Ελληνικό Μεσογειακό Πανεπιστήμιο

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή εργασία

Τίτλος:

Ανάλυση ευπαθειών ως υπηρεσία σε υποδομές λογισμικού/Vulnerability Assessment as a Service over SDN infrastructures

Ιωάννης Γεώργιος Κεφαλούκος (τπ3736)

Επιβλέπων εκπαιδευτικός :Ευάγγελος Μαρκάκης

Επιτροπή Αξιολόγησης :

- **Ευάγγελος Μαρκάκης**
- **Σπυρίδων Παναγιωτάκης**
- **Ιωάννης Παχουλάκης**

Ημερομηνία παρουσίασης : 27/09/19

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στους καθηγητές μου Δρ. Ευάγγελο Πάλλη, Δρ. Ευάγγελο Μαρκάκη καθώς και τον υποψήφιο διδάκτορά κ. Ιωάννη Νικολουδάκη για την καθοδήγηση και την βοήθεια, που συνέλαβαν στην βελτίωση της πτυχιακής μου εργασίας καθώς και στην εκπόνηση της, όπως και για τον ζήλο που μου δημιουργήσαν για να συνεχίσω την ενασχόληση μου στον κλάδο της έρευνας στα δίκτυα και μετέπειτα στην ασφάλεια των δικτύων. Τέλος, θέλω να ευχαριστήσω ολόκληρο το ερευνητικό εργαστήριο Pasiphae Lab για την όμορφη ατμόσφαιρα και την βοήθεια που μου παρείχαν οπότε το χρειαζόμουν.

Abstract

The eruption of new technologies and paradigms such as cloud/edge computing and the Internet of Things, has brought a new era in the ICT domain, by extending ICT resources to infinity, thus allowing for the development and deployment of complex and resource-demanding applications and services, and by introducing millions or even billions of diverse network-enabled devices, providing context and valuable information. Apart from the tremendous positive aspects of this technological revolution, several issues have also been risen, the majority of which concern the security and privacy of infrastructures, data and by extension, the end-users/stakeholders. Large infrastructures face the pitfall of devices entering and exiting their networks, services and terminals operated by untrained and (Cyber) security unaware personnel, render them prone to malicious attacks. Towards addressing these issues, this thesis presents a pure-SDN automated framework that monitors and detects existing and newly-introduced network-enabled entities (devices, services, Virtual Machines, etc.) and assesses them against known vulnerabilities, produces a vulnerability score, based on the CVSS V3.0 standard, and assigns them to a connection-appropriate network slice, depending on the severity of the result/score. This framework was evaluated through a series of measurements and by-far outperformed other research initiatives by more than 70%.

Περίληψη

Η ραγδαία εξέλιξη στον τομέα της τεχνολογίας και η ολοένα αυξανόμενη εμφάνιση νέων παραδειγμάτων, όπως το cloud/edge computing και το Internet of Things (IoT), μας εισάγουν σε μια νέα εποχή στον τομέα των Information and Communication Technologies (ICT), επεκτείνοντας δυναμικά τους πόρους τους, επιτρέποντας μας έτσι να αναπτύσσουμε και να εγκαθιστούμε πολύπλοκες και απαιτητικές εφαρμογές και υπηρεσίες. Επιπρόσθετα, έχουμε ένα πλαίσιο στο οποίο εισάγονται εκατομμύρια η ακόμα και δισεκατομμύρια ποικίλες δικτυακές συσκευές το οποίο μας παρέχει πολύτιμες πληροφορίες για αυτές. Εκτός από τις τεράστιες θετικές πτυχές αυτής της τεχνολογικής εξέλιξης, έχουν δημιουργηθεί καινούργια ζητήματα και έχουν αυξηθεί μερικά από τα ήδη υπάρχοντα, τα περισσότερα από τα οποία αφορούν την ασφάλεια και το απόρρητο των υποδομών, τα δεδομένα και κατ' επέκταση τους τελικούς χρήστες/εμπλεκόμενα μέλη. Οι μεγάλες υποδομές αντιμετωπίζουν το πρόβλημα ότι δικτυακές συσκευές εισέρχονται και εξέρχονται από τα δίκτυα τους συνεχώς, επίσης η ύπαρξη υπηρεσιών και τερματικών τα οποία διαχειρίζονται από μη εκπαιδευμένο προσωπικό/απληροφόρητο σε θέματα ασφαλείας, τις καθιστά επιρρεπείς σε κακόβουλες επιθέσεις. Για την αντιμετώπιση αυτών των προβλημάτων, η πτυχιακή αυτή μελετά, υλοποιεί και παρουσιάζει ένα αυτοματοποιημένο πλαίσιο εφαρμογών SDN το οποίο, παρακολουθεί και ανιχνεύει υφιστάμενες και νεοεισαχθείσες δικτυακές οντότητες, αξιολογώντας τις έναντι γνωστών ευπαθειών βάση του CVSS V3.0 standard και τα εκχωρεί σε ένα κομμάτι του δικτύου του (network slice),ανάλογα με τη σοβαρότητα του βάση του αναφερόμενου αποτελέσματος. Αυτό το πλαίσιο εφαρμογών αξιολογήθηκε μέσω μιας σειράς μετρήσεων των οποίων τα αποτελέσματα συγκρίθηκαν με άλλες υφιστάμενες εργασίες και παρατηρήθηκε ότι η παρούσα λύση είναι κατά 70% αποδοτικότερη.

Table of Contents

List of Figures

List of Tables

1. Introduction

The ongoing growth of Cloud Computing (CC) Edge Computing (EC) and the Internet of Things (IoT) in the ICT domain, offers the ability to develop resource-demanding services and applications especially for complex and multi-layered infrastructures [1][2]. The recently adopted Bring Your Own Device (BYOD) paradigm, only adds up

to the ever-increasing number of connected devices in ICT infrastructures, wherein network-enabled devices and entities expose their resources, services and interfaces, creating new opportunities for attackers, since the attack surface is exponentially widened. Moreover, the operators of these devices/entities, are often employees with limited knowledge or even complete lack of awareness concerning (Cyber) security aspects/best-practices, further adding up to the problem at hand [3]. The vulnerabilities imposed by the afore-mentioned entities, are often missed, or even neglected by System Administrators, rendering the fortification and maintenance of the underlying production network, nearly impossible. To tackle this issue, installation, configuration and maintenance of dedicated appliances/services (e.g. firewalls, proxy-servers, etc.) is required, which is a costly and complex task.

According to ENISA's "Cyber Security breaches Survey", 2018, over four out of ten businesses in the UK (43%) suffered a breach or attack, whereas 74% of businesses stated that cyber security is a high priority for their organizations' senior management, wherein only three out of ten (27%) businesses have a formal cyber security policy or policies enforced, which is a huge contradiction. Additionally, according to Gartner's statistical report, more than 8 million IoT devices were installed during 2017, and the projection for 2020, was more than 20 million connected IoT devices, Thus, more than 250% increase.

All the above-mentioned issues illustrate the need for a solution that will allow administrators to manage and fortify their networks, in an automated manner, without imposing manual interaction. There have been several research endeavors towards this direction, whereas most of the literature handles the issue either in a non-fully automated manner, or just focus on the monitoring and detection of vulnerabilities. Finally, to the best of our knowledge, all existing research initiatives depend on third party components, not utilizing novel ICT paradigms, such as SDN, at their full capacity. Therefore, to address these issues, this thesis introduces Vulnerability Assessment as a Service (VAaaS), by presenting an automated, (private) cloud-based, pure-SDN framework that monitors, detects and assesses existing and newly introduced network-enabled entities (devices, services, sensors, etc.), against public databases of reported vulnerabilities. It produces a classification score, based on the standardized Common Vulnerability Scoring System (CVSS) V3.0, and finally assigns them to a connectivity-appropriate network slice. The proposed framework was evaluated through a series of experiments, which illustrated a significant performance boost (more than 70%), compared to other research initiatives.

The rest of the paper is structured as follows. Section II, presents the state of the art regarding vulnerability assessment in the ICT domain. Section III, presents the technology enablers. Section IV, presents the implementation details (System Architecture and Use Case) of the presented framework. Section V, presents the evaluation procedure of the proposed framework and its outcomes. Finally, Section VI concludes this paper with a brief discussion on the outcomes of this thesis and the presentation of foreseen future steps

2. State of The Art

There have been various studies and research initiatives that tried to tackle the issue of untrusted devices entering and leaving a network. In this section we will present the most recent and most relevant ones.

S. Lee et al. proposed a security assessment framework specifically designed for Software Defined Networking [1]. The framework automatically produces several attack scenarios for SDN networks and assesses the underlying network, based on them. In addition, blackbox fuzzing techniques are deployed to detect potential unknown attack scenarios. Although reproducing the existing attack scenarios is a great way to assess the network, it still requires human interaction. In addition, new attack patterns can only be detected from a log file, leading to additional human interaction needed to assess it and act. Following this example, F. Loi et al. proposed a suite consisting of security tests[2]. The security tests entail assessments on i) Confidentiality (whether the data is in plaintext, encoded or encrypted) ii) Integrity (checks for replay attacks and DNS security), iii) Access Control and Availability (DoS attacks) iv) Reflection (malformed packets that sends ICMP messages, SSDP broadcasts and SNMP requests). While these security tests assess the system for potential vulnerabilities that each device may be susceptible to, F. Loi et al. have not taken any measures to address those vulnerabilities.

Taking public networks into consideration, E. T. Tchao et al. presented an assessment framework, which was evaluated on a University campus, using the Bring Your Own Device (BYOD) paradigm [3]. In their paper, they proposed a Multi-faceted authentication model to recognize patterns and usual threats to alert the network administrator. Even though the authors offer solutions for monitoring and assessment, these solutions also require human supervision.

A solid contribution for security enforcement in the IoT domain, IoT Sentinel, was proposed by M. Miettinen et al [4]. IoT Sentinel restricts communications between the vulnerable device and the attacker. It identifies the devices' types and uses a vulnerability database to pinpoint the vulnerable devices on the network. Although Sentinel is a well-developed framework, it utilizes a non-standardized assessment scoring system. Additionally, regardless of the magnitude of the vulnerability, the vulnerable device will be assigned to a non-trusted virtual network, thus blocking it even if it has little to no impact at all to the security of the network.

M. Ficco et al. presented a hybrid simulation (Emulation and simulation) platform by utilizing OpenVAS[5] agents for critical infrastructure systems, to perform penetration testing, vulnerability analysis and virtual resource allocation to allow the assessment of virtual assets, in a non-direct manner [6]. Although M. Ficco et al have detailed data from the OpenVAS agent and the penetration testing, they refrain from taking semi or fully automated actions about the vulnerable virtual/physical devices.

Ali et al. approached the issue by adopting the OCTAVE Allegro methodology [7]. This methodology analyzes how the information is used by devices and users in a system, while it provides guidance, worksheets and questionnaires for the assessment process. OCTAVE Allegro is a well-tailored assessment tool for smart homes. While countermeasures have been proposed, the main focus of Ali et al. lies in identifying the threats.

Ziegler et al. proposed ANASTACIA that demonstrates a holistic solution enabling trust and security-by-design for cyber-physical systems [8]. ANASTACIA is a suite of distributed trust and security components and enablers that are able to dynamically orchestrate and deploy security policies, while assessing risks in complex architectures. ANASTACIA also has an isolation mechanism that assesses the risk by monitoring information related to system behavior and real-time monitoring. ANASTACIA is an advanced framework, offering abounding benefits for security and trust assessment but at the same time not a pure SDN solution, as it presents a more complex architecture that could lead to possible issues at securing them from untrained or (cyber) security unaware personnel.

Nikoloudakis et al. proposed a vulnerability assessment framework utilizing an OpenVAS agent that based on the results of the CVSS score, it assigns each device to a specific VLAN, limiting traffic, granting WAN and LAN traffic or blocking its inbound and outbound traffic from the network [9].The proposed framework, as a mitigation action assigns devices to connection-appropriate VLANs, according to their vulnerability status (CVSS score), providing a layer 2 solution, thus not utilizing SDN at its full capacity.

In contradiction to some and complementary to some other research initiatives mentioned above, we propose a pure SDN-based framework that:

- Monitors existing and newly introduced network entities (devices and services), in real-time
- Maintains a database containing various META-data concerning their vulnerability status, connectivity, IP/MAC address, etc.
- Performs vulnerability assessment on entities, against a wide range of known vulnerabilities, periodically and upon discovery, utilizing a VAaaS scanner, based on OpenVAS
- Produces a detailed report and a standardized CVSS score that reflects the vulnerability status of the assessed entity

Assigns the assessed entity to a connection-appropriate layer 3 network slice

3. Technology Enablers

In this chapter we will present all the technologies used, while explaining the use and tasks expected from each one of them. The afore-mentioned technologies used are all Open-Source and are available for academic purposes for free.

3.1 Cloud Computing

Cloud Computing[10] provides a way for the end user to have compute and data resources available on-demand. More specifically, there are advanced data centers that provide computing power and data storage without the need for the end-user to manage these resources. The key aspect of cloud computing when paired with SDN, is the scalability it can offer and the minimization of costs in the up-front IT infrastructures. In addition, cloud computing offers data loss prevention through its policies and backups. There is also a noticeable improvement regarding security in cloud computing infrastructures. Lastly, Cloud Computing offers almost no downtime, with the dynamic allocation of resources offered, software updates can be done with minimum downtime. Figure 1 illustrates a minimal architecture of Cloud Computing and Figure 2 depicts the benefits of Cloud Computing.

