الجمـــهوريـــة الجـــزائريـــة الديـمقراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمـي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـلـيـــــة الـعـلـــــوم
FACULTE DES SCIENCES
قـــسـم الاعـــلام الآلـــي
DEPARTEMENT D'INFORMATIQUE

Mémoire de MASTER

Domaine : Mathématiques et Informatique
Filière : Informatiques
Option : Réseaux, Systèmes et Applications Réparties

Par:
Hamza Ahmedi Tarek Bouzid

THEME

Software Defined Networking

Soutenu publiquement le 01-06-2017 devant le jury composé de:

Mlle FATIMA BOUSBAA	*M.A.(A)*	*Président*
Mr NOUREDDINE CHAIB	*M.A.(A)*	*Examinateur*
Mr LAHCEN BENSAAD	*M.C.(B)*	*Encadreur*

Année Universitaire 2016/2017

Acknowledgments

First of all, we are grateful to God for blessing us with the gift of knowledge, the wellbeing, the good health and the will necessary to complete our studies, and this thesis.

We would like to express our sincere thanks to our thesis supervisor Dr. Mohamed Lahcen Bensaad. His office was always open whenever we ran into an issue or had a question about our research. He consistently allowed this paper to be our own work, but steered us in the right direction whenever he thought we needed it.

We are so grateful to our parents, who have provided us with moral and emotional support, attention and love in our lives. We are also grateful to our other family members and friends who have supported us along the way.

And finally, last but by no means least, we want to take this special opportunity to thank all of the Department of computer Science's members for their help and support through the course of the 5 years we spent in this university.

This accomplishment would not have been possible without all of you. Thank you so much.

Ahmedi Hamza – Bouzid Tarek

ملخص

أصبحت الإنترنت جزءا كبيرا من الحياة اليومية للجميع، من الأفراد، لرجال الأعمال والمطورين والباحثين، وغيرهم. ومن ناحية أخرى، أصبحت الشبكات التقليدية تشكل عائق وليست قادرة على تلبية جميع احتياجات المستخدمين المختلفة بعد الآن.

الشبكات المعرفة برمجيا (SDN) هو مفهوم شبكي جديد مثير للاهتمام، يسمح للشبكات بأن تكون ديناميكية للغاية ومفتوحة لجميع التطبيقات. ويركز مفهوم الشبكات المعرفة برمجيا أساسا على الفصل المادي لمستوى التحكم من مستوى اعادة التوجيه.

Open Network Operating System (ONOS) هو وحدة تحكم للشبكات المعرفة برمجيا ، و هو يوفر اتاحة عالية، قابلية للتطوير، و أداء عالي لكي يسهل إدارة شبكة SDN.

هذه المذكرة تقدم فوائد الشبكات المعرفة برمجيا بما في ذلك مقارنة مالية واقتصادية بين استخدامها واستخدام الشبكات التقليدية ، بعد إجراء الأمثلة الخاصة بنا وتحليل النتائج، وجدنا أن نشر شبكة معرفة برمجيا أقل تكلفة من الشبكة التقليدية بحوالي 60٪.

الهدف من هذه المذكرة هو تكوين مقدمة للابتكارات المستقبلية في مجال الشبكات المعرفة برمجيا والشبكات الافتراضية، فضلا عن إدخال مشروع ONOS وحالات استخدامه وقدراته.

كلمات مفتاحية: الشبكات المعرفة برمجيا (SDN) ، الشبكات التقليدية ، مستوى الشبكة، Open Network Operating System (ONOS) ، OpenFlow.

Résumé

Internet est devenu une grande partie de la vie quotidienne de tous, des individus, hommes d'affaires, développeurs, chercheurs et bien plus encore. D'autre part, les réseaux traditionnels sont devenus un obstacle et ne peuvent plus répondre à tous les besoins des utilisateurs.

Software Defined Networking (SDN) est sûrement un nouveau concept intéressant de réseau, qui permet au réseaux d'être très dynamique et ouvert à toutes les applications. SDN est principalement concentré sur la séparation physique des fonctions de contrôle et de transfert des données.

Open Networking Operating System (ONOS) est un contrôleur SDN qui offre l'évolutivité, haute disponibilité et performances pour gérer facilement un réseau SDN.

Ce mémoire présente les avantages du SDN, y compris une comparaison financière et économique entre l'utilisation d'un réseau SDN et un réseau traditionnel, après avoir fait nos propres exemples et analysé les résultats, nous avons constaté que le déploiement d'un réseau SDN est moins cher qu'un réseau traditionnel d'environ 60%.

Ce mémoire est destinée à être une introduction pour les futures innovations dans les domaines de la virtualisation réseau et du SDN, ainsi que l'introduction du projet ONOS, ses cas d'utilisation et ses capacités.

Mots clés : Software Defined Networking (SDN), réseaux traditionnels, plan de réseau, Open Network Operating System (ONOS), OpenFlow.

Abstract

Internet has become a big part of everybody's daily life, from individuals, to businessmen, developers, researchers, and much more. On the other hand, traditional networks are becoming a liability and are not able to satisfy all the different users' needs anymore.

