaction occurs between a UAC and UAS and consists of all messages starting with the initial request of the UAC and ending with the final response of the UAS, including all intermediate responses. A SIP dialog represents a relationship between two UAs that persists for some time. It is identified by a dialog ID. Refer to Figure 2 [29] for an illustration of transactions and dialogs.

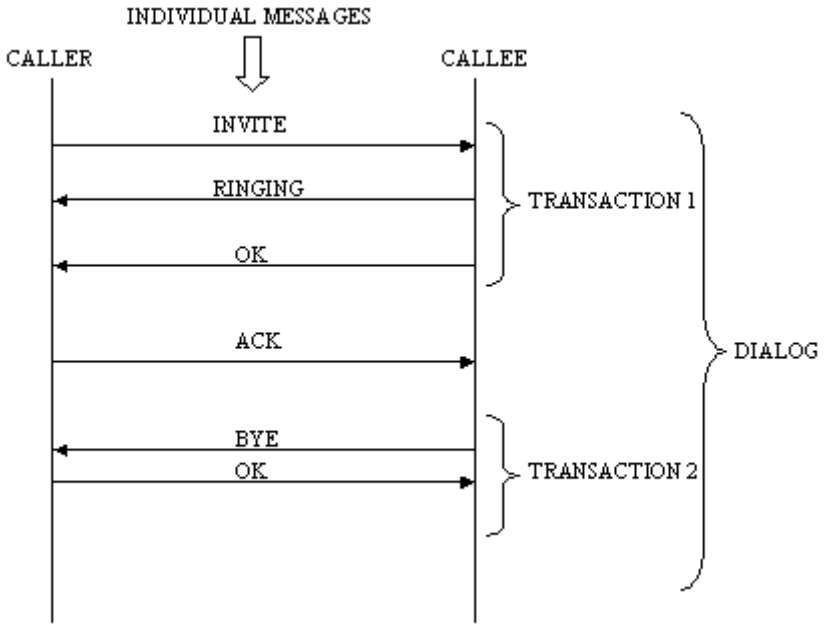

Figure 2. Transactions and dialogs

2 System Design

An SBC acts like a firewall in a SIP network. In the topology described here it is positioned between the client's and service provider's networks. From the service provider side, the SBC connects to intermediate proxy servers and not to the internet. The service provider network is guarded from the internet by a firewall or another SBC. For this reason, the traffic that comes from the service provider network can be trusted and this side of the network is called Internal. The client's network is External: it is not controlled by the service provider and therefore cannot be trusted (Figure 3). It is worth noting, that the Internal and External networks are not necessary from different subnets. This work describes a scenario where they are in the same subnet since it requires additional configuration.

This chapter describes what technologies were employed to implement SBC features and explains the configuration files and scripts used.

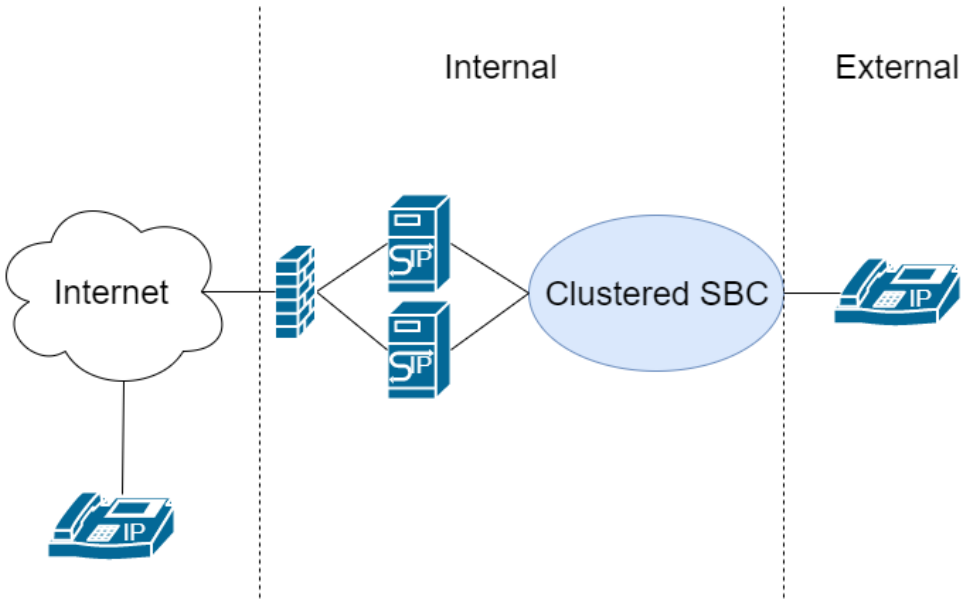

Figure 3. System design

A clustered SBC guards the service provider network (Internal) from the client's network (External). Internal network itself is usually guarded from the internet by a firewall or another SBC.

2.1 Configuration and Deployment

An open-source configuration management tool Ansible [30] is used in this work as suggested by Singh et al [31] and Ebert et al [32]. It takes two configuration files as an

input: an inventory file, that names and groups all hosts to be configured, and a so-called playbook file, which narrates what commands must be run and what files must be deployed to certain hosts or group of hosts. The playbook file can be found in Appendix 1 – Ansible Playbook File. The deployment is initiated with a command in Figure 4, where the arguments are the two aforementioned files. In the current setup the inventory file defines three groups of hosts: *proxyhosts* – hosts that will run OpenSIPS proxy server software, *dbhosts* – hosts that will run a database server, and *all* – all hosts from previous two groups. Refer to the Figure 5 for inventory file code.

```
ansible-playbook -i hosts ansible_playbook.yaml
```

Figure 4. Initiating deployment with ansible

```
[proxyhosts]
w520opensips
w530opensips

[dbhosts]
dbhost

[all:children]
proxyhosts
dbhosts
```

Figure 5. Ansible inventory file

GNU M4 macro processor [33] is used when multiple configuration files share a common part and only differ in a few key points. A set of keywords and the corresponding values called macros must be defined in one file. Another file contains a configuration template for some program, with specific macro keywords instead of specific values. M4 copies an input file to the output and expands macros in the process. All M4 macros defined in this work use capital letter to denote the keyword to make them distinguishable for the reader. Table 2 lists and describes all macros used in current work and Appendix 2 – M4 Macro Definitions shows the actual values of these macros.

Table 2. Descriptions of macros

Macro keyword	**Description**
DB_HOST	Hostname of the database server

DEFAULT_SIP_PORT	Default port used in SIP applications
GATEWAY	IP address of network gateway in the lab setup
INTERNAL_SOFTPHONE_IP	IP address of a softphone located in internal network
INTERNAL_SOFTPHONE2_IP	IP address of a second softphone located in internal network
NETMASK	Length of the network mask
NETMASK_ADDR	Netmask address in decimal notation
NETWORK	IP address of the network
NODE1_CLUSTERER_ID	Identification number of the first OpenSIPS cluster node
NODE1_EXTERNAL_IFACE	External-network-facing network interface name of the first OpenSIPS cluster node
NODE1_EXTERNAL_IP	External-network-facing network interface IP address of the first OpenSIPS cluster node
NODE1_INTERNAL_IFACE	Internal-network-facing network interface name of the first OpenSIPS cluster node
NODE1_INTERNAL_IP	Internal-network-facing network interface IP address of the first OpenSIPS cluster node
NODE2_CLUSTERER_ID	Identification number of the second OpenSIPS cluster node

NODE2_EXTERNAL_IFACE	External-network-facing network interface name of the second OpenSIPS cluster node
NODE2_EXTERNAL_IP	External-network-facing network interface IP address of the second OpenSIPS cluster node
NODE2_INTERNAL_IFACE	Internal-network-facing network interface name of the second OpenSIPS cluster node
NODE2_INTERNAL_IP	Internal-network-facing network interface IP address of the second OpenSIPS cluster node
POSTGRES_U_AND_PWD	Username and password of postgres user in DBHOST
REGISTRAR_IP	IP address of the dedicated Registrar server
REGISTRAR2_IP	IP address of the second Registrar server
REGISTRAR_PORT	Network port of the dedicated Registrar server
VRRP_EXTERNAL_IP	IP address of the external VIP
VRRP_INTERNAL_IP	IP address of the internal VIP

2.2 Network Configuration

Each proxy host has two physical network interfaces that have static IP addresses. Keepalived is used to route traffic from Virtual IP (VIP) addresses to the cluster nodes and Linux policy routing is used to enforce the outgoing traffic to exit through the correct Network Interface Controller (NIC).