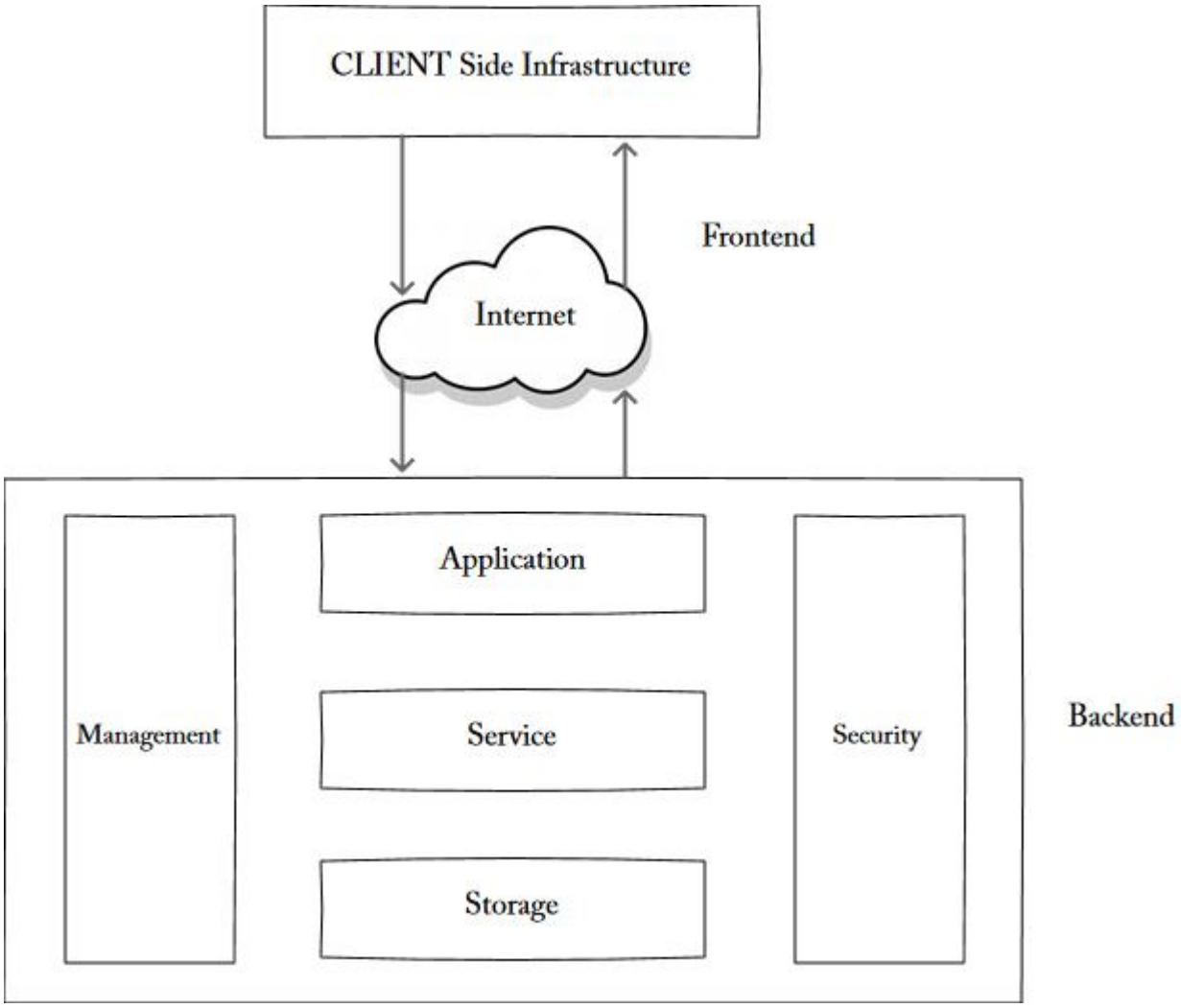

Figure 1 Cloud Computing Architecture

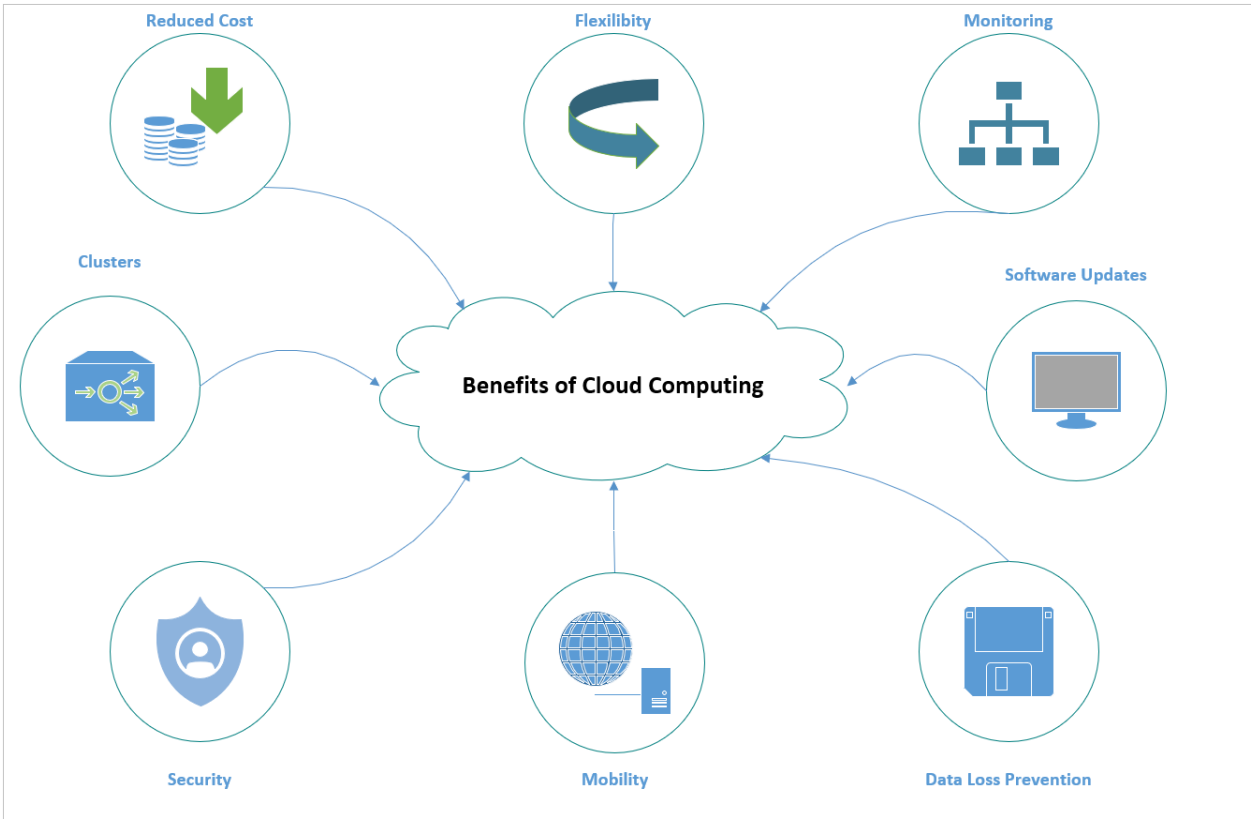

Figure 2 Cloud Computing Benefits

3.2 Software Defined Networking (SDN)

Software-defined networking (SDN)[11] is a technology that allows us to easily and dynamically configure our networks, by centralizing the network intelligence (control plane) in a network component (SDN Controller) and manage it programmatically, thus erasing the need to manually change the configuration of thousands or even million switches whenever there a need. SDN paradigm separates the logic of the network (control plane) from the forwarding (data plane). SDN offers considerably better network performance and monitoring utilities. Traditional networks are decentralized and complex, making it hard to operate and troubleshoot. To establish communication within our network, the SDN controller creates specific flows for every SDN-enabled switch in its topology (in a Cluster each controller is aware only of its own switches and does not interact with the others). In conventional networks we have a forwarding table established in every router. Flows are a set of rules based on which SDN-enabled switch packets will be forwarded, thus eliminating the need for routing tables. In addition, SDN-enabled switches accept flows only from their own controller and don't have the ability to create their own flows. The control plane can work with one or more controllers(cluster). The main problem with the SDN paradigm is the new and old security issues arise with the paradigm. While the security issues are not to be ignored, we also have new ways to detect attack patterns and mitigate attacks. Lastly, for the communication between the controller and the underlying switches, SDN utilizes the OpenFlow[12] protocol that Is widely used for the communication between control and data plane.

It's important to state that since network configuration is provided only by the SDN controller, the need for multiple – vendor specific routers, switches and protocols is eliminated, thus simplifying the network at reduced cost. There are several SDN controllers such as i) OpenDaylight[13], ii) ONOS[14] , iii) Project Callico[15], iv) NOX/POX[16], V) Project Floodlight [17] , VI) Ryu [18] and various others. From the afore-mentioned controllers OpenDaylight, ONOS, and Floodlight are the most well-known production-ready controllers, which offer tremendous aspects into an SDN network and simplicity compared to others for the configuration

Figure 3 illustrates the simplicity of SDN networks, compared to the complexity of traditional networks

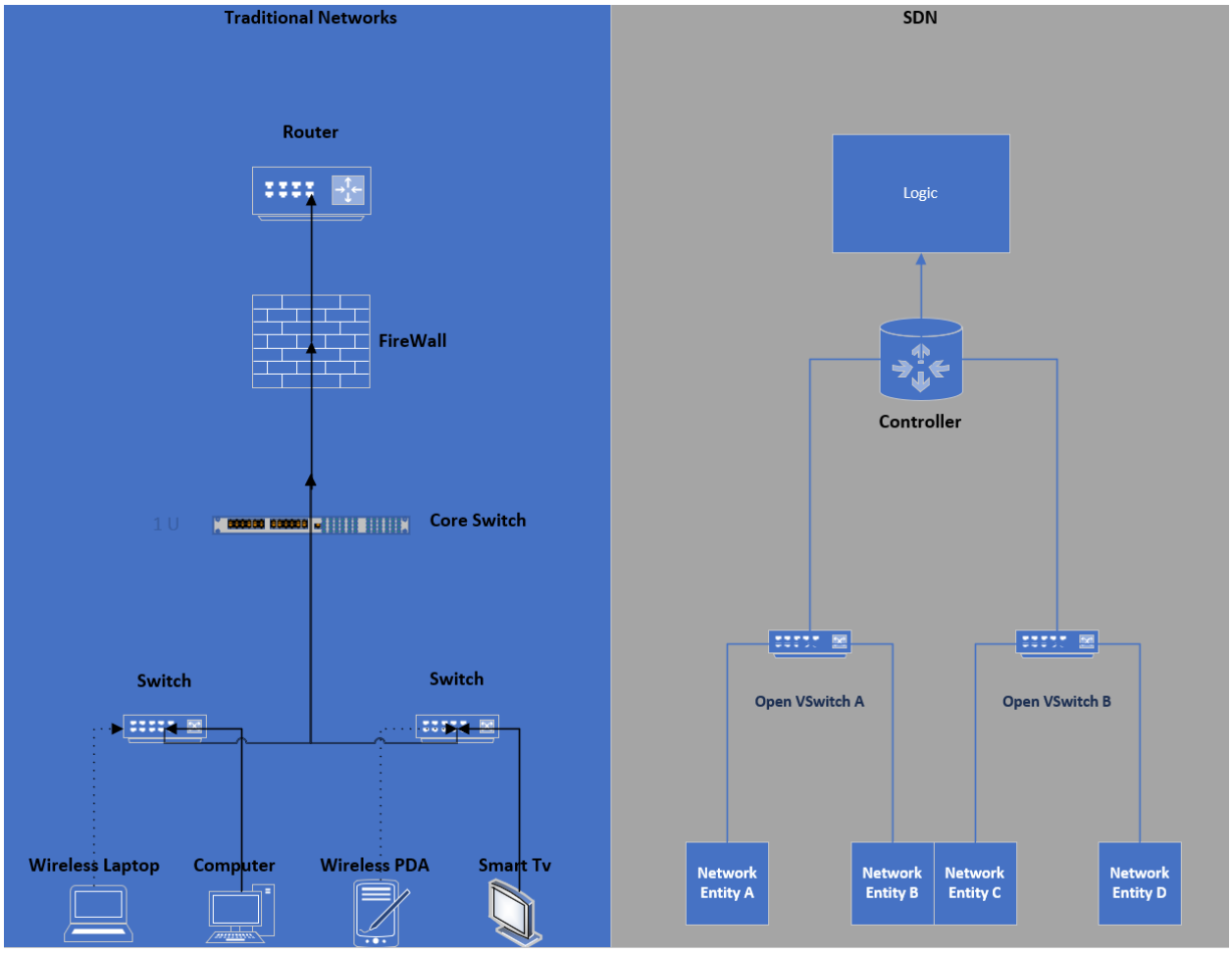

Figure 3Traditional Networks - SDN

3.3 OpenDaylight Controller

OpenDaylight is a production-ready SDN controller. It is a modular, extensible, scalable and multi-protocoled controller. The very first version of OpenDaylight (Hydrogen) was released in February

2014 and the software is written in Java. There are multiple novel characteristics regarding OpenDaylight such as multitenancy and integration to OpenStack through APIs. Figure 4 illustrates its dashboard