Software Defined Networking (SDN) is surely an interesting new network concept, that allows networks to be highly dynamic and open for all applications. SDN is mainly focused on the physical separation of the control plane from the forwarding plane.

Open Networking Operating System (ONOS) is an SDN controller, that provides scalability, high availability and performance to easily manage an SDN network.

This thesis presents the benefits of SDN including a financial and economic comparison between using an SDN and a traditional network, after doing our own examples and analyzing the results, we found that deploying an SDN network is cheaper than a traditional network by around 60%.

This thesis is meant to be an introduction for future innovations in SDN and network virtualization fields, as well as introducing the ONOS project, its use cases and its capabilities.

Keywords : Software Defined Networking (SDN), traditional networks, network plane, Open Network Operating System (ONOS), OpenFlow.

Table of Contents

Table of Contents vi

List of Figures viii

List of Abbreviations ix

Introduction and problematic 1

Manuscript Outline 2

1 Software Defined Networking 3
- 1.1 Introduction . . . 3
- 1.2 From traditional to programmable networks . . . 3
  - 1.2.1 Traditional router planes . . . 3
  - 1.2.2 New approach . . . 4
- 1.3 What is SDN ? . . . 7
- 1.4 Why SDN ? . . . 7
- 1.5 SDN Architecture . . . 8
- 1.6 SDN Protocols . . . 11
  - 1.6.1 OpenFlow . . . 11
  - 1.6.2 Other Protocols . . . 14
- 1.7 Common SDN controllers . . . 16
- 1.8 Future of SDN . . . 18
- 1.9 Conclusion . . . 19

2 Open Network Operating System 20
- 2.1 Introduction . . . 20
- 2.2 Definition . . . 20
- 2.3 History . . . 21
  - 2.3.1 ON.LAB . . . 21
  - 2.3.2 History of ONOS . . . 22
- 2.4 Architecture overview of ONOS . . . 23
  - 2.4.1 Features . . . 23
  - 2.4.2 Tiers of ONOS . . . 24
- 2.5 Core technology of ONOS . . . 26
  - 2.5.1 Subsystems . . . 26

2.5.2 Intents . . . 28
2.5.3 Technologies and implementation of ONOS . . . 28
2.6 SDN-IP . . . 30
2.7 Conclusion . . . 31

3 Testing and discussion **32**
3.1 Introduction . . . 32
3.2 Mininet . . . 32
3.2.1 Working with mininet . . . 33
3.2.2 Custom topology . . . 34
3.2.3 Python API topology . . . 35
3.3 ONOS . . . 37
3.3.1 Setting up ONOS . . . 37
3.3.2 Basics . . . 37
3.3.3 ONOS with a topology . . . 40
3.3.4 Other features . . . 40
3.3.5 Distributed Clustering . . . 42
3.4 How to contribute and get involved . . . 45
3.5 Economical benefits of SDN . . . 49
3.5.1 Example . . . 49
3.5.2 Results . . . 50
3.5.3 Analysis . . . 51
3.6 Challenges and problems with SDN . . . 52
3.7 Discussion . . . 52
3.8 Conclusion . . . 54

General conclusion and future work **55**

References **56**

List of Figures

1.1 The default router planes . . . 4
1.2 Benefits of SDN and NFV . . . 6
1.3 Standard architecture of Software Defined Networking . . . 9
1.4 Standard architecture of an OpenFlow Switch . . . 12
1.5 OpenFlow Data Plane . . . 13
1.6 OpenFlow Switch Agent . . . 14
1.7 Basic layers of SDN . . . 15

2.1 ONOS Partners and Member growth chart (Jan-2015 - Oct-2016) . . . 23
2.2 Tiers of ONOS . . . 25
2.3 ONOS subsystems . . . 27
2.4 ONOS subsystem components relationship . . . 28
2.5 SDN-IP architecture . . . 31

3.1 MiniEdit's Graphical User Interface . . . 35
3.2 Representation of the file ~/mininet/custom/topo-2sw-2host.py . . . 36
3.3 ONOS instance start . . . 38
3.4 ONOS web UI . . . 39
3.5 A topology run in ONOS . . . 40
3.6 The navigation menu of ONOS . . . 41
3.7 The device view . . . 42
3.8 Output of mininet initializing the virtual network . . . 43
3.9 Network topology view with three node cluster . . . 44
3.10 Output of the command onos> roles . . . 45
3.11 High availability test with one instance offline . . . 46
3.12 Snippet of the output after the command $ onos-create-app . . . 47
3.13 Snippet of the file pom.xml . . . 48
3.14 Output of the command onos> apps -s -a . . . 49
3.15 Graph representing the cost of different network scenarios . . . 51