2.2.1 VRRP

Keepalived Debian package is installed on each node in the clustered SBC to implement the VRRP. Two VIP addresses are configured on each node: one facing the internal network, another facing the external. The IP phones and other SIP devices in the external network route all traffic to the external VIP. The devices from the internal network route the traffic to the internal VIP (Figure 6).

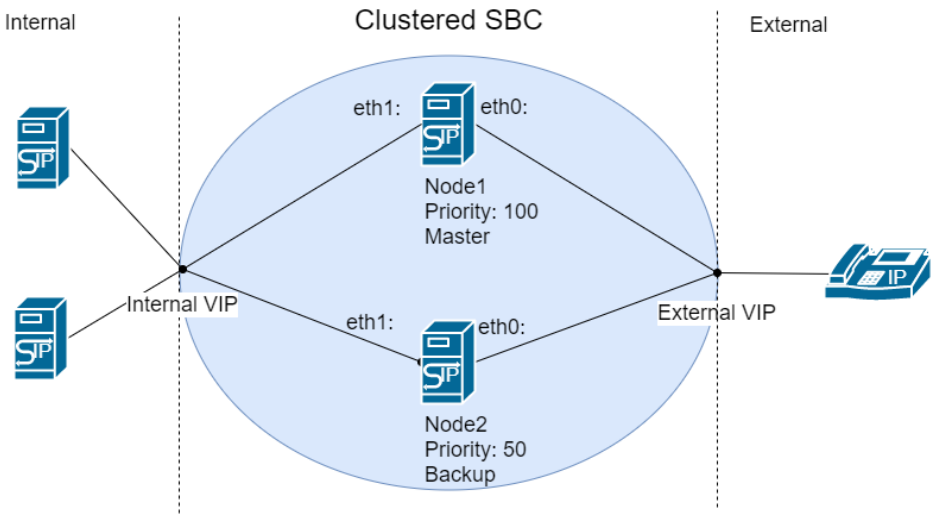

Figure 6. HA cluster using VRRP

Nodes 1 and 2 bind their eth0 interfaces to the external virtual IP address, forming a HA cluster. Network packets addressed to the External VIP will by default be forwarded to Node1, which has higher priority and acts as a master node. In case of Node1 failure, Node2 will take over the master role until Node1 comes back up. Internal VIP acts similarly.

Keepalived configuration template can be seen in Figure 7. This template will first be used to create templates for each proxy host. All M4 macros except VRRP_INTERNAL_IP and VRRP_EXTERNAL_IP will be substituted by a host-specific macro.

```
global_defs {
        enable_script_security
        script_user keepalived_script keepalived_script
}
vrrp_instance VI_1 {
    interface EXTERNAL_IFACE
    state STATE
    virtual_router_id 54
    priority PRIORITY
    advert_int 1
    virtual_ipaddress {
        VRRP_EXTERNAL_IP
    }
    notify /home/keepalived_script/notify-keepalived.sh
}

vrrp_instance VI_2 {
    interface INTERNAL_IFACE
    state STATE
    virtual_router_id 55
    priority PRIORITY
    advert_int 1
    virtual_ipaddress {
        VRRP_INTERNAL_IP
    }
    notify /home/keepalived_script/notify-keepalived.sh
}
```

Figure 7. Keepalived configuration template

Allow Keepalived service to run scripts as keepalived user, bind each physical interface of the machine to a VIP address, specify a script that will run when the cluster node state changes.

Because Keepalived only installs the VIP on the active cluster node, the nodes that are not active do not have the VIP assigned to them. OpenSIPS server software will quit with a failure if it is set up to listen to an IP address that the machine does not have. To fix this, a parameter allowing nonlocal IP binding must be forwarded to Linux kernel. Figure 8 shows the configuration line that must be placed into `/etc/sysctl.conf` file.

```
net.ipv4.ip_nonlocal_bind=1
```

Figure 8. Allowing binding to nonlocal IP addresses

2.2.2 Linux Policy Routing

Keepalived allows the HA cluster act as one machine for incoming traffic. However, the outgoing traffic must also appear to the destination users as if it is coming from one source. For the clustered SBC, the machines in the Internal network must only see the

Internal VIP as the source, and the machines in the External network must only see the External VIP.

In case where both External and Internal networks are in the same subnet, Linux kernel will by default route the traffic through the first configured interface. Linux Policy Routing allows to enforce the traffic originating from the internal VIP address to exit the machine through the internal interface. This manipulation will set the source MAC address of the outgoing packet to the MAC address of the Internal NIC. For this, a routing table must be added to /etc/iproute2/rt_tables file as seen in Figure 9.

```
200 fromInternal
201 fromExternal
```

Figure 9. Two routing tables are specified by an ID and a name

The rules to the routing tables are added to the network interface configuration file /etc/network/interfaces (Figure 10). It is then the responsibility of the SIP Server software to force the IP address of the correct VIP when sending messages.

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto EXTERNAL_IFACE
iface EXTERNAL_IFACE inet static
       address EXTERNAL_IP
       netmask NETMASK_ADDR
       gateway GATEWAY
       post-up ip rule add from VRRP_INTERNAL_IP lookup fromInternal
auto INTERNAL_IFACE
iface INTERNAL_IFACE inet static
       address INTERNAL_IP
       netmask NETMASK_ADDR
       gateway GATEWAY
       post-up ip rule add from VRRP_EXTERNAL_IP lookup fromExternal

post-up ip route add default via GATEWAY dev INTERNAL_IFACE table
fromInternal
post-up ip route add NETWORK/NETMASK dev INTERNAL_IFACE table fromInternal

post-up ip route add default via GATEWAY dev EXTERNAL_IFACE table
fromExternal
post-up ip route add NETWORK/NETMASK dev EXTERNAL_IFACE table fromExternal
```

Figure 10. Network configuration template

Static IP address is configured on both network interfaces. The additional rules instruct to route the packets that arrive from the Internal VIP out through the external interface, and the packets that arrive from the External VIP out through the internal interface.

2.3 SIP Proxy Server Configuration

Each cluster node is running the Debian [34] operation system and the OpenSIPS server software. OpenSIPS was chosen because it was claimed to offer out-of-the-box solutions for topology hiding, load balancing and clustering. The main advantage of OpenSIPS over the similar Kamailio software is that it offers more extensive documentation and tutorials on its website. OpenSIPS is configured via a single configuration script and a database. The configuration file contains global variables, a modules configuration section and a routing script. The database is used to store real-time information and

additional information that is needed to configure OpenSIPS modules. The template for full OpenSIPS configuration file used in this solution is attached in Appendix 3 – OpenSIPS Node Configuration Template. Further subsections will cover important parts of this configuration file.

2.3.1 Data Replication

As described in section 1.6, in case of an outage of a node inside the SBC cluster the new connections will not be affected by it, while the existing connections will be dropped due to the statefulness of the proxy. Data replication between cluster nodes is required to prevent dropping the existing connections in case of an outage of one of the nodes. If the state of the nodes is synchronized in real time, it does not matter which of the nodes processes a request, hence any node can take over as master at any time.

The OpenSIPS clusterer module provides an interface that other modules can use to exchange binary packets between cluster nodes. The other modules must be instructed in the OpenSIPS configuration script to produce or accept data replicates. Figure 11 shows how to configure an OpenSIPS cluster and provides an example of how to modify a module to use this cluster.