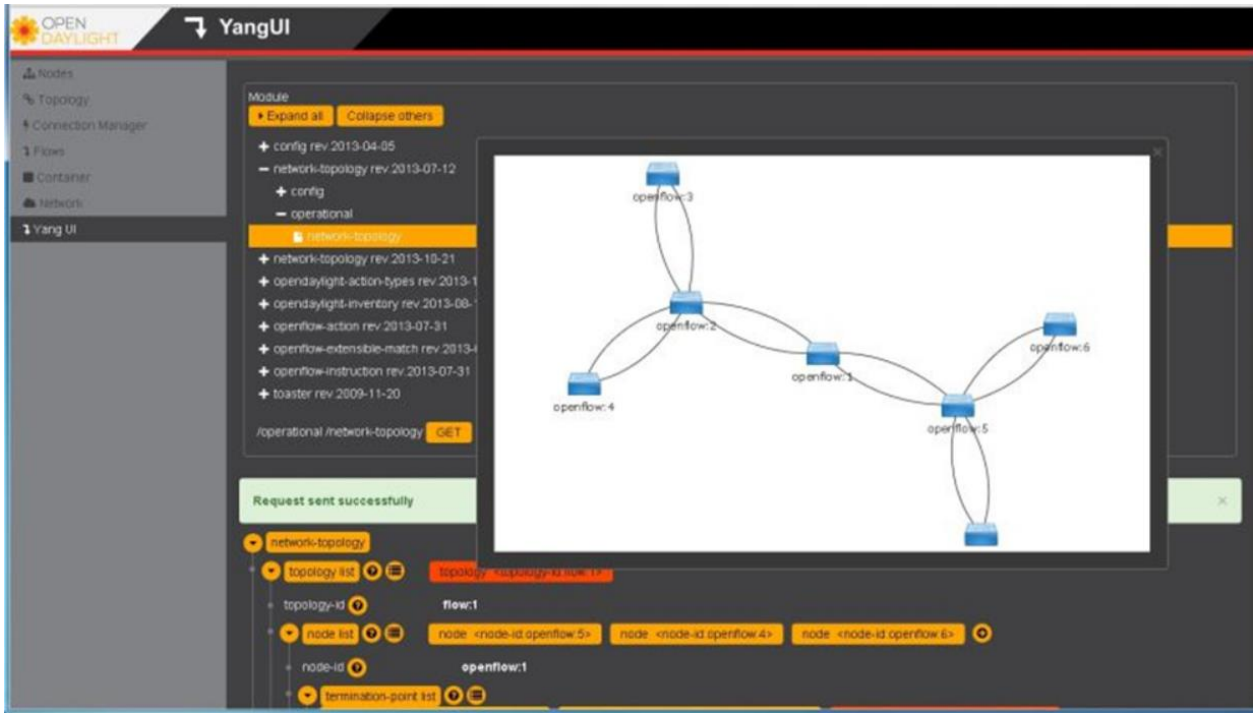

Figure 4 OpenDaylight Dashboard

3.4 POX Controller

Pox is developed in python. One of the biggest key features of POX is that it "runs anywhere" and uses reusable sample components for path selection, topology discovery etc. The main disadvantage of POX is that it's mostly a learning SDN controller and not suitable for production environments. Figure 5 shows POX terminal and *miniedit* (tool to create, configure and use network simulations)

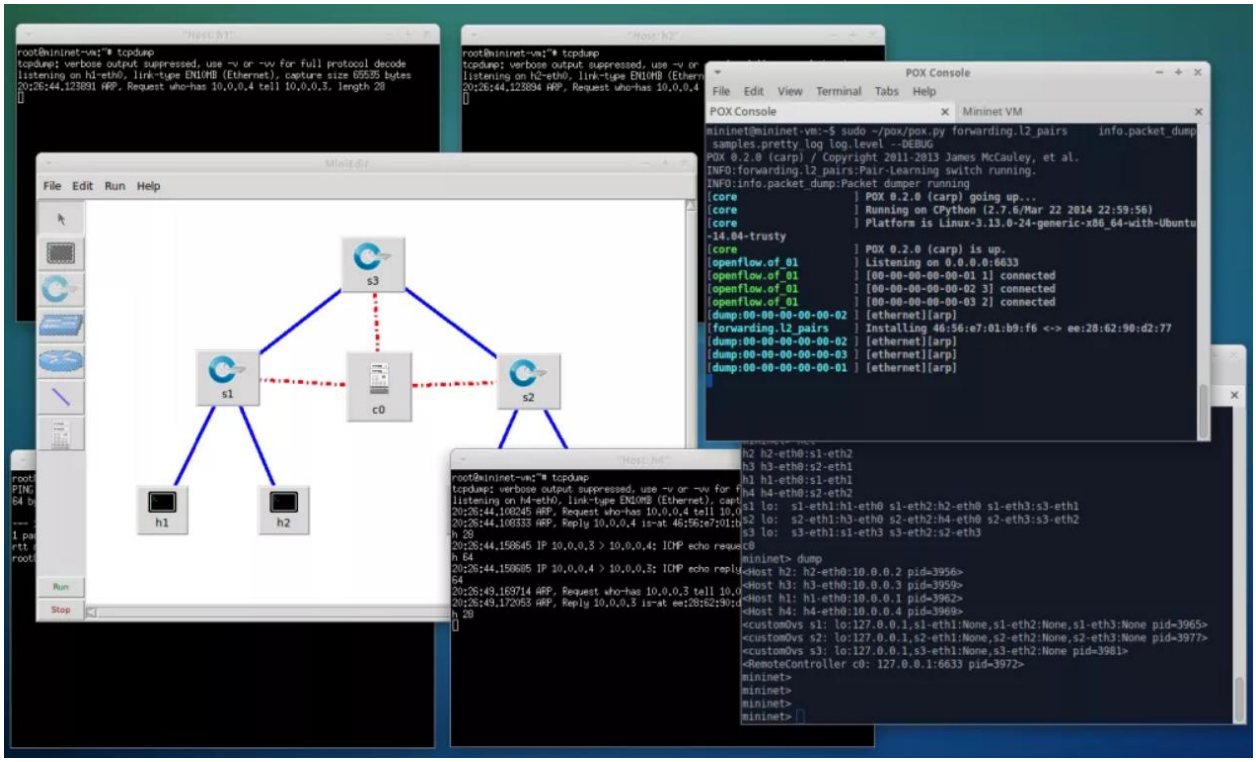

Figure 5 POX-Miniedit

3.5 Project Floodlight

Project Floodlight not only contains the OpenFlow controller but also includes a collection of applications and services on top of the controller. Floodlight is a java-based OpenFlow controller. It is considered an easy to use controller. Figure 6 below shows the interaction between the controller, the applications (java based) and the applications that interact with the Floodlight REST API, while Figure 7 illustrates its dashboard

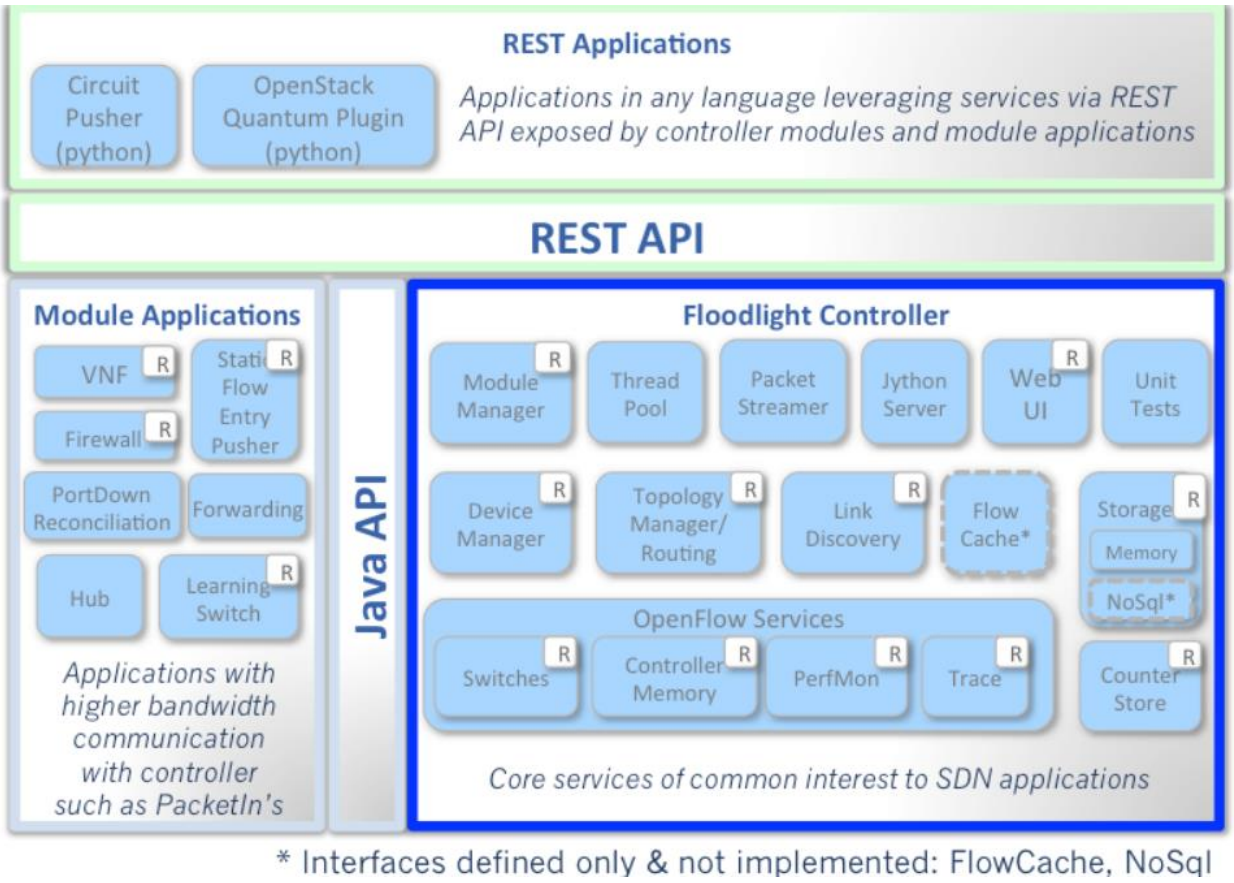

Figure 6 Floodlight Architecture Diagram

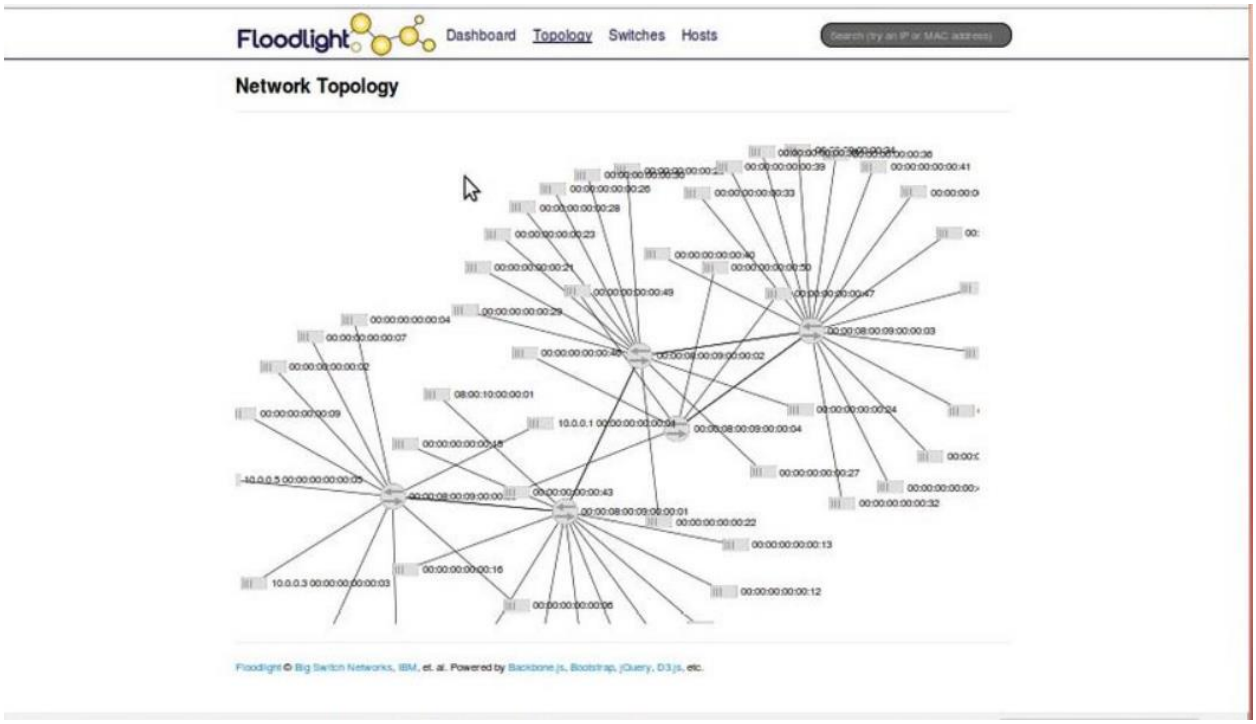

Figure 7 Floodlight Dashboard

3.6 Ryu OpenFlow Controller

Ryu is an agile framework for SDN application development. It allows modification of existing and implementation of new components. It is fully written in python and supports various protocols for networking (OpenFlow, Netconf, OF-config, SNMP etc.). It also supports integration with other projects (OpenStack, IDS (snort) etc.). Lastly, Ryu is an event driven framework that it is generic enough to be used without OpenFlow. Figure 8 presents Ryu controller real time monitoring.

