

Aalto University  
School of Electrical Engineering  
Department of Communications and Networking

**Muhammad Hassaan Bin Mohsin**

# **Security Policy Management for a Cooperative Firewall**

Master's Thesis

Espoo, September 4, 2018

Supervisor: Prof. Raimo Kantola  
Aalto University

Thesis advisor(s): M.Sc. Hammad Kabir  
Aalto University

|  |                          |                                  |
|--|--------------------------|----------------------------------|
| <b>Author:</b> Muhammad Hassaan Bin Mohsin   |                          |                                  |
| <b>Name of the Thesis:</b> Security Management System for a Cooperative Firewall   |                          |                                  |
| <b>Date:</b> 06.06.2018  | <b>Language:</b> English | <b>Number of pages:</b> XII + 70 |
| <b>School:</b> School of Electrical Engineering  |                          |                                  |
| <b>Department:</b> Department of Communications and Networking   |                          |                                  |
| <b>Professorship:</b> Networking Technology  |                          | <b>Code:</b> S.38                |
| <b>Supervisor:</b> Prof. Raimo Kantola, Aalto University   |                          |                                  |
| <b>Instructor:</b> M.Sc Hammad Kabir, Aalto University   |                          |                                  |
| <p>Increasing popularity of the Internet service and increased number of connected devices along with the introduction of IoT are making the society ever more dependent on the Internet services availability. Therefore, we need to ensure the minimum level of security and reliability of services. Ultra-Reliable Communication (URC) refers to the availability of life and business critical services nearly 100 percent of the time. These requirements are an integral part of upcoming 5th generation (5G) mobile networks.</p> <p>5G is the future mobile network, which at the same time is part of the future Internet. As an extension to the conventional communication architecture, 5G needs to provide ultra-high reliability of services where; it needs to perform better than the currently available solutions in terms of security, confidentiality, integrity and reliability and it should mitigate the risks of internet attack and malicious activities. To achieve such requirements, Customer Edge Switching (CES) architecture is presented. It proposes that the internet service provider needs to have prior information about the expected traffic of users to mitigate maximum attacks and only allow expected communication between hosts. CES executes communication security policies of each user or device. The policy describes with fine granularity what traffic is expected by the device. The policies are sourced as automatically as possible but can also be modified by the user. Stored policies will follow the mobile user and will be executed at the network edge node executing Customer Edge Switch functions to stop all unexpected traffic from entering the mobile network.</p> <p>State-of-the-art in mobile network architectures utilizes the Quality of Service (QoS) policies of users. This thesis motivates the extension of current architecture to accommodate security and communication policy of end-users. The thesis presents an experimental implementation of a policy management system which is termed as Security Policy Management (SPM) to handle above mentioned policies of users. We describe the architecture, implementation and integration of SPM with the Customer Edge Switching. Additionally, SPM has been evaluated in terms of performance, scalability, reliability and security offered via 5G customer edge nodes. Finally, the system has been analyzed for feasibility in the 5G architecture.</p> |                          |                                  |
| <b>Keywords:</b> IP, PRGW, CES, Security, Firewall, Policy, Database, Management, Security, PMS  |                          |                                  |

## Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful.

All praises to the Almighty Allah, for strengthening me and his continuous blessing on me through my hardships of life. A special praise to Prophet Muhammad (Peace be upon Him) who is forever a source of inspiration, guidance and a role model of peace and kindness for the whole humanity. I would dedicate this hard work to my grandfather and late grandmother whom I owe this success. Their prayers, motivation and trust in me was a source of encouragement to excel in life.

It was an honor to be trusted and relied on by Professor Raimo Kantalo, who has put faith in me and provided me with an opportunity to work under his professional and tactful guidance, and with his competent research group. His broad knowledge and experience in the field of my research improved my vision and approach towards technical designing and implementation.

I am also grateful to my supervisor Hammad Kabir who stood by me throughout my thesis with his technical insight in the subject and professional attitude towards the implementation approach of system. His continuous feedback regarding my thesis implementation, testing and integration was a vital source of incremental improvement of my technical skills and the thesis.

Finally, I would like to thank my parents and sister for relying on me and supporting me in building my career and acknowledging my achievements. Their love, support, prayers and kindness has always been a great source of strength. I would like to give a special thanks to Hussain Parsaiyan who became a source of guidance throughout my thesis and smoothened my transition towards the field of development. The continuous support of my friends Umar, Ahsan and Muneeb for motivating and cheering me through my thesis and during the stay in Finland was helpful in maintaining my enthusiasm for diverse efforts.

Muhammad Hassaan Bin Mohsin

Dated: 2<sup>nd</sup> September 2018

## Table of contents

|   |    |
|---|----|
| Acknowledgements.....                                     | 3  |
| Table of contents.....                                    | 4  |
| List of figures.....                                      | 7  |
| Acronyms.....   | 8  |
| 1. Introduction.....                                      | 11 |
| 1.1 Research problem.....                                 | 12 |
| 1.2 Objective.....  | 13 |
| 1.3 Scope.....  | 13 |
| 1.4 Structure.....  | 14 |
| 2. Background.....  | 15 |
| 2.1 4G/LTE architecture .....                             | 15 |
| 2.1.1 HSS.....  | 17 |
| 2.1.2 Policy and Charging Control (PCC) architecture..... | 18 |
| 2.1.3 PCC components .....                                | 20 |
| 2.1.4 Interfaces and protocols .....                      | 22 |
| 2.2 Firewalls.....  | 24 |
| 2.2.1 Introduction.....                                   | 24 |
| 2.2.2 Working methodologies and limitations.....          | 26 |
| 2.2.3 Firewall types .....                                | 28 |
| 2.2.4 SPM for firewall.....                               | 31 |
| 2.3 Policy Management System (PMS).....                   | 31 |
| 2.3.1 Terminologies:.....                                 | 32 |
| 2.3.2 Variants .....                                      | 34 |
| 2.3.3 Variant's limitations.....                          | 36 |
| 2.3.4 Research.....                                       | 37 |
| 3. Customer Edge Switching (CES) .....                    | 39 |
| 3.1 Motivation .....                                      | 39 |
| 3.2 Architecture .....                                    | 41 |
| 3.3 Communication and protocols.....                      | 42 |
| 3.3.1 Inter-CES communication.....                        | 42 |
| 3.3.2 Intra-CES communication:.....                       | 43 |
| 3.3.3 PRGW and legacy host.....                           | 44 |
| 3.4 Policy information.....                               | 45 |

|       |   |    |
|-------|---|----|
| 3.4.1 | Types of policies .....                 | 45 |
| 3.4.2 | Management Scope.....                   | 47 |
| 3.5   | Conclusion (and SPM requirements) ..... | 48 |
| 4.    | Proposed architecture .....             | 49 |
| 4.1   | Design.....                             | 49 |
| 4.2   | Policy-Database schema.....             | 50 |
| 4.2.1 | Bootstrap policies.....                 | 51 |
| 4.2.2 | Session policies.....                   | 52 |
| 4.2.3 | Host Policies.....                      | 53 |
| 4.2.4 | CES Policies .....                      | 54 |
| 5.    | Implementation .....                    | 55 |
| 5.1   | Tools and specifications .....          | 55 |
| 5.2   | System architecture .....               | 56 |
| 5.2.1 | Policy-API.....                         | 57 |
| 5.2.2 | Django-webserver .....                  | 61 |
| 5.2.3 | Policy Management Tool (PMT).....       | 62 |
| 5.2.4 | Policy-Database.....                    | 63 |
| 5.3   | Additional features .....               | 64 |
| 5.3.1 | Reputation service.....                 | 64 |
| 5.3.2 | User registration and ID service.....   | 64 |
| 5.4   | Optimizations.....                      | 65 |
| 6.    | API manual .....                        | 66 |
| 6.1   | Code Structure .....                    | 66 |
| 6.1.1 | SPM.....                                | 66 |
| 6.1.2 | Django_webserver.....                   | 68 |
| 6.2   | SPM Setup.....                          | 68 |
| 6.3   | API end points .....                    | 69 |
| 6.3.1 | Retrieval API for CES .....             | 69 |
| 6.3.2 | Policy Management API .....             | 70 |
| 7.    | Results and evaluation.....             | 72 |
| 7.1   | Tools benchmark .....                   | 72 |
| 7.2   | Security and integrity .....            | 73 |
| 7.3   | Performance.....                        | 74 |
| 7.3.1 | Test environment.....                   | 74 |

|       |                                     |    |
|-------|-------------------------------------|----|
| 7.3.2 | Policy management performance ..... | 75 |
| 7.3.3 | Policy retrieval performance .....  | 79 |
| 7.4   | Scalability.....                    | 81 |
| 7.5   | Integration testing with CES.....   | 82 |
| 8.    | Discussion.....                     | 84 |
| 9.    | Conclusion.....                     | 85 |
| 10.   | Future works.....                   | 86 |

## List of figures:

|  |    |
|--|----|
| Figure 1- Scope of Thesis.....   | 14 |
| Figure 2- 4G Network building blocks and architecture <sup>1</sup> .....         | 16 |
| Figure 3- Components of Policy & Charging Control <sup>2</sup> .....             | 18 |
| Figure 4- Firewall Placement for separation of LAN and WAN <sup>5</sup> .....    | 25 |
| Figure 5- Types of firewalls.....  | 28 |
| Figure 6- Cisco Security Manager Features <sup>7</sup> .....                     | 35 |
| Figure 7- NAT Architecture <sup>13</sup> .....                                   | 40 |
| Figure 8- Architecture of CES [26] .....   | 41 |
| Figure 9- CES to CES connection establishment [24].....                          | 43 |
| Figure 10- Inbound Communication between Legacy IP and CES host .....            | 44 |
| Figure 11- CES-CES sample policy.....  | 46 |
| Figure 12- Host-Host sample policy.....  | 46 |
| Figure 13 - Sample firewall policy .....   | 47 |
| Figure 14- Basic SPM structure .....   | 49 |
| Figure 15- LTE Architecture with SPM .....                                       | 50 |
| Figure 16- Bootstrap_Policies database schema .....                              | 51 |
| Figure 17- Session_Policies database schema.....                                 | 52 |
| Figure 18 - SPM nodes in CES Network Architecture .....                          | 56 |
| Figure 19 - Implementation hierarchy of SPM.....                                 | 58 |
| Figure 20 – Flow diagram for Request handling at API.....                        | 60 |
| Figure 21 - Architecture of Django Framework <sup>14</sup> .....                 | 62 |
| Figure 22 - Latency per packet for HTTP GET request by CES node.....             | 80 |
| Figure 23 - Impact of SPM integration on CETP-Policy negotiation at CES-CP ..... | 83 |

## Acronyms

|         |   |
|---------|---|
| 3GPP    | 3rd Generation Partnership Project            |
| 4G      | 4 <sup>th</sup> Generation                    |
| 5G      | 5 <sup>th</sup> Generation                    |
| AAA     | Authentication, Authorization & Accounting    |
| ACL     | Access Control List                           |
| AF      | Application Function                          |
| API     | Application Programming Interface             |
| ARP     | Allocation and Retention Priority             |
| AuC     | Authentication Center                         |
| AVP     | Attribute Value Pair                          |
| BBERF   | Bearer Binding and Event Reporting Function   |
| BTS     | Base Transceiver Station                      |
| C2C     | CES to CES                                    |
| CCR     | Credit Check Request                          |
| CDR     | Charging Data Records                         |
| CES     | Customer Edge Switching                       |
| CETP    | Customer Edge Traversal Protocol              |
| CN      | Customer Network                              |
| CP      | Control Plane                                 |
| CSM     | Cisco Security Manager                        |
| CSRF    | Cross Site Request Forgery                    |
| CSS     | Cascading Style Sheets                        |
| DCCA    | Diameter Credit Control Application           |
| DDNS    | Dynamic Domain Name System                    |
| DHCP    | Dynamic Host Configuration Protocol           |
| DNAT    | Destination Network Address Translation       |
| DNS     | Domain Name System                            |
| DP      | Data Plane                                    |
| DPI     | Deep Packet Inspection                        |
| E-UTRAN | Evolved UMTS Terrestrial Radio Access Network |
| EDGE    | Enhanced Data for GSM Evolution               |
| EPC     | Evolved Packet Core                           |
| FCFS    | First Come First Served                       |
| FQDN    | Fully Qualified Domain Name                   |
| FTP     | File Transfer Protocol                        |
| GPRS    | General Packet Radio Service                  |
| GSM     | Global System for Mobile Communications       |
| GTP     | GPRS Tunneling Protocol                       |
| GUI     | Graphical User Interface                      |
| H2H     | Host to Host                                  |
| HLR     | Home Location Register                        |
| HSDPA   | High-Speed Downlink Packet Access             |
| HSS     | Home Subscriber Server                        |
| HTML    | Hypertext Markup Language                     |
| HTTP    | Hyper Text Transfer Protocol                  |
| HTTPS   | Hypertext Transfer Protocol Secure            |



|        |   |
|--------|---|
| ICMP   | Internet Control Message Protocol                     |
| IMSI   | International Mobile Subscriber Identity              |
| IoT    | Internet of Things                                    |
| IP-CAN | Internet Protocol Connectivity Access Network         |
| IP     | Internet Protocol                                     |
| IPSec  | Internet Protocol Security                            |
| ISMI   | International Mobile Subscriber Identity              |
| ISP    | Internet Service Provider                             |
| JSON   | JavaScript Object Notation                            |
| KB     | Kilo Bytes  |
| LAN    | Local Area Network                                    |
| LDAP   | Lightweight Directory Access Protocol                 |
| LTE    | Long-Term Evolution                                   |
| MME    | Mobility Management Entity                            |
| ms     | Millisecond   |
| MSISDN | Mobile Subscriber Integrated Services Digital Network |
| MTU    | Maximum Transmission Unit                             |
| MVC    | Model View Controller                                 |
| NAT    | Network Address Translation                           |
| OCS    | Online Charging System                                |
| OFCS   | Offline Charging System                               |
| OSI    | Open Systems Interconnection                          |
| P-GW   | Packet Gateway  |
| PCC    | Policy and Charging Control                           |
| PCEF   | Policy and Charging Enforcement Function              |
| PCRF   | Policy and Charging Rules Function                    |
| PDE    | Policy Decision Engine                                |
| PDF    | Policy Decision Function                              |
| PDN    | Packet Data Network                                   |
| PDP    | Policy Decision Point                                 |
| PEP    | Policy Enforcement Point                              |
| PMIP   | Proxy Mobile Internet Protocol                        |
| PMS    | Policy Management System                              |
| PMT    | Policy Management Tool                                |
| PRGW   | Private Realm Gateway                                 |
| QCI    | Quality Control Information                           |
| QoE    | Quality of Experience                                 |
| QoS    | Quality of Service                                    |
| RAA    | Re-Auth Answer  |
| RAM    | Random Access Memory                                  |
| RAR    | Re-Auth Request                                       |
| REST   | Representational State Transfer                       |
| RLOC   | Routing Locator                                       |
| S-GW   | Serving Gateway                                       |
| SATA   | Serial Advanced Technology Attachment                 |
| SDF    | Service Data Flow                                     |
| SDN    | Software Defined Networking                           |

|       |  |
|-------|--|
| SIM   | Subscriber Identity Module             |
| SLF   | Subscriber Location Function           |
| SMS   | Short Message Service                  |
| SNAT  | Source Network Address Translation     |
| SPM   | Security Policy Management             |
| SPN   | Service Provider Network               |
| SPR   | Subscriber Profile Repository          |
| SSH   | Secure Shell                           |
| SSL   | Secure Socket Layer                    |
| STUN  | Session Traversal Utilities for NAT    |
| TB    | Tera Bytes                             |
| TCP   | Transmission Control Protocol          |
| TURN  | Traversal Using Relays around NAT      |
| UE    | User Equipment                         |
| UDP   | User Datagram Protocol                 |
| UI    | User Interface                         |
| URC   | Ultra-Reliable Communication           |
| URI:  | Universal Resource Identifier          |
| URS   | Ultra-Reliable Service                 |
| UUID  | Unique User Identifier                 |
| VPN   | Virtual Private Network                |
| WAN   | Wide Area Network                      |
| WCDMA | Wideband Code Division Multiple Access |

## 1. Introduction

Mobile technology is revolutionizing constantly, thus transforming a voice network to a data network. 2G Global System for Mobile (GSM) provided an initial mobile-data service with low-rate data communication using signaling channels, known as Short Message Service (SMS). With upgrades of General Packet Radio Service (GPRS) and Enhanced Data for GSM Evolution (EDGE) for Internet connectivity, the existing GSM architecture achieved the data rates of up to 384 kbps. The first version of 3G, Wideband Code Division Multiple Access (WCDMA), was only able to offer 384 Kbps [1]. High-Speed Downlink Packet Access (HSDPA) was used to improve and extend data rates of 3G to 14.4 Mbit/s [1]. It was then called as the wireless broadband because of the considerably higher data rate. With such modifications, 3G was the hybrid of data and voice network deployed on existing GSM architecture along with software upgrades in Base Transceiver Stations (BTS). The targets accomplished were short-lived and soon there was a need to achieve even higher data rates, lower latency and increased scalability.

The technology evolved with continuous upgrades in data transfer rates, latency and scalability. This, in turn, helped to confine one's world in his pocket. 4G, the last deployed upgrade, has achieved even faster data rates and now a user is able to access nearly all services on his phone such as online banking, email, multimedia, social media and TV on-demand and in nearly real-time. 4G has fundamentally replaced the core network with data infrastructures created for lower latency and better use of resources, therefore providing a higher Quality of Service (QoS). Data rates in 4G are capable of providing throughputs of up to 100Mbps during high mobility which satisfies the need for bandwidth-demanding applications such as YouTube. Retaining the confidentiality and integrity of data at last mile is important which requires the deployment of Internet Protocol Security (IPSec) cell sites in Long-Term Evolution (LTE) networks. However, LTE core network is an all Internet Protocol (IP) based network, which exposes it to millions of attacking devices as well as attacks originating from around the globe. This requires security measures in place to ensure uninterrupted and secure delivery of services [2][3].

4G is designed for consumer markets targeting humans as users. Researchers and developers have mostly focused on providing end users with high data rates, lower latencies, and better software support, but major documents regarding newer and upcoming standards of mobile communication technology lack details about safety, adaptability and elasticity of the network. During the past few years, Internet usage and the nature of users have changed significantly and this has shifted the responsibility of maintaining security, confidentiality and privacy of the users to the operators and network administrators.

Simultaneously, the next generation namely 5G is expected to serve the needs of upcoming applications such as IoT, automated driving, Industrial Internet etc. This, in turn, gives rise to the introduction of Ultra Reliable Communication (URC) in 5G [4] which demands the availability of resources for almost 100 percent of the time. While 5G is developed for consumer and corporate markets to support digitalization of economics, industry and society, at the same time, 5G extensively relies on Internet technologies and we have learned to know the Internet as an inherently unreliable and insecure network. Therefore, 5G should make significant progress in the field of security as compared to the current state-of-the-art in the Internet. Reliability and security are interlinked because malicious acts by hackers on the Internet can take down legitimate services, and such acts on the current internet are inherently unpredictable. Recalling that reliability is the probability of the lack of failures for any reason implies that reaching ultra-high-reliability of services in 5G means that it should be either very hard or impossible to take down reliable services by malicious activities (at least based on cheap and almost trivial attacks).

We have proposed the Customer Edge Switching (CES) as a new architecture for the 5G and the internet, as a component of URC for connecting the 5G core network to the Internet and other legacy IP networks. Developed at Aalto University, it helps to reach ultra-high reliability of services through policy-based communication. In this approach, when two devices or services (in the cloud) are communicating, a chain of trust on a suitable level of assurance is formed between the communicating parties. Each device has an agent (i.e., CES) in the cloud that knows expected traffic towards the served device and discards all the other traffic as anomalous or malicious. In the system, all flows to and from users are admitted based on the defined policy. CES nodes are cooperative firewalls, and they cooperatively arrive at the final admit/drop decision by exchanging user specific communication policies.

## **1.1 Research problem**

A policy is a set of conditions that needs to be fulfilled before an action can take place. In traditional networks, policies are used to control aspects of communication such as Quality of Service (QoS), security or access to data. In this thesis, we introduce and focus on communications security policies. A communication security policy is set of conditions applied in the network firewall to device admission of a flow or a packet to or from an end device. In this thesis, we use the term policy in the context of Security Policy Management (SPM) to refer to a communication security policy. These policies need to be defined on different levels of abstraction for end-users. An executable policy is bound to IP-addresses, port, protocol numbers and possibly to other fields of an IP or higher layer protocol that form the matching criteria. The policy is applied when the conditions in the policy match the fields in the packet.

There is a need to store these policies in the database usually on a more abstract level – instead of being bound to binary fields of a packet. An abstract policy may be bound to textual names. We intend to design and verify a Security Policy Management that stores and manages the user policies executable at Customer Edge Switching (CES) nodes.

Thus, there is a need to have a policy database that handles frequent read and write operations and populate changes on runtime in the network. Moreover, the database should maintain a record of subscribers belonging to different groups and shall ensure precedence of network policies over user policies.

## **1.2 Objective**

The aim of this thesis is to develop a Security Policy Management (SPM) system for storage, retrieval, deletion and editing of policies at runtime and populate changes all the way to CES firewalls for execution on demand. The database design would be capable of providing services to multiple CES nodes thus allowing sharing of resources. A Graphical User Interface (GUI) would be developed for management of policies by users, company administrators or network managers. In this thesis, we seek to define the principles for demarcation of responsibilities over the policy definition between user and domain administrators.

The objective of this thesis is to:

- a) design, implement and test the SPM design
- b) test the security, integrity and reliability of system against malicious policies
- c) evaluate the performance and scalability of system
- d) split the responsibilities between the domain administrators and the end-users for defining the content of policies.

## **1.3 Scope**

Figure 1 shows the graphical representation of the scope.

This thesis focuses on developing the back-end and front-end of a Security Policy Management (SPM) system. The thesis scope excludes the policy sourcing to the SPM. Moreover, the charging and other rules defined in Policy and Charging Rules Function (PCRF) are not in the scope of this thesis.

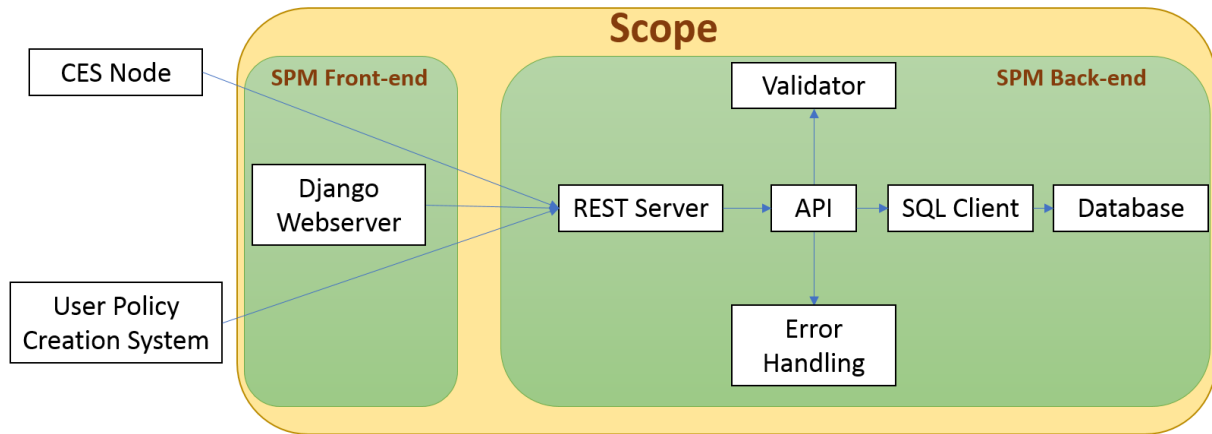


Figure 1- Scope of Thesis

This thesis focuses on the use case of Mobile Broadband in the design of the SPM and Policy-Database itself. Our working hypothesis is that by modifying some of the components and possibly redefining the policy hierarchies, we will be able to adjust the SPM and the policy control (and hence cooperative firewalling) to other use cases such as the ultra-reliable machine-to-machine communications.

## 1.4 Structure

The thesis is divided as follows.

Chapter 0 provides a brief review about the data repository in classical mobile telecommunication technologies and technical overview of firewalls. The chapter also includes the test cases used to focus the need for a separate database and management system. Chapter 3 discusses the architecture of Customer Edge Switching and its functional nodes to analyze the requirements for the Policy-Database schema. Chapter 0 proposes the architecture of SPM and schema for Policy-Database required to store security and network management policies. The chapter also discusses the possible available technologies and finalizes the technical specifications of implementation through a comparison between technologies. Chapter 0 explains the implementation of the system, hierarchy of network, the interconnection between nodes, and optimizations performed to achieve better performance and scalability results.

Chapter 6 presents a user manual of Application Programming Interface (API) of SPM for its usage, expansion and compatibility with different nodes and systems. Chapter 7 discusses the performance, scalability and security test results of API and database. Chapter 8 evaluates the SPM and its comparison to the database and management functions of LTE. Chapter 9 concludes the thesis.