```
[…]
listen=bin:INTERNAL_IP:DEFAULT_SIP_PORT
[…]
loadmodule "clusterer.so"
modparam("clusterer", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("clusterer", "current_id", CLUSTERER_CURRENT_ID)
[…]
modparam("dialog", "accept_replicated_dialogs", 1)
modparam("dialog", "replicate_dialogs_to", 1)
modparam("dialog", "replicate_profiles_to", 1)
modparam("dialog", "accept_replicated_profiles", 1)
[…]
```

Figure 11. OpenSIPS cluster configuration

The OpenSIPS server software is configured to listen on the internal IP address for updates from another node. The IP addresses of the cluster nodes are read from the shared database. The modules are instructed to replicate their state to the other nodes and accept updates from them.

2.3.2 Data Reinitialization During Failback

Real-time data replication between cluster nodes enables a seamless master failover because the backup node is constantly aware of the currently running dialogs and able to

take over at any moment. However, in the event of a failback, the new master node must sync the state with the old master node before activation. This functionality is not yet supported by OpenSIPS [35] [36] and is implemented in the current work by using a shared database. As suggested by the developers of OpenSIPS [37], before assigning the VIP to the new master node, the Keepalived software triggers synchronization with the database. The configuration that enables this trigger can be seen in Figure 7 and the script that is run can be seen in Figure 12.

```
#!/bin/bash
TYPE=$1
NAME=$2
STATE=$3
if [ $STATE="MASTER" ]
  then opensipsctl fifo dlg_db_sync
fi
```

Figure 12. OpenSIPS script for state synchronization with a database

If the state that the node is currently transitioning to is called “MASTER”, then OpenSIPS synchronization from the database must be triggered.

2.3.3 Topology Hiding

The topology hiding module of OpenSIPS is configured on both proxy cluster nodes, which enables topology hiding to be applied for each initial request received. This means that all sensitive information concerning the UAC and the route that the request followed to the current proxy is either encoded or replaced by the proxy server contact information. For each subsequent SIP request the proxy performs a lookup over the information stored and tries to fix the request by restoring the initial fields (Figure 13).

The OpenSIPS cluster must also hide its internal topology. For this the server software must force the IP address for outgoing packets. This will enable the Linux Policy Routing to select the correct NIC for the packets to use. The relevant logic is in the *RELAY* function in the OpenSIPS routing script and can be seen in Figure 14.

```
loadmodule "topology_hiding.so"
modparam("topology_hiding", "force_dialog", 1)
[…]
route{
[…]
    if (has_totag()) {
        if (topology_hiding_match()) {
            route(RELAY);
[…]
    topology_hiding("UC");
[…]
    route(RELAY);
}
```

Figure 13. Topology hiding configuration

The script loads topology hiding module and configures it to work on top of dialog module. If the request has a TO tag (it is a subsequent request), then it tries to match the dialog to one of the stored dialogs and to restore initial fields, then relays the request. Else the topology hiding, and request relay is performed.

```
route[RELAY] {
    if (is_method("INVITE|REGISTER")) {
        if ($Ri=="VRRP_EXTERNAL_IP" && $Rp=="DEFAULT_SIP_PORT") {
            route("ToInternal");
         } else if ($Ri=="VRRP_INTERNAL_IP" && $Rp=="DEFAULT_SIP_PORT") {
            route("ToExternal");
        }
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

route[ToInternal] {
    force_send_socket(UDP:VRRP_INTERNAL_IP:DEFAULT_SIP_PORT);
}

route[ToExternal] {
    force_send_socket(UDP:VRRP_EXTERNAL_IP:DEFAULT_SIP_PORT);
}
```

Figure 14. Forcing OpenSIPS to send messages from a specified socket

The script checks if the message was received on External VIP. In this case the source IP in the message header is set to Internal VIP and vice versa.

2.3.4 Data Replication with Topology Hiding

During the experimentation with the described setup, a bug in OpenSIPS was discovered: if a failover occurs during an established call, then all subsequent messages do not reach their destination because they are sent back to the proxy itself in a loop until the allowed number of hops is exceeded. However, the dialog information stored is identical in the failed master (where these messages were processed correctly) and the new master node.

A debugging session revealed that the problem lies in the dialog replication code of the OpenSIPS software. Dialogs are stored as objects. When topology hiding is performed and some the of the dialog information is scrambled, callback functions are stored in the dialog object. These functions are called when processing a subsequent message belonging to this dialog and they restore the scrambled information. However, when the dialogs were replicated to the backup cluster node, the callback functions were not. This led to a situation where although the stored dialogs appeared identical on master and backup nodes, the backup node did not have the necessary information to process the messages.

The bug was reported on the OpenSIPS GitHub page [38] along with a suggestion for a fix. Shortly the bug was fixed by the OpenSIPS developers [39] [40]. Since the bugfix is only available in the development branches of the project, the version available in the Debian repositories will not work for this setup and the project must be compiled from the source code.

2.3.5 Load Balancing

The OpenSIPS software is configured to distribute the outgoing messages amongst the intermediate proxies in the internal network. The distribution algorithm of the *load_balancer* module considers the load of the servers: if there is no available resource in one of the servers, the message will be sent to another one. The amount of resources – i.e. the number of concurrent dialogs – is configured beforehand by a database entry.

The OpenSIPS routing script also handles the case where the initial load balancing selected a proxy that is not available anymore or answers with an error. In this case another destination is picked from the set until a successful one is found, or the set has no more available destinations. This is done by setting a failure route – a function called in

case of a failure – before attempting to relay the message. The *load_balancer* configuration can be seen in Figure 15.

```
[…]
loadmodule "load_balancer.so"
modparam("load_balancer", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("load_balancer", "probing_method", "OPTIONS")
modparam("load_balancer", "probing_interval", 30)
modparam("load_balancer", "replicate_status_to", 1)
modparam("load_balancer", "accept_replicated_status", 1)
[…]
#### INITIAL REQUESTS
    if ( !load_balance("1","pstn")) {
        send_reply("500","No Destination available");
        exit;
    }
    t_on_failure("GW_FAILOVER");
    route(RELAY);
[…]
failure_route[GW_FAILOVER] {
    if (t_was_cancelled()) {
            exit;
    }
    # failure detection with redirect to next available trunk
    if (t_check_status("(408)|([56][0-9][0-9])")) {
        xlog("Failed trunk $rd/$du detected \n");
    if ( lb_next() ) {
        t_on_failure("GW_FAILOVER");
        t_relay();
        exit;
    }
    send_reply("500","All GW are down");
    }
}
```

Figure 15. Load balancing

The load balancer state is replicated to backup nodes. The script looks for suitable destinations from the pre-configured destination set with id "1". If the set is empty or some other error occurs, the 500-error message is relayed; otherwise the initial message is relayed to the selected destination. If the relay is unsuccessful, the *GW_FAILOVER* failure route is followed: *load_balancer* will keep trying the other destinations in the set until it succeeds; otherwise a 500-error message is sent.

The REGISTRAR messages are handled separately and relayed to another set of servers in the internal network – the registrar servers. Due to OpenSIPS limitations, the *load_balancer* module cannot be used for REGISTRAR messages [41]. Distribution of REGISTRAR messages is handled by the *dispatcher* module [42] and the distribution

algorithm is set to random. The logic of the failover route is analogous to the one in *load_balancer* – if the selected server failed to relay the message, next one is picked.

2.3.6 Registration Rate Throttling

The registration throttling is implemented by the *mid_registrar* OpenSIPS module (Figure 16). It is configured to lower the rate of the REGISTER requests and increase their expiration time. It forwards the initial REGISTER request to one of the main registrar servers and performs a local lookup for subsequent requests until the registration expires in the main registrar.

```
[…]
    if (is_method("REGISTER")) {
        mid_registrar_save("location");
        switch ($retcode) {
        case 1:
            xlog("forwarding REGISTER to main registrar ($$ci=$ci)\n");
            if ( !ds_select_dst("1", "6") ) { # setid=1, alg=6 random
                send_reply("500","Unable to route");
                exit;
            }
            xlog("Selected REG trunk $rd/$du \n");
            t_on_failure("REG_FAILOVER");
            route(RELAY);
        case 2:
            xlog("absorbing REGISTER! ($$ci=$ci)\n");
            break;
        default:
            xlog("failed to save registration! ($$ci=$ci)\n");
        }
        exit;
    }
    if (is_method("INVITE|MESSAGE") && ds_is_in_list("$si", "$sp", "1")) {
        xlog("looking up $ru!\n");
        if (!mid_registrar_lookup("location")) {
            t_reply("404", "Not Found");
            exit;
        }
        route(RELAY);
        exit;
    }
```

Figure 16. Mid-registrar configuration

The REGISTER message is absorbed if the contact is already registered in the main registrar and the registration has not yet expired. Otherwise the expiration time of the request is increased and forwarded to the main registrar. Requests from main registrars are forwarded to the intended destinations.

The requests from the main registrar to end-users are handled in a separate case. These are looked up locally and forwarded to the intended destination.

2.3.7 Flood Protection

Flood protection of the SBC is implemented by configuring the OpenSIPS *pike* module [43] (Figure 17). The automatic mode of the pike module detects when packets received from some IP address exceeds some limit. The packet is then processed in a separate route called *check_route* in the OpenSIPS configuration script – similarly to the failure route in *load_balancer.*

In this configuration the *check_route* does not examine the packets that come either from the destinations configured in load_balancer or from the *dispatcher* destinations. This means that the intermediate proxies and registrar servers in the internal network are trusted. If any other host exceeds the allowed threshold, it is blocked.