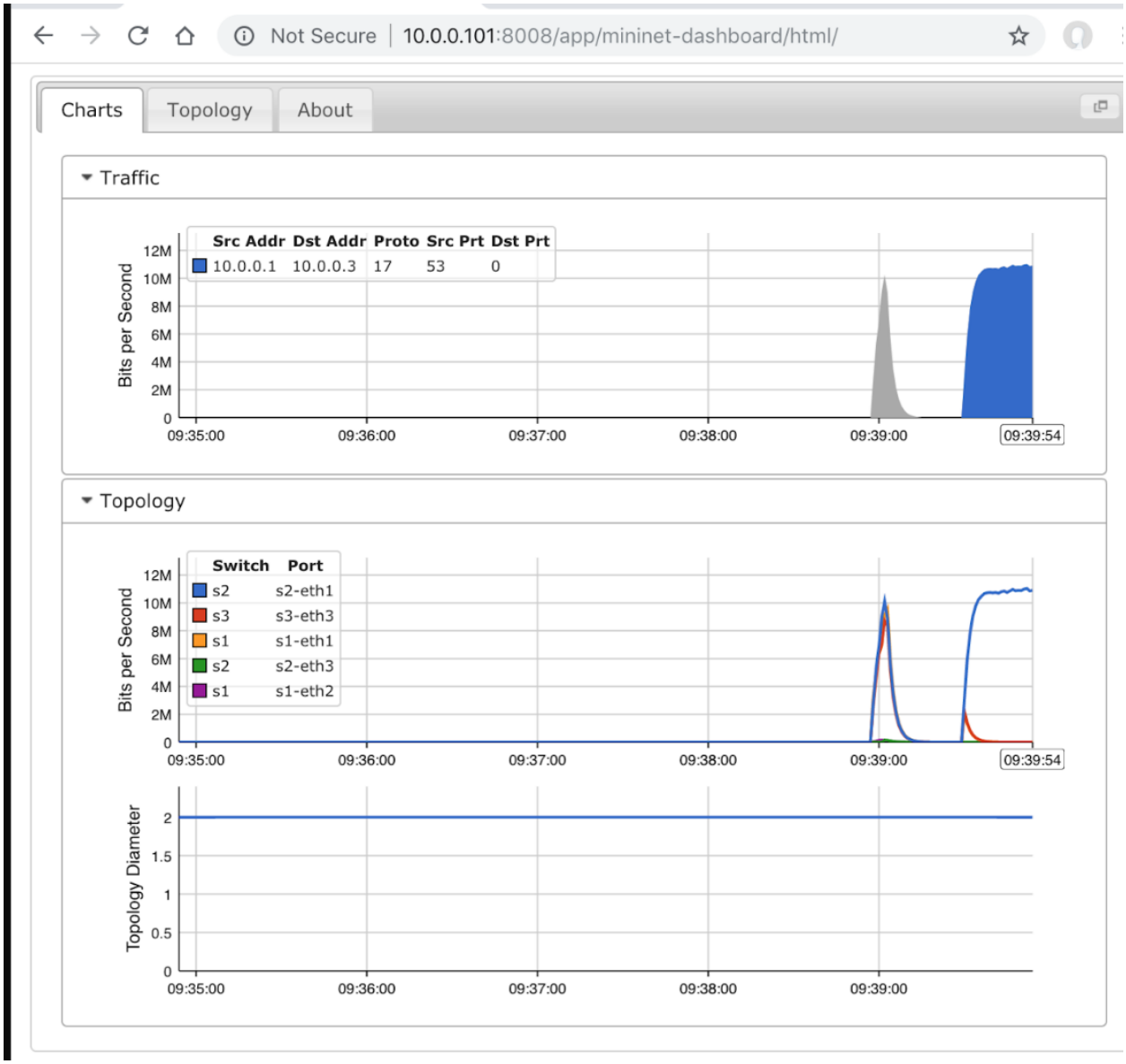

Figure 8 Real Time Monitoring (Ryu)

3.7 ONOS Controller

The Linux Foundation, developed an open-source SDN operating system-community project called Open Network Operating System (ONOS)[14]. The software is Java based, providing distributed SDN applications. A significant advantage of ONOS, in contradiction to other SDN controllers is that its system is designed to operate as a cluster, thus making it a viable solution whenever there is a failure to a specific node without disruptions. REST API, Graphical User Interface (GUI) and Command Line Interface (CLI) are the means to communicate with ONOS. ONOS offers the ability to load-unload its core extensions (services) dynamically via either CLI, REST API or even GUI. The services do not require the reboot of the system to work. In ONOS version 1.15.0 there are 172 applications included and the deployment of them is simple. Figure 9

illustrates the ONOS Dashboard, Figure 10 illustrates ONOS applications. ONOS is a production ready solution.

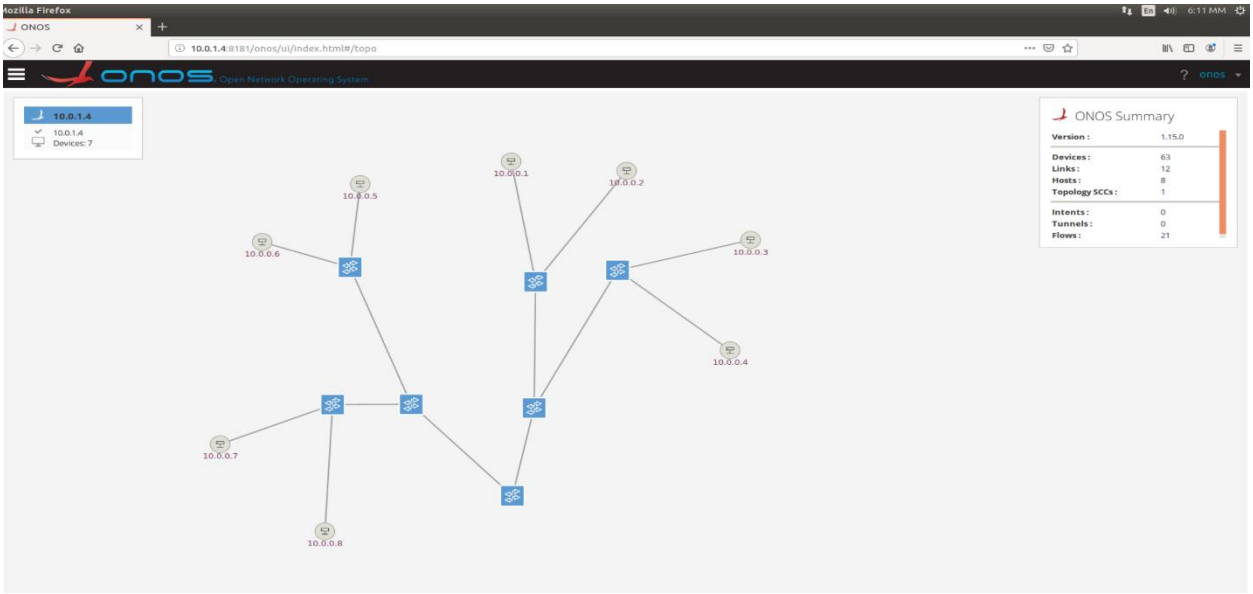

Figure 9 ONOS Dashboard

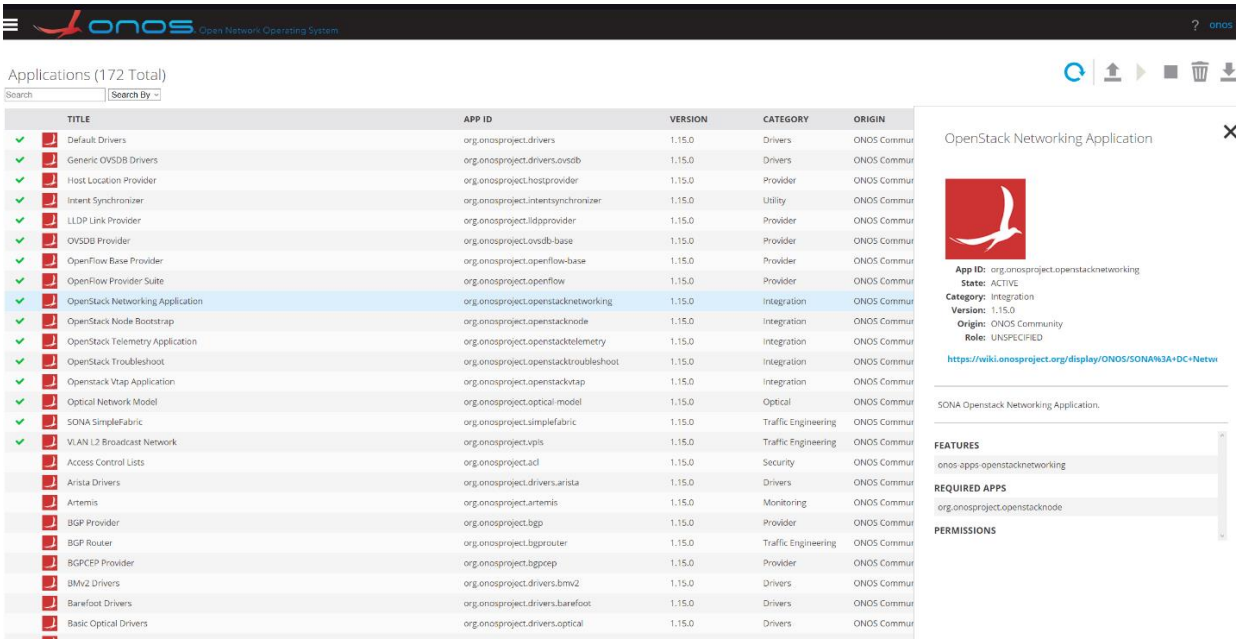

Figure 10 ONOS Applications

3.8 SDN Controller Selection

Based on the need of our thesis and taking into consideration the afore-mentioned SDN controllers, we concluded that two of them were suitable for our needs (ONOS and OpenDaylight). The aspects

we were looking for, was their production-ready status, ease of use, dynamic scaling and integration with OpenStack

OpenDaylight and ONOS both provide network management for OpenStack, they are both production-ready controllers and easy to use. ONOS has been designed and built for enhanced performance and seamless scalability. ONOS' goal is to keep the response time for requests at its northbound interface less than 50msec. To achieve that ONOS scales on-demand by introducing new instances of ONOS when more capacity is required. On the contrary, OpenDaylight is not optimized for scaling, with issues regarding VXLAN scalability, startup time, memory consumption and the use of many threads.

Sona[19] is an optimized tenant network virtualization service for ONOS. SONA consists of three ONOS applications and is responsible for OpenStack integration

- OpenStackNetworking
- OpenStackNode
- Set of assistant applications (Networking UI,Vtap,Troubleshoot,Telemetry)

OpenStackNode, manages and boostraps compute and gateway nodes

OpenStackNetworking, manages the network slices and provides the flow rules needed to have a stable network. OpenStackNetworking, calls REST APIs that Neutron (OpenStack) provides. Whenever there's a network change request (entity connecting/disconnecting, or the logic service requesting a specific entity to be assigned to a specific slice) the request is post-committed to OpenStackNetworking. Then by identifying the entity, which needs to be changed by its port universally unique identifier (UUID) the service provides the flow rules needed (installation, deletion or modification). To conclude with, OpenStackNetworking is also responsible for ARP and DHCP requests.

OpenDaylight is a network management provider for OpenStack through the Modular Layer 2 (ML2) plugin. The ML2 plugin is installed into OpenStack controller node (where Neutron is) and its available as a Python package.

To conclude, OpenStack is managed through Neutron for both controllers and the reason ONOS is selected is for the dynamic scalability feature it offers

3.9 Infrastructure as a Service (IaaS)

There are several IaaS frameworks, but in this subsection, we will present the most well-known ones that are Nimbus[20], Eucalyptus [21] , OpenStack [22] , OpenNebula [23] and XSEDE Software Stack[24]

Nimbus, is open source and the service provided is either via Web Services Resource Framework (WSRF)-based or with Amazon's EC2 WSDL web service APIs. Eucalyptus on the other hand is

mostly used for Amazon Web Services (AWS) but its paid and not opensource. XSEDE Software Stack are usually requested whenever there is a need for high performance computing. OpenNebula and OpenStack, are the key IaaS of interested and are explained thoroughly. A more detail description of every IaaS mentioned below.

3.10 Nimbus

Nimbus is a highly compatible open source IaaS, which with the tools it contains it can provide computing power and versatility. Nimbus also provides the means to combine it with OpenStack, Amazon or other clouds.

3.11 Eucalyptus

Eucalyptus

Eucalyptus works on top of Hypervisors such as KVM[25],Xen[26],VMware[27]. It can be integrated with other IaaS platforms such as Amazons Elastic Compute Cloud, to form a hybrid cloud. Through its interface we can configure compute, network and storage resources. We can also configure our systems (Controller, Cluster, Storage) to be redundant in order to make them resistant to failures. In disregard of its paid model, Eucalyptus key feature allows for dynamic scaling of its computing and storage resources on-demand, based on the load of each application.

Figure 11 illustrates Eucalyptus Dashboard

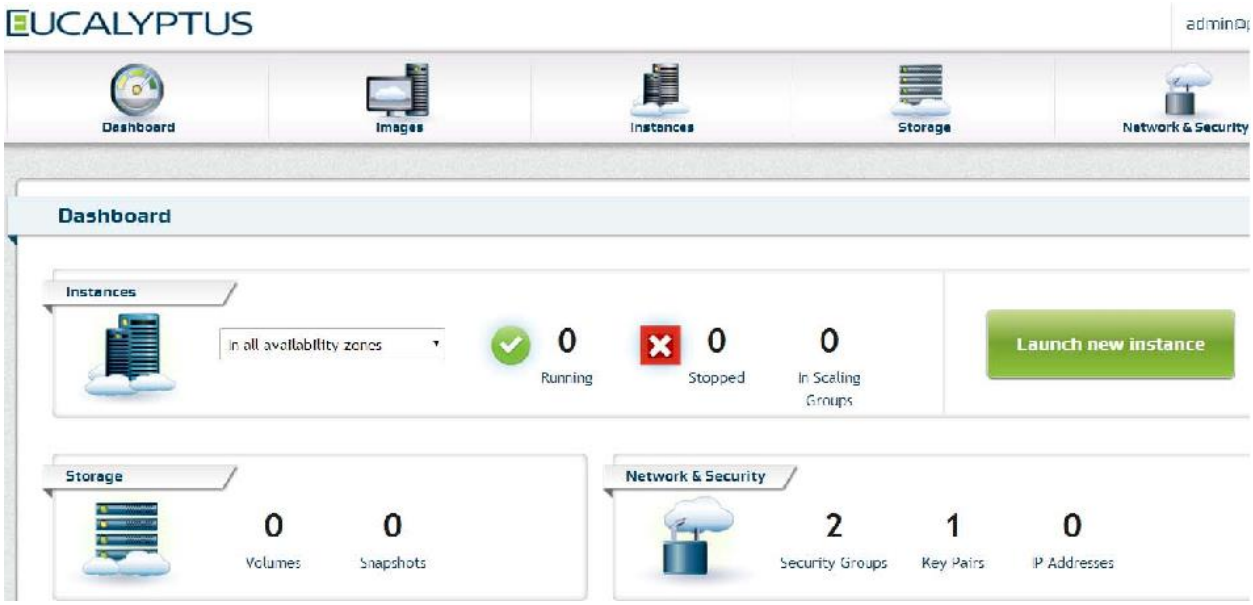

Figure 11Eucalyptus Dashboard

3.12 XSEDE Software Stack

The Extreme Science and Engineering Discovery Environment (XSEDE) acts as a virtual machine[28] (VM) where, mostly scientists use to share computing resources data and expertise. The XSEDE Software Stack includes a lot of services and software making it a supercompute-like service for sharing. Figure 12 illustrates the active XSEDE users by field of science Portal