## 2. Background

This chapter highlights the architecture of 4G LTE, its Policy Management and firewall operations. We study the already implemented infrastructure and its operation. We also describe the current firewalls, their different types and implementation approaches. Finally, we review the existing Policy Management Systems (PMS) used for handling of firewalls and their expansion possibilities to accommodate the CES requirements.

### 2.1 4G/LTE architecture

4G LTE is the currently deployed mobile network architecture, which drastically improved the data rates to the end users as well as the Quality of Experience. This required a considerable network change and a heavy investment from mobile operators as the core and access networks both required hardware modifications. The change involves the transformation of circuit-switched voice network to a packet-based core, treating user traffic as data packets. The packet-based core improved the utilization of network components by sharing resources among the services. The data-rates provided by 4G allow devices to smoothly run general office and academic applications, and even to stream HD videos. The core network in LTE stores the policies for maintaining end-user Quality of Experience and establishing (or maintaining) sessions.

The entire 4G network can be decomposed into 4 big blocks which are interconnected to give the final shape of the architecture. The blocks include UE, E-UTRAN, EPC and server Packet Data Networks (PDNs). The fourth element (PDNs) is considered as outside network.

1. User Equipment (UE)
2. Evolved UMTS Terrestrial Radio Access Network (E-UTRAN)
3. The Evolved Packet Core (EPC)

Figure 2 shows the constitutional components of the 4G network. UE communicates with E-UTRAN which is responsible for access level management that includes mobility management of UE, authentication of devices and wireless communication. User sessions are created and maintained in the EPC with the help of policies and information provided by databases. EPC plays its role in network management and handling the exchange of packets with external networks. EPC controls the complete network as it has all the necessary information for running the architecture smoothly. It also communicates with the outside network when the data needs to be exchanged with remote Packet Data Networks (PDN). Furthermore, it is responsible for maintaining the Quality of Service of user sessions along with administering internal switching within the network.

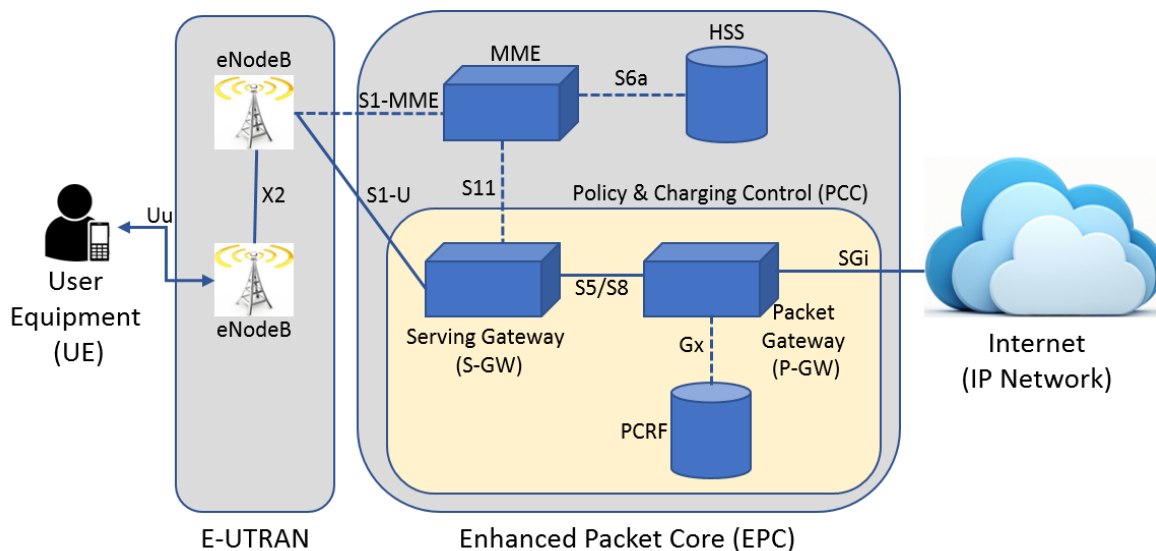


Figure 2- 4G Network building blocks and architecture <sup>1</sup>

Figure 2 gives an insight of the main building blocks of LTE architecture. In a telecommunication network, the technology deployed at last mile network differs depending on the geographical location. A city is usually equipped with LTE enabled stations, whereas a scarcely populated area can work using a 2G technology. Therefore, E-UTRAN depends on the technology being used by the telecom vendor at last mile network. ENodeB communicates with Mobility Management Entity (MME) for UE authentication and mobility management purpose. MME in turn interacts with Home Subscriber Server (HSS) to retrieve necessary information about the user.

The EPC in 4G comprises of 5 major components. MME, Serving Gateway (S-GW) and Packet Gateway (P-GW) are functional elements which are involved in enforcement of rules and management of sessions, whereas HSS and PCRF provide rules and policies governing the actions performed by the functional elements. HSS and PCRF are concerned with policy rule creation because of which, the storage entities reside with these nodes to store policy parameters. HSS is not directly involved with the session management and user-specific policies regarding data plane, instead, it is responsible for keeping the location information and authentication parameters of user devices. Serving Gateway (S-GW) or Packet Gateway (P-GW) are provided with user data policies and session parameters from Policy and Charging Rules Function (PCRF) to maintain session quality. The intermediate nodes between UE and P-GW depend on the technology used, whereas P-GW is joint for all users therefore data policies are implemented in P-GW.

PCC is the combination of components which are responsible for management of databases, retrieving policies, record user's session data and keep statistics of usage for charging purposes. The information in 4G architecture which is relevant to this thesis, is

<sup>1</sup> [online]. Available [https://www.tutorialspoint.com/lte/images/lte\\_architecture.jpg](https://www.tutorialspoint.com/lte/images/lte_architecture.jpg) [Accessed: Nov. 25, 2017]



entirely stored in components of Policy and Charging Control (PCC) because of which it will be discussed further in section 2.1.2. Once a user has successfully attached with the network, the core network does not need much information from HSS regarding user session and exchange of information. In contrary, PCRF comes into play after the user is successfully authenticated. PCRF provides underlying rules for a user or service which needs to be applied in S-GW and P-GW in order to provide satisfactory services to the user. PCRF also stores session information and charging rules which would monitor the traffic for record and logs. Two additional components, Online Charging System (OCS) and Offline Charging System (OFCS), are also part of PCC. Both of these entities are responsible for charging of user services based on the subscribed package such as prepaid and post-paid. In this chapter, we further elaborate the databases and PCC components in detail.

### **2.1.1 HSS**

Home Subscriber Server is a database unit of the network. It contains the subscriber's mobility, authentication and subscription information. HSS is the combination of two well-known units of previous mobile network generations: that are Home Location Register (HLR) and Authentication Center (AuC), because of which, it is also called as Super HLR. HSS stores the user's subscription information and provides support functions for the session and call setup along with authorization policies. As a combined alternative to HLR and AuC, HSS has the following information: [6]

- User identification and addressing: this field contains user-specific values that are unique. It includes International Mobile Subscriber Identity (IMSI) and Mobile Subscriber Integrated Services Digital Network (MSISDN) number.
- User Profile Information: This field contains some policies or profiles assigned to a user.
- Users Mobility Function: It contains the information of user's location and his mobility along with roaming parameters.
- Subscriber Location Function (SLF): Information regarding HSS associated with a user profile to redirect clients.

The purposes of AuC are security support and authentication. By using the User's unique values or codes, it generates the security parameters that are used to make the user data secure against confidentiality and integrity breaches. Correspondingly, it generates the keys for cyphering user packets. HSS is the central database for any domain but big operators might have more than one HSS. The demand for adding more HSS to system originates when the number of users in a network exceeds the handling capacity of single HSS, which in turn originates the need of having Subscription Location Function (SLF). SLF directs the client to the relevant HSS which is currently serving the user.

HSS manages the subscription information whereas S-GW and P-GW applies the session and service policies. HSS is mainly concerned with control plane and has little or no information about the data plane of user's session.

### 2.1.2 Policy and Charging Control (PCC) architecture

PCC is a combination of different modules and nodes residing at different logical or physical locations in an LTE core network. It includes Policy and Charging Rules Function (PCRF), Policy and Charging Enforcement Function (PCEF), Application Function (AF), Subscriber Profile Repository (SPR), Bearer Binding and Event Reporting Function (BBERF), Online charging system (OCS) and Offline charging system (OFCS). These components are responsible for the tasks that are most evident by their name where PCRF is the controlling component.

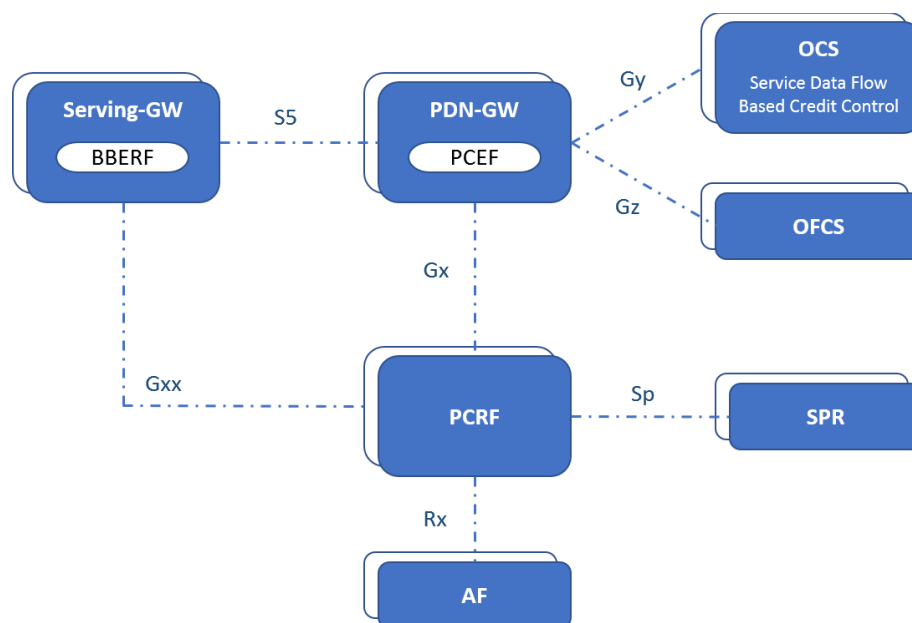


Figure 3- Components of Policy & Charging Control <sup>2</sup>

In the current architecture, P-GW behaves as the default network gateway because of which, policy application and charging rules are implemented in P-GW. QoE is maintained by EPC through policies retrieved from Subscriber Profile Repository (SPR) and Application Function (AF) through PCRF. PCRF is not connected with any of the access end entities including eNodeB, HSS and MME to achieve the modularity of nodes performing diverse functions. Figure 3 illustrates the components that together make the PCC architecture.

The figure shows different nodes in the PCC architecture of the LTE network and their interconnectivity. The rules provided by PCRF are for each Service Data Flow (SDF). SDF consists of fixed parameters which makes an SDF unique and a policy rule provided by PCRF

<sup>2</sup> [online]. Available [https://sites.google.com/site/amitsciscozone/\\_/rsrc/1468881655405/home/lte-notes/pcc-architecture/PCC%20Architecture%20non-roaming.png](https://sites.google.com/site/amitsciscozone/_/rsrc/1468881655405/home/lte-notes/pcc-architecture/PCC%20Architecture%20non-roaming.png) [Accessed: Nov. 25, 2017]

for an SDF is applied to all passing packets that match the criteria of that SDF. To define a new SDF, a template needs to be filled defining all necessary parameters to identify the flow. Each SDF has a unique identifier to be called from other nodes. There are two different types of rules that are enforced by PCC<sup>3</sup>.

- Dynamic Rules are dynamically provisioned to PCEF for enforcement by PCRF through the Gx interface
- Pre-Defined Rules are pre-configured rules in PCEF. These rules can be activated on the instructions from PCRF provided through Gx interface.

A rule provided by PCRF composes of data retrieved from the databases. SPR and AF are the main databases of PCC which are responsible for storing user-specific policies and group rules. A rule contains several fields, such as [8]:

1. **Rule Name** – An identification of the rule used to reference from PCEF and PCRF.
2. **Service Identifier** – The identification of the service or part of a service that SDF (Service Data Flow) relates to.
3. **SDF filters** – The application of rule depending on the type of traffic or data.
4. **Precedence** – Priority of the applied filter which is the screening of unwanted traffic. If the precedence or priority is the same then the dynamic rule is applicable over pre-defined rule.
5. **Gate Status** – Indicates the action of blocked or allowed for the detected traffic or SDF.
6. **QoS parameters** – The parameters like downlink and uplink bitrates, Quality Control Information (QCI) and the Allocation and Retention Priority (ARP) form the QoS parameters.
7. **Charging Key and Charging Parameters** - Online or offline charging parameters.
8. **Monitoring Key** helps to recognize a monitoring control instance that shall be used for usage monitoring control of the Service Data Flows (SDFs).

PCC is responsible for all session-related information along with charging. It ensures that each service gets enough bandwidth and resources achieve the minimum QoS target along with charging info on per service basis. The functions of all the components of PCC can be summed up to achieve the following objectives; 1) Check the correct SDF for a packet by detecting and comparing the provided fields. 2) Charge the packet according to the rule-set for specific service. 3) Provide rules for authorization control and management of service. 4) Identify the service and provide all necessary policies for session establishment and administration

**OCS** is a credit management system which is mainly responsible for the charging of Prepaid subscriptions. The information from OCS is inquired by PCEF to check for the recent

---

<sup>3</sup> [online]. Available <https://sites.google.com/site/amitsciscozone/home/lte-notes/pcc-architecture> [Accessed: Nov. 25, 2017]

credit status. Service Data Flow Based Credit Control Function performs online credit control and resides inside OCS. **OFCS**, as the name suggests, is used for offline charging of user's subscription. PCEF provides details to OFCS to formulate Charging Data Records (CDRs). These records are then interpreted to generate bills in the billing system. **BBERF** is usually located in S-GW in the 3rd Generation Partnership Project (3GPP) implementation, but its location is technology dependent and can reside in any other node. The name is formed from the two different functionalities; Bearer Binding and Event Reporting. In order to ensure the QoS for any service, there must be a mapping between rule provided by PCC and the Internet Protocol – Connectivity Access Network (IP-CAN) bearer. "Bearer Binding Function" is responsible for achieving this binding between PCC rule and IP-CAN bearer. **Event Reporting** triggers an event when a match is found on an event occurrence. This reporting is performed by Event Reporting Function, which resides either with BBERF or in PCRF. The signal for the event trigger occurs when any of the defined conditions are satisfied such as:

- A user is no more part of an operator network
- QoS values have been changed
- Mobility of User: UE has changed its location and is now in a different range
- PCRF has instructed PCEF to change a PCC rule
- User Device has received a new IP address

The remaining elements of PCC will be discussed in detail in section 2.1.3.

### 2.1.3 PCC components

The major elements of our concern are PCEF, PCRF, SPR and AF. AF and SPR are the databases for storing policy elements and other parameters, whereas PCRF and PCEF are the functional nodes. These nodes are responsible for the creation and implementation of rules. We present a study of these components and the information they contain as reference for designing our SPM, proposed in this thesis, and its integration in the mobile network architecture.

**SPR** contains the subscriber's subscription information. The information in SPR is PDN specific and it has the capability to store different information for a single user depending on the serving PDN. It contains user's information with following categories [9]:

- Allowed services to the subscriber
- Esurance of minimum QoS for the subscriber
  - ✓ Guaranteed Bandwidth for the subscriber
  - ✓ Quality of service class identifiers, maximum bit rate limit and guaranteed bit rate limit
- Charging related information of the user
- The category of the user
- Subscriber's usage monitoring related information

➤ Quota Information of the subscriber

**AF** is a database element of PCC architecture which is responsible for providing session related information to PCRF. AF is the actual server at the backend that is responsible for service session management and providing details of that service to PCRF. The Application Function contains information about the port numbers, bitrates, IP-addresses and delay sensitivity of any service to PCRF so that formation of rules can cater for these parameters. AF Contains the following fields <sup>4</sup>:

- Subscriber Identifier - typically the MSISDN of the user, if known by the AF
- IP address of the User Equipment (UE)
- Media Type or Format, Bandwidth, Flow description (For example: source and destination IP address, port numbers and the protocol)
- AF Application Identifier, Event Identifier, AF Record Information
- Flow status (for gating decision)
- Emergency indicator
- Application service provider (*i.e.*, the Diameter realm or business name)
- Quality control for the Apps that reside inside the network

**PCEF** is the enforcement agent of EPC, which applies the rules provided by PCRF. It includes a Deep Packet Inspection (DPI) module that inspects the packet for the execution of rules either provided by the PCRF or statically configured in the PCEF. It is a governing body inside PCC which guarantees the application of parameters provided in the rules thus ensuring the quality of user's sessions. PCEF is also responsible for the charging-related controls and interactions with other nodes such as OCS and OFCS. PCEF can contain pre-configured rules, but PCRF provides the activation/deactivation signals.

**PCRF** is the core element of PCC and rule providing engine for P-GW and S-GW residing in EPC of LTE. It is also sometimes referred to as Policy Server and has a former name of Policy Decision Function (PDF). PCRF is the controlling agent of the system and provides the QoS to user sessions through SDF detection, SDF filtering, filtering packets, QoS control and charging of user via PCEF. Through the information gathered from SPF and AF, PCRF creates a rule which is in accordance with the network policies, subscription and package of the user. For a complete rule creation in PCRF, along with SPF and AF, BBERF and PCEF also provide information to PCRF regarding charging and current session parameters. The information is provided through different interfaces marked on Figure 3. The information provided includes IP address of the UE, PLMN identifier, subscriber identifier, type of IP-CAN (GPRS, etc), location of subscriber, request type (Initial, Modification, etc), PDN identifier, IP-CAN bearer attributes and IP-CAN bearer establishment mode [8].

---

<sup>4</sup> [online]. Available <https://sites.google.com/site/amitsciscozone/home/lte-notes/pcc-architecture> [Accessed: Nov. 25. 2017]

PCRF plays as a negotiator between other operations that include allocating required bandwidth, QoS parameters and resources for session creation, call establishment and management. This enables an option with the operators and Internet Service Providers (ISP) to charge the customer on the basis of provided Quality of Service. A call with least interruption, delay and jitter can cost more than a normal call which has low QoS values. Moreover, it has opened the way of prioritizing calls by assigning priority based on various parameters such as dialling special codes, user-based priority, destination-based priority etc. This also enables the possibility of emergency calls which gets top priority and might be exempted from credit or charging information while dialling to make it independent from OCS and OFCS.

The functions of PCRF can be summed up as 1) it manages network and policies of users and subscribers in real time, 2) prioritizes call setup and network traffic along with dynamically handling traffic routes, 3) data gathered through device, location, billing and network policies join together to give a unified view of the user and the subscriber and 4) resource management and bandwidth allocation to assure maximum possible revenue

In terms of network management, PCRF takes the responsibility of allocating equal resources from the network to the users along with ensuring availability of services in the hour of need. It can limit the bandwidth for apps which consume maximum throughput thus dividing equal share of services amongst users. Moreover, in emergency situations, it can limit the usage of a user in order to provide necessary resources to the maximum possible number of users thus handling emergency scenarios in an optimum manner. Inside PCRF, resides a Policy Decision Engine (PDE) which is responsible for decision making tasks. This engine needs to be triggered to generate an output. This trigger can be initiated by processes, interfaces or messages from other nodes. Furthermore, internal timers can also play a role to trigger these decisions thus allowing the possibility of different resource allocation strategies according to the changing time of day.

The next section discusses the communication and interactions between different elements of PCC and the form of data exchanged.

#### **2.1.4 Interfaces and protocols**

Elements of PCC are interconnected through interfaces which have been given a name in the architecture for identification. According to Figure 3, Gx is the most important of the interfaces which connects PCEF with PCRF. Interfaces can be logical or physical depending on the nature of PCC components and their placement. The type of protocol used depends on the type of information to be retrieved, the frequency of communication and amount of data. The performance and scalability of network heavily depend on used protocols. For most interfaces, Diameter protocol is used for communication. It is an advanced form of previously used protocol Radius and is an Authentication, Authorization and Accounting (AAA) protocol.

It is a signalling protocol to retrieve data and is based on request/response principle. Major interfaces of PCC architecture shown in Figure 3 are briefly discussed below [10]:

**S5:** This interface is between PCEF which resides in P-GW and BBERF which resides in S-GW. It provides tunnelling between S-GW and P-GW for the user plane along with involvement in the exchange of information related to management of the tunnel. Furthermore, it signals the P-GW about the changing S-GW due to the mobility of the user. S5 interface is also referred as the S8 interface when a user is in roaming and there is a need to collaborate between nodes of different operators. The interface is based either on Gn/GTP (GPRS Tunnelling Protocol) or uses the Proxy Mobile Internet Protocol (PMIP). PMIP is involved in mobility management and handling of QoS parameters.

**Gy:** It behaves as Diameter Credit Control Application (DCCA) proxy between the two charging management entities, PCEF and OCS. This interface allows online charging and crediting based on SDF. Diameter protocol is used on this interface with the additional Attribute Value Pairs (AVPs) which are Diameter protocol's header fields. It controls the charging of user services based on the blocks of 100Kb (not strict) and can terminate the sessions when required. After each block consumption, the OCS checks for the remaining credit for next block usage.

**Gz:** The interface connects PCEF with OFCS for offline charging control. In offline charging, the credit information does not affect user's session and services. The interface also uses Diameter protocol as suggested by 3GPP. Charging Data Records (CDRs) are transferred between the nodes using Gz interface.

**Gx:** The interface handles communications between PCRF and PCEF. Most of the policy rules and charging based information is exchanged on this interface. It also uses Diameter protocol. The AVPs in Diameter can be altered easily by PCRF to meet the customer needs and adapt to changing scenarios. PCEF informs PCRF about the session initiation of a subscriber so that the relevant data can be retrieved from the database and formulated in the form of rules which can be implemented at PCEF. PCRF has the ability to change parameters of an ongoing session and can instruct PCEF to terminate a session. All instructions provided directly to PCEF from PCRF are sent using this interface. Two procedures are used for communication between PCRF and PCEF. A Pull Procedure in which PCEF generates a request to PCRF for fetching a rule and Push Procedure in which PCRF can instruct PCEF to enforce a rule without receiving any request. An example of Pull Procedure is Credit Check Request (CCR). PCC rules pushed to PCEF uses Re-Auth Request (RAR) messages which are replied by PCEF in Re-Auth Answer (RAA) message.

**Gxx:** The interface connects BBERF with PCRF. The rule formation at PCRF can base on the information collected from BBERF allowing PCRF to have dynamic controls over charging policies. The interface also works over Diameter protocol. Gxx interface can be further specified as Gxa and Gxc depending on the location of BBERF. If it is located in S-GW, the interface is more precisely called as Gxa whereas Gxc applies when the BBERF resides in non-

3GPP access. The interface is used for management of QoS rules from PCRF to BBERF of S-GW and also vice-versa for the recording of events occurred in the traffic plane and are monitored in S-GW.

**Sp:** This interface is used to connect PCRF with the SPR (database). Lightweight Directory Access Protocol (LDAP) is used for the retrieval of information from SPR as the node does not include any processing task and objectively is a storage device. The interface is used to retrieve subscription related data and policies of user which are implementable at the transport layer. Many parameters can be used as querying values such as subscriber identity, IP-CAN session Attributes, PDN identity. The interface can also be used in 2 ways, push and pull policies. Pull is used when an information needs to be retrieved by PCRF so it initiates a request whereas Push is used by SPR to notify PCRF about any change occurred if PCRF has subscribed for such changing notifications. PCRF can also issue a cancellation notice to SPR to stop such messages.

**Rx:** The interface supports communication between PCRF and AF in the current 3GPP specification and is used to exchange application and session level information of user's communication sessions. The information is then used for policy decision making and rule formation in PCRF. Diameter protocol is used on this interface.

**Sy:** The interface connects PCRF directly with OCS to send the credit related information and reports. The information exchanged on this interface includes the status of policy counter for its request, notification and cancellation between the two nodes. The Sy interface also uses Diameter as its underlying protocol. Credit accounting in roaming is handled in a similar manner where charging information is transferred to PCRF occasionally for keeping the log of usage and credit.

## **2.2 Firewalls**

The section introduces the concept of firewall, its functioning methodologies, types of tasks they perform and limitations. After introducing firewalls, we discuss their operative procedures, usage scenario, inspection and filtering tasks. Next, we describe different types of firewalls. Finally, their implementations would be analysed according to the need of the hour, which is the protection of networks against malicious activities and attack.