```
loadmodule "pike.so"
modparam("pike", "check_route", "pike")
[…]
route[pike] {
    if ds_is_in_list("$si", "$sp", "1"){
        drop;
    }
    if lb_is_destination("$si","$sp"){
        drop;
    }
}
```

Figure 17. Pike module configuration

The Pike module is configured to automatically detect spikes in network traffic from any destination and redirect the logic flow into a route called “pike”. The default behaviour is to block the violating source and drop the packets. However, “drop” command in *check_route* instructs the module if this source IP does not need monitoring.

2.4 Database Configuration

A separate machine from the cluster nodes is set to host the shared database. The OpenSIPS database control utility [44] is used to create the database by issuing the opensipsdbctl create command, which must be run manually since it requires interactive input. The database is configured to allow remote access by the OpenSIPS server software from the cluster nodes. It is important to note, that the OpenSIPS versions on the cluster

nodes and the database host must match. The necessary database configuration files and the populating script can be seen in Appendix 4 – Database Configuration Files.

3 System Validation

To validate that the system conforms to requirements of a clustered SBC several modifications to the network were made. The load balancing of the REGISTER messages was tested against two FreePBX [45] Registrar servers with Asterisk [46] backend. A single machine with Debian OS and PostgreSQL database was used as a database server. Two softphones were used as intermediate proxy servers and another one as an IP phone in the external network, referred to as destination softphones and a source softphone, respectively. The proxy and register server fields are set to Internal VIP in the destination softphones and to external VIP in the source softphone.

This simplifies the test setup by eliminating the need to configure at least two more intermediate SIP proxy servers. However, user authorization is not an SBC functionality and should be implemented in the intermediate proxies. Hence, in the current setup it does not matter what number the source softphone dials. Based on its load balancing configuration, the SBC will relay all incoming calls to the destination softphones.

Several test cases were designed and manually executed to validate conformity to the requirements introduced in the Problem Statement. Numeration of connections and names of network elements used in the following test descriptions reference the Figure 18.

Network packets were captured and examined using the Wireshark [47] network protocol analyzer. The OpenSIPS Management Interface [48] was used to examine the internal state of OpenSIPS servers.

In all the test cases described the expected result was achieved. The “requirements covered” section in the test case descriptions follows the requirement numbers in the Problem Statement.

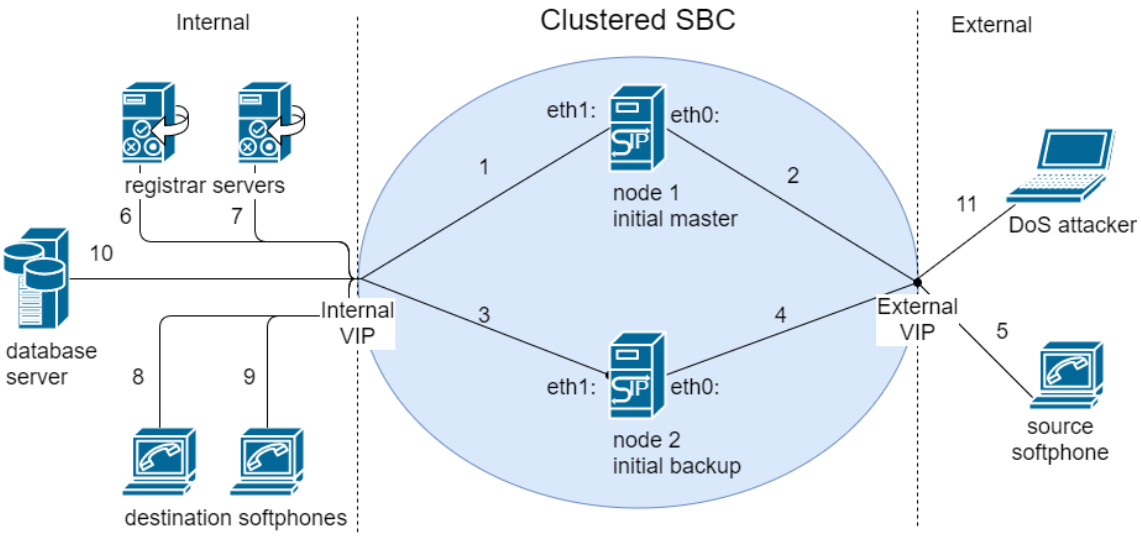

Figure 18. Test setup

Two machines form a HA cluster: their eth0 interfaces bind to a virtual IP address – External VIP. When node 1 is the master, virtual connection 2 is active. Connection 4 is active when node 1 is down and node 2 is the master. A source softphone simulating a device in a client's network is connected to External VIP via connection 5. A DoS attacker machine that simulates a malicious user is connected to External VIP via connection 11. The eth1 interfaces of cluster nodes bind to Internal VIP through connections 1 and 3. Registration requests are routed to registrar server servers via connections 6 and 7. Other messages from the external network are routed through connections 8 and 9 to the destination softphones based on their load. The master node communicates its state to the database server using connection 10.

3.1 Test Case: Call Flow

Description: a new call can be established from the *source softphone* to one of the *destination softphones*. The call can then be paused and ended.

Precondition: the test setup is not in a degraded state – all cluster nodes, registrar servers, destination softphones and the database server are operating correctly, all network connections are up.

Test steps: first, dial any number in the *source softphone*. Then, accept the call in the ringing *destination softphone*. After that, pause the call either in *source* or *destination phone*. Finally, end the call.

Expected results: one of the *destination softphones* rings. After accepting the call, a media session starts. The OpenSIPS management interface command `opensipsctl fifo dlg_list` shows the ongoing dialog in the *node 1*. The call is paused, then ended.

Requirements covered: 2

3.2 Test Case: Call Flow After Failover

Description: a new call can be established from the *source softphone* to one of the *destination softphones* after a cluster failover has occurred. The call can then be paused and ended.

Precondition: the test setup is in a degraded state – *node 1* is disconnected, network *connections 1* and *2* are down, *node 2* becomes the new master node.

Test steps: first, dial any number in the *source softphone*. Then, accept the call in the ringing *destination softphone*. After that, pause the call either in *source* or *destination phone*. Finally, end the call.

Expected results: one of the *destination softphones* rings. After accepting the call, a media session starts. The command `opensipsctl fifo dlg_list` shows the ongoing dialog in the *node 2*. The call is paused, then ended.

Requirements covered: 1, 2

3.3 Test Case: Call Flow After Failback

Description: a new call can be established from the *source softphone* to one of the *destination softphones* after cluster a failback has occurred. The call can be then paused and ended.

Precondition: the cluster failover and failback have occurred. The test setup is not in a degraded state. *Node 1* is currently the master node and *node 2* has reverted to backup state.

Test steps: first, dial any number in the *source softphone*. Then, accept the call in the ringing *destination softphone*. After that, pause the call either in the *source* or *destination phone*. Finally, end the call.

Expected results: one of the *destination softphones* rings. After accepting the call, a media session starts. The command `opensipsctl fifo dlg_list` shows the ongoing dialog in the *node 1*. The call is paused, then ended.

Requirements covered: 1, 2

3.4 Test Case: Topology Hiding – Internal Network

Description: the SIP packets arriving to the *destination softphones* should not contain any information about the *source softphone* or cluster nodes, only about the *VIPs.*

Precondition: clustered SBC is operating.

Test steps: intercept the incoming and outgoing packets in the machine where the *destination softphone* is installed. First, dial any number in the *source softphone*. Then, accept the call in the ringing *destination softphone*. Finally, end call.

Expected results: the source and destination address of the incoming packets are the *Internal VIP* and IP address of the current *destination softphone* correspondingly, and vice versa for the outgoing packets. The SIP message headers do not contain the address of the *source softphone*.

Requirements covered: 2

3.5 Test Case: Topology Hiding – External Network

Description: the SIP packets arriving to the *source softphone* should not contain any information about the *destination softphones* or cluster nodes, only about the *VIPs.*

Precondition: clustered SBC is operating.

Test steps: intercept the incoming and outgoing packets in the machine where the *source softphone* is installed. First, dial any number in the *source softphone*. Then, accept the call in the ringing *destination softphone*. Finally, end call.

Expected results: the source and destination address of the incoming packets are the *External VIP* and IP address of the *source softphone* correspondingly, and vice versa for

the outgoing packets. The SIP message headers do not contain the address of the *destination softphone*.

Requirements covered: 2

3.6 Test Case: Register

Description: users authorized by the main registrar can register to the SBC. Unauthorized users cannot.

Precondition: a test user account is registered in *registrar servers.*

Test steps: first, enter an incorrect user and password into any of the softphones. Then, try to register to SBC. Finally, enter the correct user and password, try to register.

Expected results: the softphone fails to register with the incorrect user and password. The Asterisk log in one of the registrar servers shows a failed registration attempt. The softphone successfully registers with the correct user and password. The Asterisk log shows a successful registration.

Requirements covered: 5

3.7 Test Case: Register Throttling

Description: high-rate incoming REGISTER requests is throttled by the SBC, while the main registrar servers receive low-rate requests.

Precondition: clustered SBC is operating.

Test steps: first, configure all softphones to have the highest registration rate possible. Then, examine the sip proxy logs and the *main registrar* logs.

Expected results: the softphones send out the high-rate REGISTER requests. OpenSIPS log in the active cluster node shows that the requests are relayed at a lower rate, and the high-rate requests are absorbed. The *main registrar* log shows the low-rate requests.

Requirements covered: 5

3.8 Test Case: Register Dispatching

Description: if one of the *registrar servers* is unresponsive, all the REGISTRAR requests are routed to another one.

Precondition: one *Registrar server* is shut down.

Test steps: attempt registration with one of the softphones.

Expected results: registration is successful. The Asterisk log in the operating *registrar server* shows the successful attempt.

Requirements covered: 3, 5

3.9 Test Case: Load Balancing

Description: calls are distributed among the *destination softphones* based on their load.

Precondition: command `opensipsctl fifo dlg_list` shows no active dialogs in the SBC nodes. The softphone software is turned off in the *destination* and *source softphone* machines. One destination is configured to handle ten concurrent calls, the other thirty-two.

Test steps: first, run Sipp in the server scenario on both *destination softphone* machines using command `sipp -sn uas`. Then, run Sipp in the client scenario in the source softphone machine: start forty calls with a pause set to sixty seconds. Start each new call with one-second delay to avoid triggering flood protection mechanism. For the client Sipp scenario use the command `sipp -sn uac EXTERNAL_VIP -m 40 -r 1 -d 60000 -I SOURCE_SOFTPHONE_IP -p 5060`.

Expected results: sipp in the *source softphone* machine shows forty successful calls. Sipp-s in the *destination softphone* machines show at most ten and thirty-two successful calls respectively.