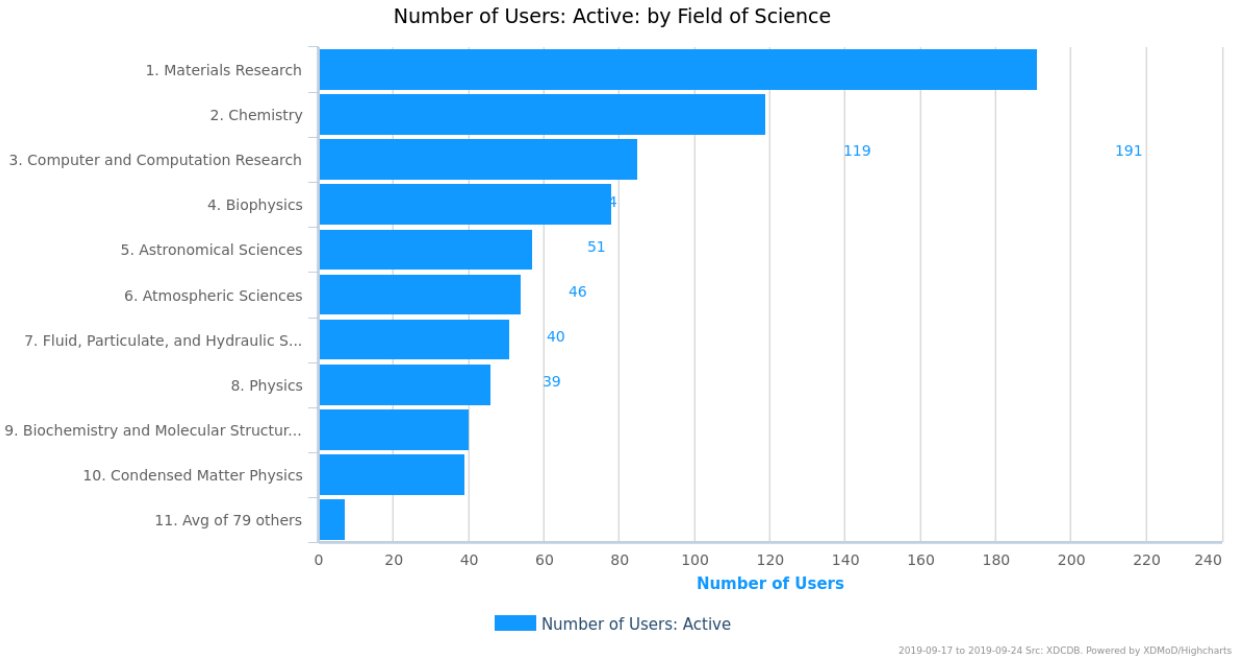

Figure 12 XSEDE Portal (Users Active by Field of Science)

3.13 OpenNebula

They key aspect of OpenNebula's platform, is the management and creation of public, private and hybrid cloud implementations of IaaS. The platform can either be used for data center virtualization or cloud infrastructure solutions. Through specific policies, OpenNebula can combine both cloud and data center resources. OpenNebula, is responsible for the orchestration of storage, network, security and monitor services, in order to deploy VMs on distributed cloud infrastructures. OpenNebula, is compatible with several cloud interfaces (Amazon EC2 Query, OGF Open Cloud Computing Interface and vCloud) and hypervisors such as Xen, KVM and VMware. Figure 13 illustrates the OpenNebula Dashboard

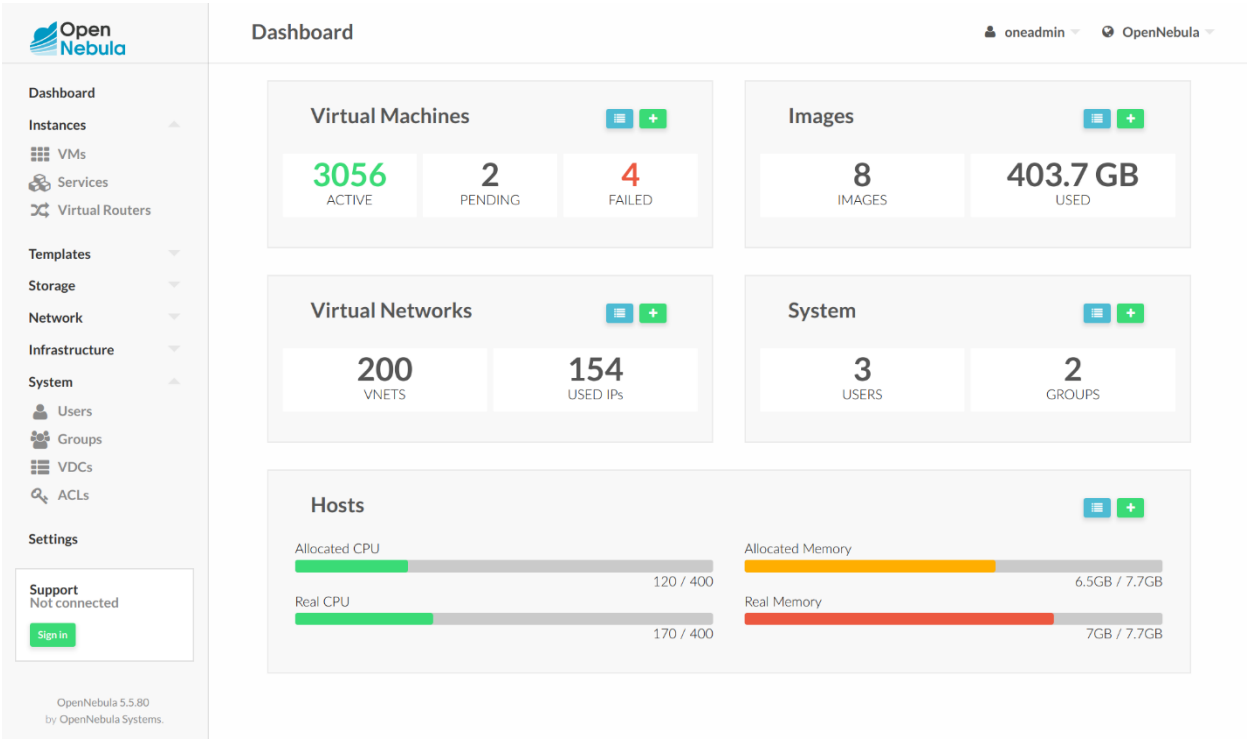

Figure 13 OpenNebula Dashboard

3.14 OpenStack

OpenStack[22], as mentioned at the beginning of the chapter, is an open-source software framework for creating public and private clouds. The communication, between users and OpenStack, is managed through either CLI or RESTful API. The framework utilizes tools for the creation and configuration of virtual machines that can have various operating systems (OS). There are 2 services that are of vital importance to this thesis, Neutron and Nova

- Neutron
  Neutron, provides the network configuration needed in order to successfully connect the compute node. Neutron is included to the core part of OpenStack
  The main idea behind Neutron is the modular layer 2 (ML2) plugin that its main function is to utilize a variety of layer 2 (l2) network technologies at the same time, it implements a lot of network types (local, GRE, VXLAN, VLAN) and the means to access them.
- Nova
  Nova is responsible to create virtual machines and bare metal servers. There are 3 ways to interact with Nova: Horizon (web GUI), OpenStack Client (CLI), Nova Client (advanced configuration, not recommended). Most of the features are available to be configured through REST API.
  In order for Nova to have some basic function Keystone, Glance, Neutron and Placement services are required. Figure 14 illustrates OpenStack Dashboard and Figure 15 illustrates a basic diagram of the components needed in order to have a basic function

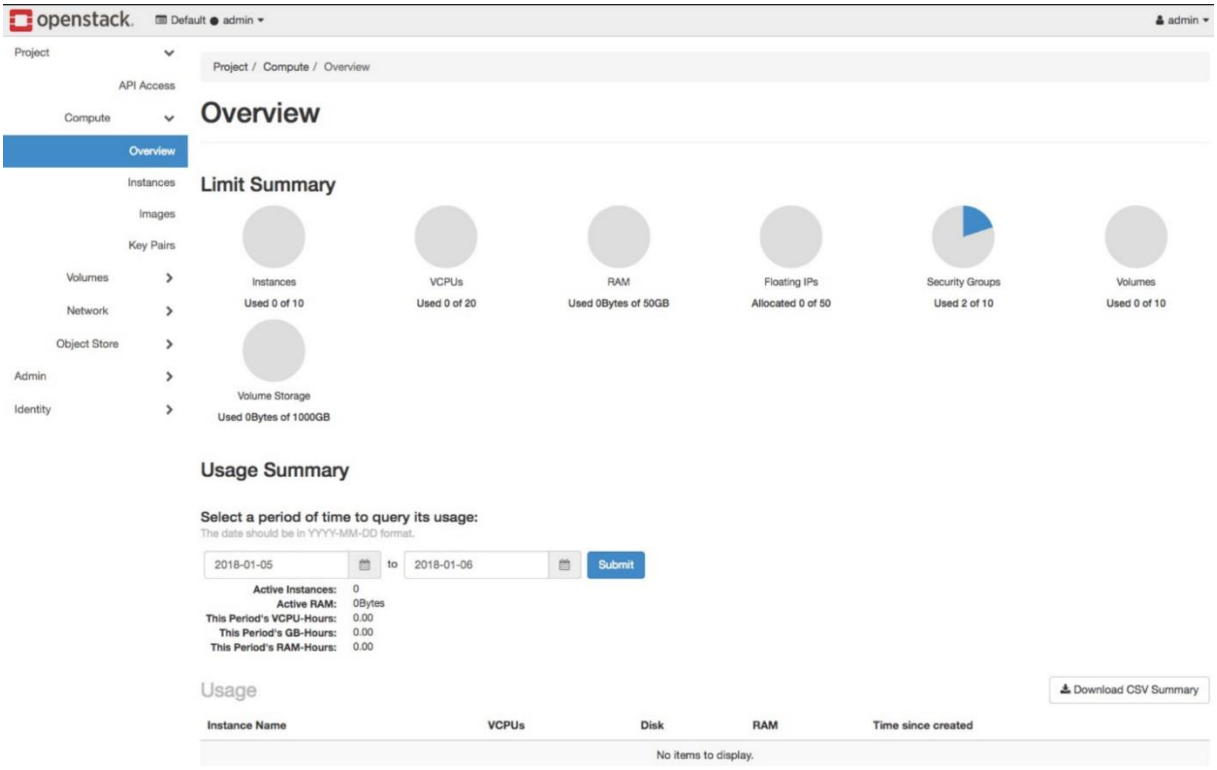

Figure 14 OpenStack Dashboard (Horizon)

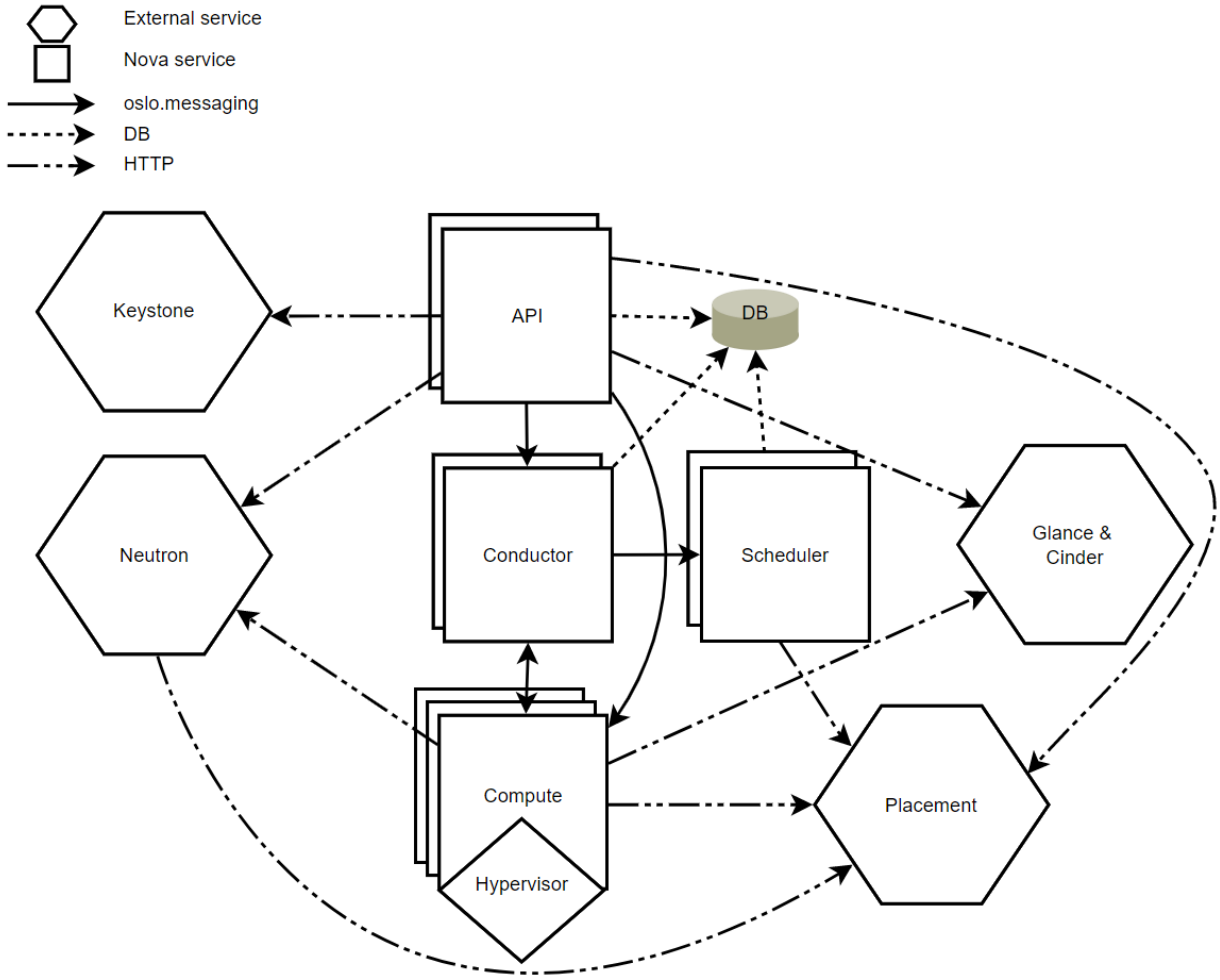

Figure 15 Nova Function

3.15 Infrastructure as a Service Selection

For the needs of the presented solution, OpenStack and OpenNebula were the best choices available. While, OpenNebula is flexible robust and powerful, OpenStack has the advantage at networking, computing power and storage[29]. Firstly, OpenNebula is a valid contender taking into consideration its ease of use, but OpenStack, with the use of Neutron and Nova services provides us the solution needed to realize network slicing, therefore we selected OpenStack as our Infrastructure-as-a-service (IaaS) framework. Table 1 presents some factors for comparison between OpenStack and OpenNebula. Table 2 shows the OpenStack Components and the OpenNebula equivalents