### **2.2.1 Introduction**

A firewall is used to enforce security policies on a communication between different entities. A firewall is needed because the communicating parties can be of different trust and security level. A firewall is also used to enforce the security policies. For example, it can restrict the internal users of a network from accessing services residing outside the network or restrict global users from accessing services on local systems of a network. thus, making the networks more trust-worthy and secure [11]. Firewalls are of two different types based



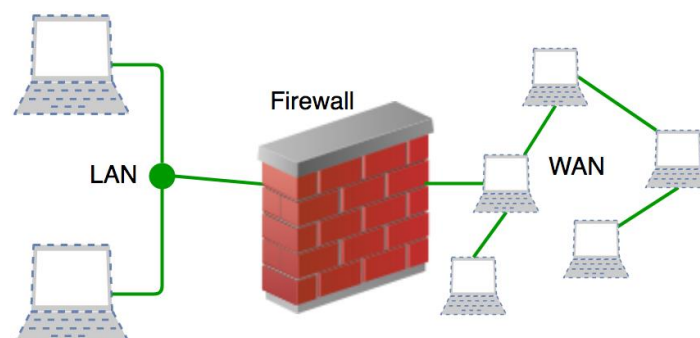
on the location of its deployment. A firewall is operated on an end-user device or in a network node running the firewall software. This, in turn, requires a secure host machine with all the latest updates and security patches installed and disabling all non-required features of the operating system thus protecting the processes from unwanted interrupts.

A firewall also behaves like a router, thus connecting different networks and providing interconnectivity between them. It is not exactly a router as a normal home class router is comparatively a dumb device used to just route packets rather than inspecting all 5 layers of Open Systems Interconnection (OSI) model. A firewall needs to perform faster processing since it needs to process every packet in detail for enforcement of all the security policies defined in rule set. A slow firewall would induce an unnecessary delay which would affect the QoS and QoE for user services. Moreover, the primary purpose of a router is to route packets but the newer versions of business class routers of Cisco provides few firewall capabilities internally. This security feature of a router is just an added benefit over the baseline purpose. A firewall is used to filter packets so it is primarily used for security that includes authorization, authentication, content security and encryption whereas routing is an added or required functionality for inter-network communications without which the packets cannot be forwarded and routed.

When a firewall receives a packet, it inspects the packet and takes any of the three predefined actions. The actions comprise 3 responses which include Reject, Drop and Accept.

- **Accept:** Allow traffic to pass through the firewall
- **Drop:** Drop the packet without any notification to the sender
- **Reject:** The packet is dropped and a notification of “Unreachable” is replied back

There are two types of Accept as action of firewall. An accept decision can be either hard accept or soft accept. Hard accept means a packet is accepted and it passes through the firewall. Soft accept passes the packet through the current chain and forwards the packet to the next chain or rule set. This would further be discussed in section 2.2.2.



*Figure 4- Firewall Placement for separation of LAN and WAN* <sup>5</sup>

---

<sup>5</sup> [online]. Available <https://cdncontribute.geeksforgeeks.org/wp-content/uploads/introduction-to-firewall-1.png> [Accessed: Nov. 25. 2017]

Figure 4 shows the placement of a network firewall which resides at the edge of the Local Area Network (LAN) towards the internet or the Wide Area Network (WAN). In case of home or small networks, a firewall is placed with a router. Business class routers are installed in small companies where some functionalities of firewalls are achieved through Access Control Lists (ACLs) configured in routers. ACLs have the capability to check packet headers and deny or allow the access according to the rules, but they lack the capability to inspect packet above transport layer. Firewalls not only allow or deny services but also impose limitations on the allowed services such as number of simultaneous connections for File Transfer Protocol (FTP). An advanced firewall has captive portal which requires users to enter credentials before their packets could be routed.

## **2.2.2 Working methodologies and limitations**

Firewall works on rules or policies defined by the network administrator. A set of rules are deployed in the firewall and a given packet is passed through all the rules till it matches any. Once a rule is matched with the headers of the packet, the actions against that rule are applied to that packet. A firewall mostly protects the internal LAN network from attacks originating from WAN, but it can also be used to block any destination for the local users such as blocking Facebook in universities. Firewalls work primarily by matching source IP, destination IP, source port, destination port and protocol which is commonly known as a five-tuple match. In the current internet world, the mostly used protocols include Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP). TCP and UDP use port numbers that identify the application level service. Firewalls also maintain a default policy which is executed if a packet does not match any rule. The default policy can either be Reject, Drop or Accept. In a very secure implementation, where the incoming traffic is known beforehand, Drop or Reject is used as a default rule. A heavy loaded network where there is a need to handle huge traffic, usually Drop is used instead of Reject because, in case of Reject, the node needs to do additional work in the creation of a response packet and sending it back towards the sender. This introduces an additional processing load on firewall node [12]. Contrary to this, a general ISP's firewall works with Accept as a default rule and all other forbidden domains are added in the rule set with action as Drop or Reject.

There are firewalls which can go up to the application layer to provide deep packet inspection and filtering. These firewalls work in today's world where the forbidden traffic is passed through HTTPS encrypted sessions. Firewalls have a limitation where it cannot drop a packet containing forbidden content wrapped within an authenticated protocol. Some workarounds have been enforced in practical implementations to deal with this aforementioned issue but the concept of Virtual Private Network (VPN) still exists. Packet filtering on the basis of five-tuple match is the easiest to implement and configure and is more efficient on the firewall end. Inspecting layers above transport layer of packet induce a delay

in packet processing and also requires specific configurations in order to avoid interruption in the normal communication of services. A good firewall does not only provide adequate security but also caters for other functional variables such as performance, easy management, reliability, event logging and intelligence. Event logging and intelligence of firewall mean to record the interesting events and learn from them to either notify the network administrator or to dynamically create rules for such events.

Next, we study the limitations of firewalls. This helps the administrator or manager to make a better analysis about the potential risks that exist. If the connections are not monitored and the firewall does not interfere in encrypted sessions or the policy executed by firewall is not effective, then it would be of no use to install such a system for protection and degrade the network performance. Following are the well-known shortcomings of firewalls [13]:

1) Authorized or legitimate services can be used for malicious purposes. Firewall is not intelligent enough to recognise any malicious attempt over a legitimate authentic session such as telnet. After connecting to a Telnet session, an attacker can run commands and get data from target machine for which, firewall would have no information. Moreover, tunnelling a forbidden service or protocol over a legitimate session of an authentic protocol is not identifiable by firewall. An example of such is the VPN where a blocked website can be viewed using this service. 2) A firewall can only monitor the traffic passing through it. For example, if an internal host of LAN wants to attack a local user, it cannot be detected by a network firewall. Firewall works on the principle of packet inspection and matching the packet with the defined rule set. If the packet does not pass through the firewall, the required filtering cannot be achieved. Internal intrusion can be detected by routing all the traffic through a firewall or installing intrusion detecting systems. Additionally, if the attacker still uses some alternate means to reach his target, such as attack through a backup dial-up server connection, the firewall would not be able to intercept such threats.

3) A firewall cannot prove beneficial against the attacks caused by social engineering. If an attacker is successful to obtain the password of a user, then he might be able to access the system or restricted services by pretending to be an authentic user. Similarly, an attacker can fake himself to be a network administrator to get the user's credentials. 4) Security and robustness of firewall are dependent on the operating system over which, firewall is installed. A firewall cannot rectify the security risks originated through flaws in the operating system. It is necessary to install latest security updates for making the host machine as secure as possible. Thus, the operating systems with embedded firewall software have gained more popularity such as, developed by Nokia. 5) There is always a lag between the firewall developers and attackers which keep attackers ahead of firewall manufacturers. An attack is first attempted after which its counter is updated in the firewall software. Because of such potential risk, an administrator must always devise ways to minimize loss and create recovery methods.

### 2.2.3 Firewall types

Various types of firewall solutions are available on the market depending on the usage scenario and the required functionalities. More advanced features and deeper packet inspection require more investment which then provide reliable security and robust framework. The firewalls can be categorized based on the location of their installation and the type of functionality performed. Firewalls are fundamentally categorized into two different types, Host-based firewalls and Network-based firewalls. Classification of firewalls is shown in Figure 5. Each component of the figure is discussed briefly [14].

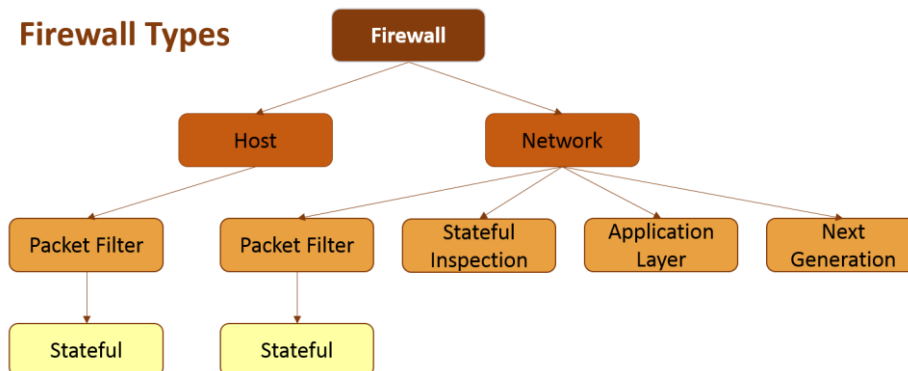


Figure 5- Types of firewalls

**Host firewall** is the installation of a firewall software in an operating system running over an end-user hardware. In recent years, many of the operating systems are offering built-in firewall tools and are provided as an added feature. Windows, Linux, and Mac are common well-known examples of such operating systems. The firewall is installed on an individual machine and is configured, maintained and controlled by the administrator/end-user through physical or remote access to the machine. It can also be pre-configured by vendor to monitor and eliminate the known attacks. Host firewall is typically concerned with protecting an individual against threats offered by WAN users as well as internal LAN attackers. A local host firewall can protect against viruses and other malicious activities which might cross network firewall due to loosened security checks. Host firewall provides following advantages:

1) Network Firewall Failure: Installation of host firewall protects the computer in case of network firewall failure which is installed at the network edge. If network firewall is crashed due to an attack, a host firewall should be more robust to tackle and mitigate such activity and protect the user. Though backup firewalls can work against attack, there are some attacks which are still out of firewall's domain such as spam. 2) Simplicity: It is relatively simpler to configure a host firewall than network firewall and likewise it gives more security against threats and attacks. A host is more service specific and knows about the expected traffic whereas network firewall needs to meet the demands of all internal users. Due to this, security level of network firewall cannot be set very strict in order to avoid interference with legitimate session and services.

3) Wide Protection: Host firewall can inspect packets thoroughly because it is not concerned much about the delay caused since it only needs to inspect packets for a single

machine. Whereas, the network firewall cannot spend considerable time in inspecting a single packet as it would induce an overall delay to packets of all users in the network. 4) Relevancy and Specificity: The host firewall can be configured precisely to allow applications and services which a host machine is expected to use and block all other traffic. Network firewalls are usually configured other way around where the default policy is to accept the packet and only the packets with blocked parameters are dropped. It is due to the reason that services used by the clients of a huge network cannot be presumed.

A **Network firewall** is installed at the edge of the network to protect a group of users from potential attacks by WAN users and block specific services and applications for internal users. Network firewalls include an added functionality of routing capability where it is connected with more than one networks and provide interconnectivity and Network Address Translation (NAT). A network firewall is static and can only protect the end devices as long as all incoming traffic is routed through this network. But if a device has more than one communication interface, then the device is subjected to potential threats from all of these interfaces. Thus, in such scenarios, host firewall is recommended which does not only protects the system from attack through all the peripheral devices and interfaces but is also mobility independent. Network firewall can crash which then shifts all the security responsibility to the host firewall whose absence might open a door to threats [13]. Network firewall provides following advantages:

1) Improved Security: A network firewall has the responsibility to protect all the nodes inside the network and thus contains rather flexible rules in order to avoid interruption in the normal functioning of most of the end users applications. Similarly, the firewalls need to be updated with all the latest patches and virus detection updates according to their latest releases to keep the network secure. A host firewall might not get updated in time as it would depend on user's end device processing power and available resources etc. but network firewall is updated right away, thus, offering better security against latest threats. Moreover, network and host firewall provide backup for each other in case of need. 2) Availability: Network firewalls and solutions are usually robust and reliable because of the need of today's world where attackers are more active than defenders to find and exploit vulnerabilities. In such cases, network firewalls are usually equipped with recovery plans and backup systems which comes into force when needed.

3) Scalability: Network firewalls are more scalable and perform better in terms of packet processing delay, parallel processing of packets and number of packets processed in unit time. They are designed to handle more flexible Maximum Transmission Unit (MTU) and higher bandwidths which host firewalls lack and might need an update or change of software when handling different network parameters and values. 4) Reach: Network firewall solutions are mostly provided by well-known vendors which have inter-company collaborations to provide up-to-date security patches and information about the latest and upcoming threats. This, in turn, makes network firewalls more updated than host firewalls and thus greatly expands the protection scope to meet the network requirements.

5) **Affordability:** Network Firewalls are usually auto-configured and all the major functionality features have an impressive user interface which could be handled by a minor IT administrator. A host firewall needs a user to configure it and does not provide easy to operate Graphical User Interface (GUI). Moreover, a network firewall does not require any configurations on each client system and configurations at one device serves the purpose for all the clients in the network.

Host firewalls are usually Packet Filter type of firewalls whereas network firewalls are further divided into four types, based on the methodology of packet filtering and OSI layer inspection level, as shown in . These types of network firewalls are also referred to as generations of firewall. The types of network firewall include Packet Filtering Firewall, Circuit Level Firewall/Stateful Inspection Firewall, Application Level/Proxy/NAT Firewall and Next Generation Firewalls will be discussed briefly.

**Packet Filtering Firewall:** These are the simplest type of firewalls which filter packets on the basis of information provided by network layer and transport layer. The information is then matched with the already stored rules and relevant action is executed over the packet. Filtering is performed on the basis of five-tuple match as explained before which include source IP, destination IP, source port, destination port and transport protocol. In addition to this, an entry interface can also be considered as a filtering parameter. Packets are handled individually and there is no information about the session or stream of packets. When a packet reaches the firewall, it passes the packets through all the rules and when a packet is matched, a relevant action is executed.

**Stateful Inspection Firewall:** This firewall inspects packet up to the session layer to get more information about application and session that lies above the transport layer. It keeps a record of the state of session in order to extend the capabilities of packet filtering firewall rules to include the information collected from the previous history of session or packet. This, thus, transforms the firewall to monitor the session initiation, handshake and termination information. As the firewalls decisions are also based on the previous session information, therefore stateful inspection firewall is also named as circuit-level firewall in which firewall keeps track of a circuit which is session in our case.

**Application Level Firewall / Proxy Firewalls / NAT Firewalls:** Application level firewall is also known as the proxy firewall or NAT firewall because of its extended functionality, provided over the previous generation of firewalls. Connections and sessions of clients inside the network are forced to go through a configured proxy, thus performing deeper packet inspection up to application level information. These firewalls require rules to be declared more carefully with additional granular level information for matching with data retrieved from the packet. These firewalls not only filter packets related to application level information but also ensures the legitimate use of generally allowed services such as File Transfer Protocol (FTP) and Hyper Text Transfer Protocol (HTTP). There is no direct connection between hosts in presence of this firewall and all sessions have to go through a proxy

**Next Generation Firewall:** This is the most advanced form of firewalls which are currently being deployed and are most robust against threats and attacks. It employs deep packet inspection which inspects application layer of packets along with the exploration of information inside the encryption protocols such as Secure Socket Layer (SSL) and Secure Shell (SSH). These firewalls are not only responsible for ensuring authorization of packets but also detect malware, virus and other application-level threats. The security provided by these firewalls is deployed at the cost of high financial investment and delay in packet processing. No security is fully reliable as NG firewall cannot prove robust against many threats such as email attacks, social engineering and insider attacks which are originated from inside the network.

#### **2.2.4 SPM for firewall**

CES, a proposed architecture for core network communication, proposes deployment of firewall at network edge. This firewall performs extended functions than a typical and conventional firewall and is responsible for securing not only the users but also less intelligent devices that makes IoT such as cameras inside the network. As the responsibility of providing protection is solely handled by edge node, therefore it is given the name of Realm Gateway. The realm gateway resides at the edge of CES network which performs negotiation functions with remote firewalls along with execution of firewall rules. The filtering and negotiation rules are stored in SPM, designed in this thesis, and are retrieved by Realm gateway when required.

### **2.3 Policy Management System (PMS)**

A policy is a set of rules that comprises of matching parameters and an action to be taken on the matched packet. It is a separation between functional elements of system and the rules that govern these functions. It makes system compatible with different work scenarios using changing rules to adapt through the new environment without changing any functional coding of the system. Because of its use and appealing approach, the concept has been used by various companies such as Cisco in making their systems flexible for deployment, use, update and management. Furthermore, there is an ongoing research to improve the architecture of such systems to enhance the performance, security and scalability which are the key factors of any management system. This section discusses the terminologies used for designing such a system. Next, it will describe few already implemented policy management systems and their functioning. Finally, the section would evaluate the feasibility of using existing systems with CES.

### 2.3.1 Terminologies:

To understand and implement SPM, there is a need to get familiar with all the architecture and functional terms of such system in order to have smooth implementation and reasonable documentation. This, thus, requires mentioning most of the related terms that would be part of the SPM developed in this thesis. The major terms of SPM are mentioned below [16]:

**AAA:** It is an acronym for Authentication, Authorization and Accounting. The basic parameters included in AAA excludes the parameters for encryption of data. AAA only includes access and accounting information which is important for SPM as firewalls and policies mostly deal with authorization and authentication.

**Action:** It is the response to be taken on a packet when it matches with any stored rule. A policy table includes a set of parameters with relevant action which are executed on the packet in case of rule match. It includes three actions Accept, Reject and Drop.

**Configuration:** It includes static or dynamic parameters of a network element. It can either be defined by manufacturer or by the administrator. There are some static parameters which are predefined and cannot be altered whereas the functional parameters are usually left to customer discretion. Packet queue length is an example of static parameter whereas an action taken against a policy match is usually dynamic.

**Filter:** The set of parameters or matching variables used to either categorize packet or to separate it from a list is called a filter. Filtering is achieved through matching of packet fields with the parameters of defined rule and then executing the defined action against the matched packet.

**Outsourced Policy:** It is the delegation of authority to another entity for taking a decision against any event. This is usually done when there is a need to reduce the delay in packet processing. An example of it is the emergency call which faces the minimum delay in a network.

**Policy:** It is a rule or set of rules to authorize, manage, maintain and control access to services and entities. It can also be defined as the rules to handle packets in a network and take decisions about actions to be performed based on the matched rules, past information and statistics.

**Policy Conflict:** It is the conflict of the actions to be taken on a packet when more than one rule matches with the packet with different actions. In such conditions, an administrator needs to define and set rules for resolution of policy conflicts. First Come First Serve (FCFS) is one approach to handle such conflicts where a packet is matched with rules and the action against first matched rule is executed.

**Policy Decision:** It is the actual action executed against a packet which matched a rule based on various other information inputs such as past record and current state. The decision can be based on parameters involved in processing of a packet or the result that includes the action taken on a packet.



**Policy Decision Point (PDP):** It is a node in a network or an entity that takes a policy decision for its internal process or for an authorized entity.

**Policy Domain:** It is a physical or logical set of devices or a geographical region of devices which are administered using a common entry point where the communication is controlled through defined set of policies. It also constitutes the collaboration of policies between different sub nodes of a domain.

**Policy Enforcement:** It is the execution of an action or a policy decision on a packet. It is usually done in the functional or edge node of a network.

**Policy Enforcement Point (PEP):** It is an entity where a policy enforcement action takes place.

**Policy Error:** It is the error which occurred during the execution of a policy decision. This failure of policy action can be due to various reasons such as compatibility of host with the defined action or change of state of packet. It can also occur if the policy conflicts are not accurately resolved.

**Policy Management Tool (PMT):** It is a software or tool to manage policies in the database and enforce any set of policy to be executed in PEP. This tool interacts with policy repository for formulation and management of policies and provides a GUI to the administrator or user for interaction with the database.

**Policy Repository:** It is the repository for storing policy rules and their related information and actions. Policy Repository holds the database and its schema for policy storage, retrieval and management. It usually consists of a Directory Service or a Database such as MySQL.

**Policy Request:** A query sent from the client to a server for retrieval, storage, management and editing of policy or its related information is a policy request. When a policy request is sent from PEP to Policy Repository, it would probably be a policy retrieval, addition or edit request whereas a request origination from PEP to PDP would possibly be a policy decision request.

**Policy Server:** Initially the term was not defined but according to the RFC2753, policy server is the PDP. It is an abstract term which can be considered as one node or set of nodes working in collaboration to achieve the required functionality.

**Policy Translation:** It is the transformation of policy from one state to another. A policy entered through frontend in plain text format using text fields can be reformed in a different layout to be stored in database which is called policy transformation. Another example is the retrieval of policy from database and then transforming it in such a format which is executable at firewall node. This task involves conversion to another form using functions such as encoding, plain text mapping or translating functions.

**Schema:** It is a layout of database and set of minimum parameters which a policy must possess in order to enter the database. Schema includes different database tables, which are also known as models, with table definitions that needs to comply with policy structure. A

policy complying with the database table definition would be successful to get inserted in the database table.

### 2.3.2 Variants

Firewall vendors and network administrators have developed several management and storage systems to store and retrieve policies for the smooth functioning of a network. This makes the system more flexible and adaptable to changing scenarios. Each system developed for the purpose of PMS is designed to address a particular usage scenario. These systems are requirement specific and do not offer much more than what is needed. Moreover, they lack proper documentation, usage manual and not give assurance of bug free software. Comparatively, some business solutions have been provided by Cisco and other vendors which provide better documentation and a user-friendly interface but is not flexible enough to meet all requirements of SPM. They, too, are designed for storage and management of specific expected information and it is not possible to change the core code or database schema of such solutions to comply with the changing needs of CES PMT. Some deployed and well-known solutions from renowned companies would be discussed hereafter.

Cisco has developed a policy management system under name of Cisco Security Manager (CSM) <sup>6</sup>. The software is able to manage policies from small networks to huge sized networks consisting large number of devices. The software provides the ability to share resources such as policies and objects to manage more users in a large enterprise. This feature of sharing resources helps in improving scalability of the system to accommodate high number of devices and new users. The views are customized for different users depending on the proficiency and role of the user which provides flexible interface for policy management. Moreover, it provides logging information to track events of attacks and request for change of configuration. Besides policy management, the software tends to provide other networking control and provisioning options such as routing control, assurance of QoS and interfaces management. Moreover, an instance of CSM is able to handle several security devices and networks.

The API module of CSM uses Representational State Transfer (REST) interface for communication with PEP. It is a java-based application providing a REST interface with all management and functional components programmed in Java. The application provides a number of features such as showing graph of statistics, processor and memory usage information by application and also keeps a log of states of sessions. Furthermore, it also saves the record of actions taken on previous packets which is then analysed to make future. It has an interface to connect to devices and listen to the incoming originated alert messages from connected entities.

---

<sup>6</sup> [online]. Available <https://www.cisco.com/c/en/us/products/security/security-manager/index.html> [Accessed: Jan. 15, 2018]



Figure 6- Cisco Security Manager Features <sup>7</sup>

The application is able to sort history in terms of all 5-tuple. To address the future threats and keep record of previous malicious activities, the application records identities (IP addresses) from which a vulnerability has been attempted to be exploited. Usage monitoring is a necessary feature of such applications where it records the most hit destinations, top Victims, most active hosts etc. For scalability purpose, it makes user groups for policy implementation. Like a quality analysis software, CSM also provides periodic reports in form of graph and sheets for analysis. Figure 6 summarizes the features provided by CSM.

AlgoSec is another network policy management software, which is developed by Bell Laboratories, primarily used to manage firewalls <sup>8</sup>. Besides firewalls, software is used to manage various security devices and protocols such as Virtual Private Networks (VPN) and routers. Corporate world uses AlgoSec to improve network security by analysing the network for vulnerabilities and test the policy rule set against possible and well-known threats. The software has three different variants which can either work together or any of these components and can be purchased and deployed to work as standalone. These components include Firewall Analyzer, FireFlow and BusinessFlow. Firewall Analyzer is responsible for auditing network elements and analysing security risks. Fireflow is used for automated transition from an old state to a new updated state with different security policies whereas BusinessFlow is used to translate business application policies to firewall rules that can be executed in firewall to allow necessary services and protect the application from threats.

AlgoSec is vendor independent and can be used with any number of devices. Along with physical and logical devices, it can also be used to manage policies on cloud and Software Defined Network (SDN). AlgoSec key feature is to automate policy creation and implementation according to the changing network scenarios which is achieved 4x faster than

<sup>7</sup> [online]. Available <https://www.slideshare.net/CiscoGreece/cisco-john-tzortzakakis> [Accessed: Dec. 19, 2017]

<sup>8</sup> [online]. Available <https://www.algosec.com/> [Accessed: Jan. 19, 2018]

conventional methods. With minimal changes, AlgoSec provides same features as provided by CSM for network management, control and monitoring. A key feature of AlgoSec is the compatibility with a hybrid network where different equipment from various vendors are installed and being monitored by AlgoSec Management Tool. Moreover, it graphically shows a complex network for easy modifications and automatically mitigates suspicious firewall rules in order to ensure system integrity and security.

The most commonly deployed firewall management tools are Tufin and Firemon<sup>9 10</sup>. Like AlgoSec and CSM, Tufin and Firemon are also network management software to automate the changes in network infrastructure or security module. Besides already mentioned features, Tufin and Firemon are also able to provide support for network switches, next generation firewalls, web proxies and cloud and network switches. Furthermore, Tufin also has three suites that include SecureTrack, SecureChange and SecureApp. These three suites are the alternatives to AlgoSec's Firewall Analyzer, FireFlow and BusinessFlow and provide similar functionalities. Besides CSM, all other tools provide technologies for automatic policy formation with policy creation tools to adapt with changing network scenarios and automate security management.