Requirements covered: 3

3.10 Test Case: Denial of Service Attack

Description: a legitimate user can establish a call during a DoS attack.

Precondition: command `opensipsctl fifo dlg_list` shows no active dialogs in the SBC nodes. The softphone software is turned off in the *destination* and *source softphone* machines. Each destination is configured to handle 100 concurrent calls.

Test steps: first, run Sipp in the server scenario on both *destination softphone* machines using command `sipp -sn uas`. Then, in the *DoS attacker* machine run a DoS simulation with Sipp using command `sipp -sn uac EXTERNAL_VIP -p 5060 -m 1000 -r 100 -d 60000`, which tries to establish a thousand new one-minute calls, a hundred every second. After a few seconds delay, in the *source softphone* machine run a legitimate user call with Sipp utilizing the `sipp -sn uac EXTERNAL_VIP -p 5060 -m 1 -d 60000` command, which tries to establish one call lasting sixty seconds.

Expected results: a call is successfully established by the *source softphone.* The OpenSIPS logs in the SBC master node shows the pike module blocking the *DoS attacker*. Most of the requests sent by the *DoS attacker* do not reach the *destination softphones.*

Requirements covered: 4

3.11 Results

The system was configured using an approach inspired by test-driven development. First a test case was designed and executed to validate a new feature. Then the feature was configured until the new as well as previous tests passed. In the process some test cases became obsolete or were merged with new tests. As a result, the final iteration of the system meets the HA and SBC requirements by passing all tests.

4 Conclusions

SIP is a widely used protocol in VoIP applications. Although there are many comprehensive proprietary SIP server solutions available, there is also a need and motivation for open-source alternatives.

This work provides a background on SIP, SIP's security features, HA clusters and the open-source technologies to implement them. Next, it describes an alternative solution to the complex SIP SBC software based on open-source technologies. A customizable deployment script is provided to ease the setup of the solution in production or test environments, alongside with configuration scripts. Successful execution of a series of test cases designed within the scope of this work shows the conformity of the solution to the HA SBC requirements.

During experimentation stage, a bug was discovered that prevented topology hiding in an OpenSIPS cluster. The bug was reported on GitHub along with a suggestion for the fix, which has consequently been accepted by OpenSIPS developers.

As future work, the database server should be transformed into a separate cluster with redundant connections to the SIP SBC to improve the availability of the shared storage. Alternatively, implementing the high availability of the database within the SIP cluster nodes will achieve the same goal. Moreover, testing under near-real-life circumstances would help to reveal potential problems.

References

[1] Oracle Corporation, "Oracle and Acme Packet," [Online]. Available: https://www.oracle.com/corporate/acquisitions/acmepacket/index.html. [Accessed 14 09 2017].

[2] F. Almeida and J. Cruz, "Open source unified communications: The new paradigm to cut costs and extend productivity," in *Proceedings of the Workshop on Open Source and Design of Communication*, Lisboa, 2012.

[3] "openSIPS," [Online]. Available: http://opensips.org/. [Accessed 14 09 2017].

[4] "CLUSTERER Module," [Online]. Available: http://www.opensips.org/html/docs/modules/devel/clusterer.html. [Accessed 14 09 2017].

[5] "Keepalived for Linux," [Online]. Available: http://www.keepalived.org/. [Accessed 14 09 2017].

[6] "topology_hiding Module," [Online]. Available: http://www.opensips.org/html/docs/modules/devel/topology_hiding.html. [Accessed 14 09 2017].

[7] "Load-Balancer Module," [Online]. Available: http://www.opensips.org/html/docs/modules/devel/load_balancer.html. [Accessed 14 09 2017].

[8] "pike Module," [Online]. Available: http://www.opensips.org/html/docs/modules/devel/pike.html. [Accessed 14 09 2017].

[9] "SIPp," [Online]. Available: http://sipp.sourceforge.net/. [Accessed 14 09 2017].

[10] J. Stanek and L. Kencl, "SIPp-DD: SIP DDoS Flood-Attack Simulation Tool," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, Maui, 2011.

[11] ACME PACKET, INC, "ANNUAL REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE ACT OF 1934 For the fiscal year ended December 31, 2008," United States Securities And Exchange Comission, Delaware, 2009.

[12] Oracle, "Oracle Buys Acme Packet," [Online]. Available: http://www.oracle.com/us/corporate/press/1903221. [Accessed 13 11 2017].

[13] Oracle, "Trusted Communications across IP Network Borders," [Online]. Available: https://www.oracle.com/industries/communications/service-providers/products/session-border-controller.html. [Accessed 13 11 2017].

[14] P. Segec and T. Kovacikova, "A Survey of Open Source Products for Building a SIP Communication Platform," *Advances in Multimedia,* vol. 2011, no. 10.1155/2011/372591, 2011.

[15] The Kamailio SIP Server Project, "Kamailio SIP Server," [Online]. Available: https://www.kamailio.org/w/. [Accessed 14 11 2017].

[16] C. A. Thompson, H. A. Latchman, N. Angelacos ja B. K. Pareek, „A Distributed IP-Based Telecommunication System using SIP,“ *International Journal of Computer Networks & Communications (IJCNC),* kd. 5, nr 6, 2013.

[17] FreeSWITCH.org, "FreeSWITCH," 2016. [Online]. Available: https://freeswitch.org. [Accessed 14 11 2017].

[18] Q. Yang, Z.-K. Wei and J.-H. Kim, "A hot backup sip server system based on VRRP," in *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on*, Seoul, 2010.

[19] OpenSIPS, "Available Versions," [Online]. Available: https://www.opensips.org/About/AvailableVersions. [Accessed 15 11 2017].

[20] OpenSIPS, "Modules," [Online]. Available: http://www.opensips.org/Documentation/Modules-2-2. [Accessed 15 11 2017].

[21] "The Smartvox Knowledgebase," [Online]. Available: http://kb.smartvox.co.uk/. [Accessed 15 09 2017].

[22] Smartvox Limited, "Clustering OpenSIPS for High Availability – Part 1," 16 11 2010. [Online]. Available: http://kb.smartvox.co.uk/opensips/clustering-opensips-part-1/. [Accessed 17 11 2017].

[23] Smartvox Limited, "Clustering OpenSIPS for High Availability – Part 2," 23 11 2010. [Online]. Available: http://kb.smartvox.co.uk/opensips/clustering-opensips-part-2/. [Accessed 17 11 207].

[24] Smartvox Limited, "Clustering OpenSIPS for High Availability – Part 3," [Online]. Available: http://kb.smartvox.co.uk/opensips/clustering-opensips-part-3/. [Accessed 15 09 2017].

[25] Smartvox Limited, "Using ClusterLabs Pacemaker with OpenSIPS," 08 06 2017. [Online]. Available: http://kb.smartvox.co.uk/opensips/using-clusterlabs-pacemaker-with-opensips/. [Accessed 17 11 2017].

[26] ClusterLabs, "Pacemaker," [Online]. Available: http://clusterlabs.org/pacemaker.html. [Accessed 17 11 2017].

[27] ClusterLabs, "Corosync," [Online]. Available: http://clusterlabs.org/corosync.html. [Accessed 17 11 2017].

[28] Smartvox Limited, "HA Scenarios and the OpenSIPS Clusterer Module," 28 06 2017. [Online]. Available: http://kb.smartvox.co.uk/opensips/ha-scenarios-and-opensips-clusterer-module/. [Accessed 17 11 2017].

[29] K.Banerjee, "Relation among Call, Dialog, Transaction & Message," 12 04 2015. [Online]. Available: http://www.siptutorial.net/SIP/relation.html. [Accessed 17 11 2017].

[30] "Asnible Automation For Everyone," Red Hat, Inc., [Online]. Available: https://www.ansible.com. [Accessed 09 12 2017].

[31] N. K. Singh, S. Thakur, H. Chaurasiya and H. Nagdev§, "Automated Provisioning of Application in IAAS Cloud using Ansible Configuration Management," in *International Conference on Next Generation Computing Technologies*, Dehradun, 2015.

[32] C. Ebert, J. Hernantes and N. Serrano, "DevOps," *EEE Software,* vol. 33, no. 3, pp. 94-100, 2016.

[33] "GNU M4," Free Software Foundation, Inc., [Online]. Available: https://www.gnu.org/software/m4/m4.html. [Accessed 09 12 2017].

[34] SPI and others, "debian The universal operating system," [Online]. Available: https://www.debian.org. [Accessed 05 12 2017].

[35] "clusterer sync function," [Online]. Available: https://github.com/OpenSIPS/opensips/issues/1194. [Accessed 14 12 2017].

[36] "rvlad-patrascu commented on Oct 18," [Online]. Available: https://github.com/OpenSIPS/opensips/issues/1189#issuecomment-337665365. [Accessed 14 12 2017].

[37] "rvlad-patrascu commented on Oct 26," [Online]. Available: https://github.com/OpenSIPS/opensips/issues/1189#issuecomment-339612358. [Accessed 14 12 2017].

[38] "Sequential topology hiding message not fixed after call failover to backup cluster node," [Online]. Available: https://github.com/OpenSIPS/opensips/issues/1189. [Accessed 16 12 2017].

[39] "dialog: fix a runtime bug with DLGCB_LOADED callbacks," [Online]. Available: https://github.com/OpenSIPS/opensips/commit/8c8f27f6091289061f6aaf0d3c85ccd27db80a0d. [Accessed 20 12 2017].

[40] "dialog: also share module flags when replicating dialogs," [Online]. Available: https://github.com/OpenSIPS/opensips/commit/ad35e7cedc44efd90c58f26a853585ece5883bac. [Accessed 21 12 2017].

[41] "[OpenSIPS-Users] Load Balancer module for REGISTER as well as INVITE?," 20 03 2013. [Online]. Available: http://lists.opensips.org/pipermail/users/2013-March/025062.html. [Accessed 16 12 2017].

[42] "DISPATCHER module," [Online]. Available: http://www.opensips.org/html/docs/modules/2.4.x/dispatcher.html. [Accessed 16 12 2017].

[43] "pike Module," [Online]. Available: http://www.opensips.org/html/docs/modules/devel/pike.html. [Accessed 20 12 2017].

[44] "Database Deployment v2.4," [Online]. Available: https://www.opensips.org/Documentation/Install-DBDeployment-2-4. [Accessed 17 12 2017].

[45] Sangoma Technologies, "FreePBX let freedom ring," [Online]. Available: https://www.freepbx.org. [Accessed 18 12 2017].

[46] "Open Source Communication Software | Asterisk Official Site," [Online]. Available: https://www.asterisk.org. [Accessed 20 12 2017].

[47] "Wireshark - Go Deep.," [Online]. Available: https://www.wireshark.org. [Accessed 12 19 2017].

[48] "openSIPS | Documentation / Management Interface - 2.4," [Online]. Available: https://www.opensips.org/Documentation/Interface-MI-2-4. [Accessed 19 12 2017].

[49] "Graphic by Georgewilliamherbert at English Wikipedia," 10 02 2006. [Online]. Available: https://commons.wikimedia.org/wiki/File:2nodeHAcluster.png. [Accessed 07 11 2017].

[50] Smatrvox Limited, "Using the Clusterer Module for contact replication," 16 11 2010. [Online]. Available: http://kb.smartvox.co.uk/opensips/using-the-clusterer-module-for-contact-replication/. [Accessed 15 09 2017].

Appendix 1 – Ansible Playbook File