Table 1 Openstack VS OpenNebula

	OpenStack	OpenNebula
License	Apache License v2.0	Apache License v2.0
Cloud Types	Private & Public	Private, Public & Hybrid
OS	Most Linux Dist	Most Linux Dist
Programming Language	Python	Java & Ruby
Data Memory	Swift	Shared FS or SCP
Compatibility (public clouds)	Amazon EC2,S3	Amazon EC2
Commercial Model	Free	Free

Table 2 OpenStack - OpenNebula Components

OpenStack Component	OpenNebula equivalent
Compute (Nova)	Builtin
Object Storage (Swift)	No match
Image Service (Glance)	Builtin
Identity (Keystone)	Builtin
Dashboard (Horizon)	SunStone
Networking (Neutron)	Builtin
Block Storage (Cinder)	Builtin + Plugins
Telemetry (Ceilometer)	Builtin
Orchestration (Heat)	Flow
Database Service (Trove)	No match
Data Processing (Sahara)	No match
Bare Metal (Ironic)	No match
Queue Service (Zaqar)	No match
Key management (Barbican)	No match
DNS Services (Designate)	No match

3.16 OpenVAS

OpenVAS[5] is a Vulnerability Assessment Scanner developed by Greenbone Networks GmbH .It can detect security issues/loopholes and contains vulnerability tests for all kind of OSs, Servers and network devices. Initially it checks for open ports (port scan), depending on the ports found open It will start the assessment for several services, for known vulnerabilities and miss configurations using its large database of Network Vulnerability Tests (NVT). Figure 16 illustrates OpenVAS Dashboard

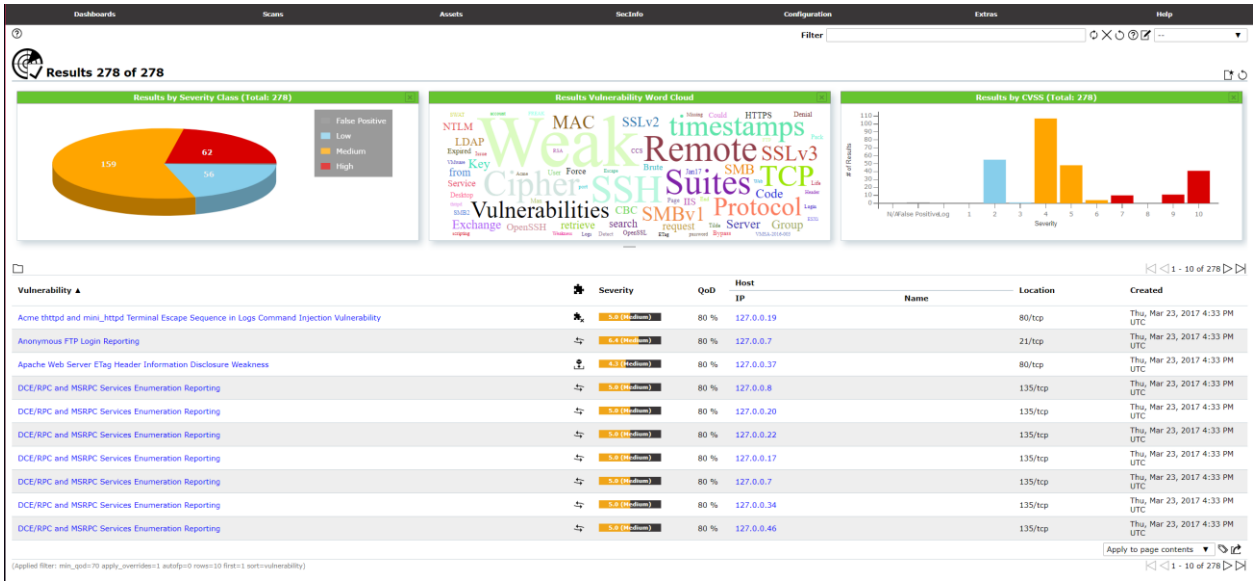

Figure 16 OpenVAS Dashboard (Results Section)

3.17 MongoDB

A database is a mass collection of data stored in a server. These data are accessible through terminals. There are 2 major categories that databases fall into, SQL Database management system (DBMS) and NoSQL. MongoDB is a NoSQL database (document-based). The difference between a NoSQL and an SQL database is how the data is processed. Document-oriented databases, have no need to map the data that are being loaded to the database in contradiction to SQL. For the needs of the thesis we selected MongoDB[30].

3.18 OpenFlow

OpenFlow[12] is the communication protocol used by the majority of SDN networks, between data and control plane. The OpenFlow protocol, is used on top of Transmission Control Protocol [31](TCP) and can work as well with the use of Transport Layer Security[32] (TLS) protocol. OpenFlow, provides us with means to remotely administrate our network by adding, removing or even modifying flow rules. Figure 17 illustrates OpenFlow Architecture

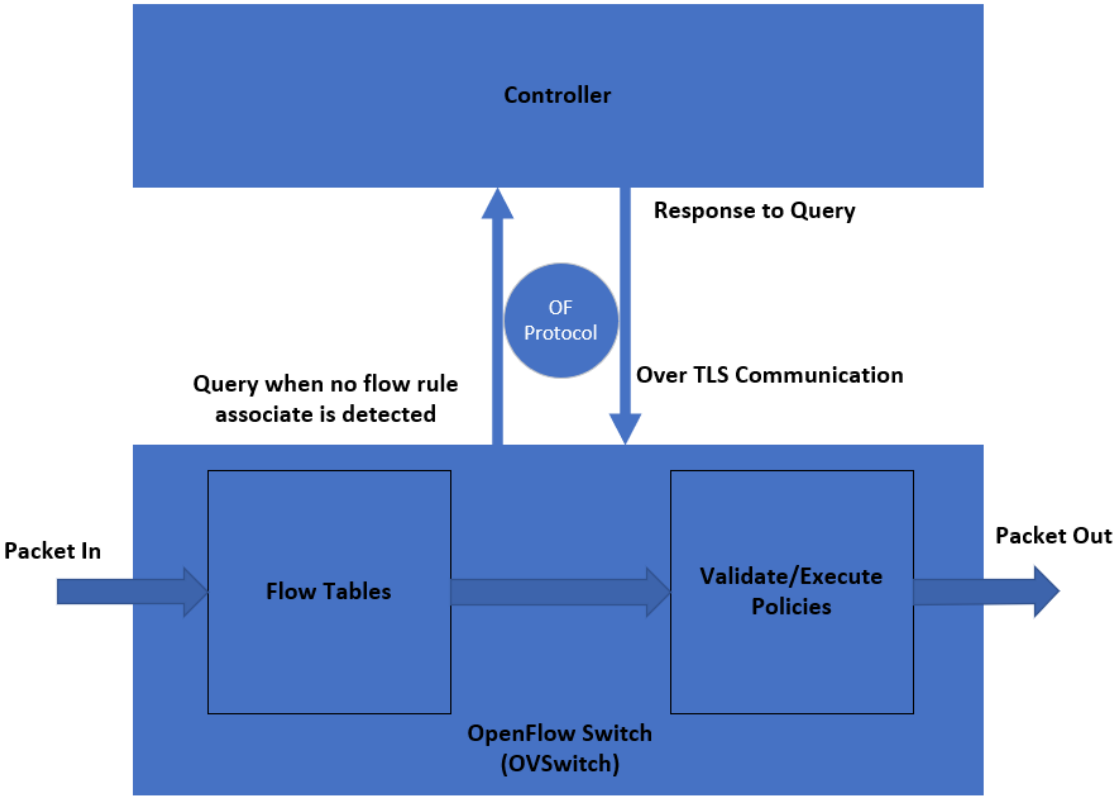

Figure 17 OpenFlow Architecture

4. Implementation

4.1 System Architecture

The proposed framework is split into two abstract layers, the private cloud and the edge. Figure 18 illustrates the High-Level Architecture of our work.

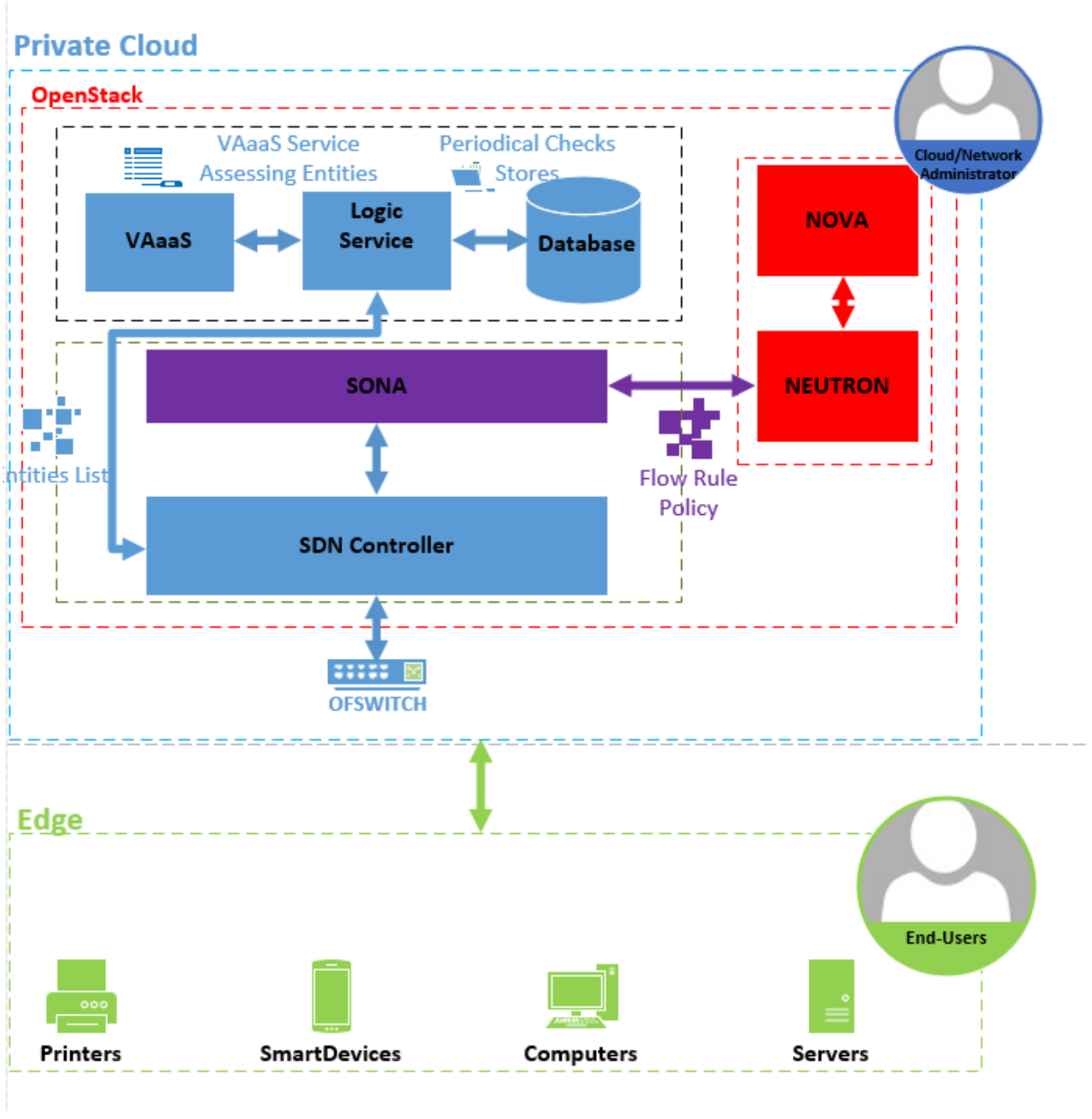

Figure 18 High-Level Architecture Diagram

4.1.1 Private Cloud

The private cloud is orchestrated by OpenStack, and the network is managed by an SDN controller (ONOS). The two entities communicate through an ONOS third-party component (SONA). Our framework operates on top of the SDN controller and comprises three components/services that constitute the overall proposed functionality. Namely, the respective components are the *logic service*, the *VAaaS service* and the persistence *database*. The subsections below present each deployed component and its main functionalities.

4.1.2 OpenStack

Openstack contains Neutron and Nova. Neutron's ML2 mechanism driver and L3 plugin backend expose REST APIs that networking-onos calls. OpenStack provides us with a way to virtually separate our network into 4 slices with the assistance of Neutron. The Neutron component talks directly to the SDN controller via the SONA component that's on top of the SDN controller.

4.1.3 Logic Service

The logic service continuously retrieves the list of connected network entities through the ONOS northbound RESTful API. Whenever a new network entity is discovered, the logic service acquires its information (IP address, MAC address e.tc.) and stores it into the database. Consequently, it checks every entity in the database, to find whether they have been assessed or not. The entities that have not been assessed, are assigned to the assessment network slice (restricted connectivity). Afterwards, the logic service sends the entities' information (IP, MAC) to the VAaaS service. The moment the VAaaS receives the IPs list, the assessment process for every IP in the list begins. The outcome of the assessment that the VAaaS produces, is a score value, based on the Common Vulnerability Scoring System (CVSS). Depending on the reported score, the logic service assigns the assessed entity to one of the four flavors of the predefined layer 3 network slices. Each flavor enforces different connection policies. Namely, the first slice, restricts all connectivity, the second only allows WAN connectivity, the third allows all incoming and outgoing traffic towards all network resources and the last, restricts connectivity and is used as a landing network for newly introduced entities, until they are assessed. The network slicing and the assignment of entities to the appropriate network slice, is performed by the SONA controller component and the Neutron OpenStack service. In more detail, the network slices have been initially created by the administrators, through OpenStack and the Neutron service. The SONA component, as instructed, installs the appropriate flow rules, so that target network entities only interact with the appropriate network slice.