### 2.3.3 Variant's limitations

The famous solutions stated in aforementioned sections have different shortcoming because of which they are not suitable to accomplish tasks of Security Policy Management for CES. Some limitations of the variants of solutions presented above have been discussed in this section. All these solutions are propriety and no opensource code is available for customization of software because vendors do not allow editing of source code in order to ensure software integrity. Therefore, the binaries of such software are not opensource. The limitation arises because of the limited options integrated in the developed software and a client is not provided with the flexibility to introduce a new option but to stick with only available features. A SPM for CES requires an interface to handle frontend for administrator and user to input and manage policies. Moreover, this interface would be used by CES node to retrieve policies. The policy parameters are flexible and can vary from very simple policies to diverse service specific parameters. It requires validation of fields to be saved and also the policies that are retrieved should be in a format that is implementable in edge node.

Cisco Security Manager is principally formed to manage and control only Cisco devices. The solution is currently available on Windows platform and no support has yet been provided for MAC and Linux based system by the vendors<sup>11</sup>. There is no command line interface that can be accessed remotely for the retrieval of policies. Though REST interface is provided in

---

<sup>9</sup> [online]. Available <https://www.tufin.com/> [Accessed: Jan. 21, 2018]

<sup>10</sup> [online]. Available <https://www.firemon.com/> [Accessed: Jan. 21, 2018]

<sup>11</sup> [online]. Available <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/> [Accessed: Jan. 29, 2018]

AlgoSec but with a limited use. The software offers pre-defined filtering parameters and the packet inspection based on distinctive fields is hard to implement. Moreover, the solutions mostly run using a dedicated hardware module for quick inspection of packets. The database of such software is write-protected because of which, new features are difficult to implement.

Due to the aforementioned reasons, a new Security Policy Management needs to be designed and developed which is opensource, relevant, and flexible for future extensions. Furthermore, this system would be performing tasks defined in policy requirements of CES node including policy validation and formatting. The database has been designed to handle device specific policies and organizational policies where an admin set policy for its dependent users. Additionally, it provides security for IoT devices and enables dynamic control over the diverse offered services.

### **2.3.4 Research**

In recent years, considerable research has been done on the designing and development of policy management systems. Few of the developed policy manage software are inspired by the proposed designs from research. Edward and Stockwell proposed a policy management system which is a United States patent and aims to address the firewall policies [17]. This system has a database for storing of policies and a query is received by the system for fetching of policy set. The requirement of authentication between the sender and the receiver depends on the confidentiality of information that is requested. The system is simple compared to its counterparts and the SPM developed as part of this thesis resembles this system but the requirements of SPM are far more than merely storing firewall rules. Similarly, David Matthew developed another United States patent for Policy-Based Network Management [18]. The system proposed the storing of underline rules upon which, the policies can be created on run-time and sent to central node for execution. The current SPM needs the policies for already registered users and work on the principal of knowing the expected traffic. Hence, it would not be feasible to create the dynamic rules on runtime.

Morris Sloman suggested to provide flexibility to the already developed policy management system to comply with the changing needs of application [19]. The solution provided in this Journal is the mechanism of adaptation of the current system with the device and application requirements. The solution would need to incorporate the machine learning principles to get the optimum result in today's diverse network traffic. Dinesh proposed a policy management architecture in [20] which suggests to divide the management of policies into two layers including business policies and technical policies. The paper aims to address the areas of managing performance service level agreements and supporting enterprise extranet. The solution does not suggest the handling of firewall policies and to store information in database related to counter the malicious traffic and attack events. A domain-based policy management system is proposed in [21] which uses LDAP to store policies and other data and aims to provide flexibility to the network. The development of SPM requires a

system to store firewall rules along with policy negotiation parameters. The leverage of policy management to end-user demands a secure full stack with a web-frontend which the above research solution clearly lacks.

### 3. Customer Edge Switching (CES)

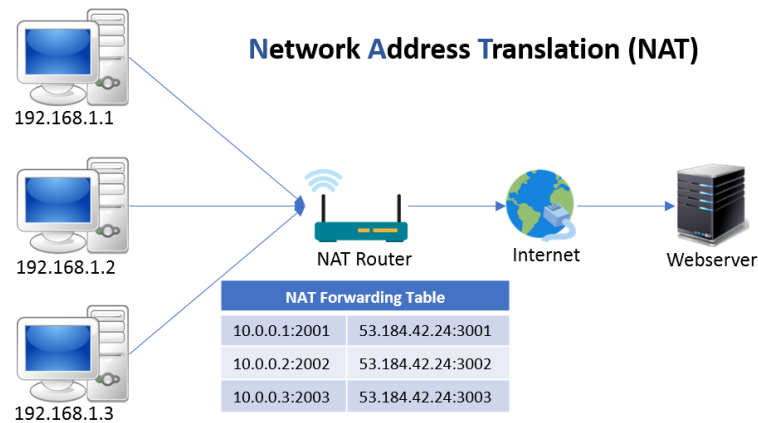
The Security Policy Management designed as a part of this thesis, provides policy service to a network edge node called CES node, which is a cooperative firewall and tunneling endpoint residing at the network edge. Conceptually, a CES node acts as an agent of end host for ensuring minimum level of security and trust before the end-to-end communications can take place with a remote entity. This chapter introduces the CES and focuses on its design and describes the policies used by CES in the establishment of a connection and to maintain uninterrupted communication between hosts. Based on the operations and requirements defined in this chapter, a proof-of-concept SPM is presented in the subsequent chapters. Finally, the chapter discusses the flexibility of CES node policies and their possible future extensions. This flexibility must be fused in the design of SPM to make it suitable for future upgrades and use cases.

#### 3.1 Motivation

The widely used version of Internet Protocol version 4 (IPv4) uses 32 bits addresses which can be allocated to approximately 4 billion devices. According to statistics<sup>12</sup>, 23 billion devices are connected to the internet and this number will increase to 75 billion by 2025 due to ever increasing number of connected and Internet-enabled devices. It has been possible to connect the 23 billion devices to internet despite the limited number of IPv4 addresses due to the introduction of a technology called Network Address Translation (NAT). It is a mechanism which allows large number of user devices in a private network to share a set of limited number of public IP addresses towards internet for connectivity. One such example is a home router where all home devices communicate with the internet through an allocated public IP. Inside the network, a range of private IP addresses is used which are then translated to the public IP at NAT Router using transport layer ports and source IP addresses. This scenario is shown in Figure 7.

---

<sup>12</sup> [online]. Available <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/> [Accessed: Jan. 29, 2018]



*Figure 7- NAT Architecture<sup>13</sup>*

Besides providing scalability, NAT introduces some new challenges for which, no universally deployed solution is adopted yet. Internet protocols require end to end connectivity which are affected by the presence of intermediate router (i.e., NAT) between the communicating hosts. Various methods have been proposed to rectify the problem such as Traversal Using Relays around NAT (TURN) [22] and Session Traversal Utilities for NAT (STUN) [23], but they lack scalability with large number of applications, slow down session establishment and require applications specific port forwarding. Moreover, NAT hides the identity of the end-user as packet takes the identity of the NAT router, which dynamically allocates the source IP and port to the outgoing packet. Therefore, a server cannot differentiate between various senders residing behind NAT devices which in turn raises the problem of applying the sender-receiver security. Tunneling protocols are complicated to use with NAT because NAT router modifies the header of the packet which effects the integrity check of packets performed at the tunneling end-nodes. The translation of the packet at NAT node needs additional processing on both ways which induce a minimal delay in packet round trip time. A solution to this problem is using IPv6 addresses in which all devices around the globe can have a globally unique IP addresses, thus eliminating the need of NAT. Deployment of IPv6 is under process but not yet implemented in the last mile of networks.

A solution pursued by research group of Prof. Raimo Kantola (at COMNET Department of Aalto University) is published as Customer Edge Switching, where the identity of the device, which classically is its IP address, has been replaced with a Fully Qualified Domain Name (FQDN). For global reachability, the system relies on the Domain Name System (DNS) to map the FQDN of host to IP address of the network edge node serving the host. Network algorithms and functions that are part of CES then solve the security and reachability issues introduced by NAT. The idea is that the communication security relies in the trust relation created between the network nodes based on provided parameters during the session establishment phase. The system keeps track of all the connected network nodes around the

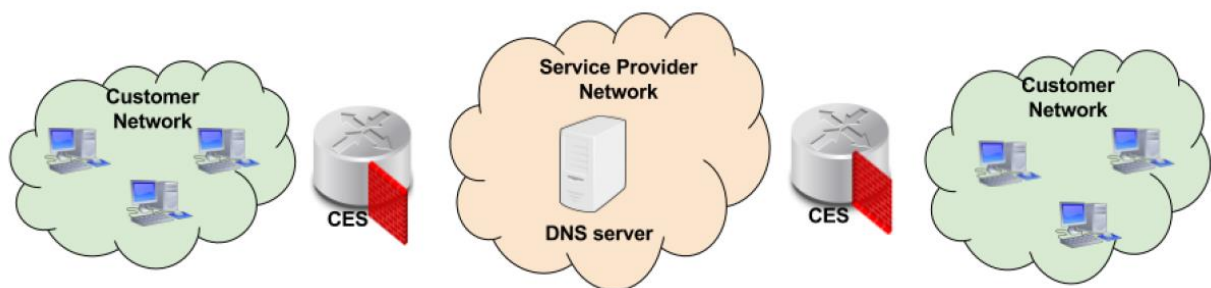
<sup>13</sup> [online]. Available <http://www.learnabhi.com/wp-content/uploads/2018/02/NAT-min.jpg> [Accessed: Jan. 15, 2018]



globe and the host devices have their unique FQDN. Therefore, a user can be identified in case of tracking any malicious activity. A typical use case of CES is in providing and ensuring adequate security for IoT devices in the network edge nodes according to the policies set by the users and administrators. Besides FQDN, the system can use any alternative stable identifier which then can dynamically and quickly mapped to the host IP address.

### 3.2 Architecture

The adaptation of CES at network edge splits the Internet into Customer Network (CN) and Service Provider Network (SPN). At the edge of the CN, resides the CES node which employs different forwarding technologies to suit changing set of requirements. The edge node also acts as a trust boundary where a different network, and therefore a different level of trust starts. Figure 8 shows a sample architecture of CES where customers together form a CN, which is separated from SPN by a CES node. The hosts have their unique FQDN through which they are accessible on the global internet. The CES node maintains a table for mapping inward and outward connection packets, which enables packet forwarding locally and towards a dissimilar trust domain.



*Figure 8- Architecture of CES [26]*

In terms of its fundamental composition, the CES uses Dynamic Host Configuration Protocol (DHCP) server which allocates the IP address to a newly connected device. The device is then registered and the DNS server, normally residing on the CES node, is updated accordingly with the IP address and FQDN of the newly connected host. Once a user has been registered in DNS, it can be reached globally via its allocated FQDN. Besides FQDNs, CES also introduces Service FQDN, which allows a more specific addressing of a service running on the host. “ssh.abc.aalto.fi” is an example SFQDN of the Secure Shell (SSH) service running on a host with FQDN “abc.aalto.fi”. Identification of hosts in CES architecture relies on the principle of Dynamic Domain Name System (DDNS) which prevents the need of giving the static IP addresses to hosts. This feature thus helps to develop web or other servers using a unique SFQDN which is independent of dynamic IPv4 addresses allocated in the Customer Network.

### 3.3 Communication and protocols

(The communication in) CES is divided into the control plane and the data plane. The control plane includes signaling for establishing the required level of trust between networks serving the sender and destination host, whereas data /forwarding plane is concerned for routing of user's data packets. The signaling cases are Inter-CES Communication and Intra-CES Communication. When a communication takes place between hosts of the same CES, it is termed as Intra-CES communication, whereas Inter-CES communication commences between the hosts of different networks served by different CES nodes. For interworking with the legacy Internet, a CES node will have a Private Realm Gateway (PRGW) that uses well-defined protocols to provide a source NAT (SNAT) and a destination NAT (DNAT) functionality.

The CES node uses the well-known and standard protocol called DNS as communication trigger. However, the CES nodes rely on a new protocol for network and host policy negotiations. The introduction of this dedicated protocol customizes the behavior of CES node according to policy requirements. For real deployment, this new protocol would have to be standardized by IETF to allow multivendor interoperation. For verifying our ideas and required algorithms, we have designed the experimental version of the protocol that we call Customer Edge Traversal Protocol (CETP).

#### 3.3.1 Inter-CES communication

In this case, the sender and destination host reside in different private networks served by their respective CES node. For connection establishment at the initiator's end, the outbound CES performs a DNS resolution to get the CES-ID and control plane Routing Locators (RLOCs) of the inbound CES for session initiation. After getting the RLOCs (the publicly reachable IP addresses of the remote CES nodes), the two CES nodes negotiate various policy parameters according to the defined policies. Upon successful negotiations, a connection state is established in both the CES nodes. The connection state locally represents the remote host in the private network with a proxy address. At conclusion of policy negotiation, a DNS reply with the destinations proxy address is sent to the sender host, and the data communication takes place via connection states established in the CES data plane.

Figure 9 shows a scenario where a host served by CES-A triggers communication with the Host-B served by CES-B. For connection initiation, Host-A generates a DNS resolution request with the FQDN of Host-B, which reaches to CES-A. CES-A does not recognize the destination as a local host, so it translates the request to a NAPTR query and forwards it to the DNS infrastructure of internet. DNS identifies the remote CES based on its name server record, which is CES-B in our case, and forwards the query to the DNS server of CES-B. CES-B acts as the leaf node of the DNS hierarchy and replies with the corresponding CES-B routing locator(s) (RLOC), implying that the destination host is reachable through these RLOCs on the CES-B node using CES service.

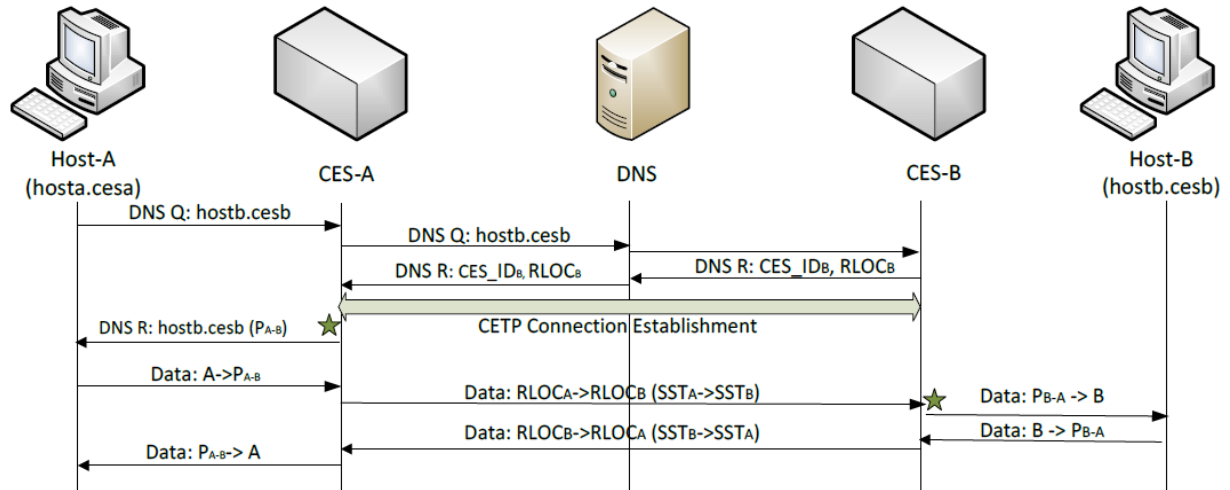


Figure 9- CES to CES connection establishment [24]

After receiving the DNS NAPTR response, CES-A initiates a policy negotiation phase to establish a connection state with CES-B so that the Host-A can communicate with Host-B, based on the principles and parameters agreed between CES nodes during policy negotiations. Subsequently, inbound and outbound CES create a state entry in their connection tables and store the address binding between private address, the proxy address, local/remote RLOCs ports and session tags etc. The NAPTR DNS response is received by CES-A, which then modifies it to carry proxy address (for destination Host-B) to represent the destination host locally and forwards the DNS response to Host-A. Subsequently, the Host-A then communicates with this proxy address assuming it to be Host-B and starts sending data packets. When this data packet reaches CES-A with the destination of proxy address of Host-B, CES-A matches it with the stored state entry in the connection table and forwards it to CES-B complying with the negotiated parameters. CES-B examines the inbound packet according to its stored state and forwards the translated packet to Host-B. Finally, CES-B forwards the packet to Host-B and the reverse packet from Host-B to Host-A follows the similar procedure.

### 3.3.2 Intra-CES communication:

Intra-CES communication also involves policy negotiation between sender and destination host, but in such scenario, the DNS of the local CES node serves the senders DNS query, and CETP policy negotiations occurs locally. All communication packets between the local hosts still passes through the CES node to ensure the policy application. When Host-A needs to communicate with Host-B, both of which reside behind the same CES, Host-A generates a DNS request which is received by CES node. The node then creates a session state containing the information of addresses of both the hosts and other defined policy parameters. Once connection state is established, the user data flow starts, where each packet is inspected to comply with the policies of both hosts. The normal case is that the hosts reside in different private address spaces although the hosts are served by the same CES. This follows the addressing model adopted for mobile devices in mobile networks.

In Intra-CES communications, we assume that the two hosts belong to trust domains under the same CES. So, CES level policy negotiation is not required, but host-to-host policy negotiations are required to negotiate and implement user policies. These two negotiation steps are introduced by CES architecture after which, a user data transfer commences. CES node allocates proxy addresses to represent remote hosts, therefore a DNS response with the allocated IP resource is sent to the Host as a reply to DNS request. CES architecture imposes an additional delay in session creation because of the additional step of policy negotiations but this in turn provides robust security for clients and servers. In an Intra-CES communication, the connection establishment is comparatively faster because the communication packet does not need to undergo the propagation delay. This thesis focuses more on the Inter-CES communication because of its rich set of parameters included in session creation and handling. Whereas, Intra-CES communication can be considered as a sub-set of Inter-CES communication.

### 3.3.3 PRGW and legacy host

For real world deployments, any new technology must be backwards compatible with legacy networks for its smooth adoption. For this reason, the CES architecture contains PRGW functions for communication with legacy IP networks. The architecture and detailed description of Primary Realm Gateway (PRGW) is available at [25].

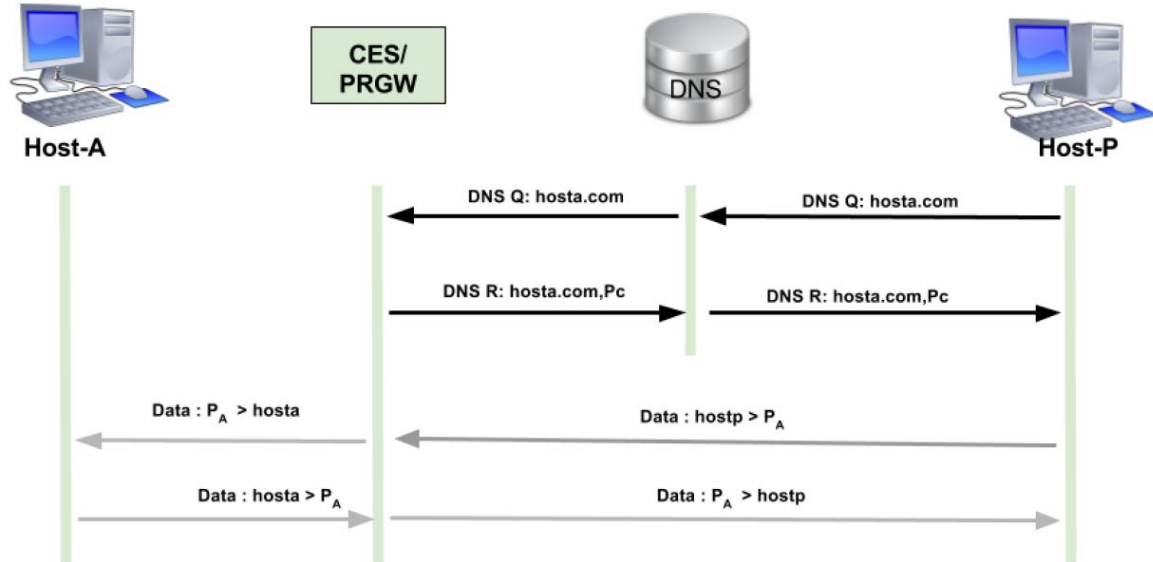


Figure 10- Inbound Communication between Legacy IP and CES host

Figure 10 shows the CES interaction with a legacy IP host in internet. Host-P in the figure is a public host that wants to create a connection with Host-A behind CES. For connection establishment, Host-P will originate a DNS query which will first reach to the global DNS server. The global DNS does not have information about the Host-A, but it has records for CES domain, the query would be forwarded to the CES node by public DNS. The authoritative DNS at CES would reply to this query (with TTL=0). Host-P will start direct communication with

Host-A as it now has the addressing information of Host-A. When the data packet reaches CES (PRGW), it gets mapped with a local IP addresses and other parameters, which is then reverse mapped to the public address when Host-A sends a reply to Host-P. In this scenario, CES(PRGW) acts as destination NAT (DNAT). A detailed description of connection and session information is described in [25].

### **3.4 Policy information**

A policy is a description of what a host or a network entity expects and what it is willing to offer to a remote entity so that a desired action takes place. In CES, the policy is defined as a set of rules and parameters upon which a connection establishes and data communication occurs. A policy is formed using several keywords and parameters. Different types of policies in the context of CES are described next.

#### **3.4.1 Types of policies**

CES architecture is divided into Control Plane (CP) and Data Plane (DP), where each deal with different types of policies. CP is responsible for initiation and establishment of signaling sessions and user connections, whereas DP governs the rules for packet forwarding. Functionally, there are three types of policies; CES-CES Policies (C2C), Host-Host Policies (H2H) and firewall policies. A C2C policy and H2H policy are involved in session creation, and thus come under the category of CP policies, whereas firewall policies are concerned with packet filtering and thus are DP policies. Each of these policies will be discussed in following paragraphs.

An inter-CES communication undergoes both C2C and H2H policy negotiations. The C2C policy is negotiated first between the two CES nodes and it is concerned with network-specific policy elements which are independent of communicating hosts. A C2C policy comprises of three sections of policies which includes “request”, “offer” and “available”. Request policies are demanded from remote CES, and express some constraints, which means that CES will not establish a signaling session if the provided parameters by remote CES do not comply with the defined constraints. The second part is the available parameters which is used to provide the available policy parameters of a CES. Whereas, offer parameters are the subset of available parameters, which a CES node provides at start of policy negotiation. Figure 11 shows a sample section of the C2C policy.

```

"available": [
  {"ope": "info", "group": "ces", "code": "cesid", "value": "cesb.lte."},
  {"ope": "info", "group": "ces", "code": "pow"},
  {"ope": "info", "group": "ces", "code": "caces", "value": "195.148.124.145"},
  {"ope": "info", "group": "ces", "code": "fw_version", "value": "0.1"},
  {"ope": "info", "group": "ces", "code": "ttl", "value": "3600"},
  {"ope": "info", "group": "ces", "code": "session_limit", "value": "200"},
  {"ope": "info", "group": "ces", "code": "evidence_format", "value": "IETF-IOC1"},
  {"ope": "info", "group": "rloc", "code": "ipv4"},
  {"ope": "info", "group": "rloc", "code": "ipv6"},
  {"ope": "info", "group": "payload", "code": "vxlan"},
  {"ope": "info", "group": "payload", "code": "gre"},
  {"ope": "info", "group": "payload", "code": "geneve"}]

```

*Figure 11- CES-CES sample policy*

This sample policy set only contains the available parameters. An offer and request section looks similar to this but with different values to “ope” parameters. The parameters are CES specific, which implies that a CES can dynamically change any of its policies to create a lax connection with known or trusted remote CES nodes, whereas untrusted CES networks can be treated severely. The representation of policies is in JavaScript Object Notation (JSON) format. Similarly, an H2H policy has the same policy structure but with a different set of parameters. A complete sample H2H policy is shown in Figure 12.

```

"request": [
  {"ope": "query", "group": "control", "code": "caep"},
  {"ope": "query", "group": "control", "code": "ttl"},
  {"ope": "query", "group": "payload", "code": "gre"},
  {"ope": "query", "group": "id", "code": "fqdn", "value": ["hostb1.cesb."]}],
"offer": [
  {"ope": "info", "group": "payload", "code": "gre"},
  {"ope": "info", "group": "id", "code": "fqdn", "value": "hosta1.cesa."},
  {"ope": "info", "group": "control", "code": "caep", "value": "195.148.124.145"}]
"available": [
  {"ope": "info", "group": "control", "code": "ttl"},
  {"ope": "info", "group": "payload", "code": "gre"},
  {"ope": "info", "group": "id", "code": "fqdn", "value": "hosta1.cesa."},
  {"ope": "info", "group": "control", "code": "caep", "value": "195.148.124.145"}]

```

*Figure 12- Host-Host sample policy*

The H2H policy in Figure 12 shows all three sections of a H2H policy. The H2H policy needs to be negotiated for each source and destination user upon need of establishing a connection between the end-nodes. The parameters are host specific and thus allow a user to restrict access to a host or impose an additional requirement from the remote entity.