```
- name: build opensips
  hosts: all
  tasks:
    - name: "clone opensips"
      git:
        repo: https://github.com/OpenSIPS/opensips.git
        dest: /home/user/opensips
    - name: Install dependencies
      become: true
      apt: name={{item}} state=installed
      with_items:
           - build-essential
           - postgresql
           - postgresql-client
           - linux-headers*
           - flex
           - bison
           - libncurses5-dev
    - name: build OpenSIPS
      shell: TLS=1 make -j4 include_modules="db_postgres" modules
      args:
        chdir: /home/user/opensips
    - name: install opensips
      shell: TLS=1 make include_modules="db_postgres" install
      become: true
      args:
        chdir: /home/user/opensips
- name: Configure opensips proxies
  hosts: proxyhosts
  tasks:
    - name: "CLEANUP: ensure opensips is stopped (and disable it at boot)"
      service: name=opensips state=stopped enabled=no
      become: true
    - name: "VRRP: generate network interfaces config"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 interfaces-{{inventory_hostname}}.m4 >
../{{inventory_hostname}}-interfaces
    - name: "VRRP: configure network interfaces"
      template: src={{item.src}} dest={{item.dest}}
      become: true
      with_items:
```

```
        - {src: "opensips/{{inventory_hostname}}-interfaces", dest:
"/etc/network/interfaces"}
        - {src: "opensips/configs/rt_tables", dest:
"/etc/iproute2/rt_tables"}
    - name: "CLEANUP: ensure network-manager is stopped (and disable it at
boot)"
      service: name=network-manager state=stopped enabled=no
      become: true
    - name: "CLEANUP: kill opensips"
      shell: "pkill -9 opensips || true"
      become: true
    - name: install the latest version of keepalived
      become: true
      package:
        name: keepalived
        state: latest
    - name: "create group keepalived_script"
      become: true
      group:
        name: keepalived_script
        state: present
    - name: "create user keepalived_script"
      become: true
      user:
        name: keepalived_script
        group: keepalived_script
    - name: "VRRP: generate keepalived config"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 keepalived-{{inventory_hostname}}.m4 >
../{{inventory_hostname}}-keepalived.conf
    - name: "VRRP: deploy keepalived config"
      template: src={{item.src}} dest={{item.dest}}
      become: true
      with_items:
        - {src: "opensips/{{inventory_hostname}}-keepalived.conf", dest:
"/etc/keepalived/keepalived.conf"}
        - {src: "opensips/configs/notify-keepalived.sh", dest:
"/home/keepalived_script/notify-keepalived.sh"}
        - {src: "opensips/configs/sysctl.conf", dest:
"/etc/sysctl.d/vrrp.conf"}
    - name: "VRRP: chown keepalived script"
      become: true
      file:
        path: /home/keepalived_script/notify-keepalived.sh
        owner: keepalived_script
        group: keepalived_script
        mode: 755
    - name: "VRRP: allow binding to non-existent ip"
      shell: "sysctl -p /etc/sysctl.d/vrrp.conf"
      become: true
```