4.1.4 VAaaS

The VAaaS service initiates its assessment process, the moment it receives the list of IP's to be assessed. It utilizes online Vulnerability Assessment Patterns repositories (NVTs) that store, maintain and daily update thousands of new and well-known vulnerability detection schemes. The produced outcomes (CVSS scores) of the assessed network entities, are propagated to the logic service, which with its turn stores those results in the database, and instructs appropriate actions (network assignments according to score). In order to dynamically communicate with the VAaaS

service we created an API for the OpenVAS that our logic service use for automation of the service the code can be found at Appendix

4.1.5 Database

The database stores all the information of every entity in the infrastructure. It only interacts with the logic service, which periodically pushes new entities to store and check the tables of the database for entities that have not been assessed yet.

4.1.6 The Edge:

Every entity which is deployed in this layer, can potentially bear vulnerabilities. Either the entity itself could be susceptible to attacks, or the end-user that has no experience and knowledge on cybersecurity could pose a threat to the other entities of the network as a whole. On the grounds that the SDN Controller has a full view of the underlying network topology, any unassessed network entity will be assessed for vulnerabilities. The connectivity of every entity until it gets assessed, is restricted.

4.2 Use Case

In this section, we will present a general use case of the proposed framework. Figure 19 presents the pseudo code that describes the sequence of actions that take place during the logic service lifecycle in detail.

Infrastructures that have free available network connectivity such as healthcare institutions, Municipality structures and generally public and private networks where untrusted devices connect and operate, are in need of a dynamic vulnerability assessment service that fast tracks the assessment (minimum wait time to have access), which creates no conflict over the policy rule since no one else will have the privileges to modify flow rules than SONA (application of ONOS SDN Controller) , provide real time monitoring through the capabilities of ONOS and lastly make administrative work easier through the capabilities of the SDN. Our Use-Case will be explained thoroughly below.

The SDN controller by nature is aware of any new entity, that connects to the network (OpenFlow messages instantiated by the OFSwitch that are sent to the controller whenever the entities interact with the network (DHCP Requests, API Requests, Applications)). The logic service, periodically initiates a script that acquires the entity list of the connected devices, through our controller (Northbound API – GET request). The logic service stores each entities data in the persistence database (ID, MAC address, IP address, Device Type, Port, Protocol, Assessment, Score, Slice). When the logic service detects an entity that has not been assessed, it initiates the vulnerability assessment process with OpenVAS. Prior to the assessment and while the assessment is on-going, the network entity will be assigned to a slice of the network that restricts any kind of communication. The moment the OpenVAS agent produces the score (CVSS standardized score) the logic service informs the SDN Controller about the slice the entity should be placed at, based on the score produced. Lastly the SDN Controller will inform SONA about the entity and the slice

it needs to be on. Sona then replies with flow rules suitable for the entity and the SDN controller applies them on the corresponding OFSwitch.

The OpenVAS agent produces the score based on four categories: i) None (Vulnerability Rating 0.0), ii) Low(Vulnerability Rating 0.1-3.9), iii)Medium(Vulnerability Rating 4.0-6.9), iv) High(Vulnerability Rating 7.0-10.0). According to the score reported, if a network entity is rated as “None” then the flow rules installed for that specific entity from SONA will allow full access to that entity(LAN-WAN).If the entity is rated as “Low” OR “Medium” the flow rules will allow it to interact only with the default gateway(therefore access to WAN only).Similarly if the entity is rated “High” then SONA will drop every packet originated from that entity The process described initiates whenever a new entity is detected. To conclude with the logic service, re-assess already existing entities in a set period of time

```
function policy(entitylist)(contains all the information needed to assess any entity)
entitiesToAssess = []
assessmentProgress = 0

for i of entitylist do
  if isAssessed(i) == false then
    entitiesToAssess.push(i)
  end if
end for
if entitiesToAssess.length() > 0 then
  AssessEntities(entitiesToAssess)
end if
while assessmentProgress < 100 do
  assessmentProgress = assessmentReports.progress
end while

assessmentReports = getAssessmentReports()
saveToDatabase(entitiesToAssess, assessmentReports)
end function
```

Figure 19 Pseudo Code

5. Evaluation

5.1 Aim

The proposed framework was evaluated in a controlled-conditions environment, through a simulation procedure wherein we measured the individual assessment duration for 100 and 200 devices. The aim of this evaluation was to benchmark the capabilities of the proposed framework by performing a large number of assessments.

5.2 Method

During the evaluation, we assessed 100 and 200 network entities respectively, deployed as virtual machines. The virtual machines hosted a deployed version of MySQL server, WordPress and Apache Tomcat. The selection for each VM was made randomly. All the VMs were deployed on a dedicated ESXi server (Dell EMC PowerEdge R940). The server's specifications are depicted in Table 3.

Table 3 Server Specifications

CPU	4x Intel Xeon Gold 6126 2.6G (12 cores & 24 threads)
RAM	128GB DDR4 RAM @2667 MT/s
Storage	5.6TB mixed storage

The simulation was performed in two iterations, for 100 and 200 VMs. For each iteration, different scan configurations were used. The main assumption for our evaluation, was that on a working deployment of our framework, 100 and 200 network-enabled entities join the network. The logic service detects that event and sends the list of newly introduced network entities to the VAaaS. The measurements start, the moment the VAaaS starts the first assessment for the first entity in its list.

5.3 Variables

5.3.1 Dependent

During the two-phase evaluation procedure (100 and 200 entities), we measured the assessment duration for each entity, the produced score (CVSS). Finally, we measured the overall duration of the evaluation.

5.3.2 Independent

As mentioned above, we performed the evaluation for 100 and 200 network-enabled entities. For the first iteration, the "Full and very deep" configuration was used. This is a deep and persistent scanning configuration our system can perform, but it allows for fast conclusion. For the latter

iteration, the “Full and fast” configuration was used. This is a moderately persistent and fast configuration.

5.4 Prediction

We know beforehand that the individual assessment duration for each entity will not be fast, since all entities are complex virtual machines, and the vulnerability assessment is a tedious task, taking into consideration that entities are assessed against thousands of penetration tests. But nevertheless, we presume that the assessment will be a matter of minutes to conclude. More specifically, since the scanning configurations vary in each iteration, we expect to get different results concerning the duration, as well as the produced score for each entity

5.5 Results

The graphs below depict the produced outcomes for the two-phase evaluation. Figure 27 presents the results for the assessment for 100 entities and Figure 28 presents the results for 200 entities

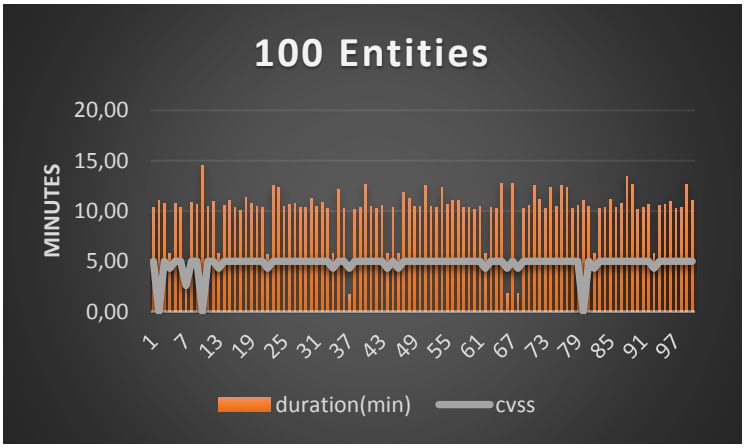

Figure 20 100 Entities Results

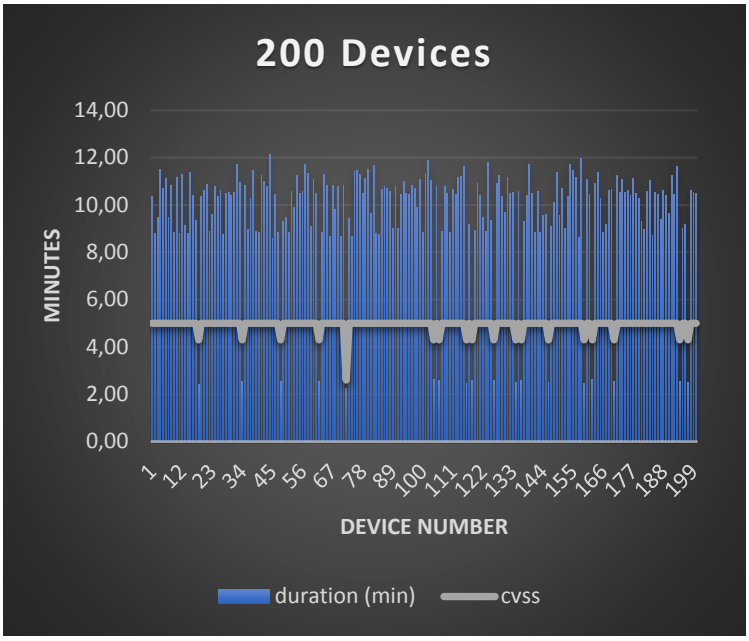

Figure 21 200 Entities Results

5.6 Discussion

By examining the presented results, we observe a rather linear behavior of our framework. We can observe some minor deviations (really low assessment duration) but we can also see the correlation with the produced CVSS score, which is also low. This means that the scanner found very few vulnerabilities on the assessed entity, thus the assessment concluded earlier. This is the case for both iterations since we used the same VMs, although the occurrence ratio is different, since the selection of VMs was made randomly.

The average assessment duration, was 13.77 and 9.59 minutes for 100 and 200 entities respectively. From a shallow point of view, these are rather contradictory results, as one would expect the duration would take longer for the assessment of 200 network entities. On a deeper observation, these results not only indicate the robustness of the evaluated framework, but also demonstrate the observable difference between the two different scanning configurations. This is obvious by observing the variation in the produced score in the first iteration, and the more static nature of the produced score in the second.

The results more or less agreed with our initial predictions, nevertheless we did not expect the duration to exceed the ten-minute barrier. To summarize, the measured results indicate the overall robustness and stability of the presented framework, by demonstrating linear behavior in both scenarios. Based on Nikoloudakis et al. [4], where they presented an average assessment time of approximately 38 minutes, these results, demonstrate a remarkable improvement. The performance difference is due to the different system architecture (cloud deployment in comparison to edge deployment) and the scanner's API redesign.

5.7 Evaluation

The presented evaluation presented the performance and behavior of the presented framework. The results were definitive of the enhanced performance and stability of our framework, compared to other research initiatives. Nevertheless, the combination of the independent variables for the experiment, could be more elaborate. From our point of view, the results would be more accurate, if we performed a two-phase iteration evaluation, but applying the same scanning configuration both for 100 as well as for 200 entities. Thus, we would have a series of measurements for 100 and 200 entities, with the "*Full and very deep" and "Full and fast" scanning configuration respectively.*

6. Conclusion

In this thesis we presented a pure-SDN automated vulnerability assessment framework that monitors the underlying network for existing and newly introduced network-enabled entities (devices, services, VMs, etc.) and performs assessments against known vulnerabilities. It produces a score based on the CVSS V3.0 standard and depending on the severity of the assessment result of each entity, it assigns it to a specific connectivity-appropriate network slice. We evaluated the framework through a series of measurements and concluded that compared to other research initiatives, it performed more than 70% better. Nonetheless there is still more room for improvement. As a future goal we firstly plan to further redesign the framework to be even more lightweight, so that we could achieve an even better performance, and finally we plan to thoroughly benchmark the framework by performing exhaustive assessments with all available scanning configurations, to gain a complete overview of the scanner's, and by extension the whole framework's capabilities.

7. References

[1] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras, "DELTA: A Security Assessment Framework for Software-Defined Networks," no. March, 2017.

[2] F. Loi, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman, "Systematically Evaluating Security and Privacy for Consumer IoT Devices," no. I, pp. 1–6, 2017.

[3] E. T., R. Y., and S. D., "Barrier Free Internet Access: Evaluating the Cyber Security Risk Posed by the Adoption of Bring Your Own Devices to e-Learning Network Infrastructure," *Int. J. Comput. Appl.*, vol. 176, no. 3, pp. 53–62, 2017.

[4] M. Miettinen *et al.*, "IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT," *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 2511–2514, 2017.

[5] "OpenVAS." [Online]. Available: http://www.openvas.org.

[6] M. Ficco, M. Choraś, and R. Kozik, "Simulation platform for cyber-security and vulnerability analysis of critical infrastructures," *J. Comput. Sci.*, vol. 22, pp. 179–186, 2017.

[7] B. Ali and A. I. Awad, "Cyber and physical security vulnerability assessment for IoT-based smart homes," *Sensors (Switzerland)*, vol. 18, no. 3, pp. 1–18, 2018.

[8] S. Ziegler, A. Skarmeta, J. Bernal, E. E. Kim, and S. Bianchi, "ANASTACIA: Advanced networked agents for security and trust assessment in CPS IoT architectures," *GIoTS 2017 - Glob. Internet Things Summit, Proc.*, 2017.

[9] Y. Nikoloudakis, E. Pallis, G. Mastorakis, C. X. Mavromoustakis, C. Skianis, and E. K. Markakis, "Vulnerability assessment as a service for fog-centric ICT ecosystems: A healthcare use case," *Peer-to-Peer Netw. Appl.*, 2019.

[10] "Cloud Computing." [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing.

[11] "SDN." [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_networking.

[12] "OpenFlow." [Online]. Available: https://en.wikipedia.org/wiki/OpenFlow.

[13] "ODL." [Online]. Available: https://www.opendaylight.org.

[14] "ONOS." [Online]. Available: https://onosproject.org.

[15] "Project Callico." [Online]. Available: https://github.com/projectcalico/calico.

[16] "POX." [Online]. Available: https://github.com/noxrepo/.

[17] "Floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[18] "Ryu." [Online]. Available: https://github.com/osrg/ryu/wiki.

[19] "SONA." [Online]. Available: https://wiki.onosproject.org/display/ONOS/SONA+Architecture.

[20] "Nimbus." [Online]. Available: https://www.nimbusframework.com.