In comparison, a firewall policy comes under the domain of DP policies. These are simple firewall rules with an added information to identify the user. A sample firewall policy is shown

in Figure 13. Policy contains five groups which include ID, GROUP, CIRCULARPOOL, SFQDN and FIREWALL. Details of functioning of these units are described in detail in [25]. A user might belong to one or more groups of policies as defined in the GROUP section. SFQDN includes all the service FQDNs that a user might have to identify/address its services.

```
test103.nest.gwa.demo.:
  ID:
    fqdn: ['test103.nest.gwa.demo.']
    ipv4: ['192.168.10.103']
    msisdn: ['0000010103']
  GROUP:
    - IPS_GROUP_PREPAID3
  CIRCULARPOOL:
    max: 3
  SFQDN:
    - {fqdn: 'test103.nest.gwa.demo.', proxy_required: false, carriergrade: false}
    - {fqdn: 'www.test103.nest.gwa.demo.', proxy_required: true, carriergrade: false}
    - {fqdn: 'sip.test103.nest.gwa.demo.', proxy_required: true, carriergrade: false}
  FIREWALL:
    FIREWALL_ADMIN:
      - {'priority': 0, 'direction': 'EGRESS', 'protocol': '17', 'udp':{'dport': '53'}, 'target':
        'REJECT', 'hashlimit': {'hashlimit-above': '5/sec', 'hashlimit-burst': '50',
        'hashlimit-name': 'DnsLanHosts', 'hashlimit-mode': 'srcip', 'hashlimit-htable-
        expire': '1001'}, 'comment': {'comment': 'Host DNS limit'}}
    FIREWALL_USER:
      - {'priority': 100, 'direction': 'EGRESS', 'target': 'ACCEPT',
        'comment': {'comment': 'Allow outgoing'}}
      - {'priority': 100, 'direction': 'INGRESS', 'target': 'ACCEPT',
        'comment': {'comment': 'Allow incoming'}}
```

*Figure 13 - Sample firewall policy*

Firewall section contains firewall rules which are further divided into two types; FIREWALL\_ADMIN rules and FIREWALL\_USER rules. This classification is performed to differentiate between the entity defining these rules and to prioritize the rules. This separation and priority resolve the conflict between policies. An admin policy is added by the network administrator for the user whereas a user can add a firewall rule for himself or for his dependent users. In both cases, the rule is stored in FIREWALL\_USER section with different priorities. Admin policies are executed first and then the user specific filtering is applied.

### 3.4.2 Management Scope

Policies described in section 3.4.1 needs to be stored in a Policy-Database. These policies will be retrieved by CES when a user attaches with the network or when a new connection needs to be established. For C2C and H2H policies, it is possible to store the policies relative

to the destination CES and Host so that a different policy can be applied to any destination. Some parameters are fixed and do not change according to the destination and thus, can be stored as default policy. A default policy must always exist in the Policy-Database in case no destination specific policy exists. CES policies are only inserted by administrator, whereas user has a restricted access to modify and add his Host-to-Host and firewall policies.

Firewall policies are divided in 5 groups as explained earlier. Most of these are only configurable by the admin whereas user can edit the FIREWALL\_USER section of the policy shown in Figure 13. The policies need to be formatted in the similar format as shown in the figure for it to be directly executable in CES node. All the policies are in JSON format.

### **3.5 Conclusion (and SPM requirements)**

There are three types of policies that need to be stored in the Policy-Database. For storage, two approaches exist with their own benefits. Either parameters can be stored and policies can be created from these parameters when fetched, or already formed policies can be inserted in the Policy-Database. The later approach significantly reduces policy retrieval time. The policies described in above section are operational policies and are retrieved when needed, whereas there is another set of policies called bootstrap policies which are retrieved by CES when it starts up. It includes the bootstrap information for setting up parameters and tables in CES node.

All mentioned policies can be inserted by a user or an administrator which requires 1) a GUI interface to interact with the SPM. Moreover, 2) to ensure that no malicious policy is inserted in the Policy-Database, various checks need to be incorporated in the SPM. 3) An additional interface is required through which a CES node will retrieve policies for attached users. 4) Another interface will interact with the Android application developed in [26] to automate the population of policies to the SPM. This, thus, requires developing a robust full stack system which needs to interact with users, network administrators and policy sourcing systems with management of the SPM and Policy-Database.



## 4. Proposed architecture

This chapter presents the architecture of a Security Policy Management to store, retrieve and manage policies needed for CES operation. The policy types are already described in Chapter 3. The major components of SPM primarily composes front-end, the central Policy-API/back-end and Policy-Database.

### 4.1 Design

SPM has been designed to provide policy services to three critical entities of a CES network, namely users/admin, CES node and a policy-sourcing Android application. A Policy-Database has been designed for storing the policies, where the Policy-API is a single-entry point to the Policy-Database. The Policy-API provides REST interface to entities for managing/retrieving policies in the Policy-Database. Android application and CES node are policy creation and application systems that directly communicate with the Policy-API, whereas the users and admin express their policy via a web interface developed in Django web framework of Python. This front-end interface is called as Django-webserver in SPM as shown in Figure 14. The GUI of Django-webserver provides easy handling of policies through a webservice and interacts with Policy-API to provide new/updated policies. Django is a robust web framework which creates frontend pages automatically along with data validation and session management. The automatically generated web pages from Django integrate diverse features to handle various front-end threats, thus eliminating malicious policies from reaching the Policy-API backend. Figure 14 shows a basic SPM architecture which illustrates Policy-API as the single-entry point to the Policy-Database and the three communicating nodes towards Policy-API.

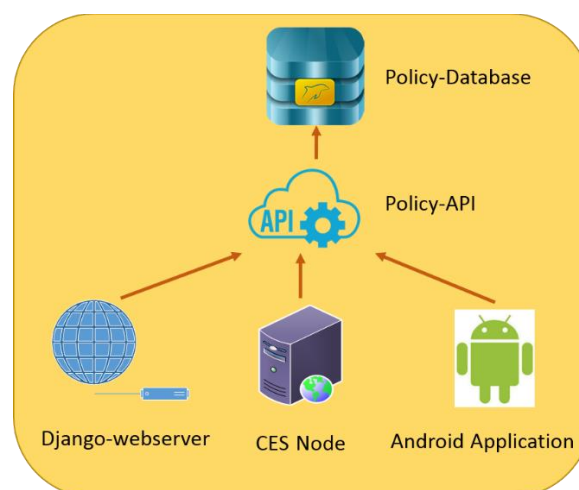


Figure 14- Basic SPM structure

Policy-API offers a REST interface to the system and allows pursuing a stateless implementation. After validation of users input, Django-webserver generates an HTTP

request to the Policy-API to perform relevant action. Similarly, CES node and Android application also generate REST queries towards Policy-API. Access to Policy-API can be restricted using certificates or IP whitelisting. Moreover, additional authentication parameters can be implemented to authenticate nodes communicating nodes but it is out of the scope of this thesis. The Policy-API is able to handle GET, POST, PUT and DELETE REST queries.

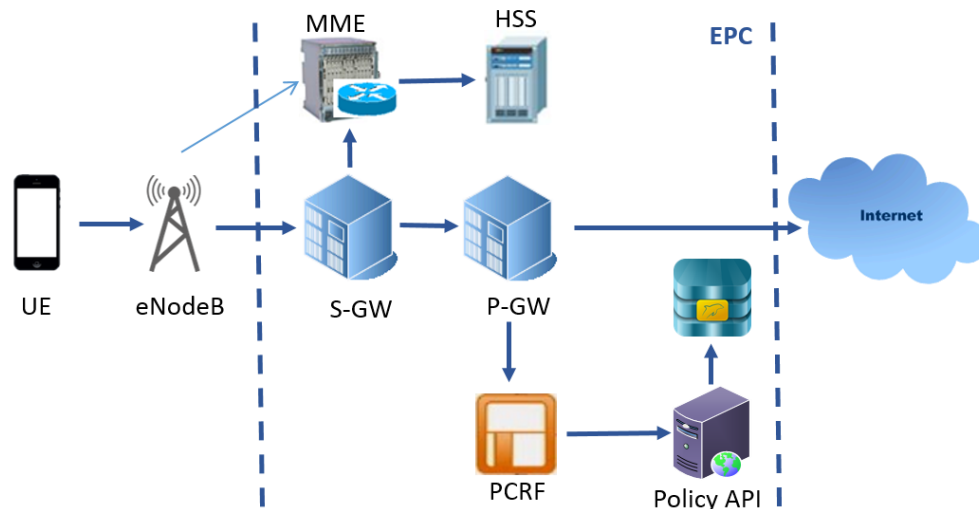


Figure 15- LTE Architecture with SPM

Backward compatibility is an essential feature of any new developed system or technology for its adoption. Our SPM draws upon LTE PCC architecture, where PCRF is the central policy management entity. In LTE, P-GW generates request to PCRF for policies. CES node can reside with P-GW as all traffic of network passes through P-GW and execution of policies would easy to achieve. Similarly, a CES node residing in the gateway generates request towards PCRF which would then retrieve policies by forwarding queries to Policy-API. The modified version of LTE architecture using SPM is shown in Figure 15.

## 4.2 Policy-Database schema

We have designed the Policy-Database for SPM to store the policy types defined in Chapter 3. Bootstrap policies and user's identifications are stored in separate tables. Database structure is GET operation centric, which allows better performance when retrieving policies. Whereas PUT and POST functions for policy insertion/modification are relatively more time consuming, since the processing assures the insertion of safe policies in to the Policy-Database, so that the response time for a GET query is less. The Policy-Database divides the policies into two separate databases namely Bootstrap\_Policies and Session\_Policies. Bootstrap policies are stored in Bootstrap\_Policies database that has only one table to store all the bootstrap policies, whereas Session\_Policies database has multiple tables to store the firewall and session establishment policies (DP and CP policies). In Bootstrap\_Policies

database, more tables can be added in future to accommodate future expansions of policy management.

### 4.2.1 Bootstrap policies

Bootstrap policies are requested by the CES node from SPM at start of the CES node. In contrary, Session\_Policies database is used on demand as the throughout CES node operation as it requires retrieving session policies for communicating nodes and need for checking updates.

The schema of the only table in Bootstrap\_Policies database is shown in Figure 16. In addition to the “id” column which is an auto-increment unique primary key, the table contains columns: “name”, “types”, “subtype” and “policy\_element”. Out of these, first three parameters are used as filtering parameters based on which, a “policy\_element” (of 2048 characters string) is retrieved. “id” column in all Policy-Database tables is to select unique row of table for update and delete operation.


| bootstrap   |               |  |
|---|---------------|--|
|  id | int(11)       |  |
| name  | varchar(128)  |  |
| types   | varchar(128)  |  |
| subtype   | varchar(128)  |  |
| policy_element  | varchar(2048) |  |

Figure 16- Bootstrap\_Policies database schema

“policy\_element” contains a JSON formatted of data wrapped as string datatype, since the database does not support JSON format in column. This stored string of JSON is converted back to actual JSON format on retrieval from database by Policy-API. On retrieval of information, the column label is used as the key of JSON whose value is the data contained in that column. This formation of policies is handled in Policy-API when a policy is retrieved. The flexibility of database comes at the cost of limited filtering parameters (or number of columns). The filter for a policy query applies on the database table columns, which allows to retrieves the rows matching the query. For example, in case of retrieving all rows where “name” is equal to “IPSET”, the database would retrieve all rows of bootstrap table in which the column “name” has “IPSET” as value.

### 4.2.2 Session policies

The second database “Session\_Policies” contains six tables. Two tables are dedicated for storing H2H policies and two tables are used for C2C policies. The remaining two tables include ID table which stores all the identifiers of the users and the “firewall\_policies” table which records all data-plane policies of users. These policies have already been discussed in Section 3.4. ID table contains all the possible identities of user which currently include FQDN, IPv4 address, username and MSISDN. This can further be extended if mobile operators need to store other identities of users such as International Mobile Subscriber Identity (IMSI) etc. All these identities are mapped to a unique 64-bit identifier in the Policy-Database, which is used internally in database as a key to retrieve the user policies thus, making policy retrieval independent from the provided identity of user. Schema of “Session\_Policies” database and linkage between tables is shown in Figure 17.

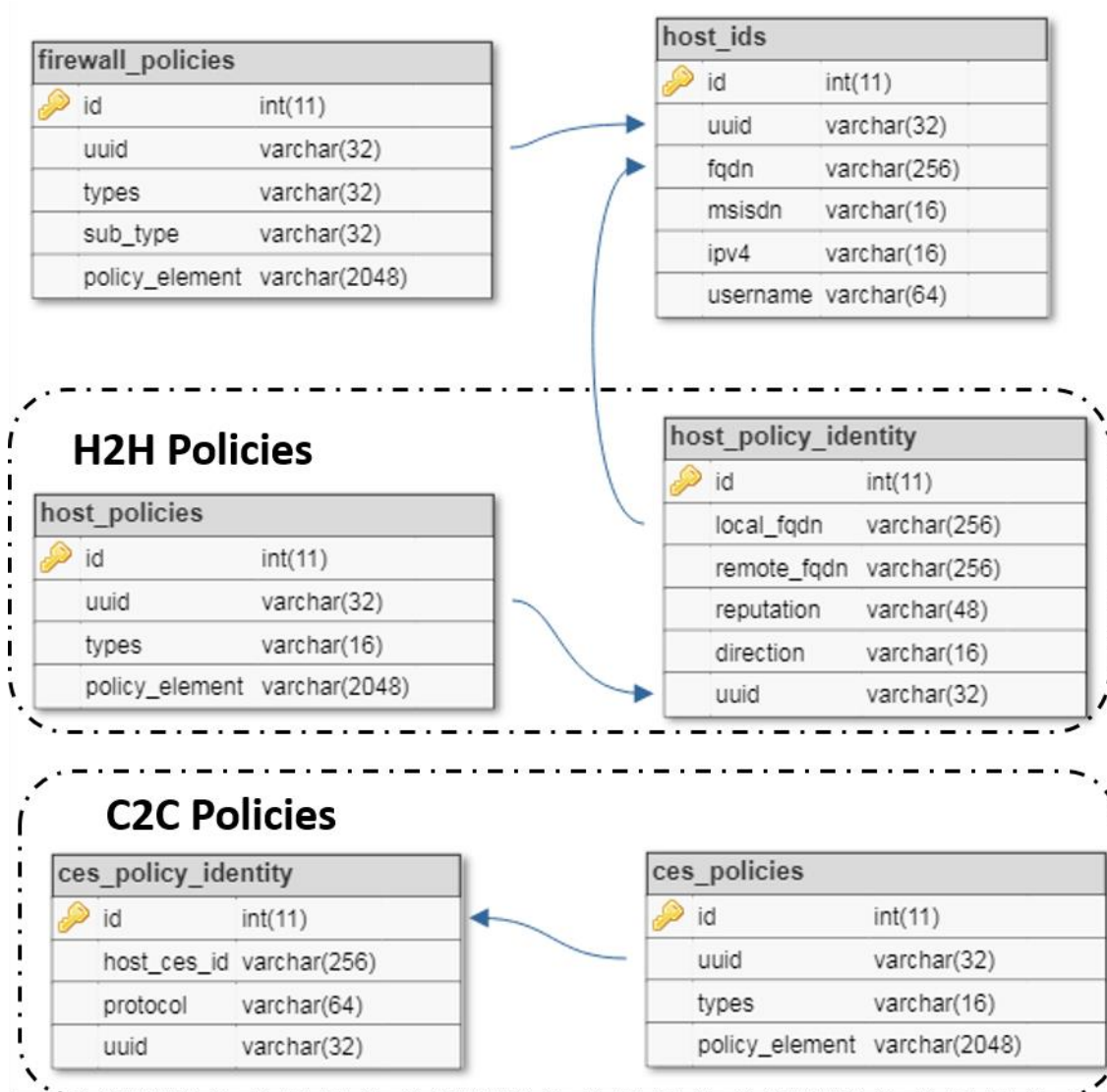


Figure 17- Session\_Policies database schema

Since H2H policies and user-firewall policies are user specific, they are shown together with the host ID table. CES policies are independent of host, therefore, the 2 tables of CES are shown separately in the bottom of figure.

### 4.2.3 Host Policies

Firewall policies are user specific policies and are identified through unique user ID (UUID). UUID is a 64-bit key used to retrieve user policies from different tables of databases and are unique for a user. Contrary to this, “id” an integer column in all database tables which is a row identifier of database table and is unique within a table. The policy data stored in firewall table is structured into “types” and “sub\_types” column. The “types” column can get the following values: ID, GROUP, CIRCULARPOOL, SFQDN and FIREWALL, whereas the “sub\_type” column allows to specify, if a FIREWALL policy is FIREWALL\_ADMIN or FIREWALL\_USER. The policies can be filtered using either “types” or “sub\_types” column.

“host\_policy\_identity” and “host\_policies” tables are again mapped with a separate unique 64-bit Universally Unique Identifier (UUID). A new key is generated when a new distinct set of parameters is inserted in “host\_policy\_identity” table. These distinct parameters are then mapped to the corresponding host policies inserted in “host\_policies” table using this 64-bit identifier. Therefore, this identifier is completely isolated from the one mentioned in ID table for user identification. Policies that need to be stored in host policies table are shown in Figure 12 which are mapped to relevant columns in “host\_policy\_identity” table. As shown in the table of “host\_policy\_identity”, the filtering of policies can be achieved using one or more of the following parameters in the incoming request that include “local\_fqdn”, “remote\_fqdn”, “direction” and “reputation” upon reception of a request. By providing these parameters, a corresponding UUID is retrieved which is then used to fetch the set of policies from the “host\_policies” table. In case of lesser number of provided parameters than maximum possible, more than one UUID can fulfil to the provided criteria. In such case, the UUID is fetched on the first match basis. The case where more than one identifier matches the provided parameters, the conflict is resolved by first come basis. “host\_policies” table has one query parameter; UUID and two columns are retrieved which include “types” and “policy\_element”. As shown in Figure 12, “types” can have three possible values including *request*, *offer* and *available*. “policy\_element” contains a JSON string of parameters against the “uuid” and “type” of policy. Two columns of Type and Policy element are then formatted into JSON in Policy-API to produce the policy that is executable at CES node.

#### 4.2.4 CES Policies

Schema of tables for storing CES policies strongly resemble with that of H2H policy tables. “CES\_policy\_identity” table contains filtering criteria for policy retrieval. It defines the “host\_CES\_ID” for which policies need to be retrieved and the “protocol” that will be used. These two unique parameters are mapped to 64-bit Universally Unique Identifier (UUID) based on which, C2C policies are retrieved from “CES\_policies” table. Like “host\_policies” table, “CES\_policies” also contain a column for UUID which acts as key. “types” column specifies the type of policy which are *request*, *offer* and *available* and the “policy\_element” contains a JSON string which is used to produce final policy in Policy-API upon retrieval. All these policies are formatted and validated upon a POST or PUT request for policy insertion/modification, while retrieval process consumes very little processing which significantly reduces the policy retrieval time.

## 5. Implementation

This chapter presents the implementation of Security Policy Management. We first describe the tools used for development of the system and the reasons for selection of chosen tools and their possible alternates. Next, the implementation of system is discussed along with the interconnection of different nodes, and functions being performed by each entity. Finally, we discuss the optimizations in the system to achieve better performance and scalability results.

### 5.1 Tools and specifications

While developing a system, the choice of tools can be an important factor in stability and performance of the system. SPM has been developed using Python 3.5. The system contains two web servers namely Django-webserver and Policy-API server as shown in Figure 14. Since the Django-webserver provides a portal for users and admins to edit their policies, it requires a GUI which is developed using Django tool. The backend system has been developed using AIOHTTP, which is an asynchronous library of Python for creating REST service. For storage of policies, MySQL has been used as the database for this system and python connects with the database using “AIOMySQL” package. It is also an asynchronous library to connect with the database by opening a pool of connections which are distributed among the using functions. The unused connections are automatically closed after being idle for a set value of time. It reuses the already opened connection through sharing, thus reducing time for establishing a new connection.

Python3 is the latest version of python and contains a built-in framework to handle asynchronous calls. Our SPM has been developed in the latest available python version (Python 3.5 at the time of development). The asynchronous implementation allows full utilization of the limited system resources. This improved performance is achieved through the concept of Co-routines where, a sequence of programming instructions, called tasks, is put to sleep while it waits for a response, and in the meanwhile, programming control is passed to handle another request.

The Policy-API server needs to interact with database to store and retrieve policies. MySQL has been selected as the database of this system, since it is easy to develop systems database with extensive online support available, and it is open-source. It is scalable and provides better performance in scenarios with frequent read and write operations. On the contrary, Oracle database is not open-source, and the free storage capacity is limited. MySQL provides unlimited storage. Furthermore, an extensive range of libraries are available in python to communicate with MySQL than any other database.

The aim of this thesis is to develop a system which can be used to store policies and provide a frontend to the user, therefore we chose Django which is a simple and powerful

web framework based on python. Django has been the choice for developing policy-frontend, because of its ability to handle various frontend-based attacks internally and form creation for policy management. It offers a Model View Controller (MVC) architecture, where a developer does not need to create the complete Hypertext Markup Language (HTML) pages, as it is mainly handled by the framework itself. Moreover, Django has built-in validation and form creation tools, which reduce the task of developer to handle HTML and web form field errors and in turn validates user's input. It also displays error message in case of error. The Django-webserver developed as part of this system is a standalone system and can be moved to any node, and it interacts with the Policy-API server through HTTP REST queries.

## 5.2 System architecture

SPM has been developed to store, validate and populate policies all the way to the Policy-Database, and to provide policies to CES node for execution. For users and admin, the function of SPM is to provide an interactive interface to manage their policies and it provides an additional interface for CES node for the retrieval of policies. The actual execution of policy is out of the scope of this thesis. Figure 18 presents the high-level implementation of SPM.

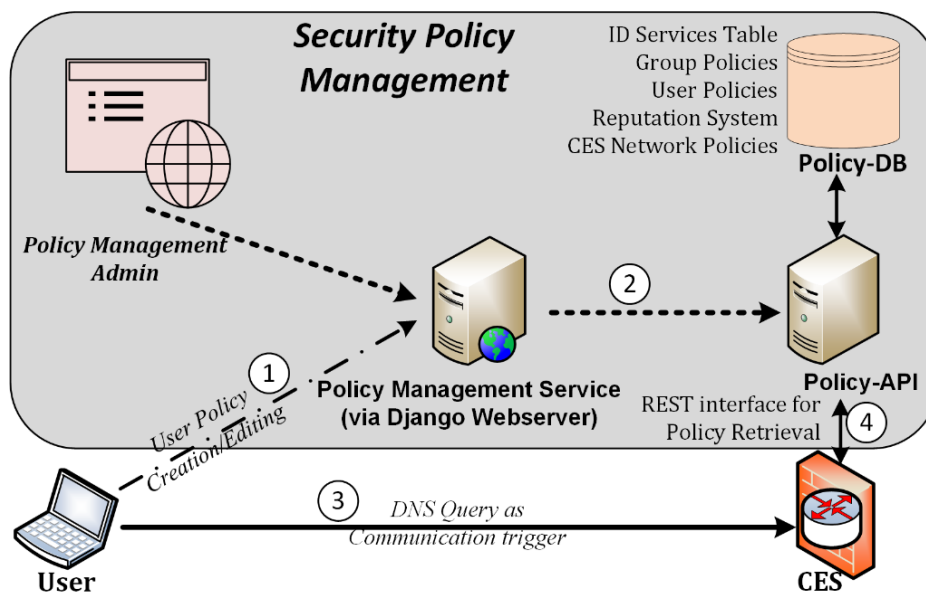


Figure 18 - SPM nodes in CES Network Architecture

Admin and user login to Django-webserver for management of policies as shown in the figure. The permissions of storing policy in particular location and the execution of conflicting policies is resolved based on login credentials. Authentication parameters limit the authorization of users to manage and edit certain policies only and correspondingly allocate a priority value to an inserted policy. A Policy-API server, which provides REST interface for management of policies, offers a protective boundary to API and Policy-Database.

To compare the SPM with RFC 3198 for the terminologies of Policy Based Management [16], the nodes have been assigned names to elaborate their functionality. A set of users



residing behind a single CES node is the Policy Domain where the internal hosts are treated as local hosts and the identifier type for CES is the same for all the entities. Policy Enforcement Point (PEP) is the CES node where the policy retrieved from the SPM is executed. Policy-API along with Django-webserver allows the task of Policy Management Tool (PMT) which manages policies in the Policy-Database using a GUI. Policy-Database designed over MySQL is the Policy Repository of SPM, where policies are stored into the database according to the tables schema. Policy Request can be initiated from the CES node or from the Django-webserver for the retrieval or management of policies respectively, and Policy-API serves the request and thus, it behaves as a Policy Server and a Policy Decision Point (PDP) because of its ability to formulate policies. This complies with the RFC 2753, where a Policy Server resides inside PDP. Policy Translation is also performed inside Policy-API where a policy entered in plain text from the frontend is translated to an executable JSON format for the CES node.