```
    - name: "VRRP: ensure keepalived is running "
      service: name=keepalived state=started enabled=yes masked=no
      become: true
    - name: "OPENSIPS: generate opensips config"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 opensips-{{inventory_hostname}}.m4 >
../{{inventory_hostname}}-opensips.conf
    - name : "OPENSIPS: deploy opensips conf"
      template: src={{item.src}} dest={{item.dest}}
      become: true
      with_items:
        - {src: "opensips/{{inventory_hostname}}-opensips.conf", dest:
"/etc/opensips/opensips.cfg"}
        - {src: "opensips/configs/opensipsctlrc", dest:
"/etc/opensips/opensipsctlrc"}
- name: Configure db host
  hosts: dbhosts
  tasks:
    - name: "DB: generate DB config pg_hba"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 pg_hba-template.m4 > ../pg_hba.conf
    - name: "DB: generate DB config populate"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 populate-template.m4 > ../populate.sql
    - name: "DB: generate DB config postgresql"
      local_action:
        module: shell
        chdir: opensips/templates
        _raw_params: m4 postgresql-template.m4 > ../postgresql.conf
    - name : "DB: deploy DB conf"
      template: src={{item.src}} dest={{item.dest}}
      become: true
      with_items:
        - {src: opensips/configs/opensipsctlrc, dest:
/etc/opensips/opensipsctlrc}
        - {src: "opensips/postgresql.conf", dest:
"/etc/postgresql/9.6/main/postgresql.conf"}
        - {src: "opensips/pg_hba.conf", dest:
"/etc/postgresql/9.6/main/pg_hba.conf"}
        - {src: "opensips/populate.sql", dest: "/home/user/populate.sql"}
#@db: opensipsdbctl create
    - name: "populate postgre tables"
      become: postgres
      shell: "psql -U postgres -f /home/user/populate.sql"
    - name: "DB: restart database"
      service: name=postgresql state=restarted
```

```
      become: true
#@registrar:
# - freepbx iso
# - add users and extensions
- name: Restart opensips
  hosts: proxyhosts
  tasks:
    - name: "VRRP: restart keepalived"
      service: name=keepalived state=restarted
      become: true
    - name: "OPENSIPS: start opensips"
      shell: "opensips -n 1 -f /etc/opensips/opensips.cfg"
      become: true
```

Appendix 2 – M4 Macro Definitions

```
define(`DB_HOST', `dbhost')dnl
define(`DEFAULT_SIP_PORT', `5060')dnl
define(`GATEWAY', `10.164.164.1')dnl
define(`INTERNAL_SOFTPHONE_IP', `10.164.164.250')dnl
define(`NETMASK', `23')dnl
define(`NETMASK_ADDR', `255.255.254.0')dnl
define(`NETWORK', `10.164.164.0')dnl
define(`NODE1_CLUSTERER_ID', `1')dnl
define(`NODE1_EXTERNAL_IFACE', `enp0s25')dnl
define(`NODE1_EXTERNAL_IP', `10.164.164.171')dnl
define(`NODE1_INTERNAL_IFACE', `enx00b56d0216c2')dnl
define(`NODE1_INTERNAL_IP', `10.164.165.124')dnl
define(`NODE2_CLUSTERER_ID', `2')dnl
define(`NODE2_EXTERNAL_IFACE', `eth0')dnl
define(`NODE2_EXTERNAL_IP', `10.164.165.118')dnl
define(`NODE2_INTERNAL_IFACE', `eth1')dnl
define(`NODE2_INTERNAL_IP', `10.164.164.75')dnl
define(`POSTGRES_U_AND_PWD', `postgres:12345')dnl
define(`POSTGRESQL_LISTEN_ADDRESS', `*')dnl
define(`REGISTRAR_IP', `10.164.165.145')dnl
define(`REGISTRAR_PORT', `5160')dnl
define(`VRRP_EXTERNAL_IP', `10.164.164.195')dnl
define(`VRRP_INTERNAL_IP', `10.164.164.190')dnl
```

Appendix 3 – OpenSIPS Node Configuration Template

```
####### Global Parameters #########

log_level=3
log_stderror=no
log_facility=LOG_LOCAL0

children=4

/* comment the next line to enable the auto discovery of local aliases
   based on revers DNS on IPs */
auto_aliases=no


listen=udp:VRRP_INTERNAL_IP:DEFAULT_SIP_PORT   # CUSTOMIZE ME
listen=udp:VRRP_EXTERNAL_IP:DEFAULT_SIP_PORT
listen=bin:INTERNAL_IP:DEFAULT_SIP_PORT

####### Modules Section ########

#set module path
mpath="/usr/local/lib64/opensips/modules/"
loadmodule "topology_hiding.so"
```

```
modparam("topology_hiding", "force_dialog", 1)

#### SIGNALING module
loadmodule "signaling.so"

#### StateLess module
loadmodule "sl.so"

#### Transaction Module
loadmodule "tm.so"
modparam("tm", "fr_timeout", 5)
modparam("tm", "fr_inv_timeout", 30)
modparam("tm", "restart_fr_on_each_reply", 0)
modparam("tm", "onreply_avp_mode", 1)

#### Record Route Module
loadmodule "rr.so"
/* do not append from tag to the RR (no need for this script) */
modparam("rr", "append_fromtag", 0)

#### MAX ForWarD module
loadmodule "maxfwd.so"

#### SIP MSG OPerationS module
loadmodule "sipmsgops.so"

#### FIFO Management Interface
loadmodule "mi_fifo.so"
modparam("mi_fifo", "fifo_name", "/tmp/opensips_fifo")
modparam("mi_fifo", "fifo_mode", 0666)

#### URI module
loadmodule "uri.so"
modparam("uri", "use_uri_table", 0)

loadmodule "proto_bin.so"
modparam("proto_bin", "bin_port", 5062)

loadmodule "clusterer.so"
modparam("clusterer", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("clusterer", "current_id", CLUSTERER_CURRENT_ID)
loadmodule "db_postgres.so"

#### AVPOPS module
loadmodule "avpops.so"

#### ACCounting module
loadmodule "acc.so"
/* what special events should be accounted ? */
modparam("acc", "early_media", 0)
```

```
modparam("acc", "report_cancels", 0)
/* by default we do not adjust the direct of the sequential requests.
   if you enable this parameter, be sure the enable "append_fromtag"
   in "rr" module */
modparam("acc", "detect_direction", 0)

#### DIALOG module
loadmodule "dialog.so"
modparam("dialog", "dlg_match_mode", 1)
modparam("dialog", "default_timeout", 21600)  # 6 hours timeout
modparam("dialog", "db_mode", 1)
modparam("dialog", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("dialog", "accept_replicated_dialogs", 1)
modparam("dialog", "replicate_dialogs_to", 1)
modparam("dialog", "replicate_profiles_to", 1)
modparam("dialog", "accept_replicated_profiles", 1)