[21] "Eucalyptus." [Online]. Available: https://en.wikipedia.org/wiki/Eucalyptus_(software).

[22] “OpenStack.” [Online]. Available: https://www.openstack.org.

[23] “Nebula.” [Online]. Available: https://opennebula.org.

[24] “XSEDE.” [Online]. Available: http://www.xsede.org/ecosystem/software.

[25] “KVM.” [Online]. Available: https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine.

[26] “xen.” [Online]. Available: https://xenproject.org.

[27] “VMWare.” [Online]. Available: https://www.vmware.com.

[28] “VM.” [Online]. Available: https://en.wikipedia.org/wiki/Virtual_machine.

[29] “Stack Vs Nebula.” [Online]. Available: https://stackshare.io/stackups/opennebula-vs-openstack.

[30] “Mongo.” [Online]. Available: https://www.mongodb.com.

[31] “TCP.” [Online]. Available: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[32] “TLS.” [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security.

8. Appendix

8.1 OpenVAS API

```
from pyvas import Client
from pyvas.exceptions import ElementExists
from flask import Response
from uuid import uuid4
from configurations.credentials import USERNAME,PASSWORD,HOST

PORT=9390
# omp --port=9390 --host=localhost --username=admin --password=UUID-PASSWD -G -i
import json

'''
Filter a list based on some predicate
'''
def filter_list(list_obj,predicate):
    items = []
    for item in list_obj:
        if predicate(item):
            items.append(item)
    return items

# Tasks endpoint
def create_new_task(name,hosts,config_uuid=None,comment=None):
    def name_predicate(item):
        config_name = "Full and very deep ultimate"
        return item.get('name') == config_name
```

```
    if  not name or not hosts:
        message = {'error':'No name or hosts were provided'}
        return Response(json.dumps(message),status = 400,mimetype='application/json')
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        result = {}
        status = 200
        try:
            # Investigate how to choose a different config
            #conf = cli.list_configs().data[0]
            conf = None
            if config_uuid is not None:
                conf = cli.get_config(config_uuid).data
            else:
                list_of_configs = cli.list_configs().data
                filtered_configs = filter_list(list_of_configs,name_predicate)
                if len(filtered_configs) == 1:
                    print('Found filtered configs')
                    print(filtered_configs)
                    conf = filtered_configs.pop()
                else:
                    print('Not one but....')
                    print(list_of_configs)
                    conf = list_of_configs[0]
                print('information for current task config')
                print(conf)
            target = cli.create_target("Task Name {}.Intermediate scan of
{}".format(name,hosts),hosts=hosts).data
            config = cli.create_config(name,copy_uuid=conf.get('@id')).data
            print('Created configuration {}'.format(config))
```

```
            task = cli.create_task(name,config_uuid=config.get('@id'),comment=comment,target_uuid=target.get('@id')).data
            result  = json.dumps(task)
        except ElementExists as e:
            status = 500
            result = {"error":"Task with same name/config exists"}
        except Exception as e:
            status = 500
            result = {'error':e.message}
        finally:
            return _json_response(result,status=status)



def create_multiple_tasks(addresses):
    def predicate_function(item):
        config_name = 'Full and very deep ultimate'
        return item.get('name') == config_name
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        print(cli.list_configs().data)


        configs = cli.list_configs().data
        conf = [config for config in configs if predicate_function(config)][0]

print("================================================================================================================================")
        print('Create multiple tasks current config {}'.format(conf))

print("================================================================================================================================")
```

```
        tasks = []
        for ip in addresses:
            name = "Automated task for {}".format(ip)
            target = cli.create_target("Task Name {}.Intermediate scan of {}".format(name,ip),hosts=ip).data
            task = cli.create_task(name,config_uuid=conf.get('@id'),comment=None,target_uuid=target.get('@id')).data
            tasks.append(task)
        return _json_response({'tasks':tasks})


def get_tasks(type=None,projection=None):
    '''
    Get all tasks. type is used to determine whether we want all tasks
    or we only want finished/pending tasks
    '''
    output = None
    tasks = []
    #Investigate if this code throws an error
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        tasks = cli.list_tasks()
        output = tasks.data if tasks else []
    if not tasks.ok:
        output['status_code']=tasks['status_code']
    if type is not None:
        status = 'Done' if type == 'finished' else 'New'
        output = [t for t in output if t.get('status') == status]
    if projection_exists(projection):
        output = [_projection(x,projection) for x in output]
```

```
    return _json_response(output)


def get_task(uuid,projection=None):
    output = {}
    task = {}
    status = 200
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        try:
            task = cli.get_task(uuid)
            output = task.data
        except TypeError as e:
            output = {'error':'Make sure that you entered a correct task uuid'}
            projection = None
            status = 400
    if projection_exists(projection):
        output = _projection(task.data,projection)
    if 'ok' in task and not task.ok:
        output['status_code'] = task['status_code']
    return _json_response(output,status=status)


def start_task(uuid):
    output = {}
    status = 200
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        try:
            res = cli.start_task(uuid)
            output = res.data
        except TypeError:
            status = 400
```

```
            output = {'error':'Make sure that you entered a correct task uuid'}
        finally:
            return _json_response(output,status=status)


def start_multiple_tasks(ids):
    for id in ids:
        start_task(id)
    return _json_response({})


def stop_task(uuid):
    output = {}
    status = 200
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        try:
            res = cli.stop_task(uuid)
            output = res.data
        except TypeError:
            status = 400
            output = {'error':'Make sure that you entered a correct task uuid'}
        finally:
            return _json_response(output,status=status)


def delete_task(uuid):
    output = {}
    status = 200
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        try:
            task = cli.get_task(uuid).data
            target = task['target']
```

```
            res = cli.delete_task(uuid)
            # Must delete target **AFTER** deleting task
            cli.delete_target(target['@id'])
            output = res.data
        except TypeError:
            status = 400
            output = {'error':'Make sure that you entered a correct task uuid'}
    return _json_response(output,status=status)

#  Targets endpoint
def get_targets(projection=None):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        targets = cli.list_targets().data
        if projection_exists(projection):
            targets = [_projection(target,projection) for target in targets]
        return _json_response(targets)

def get_target(uuid,projection=None):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        status = 200
        try:
            target = cli.get_target(uuid).data
        except TypeError as e:
            status = 400
            target = {'error':'Make sure that you entered a correct report uuid'}
            projection = None
        if projection_exists(projection):
            target = _projection(target,projection)
        return _json_response(target,status=status)
```

```
def create_target(name,hosts):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        try:
            data = cli.create_target(name,hosts).data
        except ElementExists:
            data = {"error":"target exists"}
        return _json_response(data)


def delete_target(uuid):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        data = {}
        status = 200
        try:
            data = cli.delete_target(uuid).data
        except :
            data = {"error":"Make sure that you entered a correct target uuid"}
            status = 400
        finally:
            return _json_response(data,status)
# Configs endpoint


def get_configs(projection=None):
    print('requesting configs')
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        configs = cli.list_configs().data


        if projection_exists(projection):
            configs = [_projection(conf,projection) for conf in configs]
```

```
        return _json_response(configs)


def get_config(id):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        config = None
        status = 200
        try:
            config = cli.get_config(id)
        except:
            status = 400
            config = {"error":"Make sure you provided a valid uuid"}
        finally:
            return _json_response(config,status=status)



def create_config(name,copy_uuid=None):
    print(len(copy_uuid))
    print(name,copy_uuid)
    if not copy_uuid or not len(copy_uuid) is 0 or copy_uuid is None:
        return _json_response({"error":"Provide a valid uuid"},400)
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        config = {}
        status = 200
        try:
            config = cli.create_config(name,copy_uuid=copy_uuid)
            if config.ok:
                config = config.data
        except:
            config = {"error":"Provide a valid uuid"}
```

```
            status = 400
        return _json_response(config,status)


def delete_config(uuid):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        data = cli.delete_config(uuid).data
        return _json_response(data)
# Tasks endpoint


def get_pending_tasks(projection=None):
    tasks = get_tasks('pending',projection=projection)
    return tasks


def get_finished_tasks(projection=None):
    tasks = get_tasks('finished',projection=projection)
    return tasks


def get_task_progress(uuid):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        data = {}
        projection=['progress']
        status = 200
        # prevent Unbound local error
        task = None
        try:
            task = cli.get_task(uuid).data
            if not task:
                raise TypeError
            # Do not use 'is' for string comparison
```

```
            # see here: https://stackoverflow.com/a/1504742/7180331
            if  task.get('progress') == "-1":
                data['progress'] = 100
            elif task['progress']=="1":
                data['progress'] = 1
            else:
                data['progress'] = task.get('progress')
                data['progress'] = data['progress']['#text']
        except TypeError as err:
            status = 400
            #@FIX typo
            print('Error while trying to get task progress')
            data = {'error':'Make sure that you entered a correct task uuid'}
        except ConnectionResetError as con:
            status = 500
            data = {'error':con}
        return _json_response(data,status=status)


# Reports endpoint


def get_reports(projection=None):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        reports = cli.list_reports().data
        if projection_exists(projection):
            reports = [_projection(rep,projection) for rep in reports]
        return _json_response(reports)


def get_report(uuid,projection=None):
    report = {}
```

```
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        status = 200
        try:
            report  = cli.get_report(uuid).data
            if projection_exists(projection):
                report = _projection(report,projection)
        except TypeError:
            status = 400
            report = {'error':'Make sure that you entered a correct report uuid'}
        finally:
            return _json_response(report,status=status)


def delete_report(uuid):
    with Client(host=HOST,username=USERNAME,password=PASSWORD,port=PORT) as cli:
        status = 200
        result = {}
        try:
            result = cli.delete_report(uuid).data
        except TypeError:
            result = {'error':'Make sure that you entered a correct report uuid'}
            status = 400
        except Exception as e:
            status = 400
            if "Failed to find report" in str(e):
                result = {'error':'Make sure that you entered a correct report uuid'}
            else:
                result = {'error':str(e)}
        finally:
            return _json_response(result,status=status)
```

```
def get_report_results(uuid):
    projection = "report.results.result"
    with Client(host=HOST,password=PASSWORD,port=PORT,username=USERNAME) as cli:
        status = 200
        try:
            report = cli.get_report(uuid).data
            data = {
                #
                "results":report['report']['results']['result']
            }
        except TypeError as e:
            data = {'error':'Make sure that you entered a correct report uuid'}
            # Bad request
            status = 400
        return _json_response(data,status)

# Utilities

def projection_exists(projection=None):
    '''
    Determines whether a projection string is
    is an empty projection
    '''
    return  projection is not None and len(projection) is not 0

def deep_extract(data,key):
    '''
        Wander what this does?
```

```
        It shall remain a mystery for the eternity
    '''
    base_key = None
    keys = key.split(".")
    # Take the first key as the base property name
    # e.g. for the following keys ['task','owner','name'] base would be the task
    base_key = keys[0]
    del keys[0]
    # reverse the keys to work easier with
    # ['name','owner']
    keys.reverse()
    # Let the fun begin
    data = data.get(base_key)
    while len(keys) > 0:
        cur_property_name = keys.pop()
        # If we have passed an invalid property name
        # data will become None or an empty string
        if isinstance(data,dict):
            data = data.get(cur_property_name)
    # Unicorns have finished their job
    # Time to continue our non unicorn-related work
    # Maybe I should send an empty string instead of {}
    return data if not data is None else {}

def _projection(data,keys):
    # If the projection string is an empty string
    # _projection would return as an empty object
    # but no more
    if not projection_exists(keys):
```

```
        return data
    '''
        Extract only the projected keys from a data object
        This helps save bandwith.Imagine an object having 20 maybe 30 properties.
        This would be an overkill to transfer.That's why with a projection
        you can specify what you want
    '''
    projected = {}
    for key in keys:
        # No need to check if key exists in data
        projected[key] = deep_extract(data,key)
    return projected


def parse_projection(projection):
    if projection is None:
        return []
    # Maybe projection is alreay a list
    projection_keys = projection if isinstance(projection,list) else projection.split(',')

    # Let's handle the following scenario
    # A user does a get request and then as a url parameter
    # they pass an array of keys like this
    # ?projection=[a,b,c] instead of projection="a,b,c"
    # So as a key we also get the opening/closing brackets
    if projection_keys[0] is '[':
        del projection_keys[0]
    last_key_index = len(projection_keys) -1
    #Bring the last element to the front
    if projection_keys[last_key_index] is ']':
```

```
        del projection_keys[last_key_index]
    return projection_keys



def _json_response(data,status=200):
    return Response(json.dumps(data),status=status,mimetype='application/json')



def clean_db(username,password):
    data = {}
    # There are specific targets and specific configs that cannot be deleted
    predefined_configs = ['empty','Full and fast','Full and very deep','Host Discovery','Network Diskovery']
    predefined_targets = ['Localhost']
    if(username != USERNAME or password != password):
        return _json_response({'error':'invalid credentials'},status=401)
    with Client(host=HOST,password=PASSWORD,port=PORT,username=USERNAME) as cli:
        print('Cleaning up')
        status = 200
        try:
            tasks = cli.list_tasks().data
            configs = cli.list_configs().data
            targets = cli.list_targets().data
            reports = cli.list_reports().data
            message_template = "Found {} tasks {} targets {} configs and {} reports to delete"
            print(message_template.format(len(tasks),len(targets),len(configs),len(reports)))
            for task in tasks:
                id = task['@id']
                cli.delete_task(id)
```

```
        for target in targets:
            in_use = target['in_use'] == "1"
            owner = target['owner']['name']
            id = target['@id']
            # only targets not in use and created by admin can be deleted
            # if not in_use and owner == "admin":
            #     cli.delete_target(id)
            try:
                cli.delete_target(id)
            except:
                pass
        for config in configs:
            in_use = config['in_use'] == "1"
            owner = config['owner']['name']
            id = config['@id']
            try:
                cli.delete_config(id)
            except:
                pass
        for report in reports:
            id = report['@id']
            try:
                cli.delete_report(id)
            except:
                pass
    except TypeError as e:
        data = {'error':'Make sure that you entered a correct report uuid'}
        # Bad request
        status = 400
```

```
    finally:
        return _json_response(data,status)
```