Figure 18 shows the sequence of policy insertion in the Policy-Database using the proposed SPM architecture, and the retrieval of policy for execution by CES. The user authenticates himself on Django-webserver after which he edits or inserts his policy which is then forwarded to Policy-API for storing in database. After saving the policy, a user initiates a connection with a remote internet entity. When a connection is initiated, a DNS request is originated towards the CES node. Upon reception of this request, the CES node sends a policy request to Policy-API for the retrieval of the user policies. These policies are negotiated over the control plane to establish a data plane session between the end-nodes. We discuss the detailed implementation of each component of SPM in the following sections.

### **5.2.1 Policy-API**

All functional elements of the SPM backend reside behind the REST Server, which acts as a single-entry point towards the Policy-Database. Authentication parameters are exchanged between the authenticating connected entities and Policy-API for authentication of node and identification of user. Policy-API provides functionality to three different services. First is Django, which is used to provide GUI to users and admins for managing policies. Second is the CES node that retrieves the policies from database for execution and third interface handles mobile application which has been developed for policy sourcing from user devices, but is out of the scope of this thesis. Policy-API provides interface for storing, editing, retrieving and deletion of policies. A CES node cannot execute POST or PUT operations towards the API as only safe actions, such as GET, are permitted to CES node allowing it to only retrieve policies. In order to perform an active function, such as POST, PUT or DELETE, a CES node needs to get verified with the SPM either using authentication credentials or by pre-shared keys/certificates.

Policy-API functionality has been divided into sub modules for easy management and maintenance of code while keeping modularity of system as shown in . The node title defines the functions performed by each module. In current implementation, all the nodes reside in

the same system, but connection between Django Web Server and Policy-API is based on HTTP REST queries therefore, both the nodes can reside entirely independent of each other. Similarly, SQL Client interacts with the database which resides locally, but it can also be moved to another system. However, isolating database from SQL Client is not recommended because directly opening MySQL port 3306 for external connection escalates potential threats to the Policy-Database. All other nodes or elements can be distributed to different systems (with dedicated hardware). This would distribute the task of policy creation thus improving efficiency of SPM system. However, this could introduce additional delays due to communication between different modules.

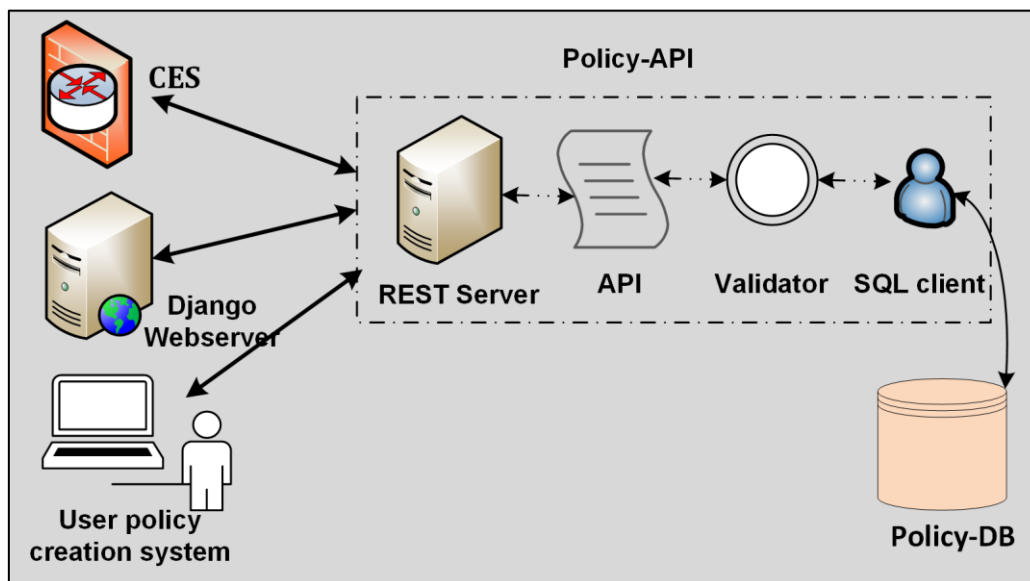


Figure 19 - Implementation hierarchy of SPM

The REST Server is an HTTP entry point to the Policy-API and thus, the Policy-Database. The REST server is responsible to receive all incoming connections and serve accordingly. It has been developed over AIOHTTP, which creates an interface for incoming REST requests and routes them to the relevant function. The response is created by the AIOHTTP (within the REST Server) containing either the data in payload or error message in case of validation error. AIOHTTP replies with the error code and message without forwarding the query to API if the request query either contains bad HTTP headers including URL or does not provide the essential processing parameters for the query. For example; if the incoming URI does not match with any routing function, an automatically generated reply of error 404: "Page not found" is sent to the user by AIOHTTP REST Server. AIOHTTP is not intelligent enough to counter malicious requests with false data, sent with the intent of attack. A new connection establishment, at REST Server, is handled by AIOHTTP without user intervention.

After establishment of a TCP connection between client and REST Server, REST server receives a REST query where, a Universal Resource Identifier (URI) is parsed to extract the parameters for GET and Delete requests. In case of POST and PUT methods, the information is extracted from the HTTP payload. The method and URL of the received query identifies the function to be called in REST Server for further processing of the request. Validation of the

HTTP query is a distributed task and a validation error can be originated from any node of the Policy-API. A dedicated Validator module validates the parameters for POST and PUT request to ensure that only safe and non-malicious policies are stored in the Policy-Database. GET and DELETE functions do not go through extensive validations as these are safe methods, and are not concerned with storing malicious payloads.

After validation of query from Validator module, the request is then returned to the API. The Validator also assigns default values for missing parameters. If the missing parameters are necessary or a validation error occurs, the validator raises an error. The error is replied to sender to modify the HTTP request parameters accordingly. In case of successful validation, the API then creates an SQL query to store policies in the Policy-Database. The execution of query returns None in case of successful execution or returns an error. The SQL Client checks the query for existence of any special restricted character, which can convert the query to a potential SQL-injection attack to Policy-Database. If such character is found, an error is returned to remove the special character. The unsafe special characters for query are combined in a variable list in SQL Client which can be modified depending on the unsafe characters for the database. Figure 20 shows the handling of a request from its reception at the REST Server till its insertion in the Policy-Database. The tasks shown in different stages of Figure 20 are performed by the different modules of Policy-API displayed in Figure 19 .

All nodes in the SPM which are dependent on INPUT/OUTPUT systems, such as MySQL connector, are programmed with 'asyncio' module for asynchronous handling of requests. Similarly, SQL Client is implemented using "AIOMySQL" which is the asynchronous socket connector with MySQL. It creates a pool of connections with the database and handles requests to database in parallel. A single SQL query to Policy-Database is handled by a single connection to which, the SQL Client then replies with either success or the error code (and message in case of unsuccessful execution). On execution of query in the Policy-Database, an error can be raised by MySQL if the provided data does not comply with the table definition or a duplicate entry exists for the provided data. This provides an additional validation of inputs avoiding duplication in tables. For GET requests, the data retrieved by SQL Client is provided to API for further action, whereas in case of POST, PUT and DELETE requests, an OK message is returned to API which shows the successful execution of the query in Policy-Database. An idle TCP connection with the Policy-Database is terminated by "AIOMySQL" after a certain time-period.

For GET requests, the API forwards the retrieved data from Policy-Database to a processing function which formulates the data into a policy format that is executable in CES node. The processing function resides locally in the Policy-API and provides formatted policy to the CES node. This policy-formatting is handled sequentially in the Policy-API, where the performance can be improved by using multiple threads.

Policy-API corresponds to the services and functionalities provided by the PCRF. This resemblance is drawn based on functionalities of validating, constructing, formulating and deciding policy rules that needs to be provided for execution. Therefore, like PCRF, Policy-API can be considered as PDP and Policy Server of our SPM.

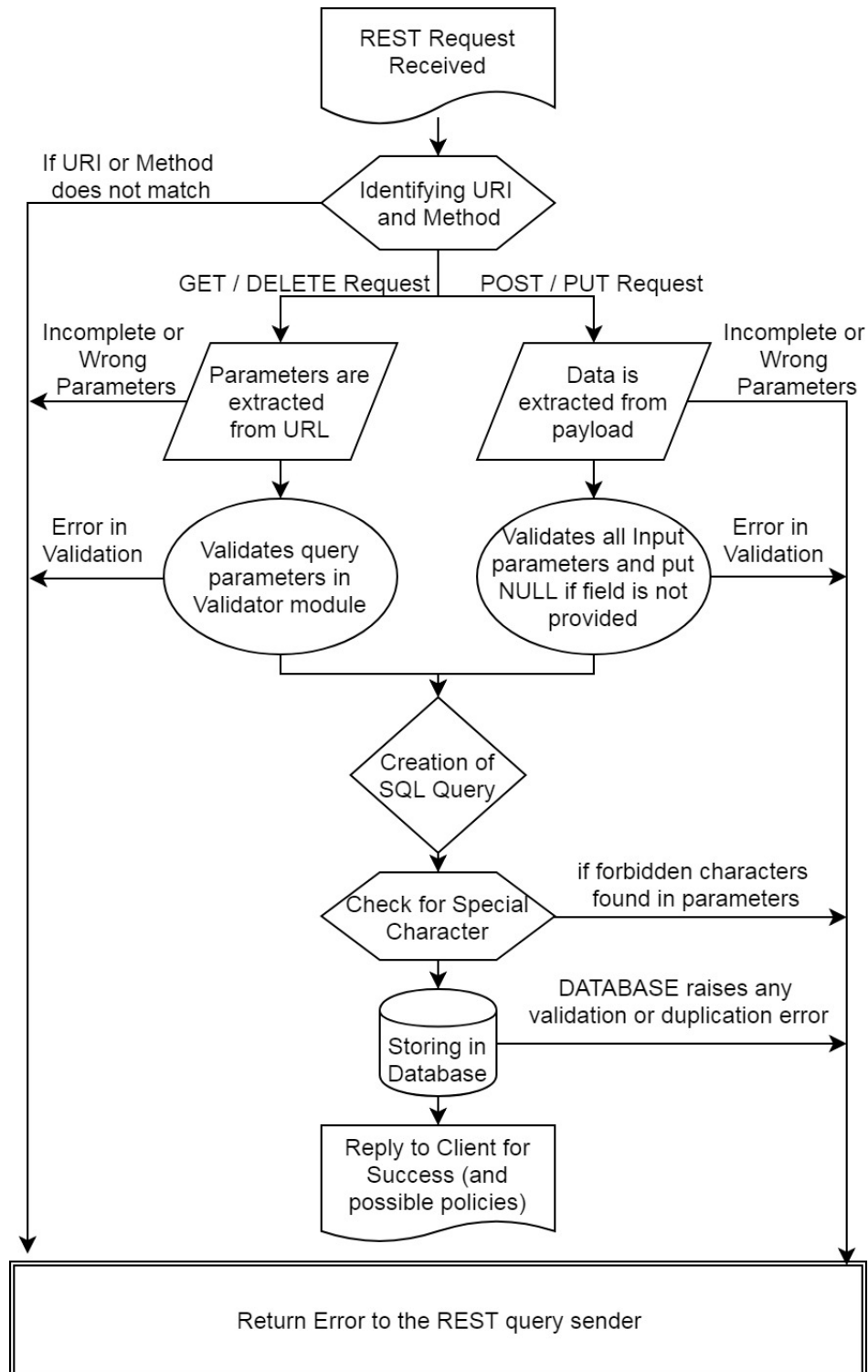


Figure 20 – Flow diagram for Request handling at API

### 5.2.2 Django-webserver

The policy management interface to the Policy-API is based on the Django-webserver. It allows users to get authenticated with the system and manage their policies as well as of their dependents. Login credentials, entered in Django login page, specifies the authorization level of user. It also defines the objects of Policy-Database that the user can modify delete or add.

Django is a MVC architecture in which a model handles the database and tables definition, controller are named as “views” in Django which contains the logic behind actions performed on templates and views are called “templates” in Django which are the frontend webpages of Django-webserver. In model, definitions of all tables have been defined though the tables do not actually reside within Django-webserver, but this assist in producing the form on templates and validate the input fields. As the Django handles user login sessions internally, therefore, an authentication database exists within Django-webserver. The models of Django contain the definition of all the tables described in Figure 16 and Figure 17. The model offers various types of column definition such as IP address, phone number, comma separated list, integer, email address and password hash. If a user enters some alphabets in IPv4 address field, the models raises a validation error which is shown to user, to correct his input.

An additional file called “form.py” in Django-webserver serves the purpose of formulating a frontend form for inserting and editing of policies. Styling of input fields, validation of input data and the selection of error message to be shown in case of validation failure are defined in this file. It also handles the dynamic change of values in choice field such as drop-down menu. While creating a dropdown-menu, form.py retrieves data via REST Server and makes a dropdown menu of the available options. A URL file in Django folder allows to route the incoming requests to specific controller (views.py) function upon a URL hit. The validation of certain input parameters in URL can be performed by this file, and it forwards the parameter values from URL to controller function. Figure 21 shows the flow of query between different modules of Django architecture. Running of the server is automatically handled by Django’s “manage.py” file in the main directory. By default, Django runs on port 8000 and though it can be altered. Default port of Django-webserver has been changed in this thesis to 80 while the REST Server running on the same machine is running on a different port. REST Server needs to be accessed by our own entities (such as CES node, Android Application) therefore, we can configure it to any port. Whereas, the Django-webserver needs to be accessed by users via their browsers who, by default, use HTTP port 80 to access the server.

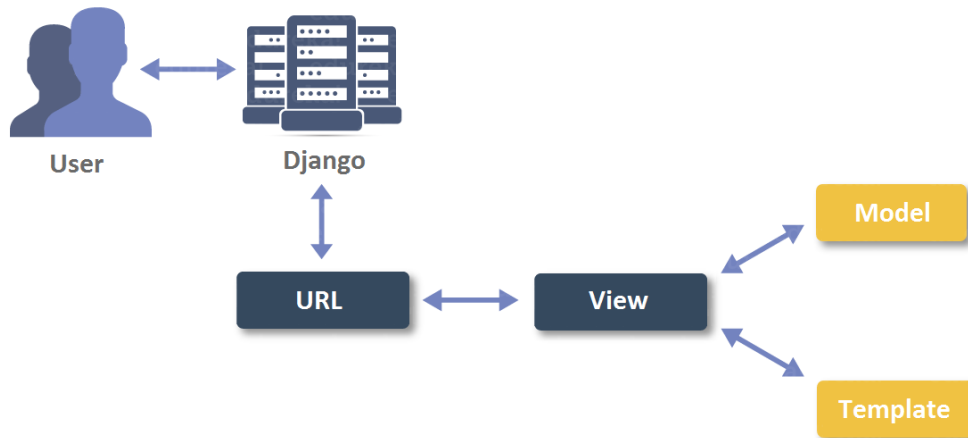


Figure 21 - Architecture of Django Framework<sup>14</sup>

“views.py” is the controller entity of Django and it handles all the logical operations performed via User Interface (UI). Additionally, it is used to manage user sessions and handle Cross Site Request Forgery (CSRF) tokens. Each action at user end has a specific corresponding function in controller. When a user submits a new POST request from frontend, it is received by URL file in Django which forwards the request to relevant controller function according to the URL mapping. Controller forwards the request to “form.py” and model for validation. After being validated, controller create a REST POST request to Policy-API for inserting data in Policy-Database. In case of success, the user is redirected to a webpage or showed an error message if the request was unsuccessful and the HTTP code in reply message from Policy-API is not 200. If validation of input field fails at frontend, the request is not sent to the REST Server to reduce validation load on the Policy-API. The user needs to authenticate himself in Django and then these credentials are sent with the query to Policy-API for user identification while processing policies. Because of this ease of controller configurations, Django has been the choice for development of web GUI as part of this SPM. The frontend which are rendered to user on their web browsers include HTML, Cascading Style Sheets (CSS) and JavaScript, while the python executes loops and variables forwarded to templates by controller (views). By providing a frontend, Django-webserver ensures that only a safe and validated policy, inserted by a legitimate user in SPM.

### 5.2.3 Policy Management Tool (PMT)

The PMT includes all the components that are linked to policy sourcing and management, where Django-webserver is a sub part of PMT. It is responsible for providing an interface for policy management to administrators and subscribers.

A network administrator can use the Django Web-server described in Chapter 5.2.2 to manage policies of complete network including user, system and network policies. The

<sup>14</sup> [online]. Available <https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2017/08/Architecture-Django-Tutorial-Edureka-1.png> [Accessed: Jan. 19, 2018]

administrator has the full access of Policy-Database and can modify any table entry. Moreover, in case of emergency or attack, a customized policy to handle such scenarios is taken into force and is activated by the administrator. An administrator can add or delete users and has the authority to terminate an ongoing session. Our Policy-Database supports adding a new administrator by using a secret key in the signup form or by the existing administrator. In either of the cases, the administrator is not added to the ID table of subscribers but to the login table for administrators. This keeps the administrator's information isolated from Policy-API thus improving modularity of database and table. A policy inserted by the administrator in the database gets higher priority in the policy formation because of which his policy supersedes the user's policy. An example for such scenario is that, a service allowed by the administrator can be blocked by user whereas the vice-versa is not possible as the administrator policy is executed first. Furthermore, a user can put a limit on the allowed services such as restricting number of simultaneous connections, restricting an end-user from accessing a service. This approach helps SPM to maintain minimum level of security and integrity.

A user is provided with two different tools for policy sourcing. The user can use the same Django web server which is used by admin but the authorization scope for management of policies is limited. The extent of limitation and access to policies is determined by the login credentials. Similarly, the priority is allocated to the policy inserted by user according to the credentials which then helps to resolve conflict between policies assigned by subscribers and their dependents. Nevertheless, the policies specified by subscribers always supersede the policies defined by their dependents. An alternative to the web-based system for policy sourcing, is an Android application, which has been developed as part different thesis in similar project [26]. The application interacts with Policy-API through REST Server to manage policies. The application is authenticated with the Policy-API to ensure that only safe policies are maintained in the database. Android Application system maintains a separate database which is then periodically synchronized to populate the changes in Policy-Database. This application automates the policy sourcing by determining the services being used on a user's device and allows to create policy templates for such services. Detailed implementation of this Android application is out of the scope of this thesis and is described in [26].

#### **5.2.4 Policy-Database**

The Policy-Database has been developed in MySQL for storage and management of policies. SQL Client in Policy-API is the only node that interacts with the database through "AIOMySQL" library of Python3. One connection of MySQL connects with only a single database, thus requiring one unique connection per database to handle its tables. In case of SPM, two databases have been defined that includes "Session\_Policies" and "Bootstrap\_Policies" which requires at least two distinct simultaneous connections with MySQL to interact with tables of

both the databases. Database Schema and table structure has already been discussed in Chapter 4.2

### **5.3 Additional features**

Apart from main functions of SPM, few additional features have been implemented to improve the operation of the system and provide flexibility in the SPM. These features add towards the flexibility and scalability of system and make the system compatible with the state-of-the-art technologies. This includes the implementation of Reputation System, User registration methodology and the mechanism of managing various identities of User in mobile and internet networks.

#### **5.3.1 Reputation service**

The communication in CES relies on trust level between various CES nodes which changes according to the malicious traffic being offered. A legitimate traffic increases the reputation of a CES whereas a CES offering potential attacks loses its trust level. A reduced trust level limits the services being communicated with the hosts of such CES.

We have added a reputation table as part of Policy-Database to keep track of remote networks host reputations, which would be populated by a trusted third party. While retrieving policies from API, based on reputation of a remote entity, one can further enforce additional restrictions over the defined policy parameters. Reputation system provides an added layer of security over conventional methods because of which, all CES networks strive to improve their security systems and filter the malicious traffic.

#### **5.3.2 User registration and ID service**

A user is identified by his FQDN within a CES network which is stored in the Policy-Database. This is because an IP address is not a reliable identity of user since it might change frequently. Thus, we devise a method to update DNS about the current IP address of the user for FQDN resolution. Captive portal is one of the solutions to address this problem where a user is identified through allocated credentials. The user ID credentials are mapped to an FQDN which is updated in the DNS when a user attaches with the network or in case of change of IP address of connected user. The technology of captive portal takes the inspiration from web-portal deployed commonly at airports [27], where by it redirects a newly attached user of a network to a login page for authentication. Credentials allocated to users are authenticated using the database and an updated information of user is registered to the DNS server. FQDN for registered users are stored in ID table of database whereas, for guest users, a temporary FQDN can be created for use and populated to DNS server. For future referrals, CES node uses FQDN of registered users as key for policy retrieval from API. This method of



user registration is more suitable for smaller networks. A solution based on tokens or other identity parameters might be more suitable for large networks with their own identity parameters. An example of such scenario is the mobile network where the Subscriber Identity Module (SIM) based credentials can be used for policy retrieval purpose.

The internet technology requires various identities of users to identify an entity. User single type of ID type might not satisfy the need to address the different type of users. A mobile operator might want to use MSISDN as identity of user which would not be an option in case of ISP. In SPM, ID table of database can contain different types of user identities. All these IDs are mapped to 64bit unique identifier which is used as key for policy retrieval from Policy-Database tables. This approach provides flexibility in user identification and possibility of policy retrieval through any of the available IDs. This can be a use case for mobile operators where a unique SIM identifier such as IMSI can be used to refer a user.

## **5.4 Optimizations**

Performance is an important feature for any system design, since the delay induced by SPM will affect all the users of a network. The delay caused by fetching of policies from SPM will add to connection establishment. Policy-Database schema and implementation of Policy-API has been optimized to minimize the delay.

Initially, the Policy-Database contained various columns to store each parameter of policy which allowed the flexibility in Policy-Database providing various filtering options. However, the policies were not in executable format and upon receiving a policy request, the Policy-API retrieved information from Policy-Database and then formatted it in JSON format which is executable at CES edge node. This approach induced considerable delay in fetching of policies from the database and thus, lead to a considerable delay in connection establishment. For optimization, the Policy-Database was modified to store the executable format of policy. The sequence of formatting of policies was changed and in the current implementation, the policies are formatted in executable format upon insertion in the database. This, thus, effected the performance of POST and PUT requests but these methods does not directly affect the performance of CES sessions.

The initial Policy-Database schema consisted of different tables for storage of various types of policy elements. This implementation required N number of round trips to Policy-Database, where N is the types of policy elements in the policy. This effected the policy retrieval time while fetching of policies. In the current implementation, the approach has been optimized by combining different policy types in lesser number of tables to decrease the round trips to Policy-Database. As an example, the different table for CIRCULARPOOL, SFQDN and FIREWALL have been merged to improve the performance. The above two optimizations helped to decrease the RTT by less than half for a policy retrieval request, compared to earlier implementation.

## 6. API manual

This chapter discusses the operational procedure of the SPM. First, we discuss the code structure of SPM along with the classes and the use of different objects of modules in the system. It also discusses the code structure of Django-webserver. The chapter then discusses the procedure for setting up an SPM on a machine along with all the package and software requirements. The setup includes the configurations of SPM along with Django-webserver. Finally, we present the manual for handling SPM through URL end points which includes URLs retrieve, add, modify and delete policies in Policy-Database. The Django-webserver uses the similar URLs mentioned in API to communication with the SPM.

### 6.1 Code Structure

The coding methodology of API is Object Oriented Programming (OOP) where the different modules are programmed in classes and the information and resources are shared among various methods of a class. The error handling is done through “try except” methodology of Python where the location of error suggests the error code for the HTTP response to the sender. For example, an error raised in case of missing necessary parameter will provide a status code of the series of 400 to be sent back to the client. As the implementation of SPM is single threaded, therefore a function taking more than usual processing time would delay all the pending requests. SPM is more optimized as compared to Web-server because the REST Server does not include any HTML and front-page information whereas Django needs to communicate with the templates. The data provided by controller of Django to web-browser is used in the formation of front-end on user’s screen. This section further explains the code structure of SPM and Django-webserver.

#### 6.1.1 SPM

As explained above, OOP is used in the implementation of the modules of SPM. The REST Server is a class which is initialized when the server is run initialized. The AIOHTTP takes host and port as input parameters, so the user can decide the port on which, the server is required to be setup. An asynchronous method “build\_server” in the REST Server is responsible for setting up the server with initialized of object for handling of incoming HTTP requests. It is an AIOHTTP function for creation of server and is called in “\_\_main\_\_”. The function has list of URLs with mapped object functions which are called according to the matched URL and method. A request with the correct URL but with wrong methods is replied automatically with “Method not allowed” error. Similarly, a wrong URL is replied with “404: Not found”. In case of correct method with URL, the request is redirected to the mapped method of object for further processing. A creation of object in REST Server, automatically creates an object for

Policy-API class. The Policy-API class is responsible for actual processing of the request along with the validation of query and creation of SQL query for execution in the database. All the mapped functions, with URL and method, are also asynchronous functions to handle multiple queries simultaneously. The purpose of these functions is to extract information from the received query either from the URL or from payload and pass this information to the relevant object method of Policy-API. These functions also incorporate logging capability, thus providing logging information for every query in the terminal.