#### LOAD BALANCER module
loadmodule "load_balancer.so"
modparam("load_balancer", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("load_balancer", "probing_method", "OPTIONS")
modparam("load_balancer", "probing_interval", 30)
modparam("load_balancer", "replicate_status_to", 1)
modparam("load_balancer", "accept_replicated_status", 1)

loadmodule "dispatcher.so"
modparam("dispatcher", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")

loadmodule "proto_udp.so"
loadmodule "usrloc.so"
modparam("usrloc", "db_mode", 0)
modparam("usrloc", "db_url",
"postgres://POSTGRES_U_AND_PWD@DB_HOST/opensips")
modparam("usrloc", "accept_replicated_contacts", 1)
modparam("usrloc", "replicate_contacts_to", 1)

loadmodule "mid_registrar.so"
modparam("mid_registrar", "mode", 1) /* 0 = mirror / 1 = ct / 2 = AoR */
modparam("mid_registrar", "outgoing_expires", 7200)
modparam("mid_registrar", "insertion_mode", 0) /* 0 = contact; 1 = path */

loadmodule "pike.so"
modparam("pike", "check_route", "pike")

######## Routing Logic ########

# main request routing logic
```

```
route{
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    }

    if (has_totag()) {
        if (topology_hiding_match()) {

xlog("============================================================");
            xlog("Succesfully matched this request to a topology hiding
dialog. \n");
            xlog("Calller side callid is $ci \n");
            xlog("Callee side callid  is $TH_callee_callid \n");

xlog("============================================================");
            route(RELAY);
        }
        else {
            xlog("====TOPOLOGY HIDING DID NOT MATCH==================");
            if ( is_method("ACK") ) {
                xlog("L_ERR", "============ METHOD ACK   ============");
                if ( t_check_trans() ) {
                    # non loose-route, but stateful ACK; must be an ACK after
                    # a 487 or e.g. 404 from upstream server
                    t_relay();
                    exit;
                } else {
                    xlog("L_ERR", "ACK WITHOUT MATCHING TRANSACTION =====");
                    # ACK without matching transaction ->
                    # ignore and discard
                    exit;
                }
            }
            sl_send_reply("404","Not here");
        }
        exit;
    }

    #### INITIAL REQUESTS

    if (is_method("REGISTER")) {
        xlog("INITIAL_REGISTER \n");
        mid_registrar_save("location");
        switch ($retcode) {
        case 1:
            xlog("forwarding REGISTER to main registrar ($$ci=$ci)\n");
            if ( !ds_select_dst("1", "6") ) { # setid=1, alg=6 random
                send_reply("500","Unable to route");
                exit;
            }
```

```
        xlog("Selected REG trunk $rd/$du \n");
        t_on_failure("REG_FAILOVER");
        route(RELAY);
    case 2:
        xlog("absorbing REGISTER! ($$ci=$ci)\n");
        break;
    default:
        xlog("failed to save registration! ($$ci=$ci)\n");
    }
    exit;
}
# initial requests from main registrar, need to look them up!
if (is_method("INVITE|MESSAGE") && ds_is_in_list("$si", "$sp", "1")) {
    xlog("looking up $ru!\n");
    if (!mid_registrar_lookup("location")) {
        t_reply("404", "Not Found");
        exit;
    }
    route(RELAY);
    exit;
}

# CANCEL processing
if (is_method("CANCEL")) {
    if (t_check_trans())
        route(RELAY);
        exit;
}
else if (!is_method("INVITE")) {
    send_reply("405","Method Not Allowed");
    exit;
}
else {
    create_dialog();
}

if ($rU==NULL) {
    # request with no Username in RURI
    sl_send_reply("484","Address Incomplete");
    exit;
}

t_check_trans();

# preloaded route checking
if (loose_route()) {
    xlog("L_ERR",
    "Attempt to route with preloaded Route's [$fu/$tu/$ru/$ci]");
    if (!is_method("ACK"))
        sl_send_reply("403","Preload Route denied");
    exit;
```

```
    }

    # record routing
    record_route();

    do_accounting("log");

    topology_hiding("UC");
    if ( !load_balance("1","pstn")) {
        send_reply("500","No Destination available");
        exit;
    }
    t_on_failure("GW_FAILOVER");
    route(RELAY);
}

route[RELAY] {
    if (is_method("INVITE|REGISTER")) {
        if ($Ri=="VRRP_EXTERNAL_IP" && $Rp=="DEFAULT_SIP_PORT") {
            xlog("L_INFO","ToInternal message");
            route("ToInternal");
         } else if ($Ri=="VRRP_INTERNAL_IP" && $Rp=="DEFAULT_SIP_PORT") {
            xlog("L_INFO","ToExternal message");
            route("ToExternal");
        }
    }
    if (!t_relay()) {
        sl_reply_error();
    };
    exit;
}

route[ToInternal] {
    xlog("L_INFO","route(ToInternal)");
    force_send_socket(UDP:VRRP_INTERNAL_IP:DEFAULT_SIP_PORT);
}

route[ToExternal] {
    xlog("L_INFO","route(ToExternal)");
    force_send_socket(UDP:VRRP_EXTERNAL_IP:DEFAULT_SIP_PORT);
}

route[pike] {
    if ds_is_in_list("$si", "$sp", "1"){
        drop;
    }
    if lb_is_destination("$si","$sp"){
        drop;
    }
}
```

```
failure_route[GW_FAILOVER] {
    if (t_was_cancelled()) {
            exit;
    }
    # failure detection with redirect to next available trunk
    if (t_check_status("(408)|([56][0-9][0-9])")) {
        xlog("Failed trunk $rd/$du detected \n");
        if ( lb_next() ) {
            t_on_failure("GW_FAILOVER");
            t_relay();
            exit;
        }
        send_reply("500","All GW are down");
    }
}

failure_route[REG_FAILOVER] {
    xlog("REG_FAILOVER \n");
    if (t_was_cancelled()) {
            exit;
    }
    # failure detection with redirect to next available trunk
    if (t_check_status("(408)|([56][0-9][0-9])")) {
        xlog("REG server answered with a failure \n");
        if ( ds_next_dst() ) {

            t_on_failure("REG_FAILOVER");
            xlog("DS_NEXT_DST \n");
            t_relay();
            exit;
        }
        send_reply("500","All REG GW are down");
    }
}

local_route {
    if (is_method("BYE") && $DLG_dir=="UPSTREAM") {

        acc_log_request("200 Dialog Timeout");
    }
}
```

Appendix 4 – Database Configuration Files

```
DBENGINE=PGSQL
DBHOST=localhost
DBNAME=opensips
DBRWUSER=opensips
DBRWPW="opensipsrw"
```

Figure 19. Opensipsctlrc file

Specifies that Postgres database is by OpenSIPS used on current machine. Provides database name and credentials to access it.

```
include(`defines.m4')dnl
host all all NETWORK/NETMASK trust
host all all 127.0.0.1/8 trust
local all all trust
```

Figure 20. Allow access to the database from localhost and the same network

```
include(`defines.m4')dnl
\c opensips
DELETE FROM load_balancer;
DELETE FROM clusterer;
DELETE FROM dispatcher;
INSERT INTO load_balancer (id, group_id, dst_uri, resources) VALUES (1, 1,
'sip:INTERNAL_SOFTPHONE_IP:DEFAULT_SIP_PORT', 'pstn=32');

INSERT INTO load_balancer (id, group_id, dst_uri, resources) VALUES (2, 1,
'sip:INTERNAL_SOFTPHONE2_IP:DEFAULT_SIP_PORT', 'pstn=32');

INSERT INTO clusterer (id, cluster_id, node_id, url, description) VALUES
(1, 1, NODE1_CLUSTERER_ID, 'NODE1_INTERNAL_IP:DEFAULT_SIP_PORT', 'Node
1');

INSERT INTO clusterer (id, cluster_id, node_id, url, description) VALUES
(2, 1, NODE2_CLUSTERER_ID, 'NODE2_INTERNAL_IP:DEFAULT_SIP_PORT', 'Node
2');

INSERT INTO dispatcher (setid, destination) VALUES (1,
'sip:REGISTRAR_IP:REGISTRAR_PORT');

INSERT INTO dispatcher (setid, destination) VALUES (1,
'sip:REGISTRAR2_IP:REGISTRAR_PORT');
```

Figure 21. Postgres populate script template