API also comprises of a class whose object is created in REST Server on initialization of webserver. On creation of API object, an object for MySQL Client is automatically created to connect to the database. As there are two databases and each connection handles single database, so two objects are created to handle queries for the two databases. The objects of MySQL Client are used in the methods of API and it includes dedicated function for using both the connections. The methods that are specific to Bootstrap\_Policies database uses the object for Bootstrap\_Policies database connector whereas the remaining function uses the connector object of Session\_Policies database. The API class has methods to receive request from REST Server, performs initial validation of query parameters, creates SQL query to be executed in MySQL database, receive the response from MySQL Client and post process the received information to provide it back to the REST Server. Advanced level validation is performed in the validator which is a separate file containing validation functions. These functions are called by the methods of API object for validation of data. The functions in validator validate the input fields of a query along with providing default values for missing parameters and raises the error in case of validation failure of missing of necessary parameters.

MySQL Client object, used in API methods for execution of SQL query in databases, comprises of various methods depending on the type of query. This module also has an internal function to check for the presence of special character. The methods of MySQL Client object methods are primarily divided into two different types including retrieval and execution methods. The retrieval function is called when MySQL provide some information on the execution of query such as GET which is used to retrieve the information from database tables. Contrary to this, execution method does return nothing in case of successful execution of query in the database or returns an error in case of error such as POST request which is used to insert a policy in the Policy\_Database. On Successful insertion of data, MySQL does not respond with any error but an error might occur in case of duplication or validation of input table column fields. Therefore, the hierarchy of information flow is through different objects created on the initialization of the REST Server. This approach optimizes the system performance by re-using the available resources.

### 6.1.2 Django\_webserver

Django has its own MVC framework that we have discussed before. Using the proposed methodology of programming in Django is recommended to achieve better performance results and to keep the smooth interaction between system components. Django has a 'urls' file which is used to map the incoming request to the relevant controller function (controller is views.py file in Django). The controller receives the request and get the query parameters if they are contained in URL or extract them from the HTTP payload if the request has POST or PUT method. The queries are then forward to forms for further validation of the input parameters. The validation in Django is distributed to Models and forms. Forms is the additional file in Django which contains constraints for the input parameters and raises an error if validation is unsuccessful. Forms, after validating the input, forwards the parameters to Models for further verification with the column definition in a table. After being verified by both these entities, the controller then generates a REST query forwarding the parameters and action required to the Policy-API. In case of GET requests, the controller directly generates a data retrieval query to the Policy-API without the intervention of Forms or Models.

Policy-API contains additional functions to retrieve information which is only required by the Django\_webserver and is not needed by CES node or Android application. The error that is raised by Django in validation of a parameter is shown above the form field in template but the error replied by Policy-API is just shown at the top of page in plain text format. All database related operations are sent to Policy-API from Django-webserver, therefore the delay at SPM is sequentially added in all the requests from client's web-browser. As Django\_webserver and Policy-API, both are running on the same machine, so this delay is minimal but can be increased if these modules are geographical placed farther from each other.

## 6.2 SPM Setup

The setup of SPM requires necessary packages to be installed on the system to get the system running. The code is separated in different folders of the project file where "src" contains the most relevant files to setup the server. The folder contains database backups, the code for running SPM through REST Server and the Django application to deploy the Django\_webserver. The transport ports can be changed on both the application through configurations. In Django, it can be changed in command line whereas in Policy\_API, the port can be set in REST Server file. As Django\_webserver interacts with the Policy\_API so a changed port should also be configured in Django\_webserver for proper communication between the nodes. A set of required software, and packages are already mentioned in the installation file. Further information for the SPM setup and the relevant commands are mentioned in the READ.ME file of the project directory.

## 6.3 API end points

The section shows that URLs for accessing or retrieving specified information. There are two different set of requirements for the usage; policy retrieval by CES and policy management by Django and Android application. The scope of both these sets is different in terms of usage, therefore they are separated into two different tables. The tables in Session\_Policies database uses the AIOMySQL connector which provides connection with Session\_Policies database whereas, while dealing with bootstrap policies, the connector with Bootstrap\_Policies database is used. The tables below show URL with HTTP method to perform the function mentioned in Function column. Parameters are the variables in the URL and the possible values for these parameters are mentioned in the column of Possible values.

### 6.3.1 Retrieval API for CES

Table 1 shows the URL which are used by CES node for retrieval of user policies and the all the retrieved data is already formatted in a form which is directly executable at CES node. CES node is involved in only retrieval of policies because of which, all the below mentioned methods are GET centric.

*Table 1 - End Points of API for CES Policy Retrieval*

| Method + Static URL   | Function   | Parameter   | Possible values         |
|---|--|-------------|-------------------------|
| <b>[GET]</b><br>/API/firewall_policy_user/<br><b>&lt;id_type&gt;/&lt;id_value&gt;?</b><br>policy_name= <b>&lt;value&gt;</b>                   | Retrieve single type or<br>all of formatted policies<br>for a user.                      | id_type     | 'fqdn'                  |
|   |  |             | 'msisdn'                |
|   |  |             | 'ipv4'                  |
|   |  |             | 'username'              |
|   |  | id_value    | Value of id of the user |
|   |  | policy_name | 'CIRCULARPOOL'          |
|   |  |             | 'CARRIERGRADE'          |
| 'GROUP'   |  |             |                         |
| 'SFQDN'   |  |             |                         |
|   | Retrieve formatted<br>H2H policies from<br>database matching the<br>filtering parameters | policy_name | 'FIREWALL'              |
|   |  |             | 'available'             |
|   |  |             | 'offer'                 |
|   |  | lfqdn       | Value of local FQDN     |
|   |  | rfqdn       | Value of remote FQDN    |
|   |  | direction   | 'EGRESS'                |
|   |  |             | 'INGRESS'               |
| <b>[GET]</b> /API/ces_policy_node?<br>ces_id= <b>&lt;value&gt;</b> &<br>protocol= <b>&lt;value&gt;</b> &<br>policy_name= <b>&lt;value&gt;</b> | Retrieve formatted C2C<br>policies from database<br>matching the filtering<br>parameters | policy_name | '*' Or Leave black      |
|   |  |             | 'available'             |
|   |  |             | 'offer'                 |
|   |  | protocol    | Transport protocol      |
|   |  | ces_id      | FQDN of CES             |
|   |  |             |                         |
|   |  |             |                         |

|  |   |             |                |
|--|---|-------------|----------------|
| [GET]<br>/API/bootstrap_policies_ces?<br>policy_name=<value> | Retrieve formatted<br>bootstrap policies for<br>CES | policy_name | 'IPSET'        |
|  |   |             | 'IPTABLES'     |
|  |   |             | 'CIRCULARPOOL' |

### 6.3.2 Policy Management API

Table 2 shows the endpoints for management of policies. GET operation in all these endpoints retrieves database information without formatting of data. The URLs, mentioned in below table, are used by Django\_webserver and Android application for the management of policies. The operation includes GET, POST, UPDATE and DELETE. The table name specifies the table in which action needs to be performed. As filtering parameter, multiple column values can be used in the URLs to filter the table data. The below table includes the URLs only for Session\_Policies database.

Table 2 - End Points of API for Policy Management for Session\_Policies Database

| Method + Static URL  | Function Required   | Parameter      | Possible values                         |
|--|---|----------------|---|
| [GET] /API/firewall_policy/<br><table_name>?<br><table_column_1>=<value>&<br><table_column_2>=<value>    | Fetches policies from<br>table matching the<br>provided filtering<br>parameters. All policies<br>in the table are<br>retrieved if no filtering<br>parameter is provided | table_name     | TABLE NAMES                             |
|  |   | table_column_1 | Column name for<br>filter and its value |
|  |   | table_column_2 | Column name for<br>filter and its value |
| [GET] /API/firewall_policy/<br><table_name>/<id>   | Retrieve a single policy<br>by id from the table  | table_name     | TABLE NAMES                             |
|  |   | id             | Policy ID in table                      |
| [POST] /API/firewall_policy/<br><table_name>   | Insert a row of policy in<br>a table  | table_name     | TABLE NAMES                             |
| [PUT] /API/firewall_policy/<br><table_name>/<id>   | Modify a row in a table<br>identified by the id   | table_name     | TABLE NAMES                             |
|  |   | id             | Policy ID in table                      |
| [DELETE] /API/firewall_policy/<br><table_name>/<id>  | Delete a policy from<br>table identified by id  | table_name     | TABLE NAMES                             |
|  |   | id             | Policy ID in table                      |
| [DELETE] /API/firewall_policy/<br><table_name>?<br><table_column_1>=<value>&<br><table_column_2>=<value> | Delete set of policies<br>from table matching<br>the filtering<br>parameters  | table_name     | TABLE NAMES                             |
|  |   | table_column_1 | Column name for<br>filter and its value |
|  |   | table_column_2 | Column name for<br>filter and its value |
| [GET]<br>/API/tables_get_columns/<br><table_name>  | Retrieves the column<br>names of a table and is<br>used by Django forms   | table_name     | TABLE NAMES                             |
| [GET]<br>/API/user_registration?<br>username=<value>&<br>ip=<value>                                      | Receives the request<br>for DNS update from<br>captive portal and<br>stores IP address  | Username       | Username of user                        |
|  |   | ip             | IP address of user                      |

In Table 2, the TABLE NAMES include 'CES\_POLICIES', 'CES\_POLICY\_IDENTITY', 'HOST\_POLICIES', 'HOST\_POLICY\_IDENTITY', 'FIREWALL', 'ID'. The filtering of parameters can be made precise to one row by providing all the necessary parameters. Table 3 shows the URL end-points for the management of 'bootstrap' table in Bootstrap\_Policies database. The end-points perform all four basic functions including GET, UPDATE, INSERT and DELETE. The parameters for Bootstrap policies are limited because the database includes only one table with fewer columns hence, less flexibility in filtering criteria.

*Table 3 - End Points of API for Policy Management for Bootstrap\_Policies Database*

| Method + Static URL   | Function Required                       | Parameter   | Possible values    |
|---|---|-------------|--------------------|
| <b>[GET]</b><br>/API/bootstrap_policies/<id>                    | Retrieve a row from table by id         | id          | Policy ID in table |
| <b>[GET]</b><br>/API/bootstrap_policies?<br>policy_name=<value> | Retrieve rows from table by policy type | policy_name | 'IPSET'            |
|   |   |             | 'IPTABLES'         |
|   |   |             | 'CIRCULARPOOL'     |
| <b>[POST]</b><br>/API/bootstrap_policies                        | Insert a row in bootstrap table         | -           | -                  |
| <b>[PUT]</b><br>/API/bootstrap_policies/<id>                    | Modify a row in bootstrap table by id   | id          | Policy ID in table |
| <b>[DELETE]</b><br>/API/bootstrap_policies/<id>                 | Delete a row in bootstrap table by id   | id          | Policy ID in table |
| <b>[DELETE]</b><br>/API/bootstrap_policies                      | Delete all entries in a bootstrap table | -           | -                  |

## 7. Results and evaluation

This chapter analyzes the implementation of SPM and discusses the results achieved in terms of scalability, reliability and performance testing of the system. First, we argue on the tools explored, their performance benchmarking and leading edge against their counterparts. Next, the performance and efficiency of the system will be debated through results obtained using benchmarking tools. Succeeding to that, the scalability of SPM is analyzed through all major interacting modules. The chapter also explains the additional important modifications which were implemented for optimizing the system. Finally, the performance and scalability are compared in terms of security and efficiency of the system.

### 7.1 Tools benchmark

As mentioned above, Security Policy Management has been developed on python web server using asynchronous HTTP library which is AIOHTTP to get the best possible results using single threaded program. There are no defined figures or numbers of improvement between asynchronous and synchronous programming as it is dependent on the implementation approach. To comply with asynchronous HTTP module, asynchronous MySQL package AIOMySQL is used as a connector to the SQL database. MySQL is an open source database providing remarkable results in performance, efficiency and scalability because of which it has been selected as a database for storing policies in SPM. MySQL provides a maximum of 64 Kilo Bytes (KB) of row size and 256 Tera Bytes (TB) of table size. According to oracle website<sup>15</sup>, MySQL can handle 1.6 million queries per second with 1024 parallel opened connections. Moreover, it can create more than 100,000 connections per second. Frontend has been developed in Django which can handle up to 45,000 requests per second. Moreover, it can handle large number of users and templates because of which it has been a popular choice of many organizations such as Instagram, NASA and Pinterest.

WRK tool has been used to benchmark HTTP server for GET and POST requests which would be the most used methods in Security Policy Management. This tool is open source and involves major contribution by NGINX. The tool allows testing with limiting number of threads, connections and duration of test. To test the POST method, LUA file containing the POST parameters has been used to send data to the server. A self-created asynchronous python code has also been used to test the performance and scalability of system. The results received through WRK and self-created python code matches closely. MySQL database performance has been tested through 'MySQLslap' which can emulate generate burst connection initiation towards the database. Functions of API includes unit testing to check for the smooth integration of new features without effecting the existing system.

---

<sup>15</sup> [online]. Available <https://go.oracle.com/LP=22041> [Accessed: Feb. 20, 2018]



## 7.2 Security and integrity

Four-layer security has been added in the system to avoid malicious policies to enter in the database. Policies are first validated in Django application. Afterwards, policies are received by AIOHTTP server from Django where the input fields are again validated using validation functions and then sent to MySQL for storing in table. MySQL connector then checks for possible SQL injection attack. MySQL database receives the policies from MySQL connector of python and generates an error while inserting if the input value does not match with the column definition. MySQL also raises the error if a duplicate row exists in the database. Django validates the query from the frontend with all the necessary elements and if a validation error occurs, the query is not passed to AIOHTTP server. This basic validation check avoids server overloading due to bogus or malicious requests from users and attackers, thus sharing server load.

Sender might pass the Django application and directly send query to AIOHTTP server where the responsibility of validation is shifted from Django. This case if possible where the Policy-API is not restricted to serve queries from the recognized entities. If the queries are directly sent to AIOHTTP server, the validation tool of server takes the responsibility and generates an error of validation and sends back the response. In case of error, the server replies with an HTTP code in header code field and replies with the relevant error message in payload. Similarly, the error can be originated from MySQL where the HTTP header is created with a predefined code along with a message in payload for sending response to sender.

Security and Integrity is also maintained through user identification in Django. All users get an account on SPM to manage their policies so that an attacker cannot change policies for other users. Moreover, it restricts unauthorized users from interacting with database. Django also caters for Cross Site Request Forgery (CSRF) through CSRF token. This token prevents the attack on web application using already authenticated session. Additionally, Django provides easy Hyper Text Transfer Protocol over Secure Sockets Layer (SSL) (HTTPS) session handling for secure communication between client and server. It also includes protection against clickjacking which is hiding of malicious commands or scripts behind legitimate inputs.

MySQL includes special characters in its queries which can execute different various tasks according to their selection and placement in query. One of such example is semicolon ( ; ) which is used to end a statement. Two queries can be executed in one line in MySQL provided that both the queries are separated by semicolon at the right place. So as a prevention to such attacks, a check for special characters has been implemented in API which checks every SQL query for special characters before execution and returns an error if it finds any.

API also includes parameters, paths, query strings and methods validation which prevents user from inserting policy in the database using a GET request or vice-versa, thus following the REST principles. Furthermore, it also limits the server to handle only predefined parameters and queries which then prevents various potential attacks on webserver and

database that might arise due to flexibility in handling of incoming request parameters at the server.

## **7.3 Performance**

After implementation of the system, testing for its scalability and performance is the most important part to analyze its usage in deployed network. Testing involves performance analysis of AIOHTTP server from the perspective of different HTTP methods and the web frontend developed on Django. Django is not very critical in terms of latency because web frontend can accommodate a delay of few milliseconds. As there can be many users using GUI so Django needs to be tested thoroughly regarding scalability of server. API server has been tested with different input parameters to analyze the performance variation.

The policies are retrieved in two different structures depending on the usage. The policies retrieved by CES node are formatted in a structure which is directly executable at the CES whereas the policies retrieved by Django\_webserver and Android application for policy management includes the plaintext information from the database tables. The same URL end-points have not been used for all the contacting entities because the formatting of policies for Django\_webserver and Android application is needless and an unnecessary processing load of the server at both the ends. The Django\_webserver also does not need policies in formatted form therefore, if it receives the format of policy that is needed by CES node, then Django\_webserver would again need to format it back to its original shape. Therefore, two different types of testing have been performed which include the testing of fetching and inserting the policies into the database through the perspective of policy management whereas the second testing includes the formatting of policies for the user in real scenario with assumed number of policies. The sections 7.3.2 and 7.3.3 shows the results respectively.

### **7.3.1 Test environment**

Tests have been performed on a Linux 16.04 operating system. Two different platforms have been used to test the performance. The performance is first tested on a normal virtual machine with 64-bit architecture over Oracle Virtual Box platform. The other system is on Aalto cloud which has dedicated hardware and the SPM runs on the local host machine without any virtual environment. The requests in both systems is generated locally as propagation time depend on external parameters that are out of the scope of SPM. The specifications of the two systems are mentioned in Table 4.

Table 4 - Test machines specifications<sup>16</sup>

| Machine       | Operating System | Environment  | RAM  | No. of processor cores | Processor clock speed | Processor Type        | Cache | Bus Speed |
|---------------|------------------|--------------|------|------------------------|-----------------------|-----------------------|-------|-----------|
| Local machine | Linux (64-bit)   | Virtual-box  | 3GB  | 2                      | 2.5 GHz               | Intel I5 2400         | 6 MB  | 5 GT/s    |
| Cloud machine | Linux (64-bit)   | Host machine | 16GB | 32                     | 2.6 GHz               | Intel Xeon E5-2640 v3 | 20 MB | 8 GT/s    |

In HTTP, the result and processing delays are dependent on the amount of data retrieved and requests per second. Therefore, same data cannot be used for performance analysis of POST and GET requests. GET test has been performed on both, Local and Cloud machine whereas POST request performance has only been done on Local machine. GET request is directly linked to the performance of the network and initiation of connection between hosts and hence, more concerned about the delay in processing of requests. All performance tests are performed for 30 seconds and the result is an average time per request in milliseconds (ms) for a specific query method.

### 7.3.2 Policy management performance

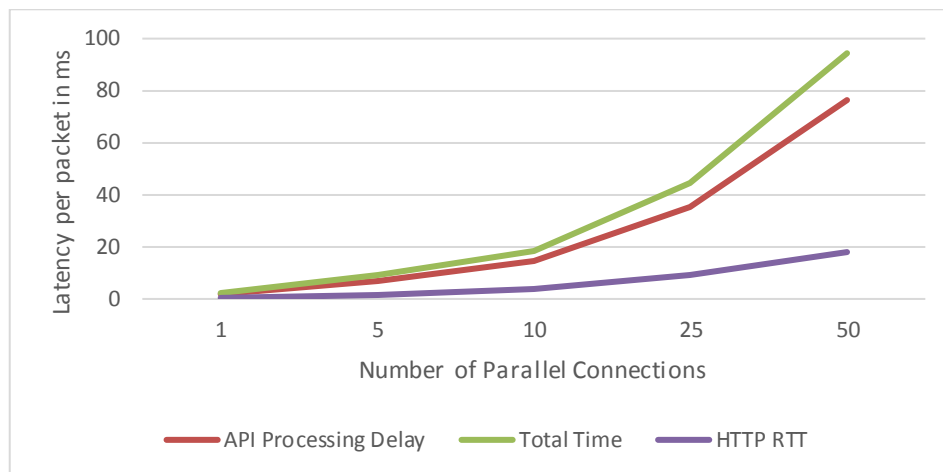
For testing of GET method, 'firewall\_policies' table has been used because it would be most extensively used table of database as user and admin, both needs to handle the firewall policies using this table. The tests have been performed with 50 policies (rows) in the table. Firewall policy that is being retrieved from the database includes policies with all distinct combinations of column values. The types of policies from firewall table include circular pool, carriergrade, user groups, SFQDNs and admin and user policy of firewall. The table contains 20 "FIREWALL" policies, 10 SFQDN policies, 10 carrier grade policies and 5 policies for user group and circular pool each. No extensive post processing is involved in this retrieval of policies because Django\_webserver and Android application needs these policies for policy management.

Table 5 - Performance Results in ms of HTTP GET to API Server with single thread

| 1 thread. 30 seconds | Response Time for Local machine |                     |            | Total Response Time for Cloud Machine |
|----------------------|---------------------------------|---------------------|------------|---------------------------------------|
|                      | HTTP RTT                        | API Processing Time | Total Time |                                       |
| 1 Connection         | 0,41                            | 1,86                | 2,27       | 1,8                                   |
| 5 Connection         | 1,86                            | 7,09                | 8,95       | 7,86                                  |
| 10 Connection        | 3,6                             | 14,83               | 18,43      | 16,89                                 |
| 25 Connection        | 9,17                            | 35,63               | 44,8       | 42,26                                 |
| 50 Connection        | 18                              | 76,39               | 94,39      | 85,23                                 |

<sup>16</sup> [online]. Available <https://ark.intel.com/compare/83359,52208> [Accessed: Sep. 1, 2018]

Table 5 shows the round-trip time of an HTTP query in the first column followed by time taken by server to process a query in second column. The time in second column is the duration of API to validate input parameters (which is table name in this test case), retrieve data from MySQL database. Sum of these two values, the total time taken by the query HTTP RTT and processing at API, is shown in third column of Table 5. The fourth column shows the performance optimization in GET request when Cloud machine was used. Graph 1 shows an increase in latency of requests as the number of connections increases due to the reason that all queries are executed sequentially except the interaction with database and HTTP request/response.



*Graph 1 - Latency per packet for HTTP GET request*

Number of parallel connections also affects the requests per second in HTTP. Number of requests increase with increase in number of connections but after a threshold it starts decreasing. Reason for decrease requests per second is the overloading of server with many requests as single request has to go through multiple sections before being sent back to client. Table 6 shows the number of requests corresponding to number of connections. First column presents the number of requests for HTTP round trip without API processing whereas the second column includes complete processing of query and sending a response. The graphical representation of Table 6 is shown in Graph 3

*Table 6 – GET Requests per Second to Server*

| Connections | Without Processing | With Processing |
|-------------|--------------------|-----------------|
| 1           | 2488               | 481             |
| 5           | 2686               | 578             |
| 10          | 2779               | 632             |
| 25          | 2736               | 542             |
| 50          | 2788               | 495             |

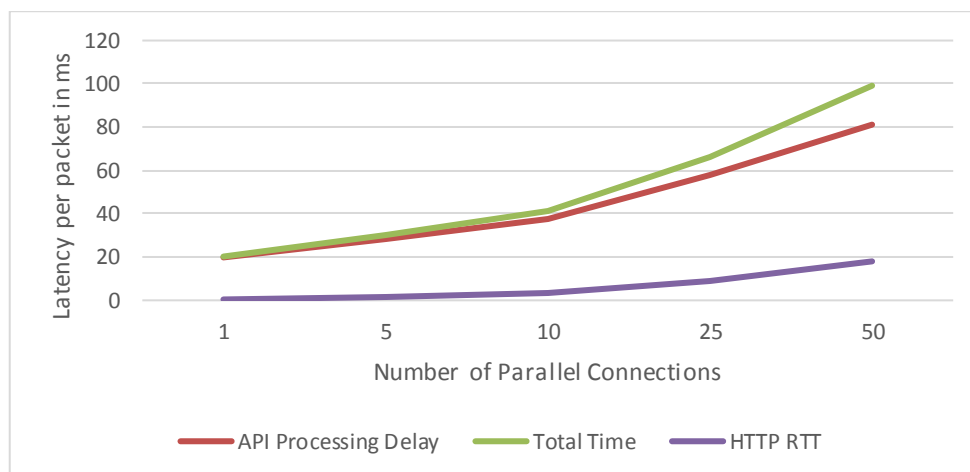
For performance analysis of POST requests, only one row is inserted in the database per query. The query comprises of one policy of firewall type which is the most extensive policy of database constituting of total 17 different fields including different data types. Processing of input data involves validation of field, formatting of fields, putting NULL for optional

parameters that are not provided and cater for duplication of policy. The performance can be improved when using a larger number of policies to be inserted in single HTTP request, a flexibility feature being offered by SPM. Also, significant delay reduction can be observed when sending multiple policies in single HTTP request because it includes single POST request to database for insertion of queries in a single table.

*Table 7 - Performance Results of HTTP POST to API Server with 1 Thread*

| 1 thread. 30 seconds | Response Time per query in milliseconds |                     |            |
|----------------------|---|---------------------|------------|
|                      | HTTP RTT                                | API Processing Time | Total Time |
| <b>1 Connection</b>  | 0,41                                    | 19,82               | 20,23      |
| <b>5 Connection</b>  | 1,86                                    | 28,58               | 30,44      |
| <b>10 Connection</b> | 3,6                                     | 37,78               | 41,38      |
| <b>25 Connection</b> | 9,17                                    | 57,46               | 66,63      |
| <b>50 Connection</b> | 18                                      | 81,15               | 99,15      |

Table 7 shows the same information as Table 5 but the results are for POST request. HTTP RTT is the HTTP round trip time. It is the time when a POST request was created and sent to the server with data and server sent the reply to client without doing any processing. Second column shows the API Processing delay, which includes the time taken by MySQL in insertion of policy and the validation of input at API and MySQL. Third column shows the sum of HTTP RTT and processing time at API which makes to the total time taken by a single request to process. A similar analysis has been shown graphically in Graph 2.



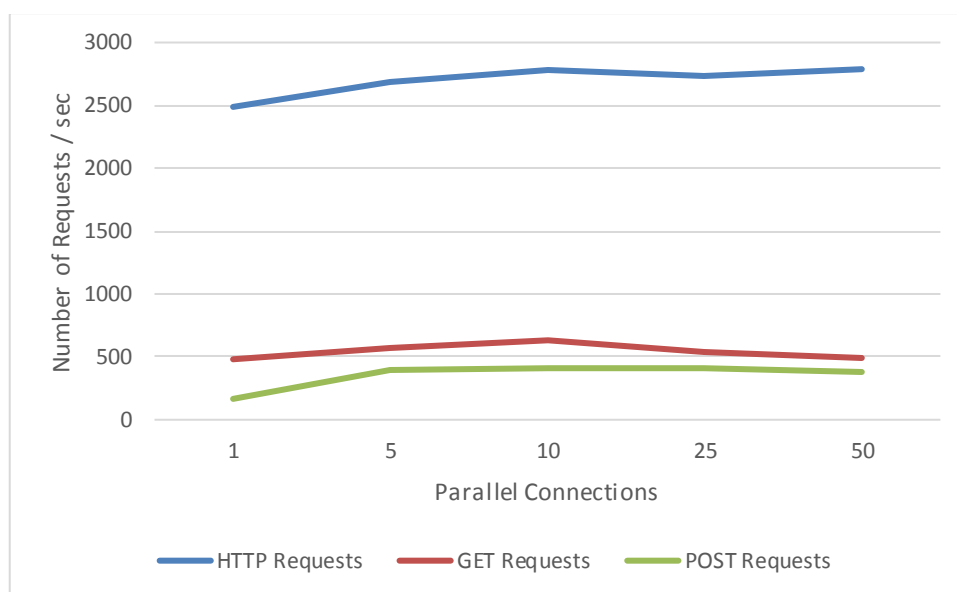
*Graph 2 - Latency per packet for HTTP POST request*

The analysis of graph shows that processing delay and HTTP RTT increase with increase in parallel connections. The reason for delay is that when the number of queries increases at server end, then the delay per query also increases because of more processing requirement. An analysis of requests per second with simultaneous parallel connections is shown in Table 8 which would help to understand the increased processing delay of Table 7.

*Table 8 - POST Requests per Second to Server*

| Connections | Without Processing | With Processing |
|-------------|--------------------|-----------------|
| 1           | 2405               | 166             |
| 5           | 2636               | 395             |
| 10          | 2719               | 410             |
| 25          | 2691               | 410             |
| 50          | 2669               | 383             |

In Table 8, first column shows the simultaneous connection to server and second and third columns show the number of requests per second without and with API load accordingly. We can see from this table that increasing the number of parallel simultaneous connections does increase the number of requests per second but not multiple it. It is due to the load at server which becomes a bottleneck. This increased requests per second induce a greater delay at server end for processing. Graph 3 shows the reduction in number of requests per second to server when a processing load is applied to the HTTP query.



*Graph 3 - GET and POST Requests per Second at API Server*

Results shows that the increase in parallel connections does increase the requests per second but also increases the processing time per query at server end.

### 7.3.3 Policy retrieval performance

The section shows results for the testing of retrieval of policies of a user by CES node. CES node only uses GET method for the retrieval of policies because of which only GET requests have been considered. For performing the test, firewall policies of a single user have been used. Firewall policy that is being retrieved from the database includes policy from two different tables. These two tables consist of the user identification and policies of user. Policy of a single user comprises of 6 different elements from policy table along with identity details. These elements include ID, circular pool, carriergrade, user groups, SFQDNs and admin and user policy of firewall. To test the system for real-time scenario, a total of 30 policies have been used for a user which include 20 firewall policies per user. Added to this, we assume a total of 10 extra policies that comprises of ID, carriergrade, SFQDN and circular pool values. We assumed that there are 20 mobile applications on average running on a user's end device and hence, 20 firewall admin and user policies in the table.

When a GET request is received at the SPM, the policies are retrieved from the database are then structured in the format which can be directly executed at CES node. This two steps procedure increases the processing time for fetching a user's policy and the processing delay at API increases in proportion as the number of rows to be retrieved increases. To analyze the performance of the system and its optimization with high-end machine, the system has been tested over Local machine and on Cloud machine. The results of testing on both the machines are shown in Table 9.

*Table 9 - Performance with high-end system compared with normal system*

| 1 thread. 30 seconds | Response Time for Local machine |                     |            | Total Response Time for Cloud Machine |
|----------------------|---------------------------------|---------------------|------------|---------------------------------------|
|                      | HTTP RTT                        | API Processing Time | Total Time |                                       |
| 1 Connection         | 0,41                            | 2,15                | 2,56       | 2,13                                  |
| 5 Connection         | 1,86                            | 9,41                | 11,27      | 9,14                                  |
| 10 Connection        | 3,6                             | 20,49               | 24,09      | 19,1                                  |
| 25 Connection        | 9,17                            | 50,37               | 59,54      | 48,4                                  |
| 50 Connection        | 18                              | 79,58               | 97,58      | 81,79                                 |

The first column of Table 9 shows the simultaneous number of connections. HTTP RTT is the round-trip time taken by HTTP query without any API processing whereas the sum of API processing and HTTP RTT constitutes the total time taken by a GET query for fetching of user policy. The total time taken by a query on Cloud machine is lesser than Local machine because of better specifications. The analysis in above table shows that the performance has been improved using more computing capacity, but the increase is not proportional. With single connection, a query takes 2,13 second on Cloud machine which is delayed to 2,56 seconds with Local machine. The code has been implemented using asynchronous programming methodology without using threads which limits the code to optimally use the multi-cored processing power. For a single threaded application, a processor with higher clock rate gives

better results than using multiple processors having low processing power per core. Figure 22 provides a graphical representation of the results of Table 9.

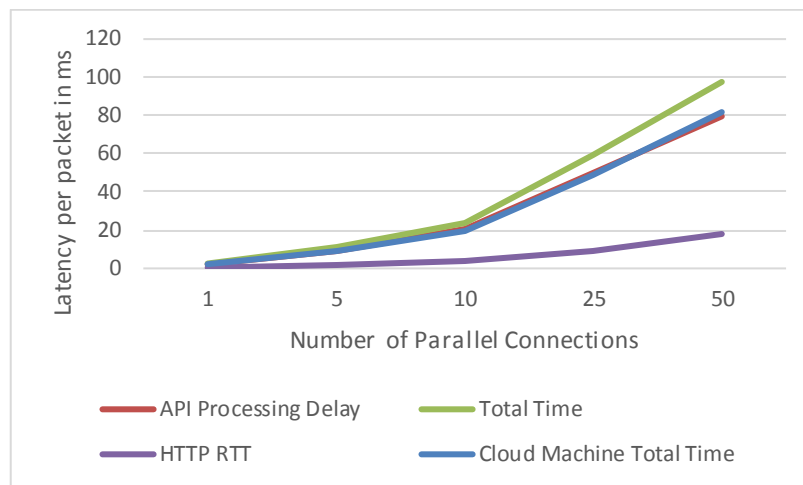


Figure 22 - Latency per packet for HTTP GET request by CES node

Table 10 compares the requests per second served by Policy-API when using a Local machine and Cloud machine. The number of requests have improved while using the system with more power but the improvement is not proportional to the increasing of system specifications. The reason stays same which is the lack of multi-threaded nature of code.

Table 10 - Requests per second with high-end system and normal system

| Connections | With Local machine | With Cloud machine |
|-------------|--------------------|--------------------|
| 1           | 417                | 439                |
| 5           | 503                | 546                |
| 10          | 491                | 523                |
| 25          | 480                | 516                |
| 50          | 475                | 503                |

To get an idea of the delay in successful attachment of user with the CES, let us consider 50 users who attaches with the network simultaneously, and CES generates a request to SPM for retrieval of Firewall policies of these 50 users. As mentioned above, we assume that each user has 20 applications and hence, a total of 30 policies including all other policy types. Therefore, a total of 30 policies per user needs to be fetched which would then make a total of  $30 \times 50 = 1500$  policies for 50 users. The time taken by server in processing of 50 requests and sending the response with formatted policies is shown in Table 11.

Table 11 - Latency for policy retrieval of 50 users by CES node

| 1 thread. 50 requests | Total response time in ms for 50 users |                    |
|-----------------------|--|--------------------|
|                       | With Local machine                     | With Cloud Machine |
| 1 Connection          | 126,41                                 | 110,65             |
| 5 Connection          | 119,53                                 | 101,43             |
| 10 Connection         | 127,40                                 | 110,41             |
| 25 Connection         | 135,72                                 | 118,74             |
| 50 Connection         | 159,01                                 | 141,84             |



The results of Table 11 shows the total time taken by SPM in replying policies for 50 users. The duration of response increases with the increased number of connection as it then effects the balance between the number of requests and the connection opening time. Initiating more than 5 connections with the server delays the query more because the tool then needs to first create connection and then start sending the requests. The results show that it takes only 119,53 milliseconds on a Local machine for 50 users to attach with the system which is acceptable because the fetching of policies is performed once for a user attached to the network. An increased number of requests with higher clock speed processor can improve the response time of queries and help to optimally use the parallel connections

## 7.4 Scalability

The next concern for the analysis of SPM is to inspect its elements in terms of scalability with increasing policy requirements, number of users and large networks. The scalability can be defined as the power of system to handle large number of clients with relatively challenging amount of data. According to the results obtained for number of HTTP requests in GET and POST queries shown in Table 6 and Table 8 reveals that SPM is scalable to the large network scenarios with substantial number of users. The results are graphically compared in Graph 3. The increase in processing delay is caused due to the asynchronous nature of programming. The results can be further improved using threads which the functions can run in parallel. A load balancer which various instances of SPM can further provide impressive performance results for commercial deployment. Having many instances of SPM would eliminate the bottleneck of maximum 1023 connections to the REST Server.

The SPM leverages many users to handle the policies of a single user having similar policy elements. The conflict of policies in such scenarios is elegantly resolved through the priority field of policies which is assigned by SPM on insertion of new policies and is mapped to the login credentials to the system. On request from CES, in final policy creation, the policy with lower priority is overridden by a higher priority policy. The thesis refers only to two stakeholders which includes a network administrator and a user.

*Table 12 - Scalability of firewall policies*

| Users | Basic Policy size (bytes) |          | 1 additional FW rule (bytes) |          |
|-------|---------------------------|----------|------------------------------|----------|
|       | All policies              | Per user | All policies                 | Per user |
| 1     | 334                       | 334      | 453                          | 453      |
| 25    | 8,639                     | 345      | 11,543                       | 461      |
| 100   | 34,845                    | 348      | 46,924                       | 469      |
| 1K    | 351,355                   | 351      | 478,230                      | 478      |
| 100K  | 36,184,413                | 361      | 48,157,144                   | 481      |

Scalability not only refers to number of users or connections being handled simultaneously but also considers the memory element. Memory aspect shows the scalability in terms of increase in storage size when new policies/parameters are added. The analysis

has been performed on different types of policies such as firewall policies, C2C and H2H policies. In these policies, most of the fields are defined by user and only limited and safe policy editing is leveraged to end user. An addition of firewall policy increases the size of database by 119 bytes whose default size is 334 bytes. To start with 100K users, the firewall policy make a storage size of 34.5 MB which then increases to 45.9 MB on addition of single firewall rule by each of these users. The results are shown in Table 12.

*Table 13 - Scalability of CETP-H2H policies*

| Users       | Basic Policy (in bytes) |          | 1 Addition in policy (in bytes) |          |
|-------------|-------------------------|----------|---------------------------------|----------|
|             | All policies            | Per user | All policies                    | Per user |
| <b>1</b>    | 190                     | 190      | 246                             | 246      |
| <b>25</b>   | 5043                    | 201      | 6483                            | 259      |
| <b>100</b>  | 20501                   | 205      | 26391                           | 264      |
| <b>1K</b>   | 215809                  | 215      | 276699                          | 276      |
| <b>100K</b> | 22660535                | 226      | 28949900                        | 289      |

Likewise, a basic H2H policy amounts to 190 bytes in database. The scalability results show that the policy for 100K users is about 21.6 MB. The results on according to increasing number of users is shown in Table 13. Therefore, the total amount of storage required to store firewall and H2H policies of 100K users is almost 56MB which then increases to 6GB for 10 million users. This amount of required storage is quite easy to manage in the advance storage options available in local system and cloud. Thus, SPM can scale to large number of users requiring limited and manageable amount of storage and resources.

## 7.5 Integration testing with CES

The basic purpose of SPM is to provide policies to CES nodes for its automation. CES node queries the REST Server for policy retrieval when a user attaches with the network and when a connection is initiated. This, thus, adds to the delay in connection established at source and destination end before data exchange. Several policies are retrieved on attachment of a user with the network such as firewall policy whereas C2C and H2H policies are retrieved on connection establishment. This affects the connection establishment time as the data communication policies have already been fetched on attachment. The analysis has been performed over the more effected type of policies which are C2C and H2H.

To perform the test, two virtual machines has been considered where one contains the SPM, which has already been discussed before. The second VM runs the test components containing two private networks having independent controlling CES nodes. LXC containers have been used aimed at establishment of test scenario for CES network and hosts <sup>17</sup>. The CES networks consist of hosts using which, a connection is established from host to destination. When a connection is initiated from a host, the host CES retrieves the policies of the host from SPM relying on which, the host CES generates a request to destination CES. On reception of

<sup>17</sup> "LXC," [Online]. Available: <https://help.ubuntu.com/lts/serverguide/lxc.html>. [Accessed: Mar. 03, 2018]

connection request from host CES, the destination CES then generates a REST query to its SPM for policy retrieval of destination user based on which, the trust negotiation proceeds. This negotiation is usually finalized in 2 or 3 RTTs [28].

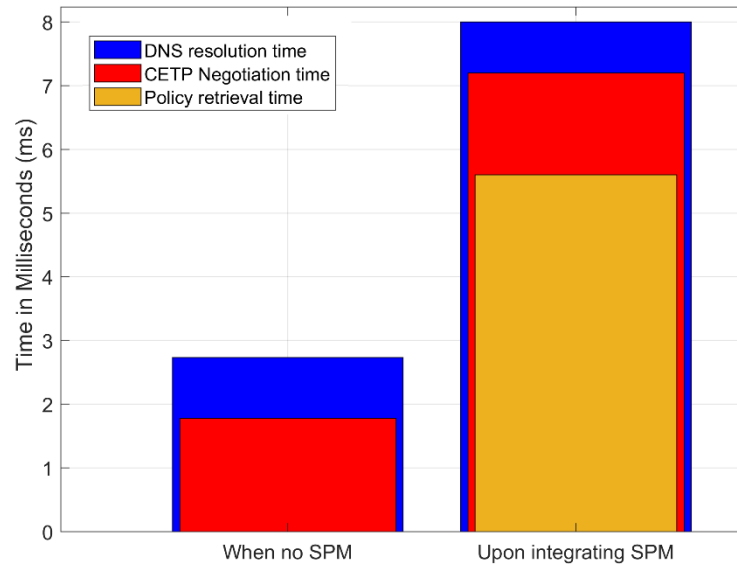


Figure 23 - Impact of SPM integration on CETP-Policy negotiation at CES-CP

The implementation of SPM in the communication hierarchy of two different CES nodes induces an additional delay of fetching policies at both ends. This delay at both the ends is sequential because the initiation message from host CES includes parameters that are fetched from database. Similarly, on destination CES, the query to SPM is generated when a request is received from the host CES. This additional delay of fetching policies from SPM is shown in Figure 23. As shown in the figure, a single RTT of policy negotiation between two CES without SPM takes nearly 2ms and the corresponding DNS request originated by user is served in approximately 3 ms.

On integration with SPM, a delay of 5.5 ms is added to the connection establishment which increases the policy negotiation time from 2 ms to 7 ms and DNS response time from 3 ms to 8ms. This delay of 5.5 milliseconds includes the delay at both the ends for policy retrieval and would be half for each CES network. Therefore, a delay of 2.7ms is added at a CES node for policy retrieval during connection initiation.

## 8. Discussion

The development of SPM over asynchronous module allows to get the maximum of single threaded code of Policy-API developed over python which can further be enhanced using a multi-threaded implementation. A load balancer with many instances of API can further improve the scalability and thus the performance results. The performance is currently comparable to state-of-the-art HTTP technologies but can further be improved using high end processors available as system hardware or in the cloud computing. With respect to scalability, the API is currently able to handle more than 100 users per second which is inspiring for testing stage and limited scale of deployment. Furthermore, the memory requirements for policy extensions can be handled either locally or through the cloud storage. Performance further relies on the upcoming technologies such as HTTP2.0 which handles all requests asynchronously that includes the sending of responses back to the sender.

The ultra-high reliability demanded from 5G network comes under the premise of Future Internet and requires the following features:

- Handling potential threats and vulnerabilities in the conventional internet architecture such as traffic floods, DDoS attacks, address spoofing, phishing and sniffing.
- Network should be able to accommodate user centric security policies which then helps to satisfy end user needs on per connection or session basis. This provides flexibility to the network and end user to allow services for an advance user whereas restrict attacks at network edge for basic or dumb host devices such as IoT.

To address these concerns, CES has been developed as an enhancement to NAT where a cooperative network firewall takes responsibility of protecting end user and only allows relax connections between trusted end nodes. This, thus, helps to mitigate attacks mentioned above along with providing reliable session and data communication between hosts and networks. The reliability is achieved through establishment of trust between the communicating networks through policy negotiations. The user centric policies can tailor the default parameters of policy negotiation based on the host/destination CES nodes, host/destination users or devices and the transport protocol or service. These policies are stored by administrator and user in the database through frontend and backend of SPM. The CES node then retrieves these policy on connection initiation and establishment before the data communication. The approach helps to modify the session between entities on per user basis, therefore providing flexibility to end users.

## 9. Conclusion

The thesis focuses on the development of Security Policy Management for dynamic handling of user policies to provide flexible session handling for ultra-reliable services. SPM, developed in this thesis, provides services to Customer Edge Switching nodes which is a proposed architecture for 5G core network that controls the malicious activities at the edge of network, thus allowing only expected traffic to pass through the gateway. The goal is to provide flexible network management to network operators and end users along with ensuring reliability and security of all entities of a network.

SPM has mainly contributed to the automation of functioning of CES nodes. The other contributions of CES are: 1) leveraging a part of SPM to manage and monitor his policies and alter them in case of need which provides the end user, an autonomous connection and session policies. This approach helps to cater for the IoT use case where the devices are dumb and the responsibility of security is shifted to operators. Therefore, 2) filtering the unexpected traffic for a host at the network edge, thus saving the end user from attacks and waste of resources on processing unwanted traffic. The approach helps to improve the reliability and ensures the availability of services. 3) FQDN has been used as identity of user in this thesis which aims to fill the gap caused by the depletion of IPv4 address and the migration of network devices handle IPv6 traffic. Furthermore, use of FQDN can help to address individual services of a user using SFQDN which then serves the purpose of using DDNS for hosting services over dynamic IP such as web server. 4) SFQDNs allows filtering of traffic and restricting of access on the per service basis which implies that a host can restrict a device for creating an SSH connection whereas allowing to access the web server running on the same machine.

The development of SPM and its experimental implementation along with scalability and performance analysis motivates the operators and researches to integrate the SPM in the traditional mobile-network policy architecture to provide better security, reliability of services and flexibility to handle sessions. The thesis proposes the addition of SPM in the policy architecture of mobile operators and using the CES methodology to cater for the security threats in current internet-networks. Besides mobile operators, the ISPs can also benefit through CES deployed with SPM to provide robust security. In this case, the identification of user can be other parameters such as interface or login credentials to identify user at CES node for policy retrieval. The computing and storing capacity can be provisioned on demand from cloud or other hardware solutions to expand the deployment of CES at large network with millions of hosts.

## 10. Future works

SPM, developed in this thesis, provides a basic framework to store, retrieve and manage policies. It also incorporates capability to handle few well-known attacks and malicious activities. But, it still provides a scope for addition of new features related to network management, network optimization, system security and reliability, and compatibility. The future work for SPM is its expansion to optimize the system by caching the most used policies to reduce their fetching time. Moreover, the SPM can be made secure through latest protocols and apply robust authentications between the communicating nodes. A valuable performance optimization would be achieved by making the SPM code multi-threaded along with asynchronous capability to achieve the better results and to improve the scalability of system. This would enable the SPM to take full advantage of multi core hardware.

Machine learning can be used in SPM in future to create policies dynamically using the rules provided in the database rather than storing the hard-coded policies. This approach would automate the usage of SPM with various networks with minimal user intervention. Furthermore, this enhancement would help system to monitor the changing requirements of users and devices and adapt to the changed network on runtime.

## References

- [1] Rysavy, Peter. "Mobile Broadband: EDGE, HSPA and LTE'." Online Material URL: [http://www. itu. int/ITU-D/imt-2000/index. html](http://www.itu.int/ITU-D/imt-2000/index.html) (2006).
- [2] Donegan, Patrick. "The security vulnerabilities of LTE: Risks for operators." Juniper Networks white paper (2013).
- [3] Jimaa, Shihab, Kok Keong Chai, Yue Chen, and Yasir Alfadhil. "LTE-A an overview and future research areas." In Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on, pp. 395-399. IEEE, 2011.
- [4] Popovski, Petar. "Ultra-reliable communication in 5G wireless systems." IEEE 1st International Conference on 5G for Ubiquitous Connectivity (5GU), 2014.
- [5] Singh, Rakesh Kumar, and Ranjan Singh. "4G LTE Cellular Technology: Network Architecture and Mobile Standards." International Journal of Emerging Research in Management & Technology ISSN (2016): 2278-9359.
- [6] 3GPP TS 29.336, "Home Subscriber Server (HSS) diameter interfaces for interworking with packet data networks and applications," V 12.2.0, Jun. 2014
- [8qqq] 3GPP TS 129.212, "Policy and Charging Control over Gx reference point" V7.4.0 April 2008
- [9] Quality of Service in LTE (2014), Sandvine Intelligent Broadband Networks.
- [10] Ali-Yahiya, Tara. Understanding LTE and its Performance. Springer Science & Business Media, 2011. Chapter 2: Network Architecture and Protocols
- [11] Singh, A., Madhu Pahal, and Neeraj Goyat. "A Review Paper on firewall." International Journal for Research in Applied Science Engineering Technology (IJRASET) (2013): 4-8.
- [12] Wesinger Jr, Ralph E., and Christopher D. Coley. "Firewall providing enhanced network security and user transparency." U.S. Patent No. 5,898,830. 27 Apr. 1999.
- [13] Stallings, W. (2003). Cryptography and network security: principles and practice. Pearson Education India. Chapter 22: Firewalls.
- [14] Bhagchandka, Dhiraj. Classification of firewalls and proxies. Department of Computer Sciences, The University of Texas at Austin, 2003.
- [15] Strassner, John. Policy-based network management: solutions for the next generation. Elsevier, 2003.
- [16] Westerinen, Andrea, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Shai Herzog, A. Huynh, Mark Carlson, Jay Perry, and Steve Waldbusser. Terminology for policy-based management. No. RFC 3198. 2001.

- [17] Stockwell, E.B. and Klietz, A.E., Secure Computing Corp, 1999. Generalized security policy management system and method. U.S. Patent 5,950,195.
- [18] Putzolu, D.M., Intel Corp, 2003. Policy-based network management system using dynamic policy generation. U.S. Patent 6,578,076.
- [19] Sloman, M., 1994. Policy driven management for distributed systems. *Journal of network and Systems Management*, 2(4), pp.333-360.
- [20] Verma, D.C., 2002. Simplifying network administration using policy-based management. *IEEE network*, 16(2), pp.20-26.
- [21] Damianou, N., Dulay, N., Lupu, E., Sloman, M. and Tonouchi, T., 2002. Tools for domain-based policy management of distributed systems. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP* (pp. 203-217). IEEE.
- [22] Matthews, Philip, Rohan Mahy, and Jonathan Rosenberg. "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)." (2010).
- [23] Wing, Dan, Philip Matthews, Rohan Mahy, and Jonathan Rosenberg. "Session traversal utilities for NAT (STUN)." (2008).
- [24] Hammad Kabir. "Security Mechanisms for a Cooperative Firewall". School of Electrical Engineering Aalto University 2014
- [25] J. S. Llorente, "Private Realm Gateway," Master Thesis, Aalto University, School of Electrical Engineering, 2012.
- [26] Fofana Ibrahima Kalil. "Policy Creation and Bootstrapping System for Customer Edge Switching". School of Electrical Engineering Aalto University 2018
- [27] "Captive Portal (Authenticated DHCP)," Infoblox.
- [28] Kantola, R., Llorente Santos, J., and Beijar, N. (2016) Policy-based communications for 5G mobile with customer edge switching. *Security Comm. Networks*, 9: 3070–3082. doi: 10.1002/sec.1253.