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
BGP	**B**order **G**ateway **P**rotocol
CapEx	**C**apital **Ex**penditure
CLI	**C**ommand **L**ine **I**nterface
NFV	**N**etwork **F**low **V**irtualization
NV	**N**etwork **V**irtualization
ONF	**O**pen **N**etworking **F**oundation
ONOS	**O**pen **N**etwork **O**perating **S**ystem
OpEx	**O**perating **Ex**pense
OSGi	**O**pen **S**ervice **G**ateway **I**nitiative
OSPF	**O**pen **S**hortest **P**ath **F**irst
OVS	**O**pen **vS**witch
QoS	**Q**uality **o**f **S**ervice
REST API	**RE**presentational **S**tate **T**ransfer API
SDN	**S**oftware **D**efined **N**etworking
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SSL	**S**ecure **S**ockets **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity
VoIP	**V**oice **O**ver **IP**

Introduction and problematic

In the early days of Internet, there was a limited number of provided services and connected users, which made traditional networks enough to fulfil the users' demands. But Internet has become an important part of everybody's life, from searching, chatting and using voice over IP (VoIP), to online shopping and e-commerce. These are just a few of the many services that have to coexist and adapt to the ever-changing needs of their costumers.

Network Changes

The ever changing nature of the network architecture is starting to become a liability because it is becoming larger and highly dynamic since multiple devices and users on the network may use it differently. They can even have different priorities such as reliability, quality of service and much more. It is becoming harder to satisfy all needs without sacrificing others.

Since computing trends are driving these ongoing network changes, any new solutions must support the existing functionalities already available in the current network technologies while improving on them and adding even more.

Some of the critical issues in today's computing that must be addressed :

- The change of the traffic models: Most applications manage a lot of distributed resources such as databases and servers through public and private cloud networks, which require extremely flexible access and routing.
- The rise of the cloud services: Using the network to access different applications and resources on demand.
- The use of big data: To use and manage today's big data needs massive parallel processing and a reliable network infrastructure.
- The level of complexity: Adding, removing or implementing devices need manual intervention and lot of maintenance time, which constitutes a risk for services' regularity and causes disruptions and extra network charges.
- The inability to scale: Due to the complexity that comes with such large networks, scalability can be very hard to achieve.
- Vendor independence: Lengthy vendor equipment product cycles, open interfaces and lack of standards, all limit the ability of network operators to tailor the network to their individual environments.

The solutions

Over the past two decades, we have seen tons of innovations in the tools we use to interact with networks, from applications and services we depend on to run our lives, to the computing and storage solutions that we rely on to hold all our big data. However, the underlying network that connects all of these things has remained virtually unchanged [1].

We have seen some ideas and concepts in these last few years, that try to provide solutions to this problem. But most of these ideas believed that increasing the bandwidth will suffice. Later on, more services came to light, these services had more critical requirements such as low latency, increasing the bandwidth was not a valid modification any more. A solution, that was not very based on bandwidth was Quality of Service (QOS), which was a great idea at the time that but it is still underutilized.

The number of connected people and devices is constantly evolving, and reaching the network's limits. This is why other approaches are needed now more than ever. The solutions we need must make radical changes to the network and allow it to be highly dynamic, highly available, scalable and programmable.

Thankfully, there are some solutions already available. Software Defined Networking (SDN), Network Functions Virtualization (NFV), and Network Virtualization (NV) are providing new ways to design, build and operate networks. But each of these approaches is trying to solve and target different parts of the network.

The objective of our thesis is to introduce SDN, its concept and architecture, and to use it, by controlling an SDN virtual network simulated by mininet and ONOS as its controller.

Manuscript Outline

The rest of this manuscript is structured in three chapters.

The first chapter will present one particular solution, which is Software Defined Networking (SDN), because it is the most interesting approach, since it gives many new concepts and proposes various innovative ideas to change the current networks for the better.

The second chapter will shed some light on some implementations of SDN, but will be fully dedicated to one specific implementation, which is Open Network Operating System (ONOS).

Then, the third chapter will provide some examples, simulations and tools needed to start working with ONOS. This chapter will also present some results and discussions regarding ONOS and SDN in general.

CHAPTER 1

Software Defined Networking

1.1 Introduction

Because Internet is so invaluable, it needs to be more accessible and easier to use. Networks, which are the skeleton of Internet, need to adapt to changes and this is where Software Defined Networking (SDN) can be exploited. SDN can radically reshape wide area networks, it has the potential to deliver the performance, scale, availability and core features valued by all Service Providers.

In this chapter we will try to explain in depth what SDNs are, how they are implemented, how they are used, what are their advantages and what do they offer compared to traditional network solutions.

1.2 From traditional to programmable networks

We have been introduced to the problem with older networks, but now we will see what causes the problem, and how the new solutions are planning to fix it.

1.2.1 Traditional router planes

In routers or layer-3 switches, we notice three distinct planes (Fig. 1.1 [2]), control and data are the two main planes, while the management plane is used for administration purposes.

1. **Management Plane** provides management, monitoring and configuration tools and services like CLI and SNMP.
2. **Control Plane** primarily concerned with being aware and updating the network topology and the routing table so it knows how to deal with incoming packets.
3. **Data Plane** sometimes referred to as the Forwarding Plane, which focuses mainly on forwarding the actual packets.

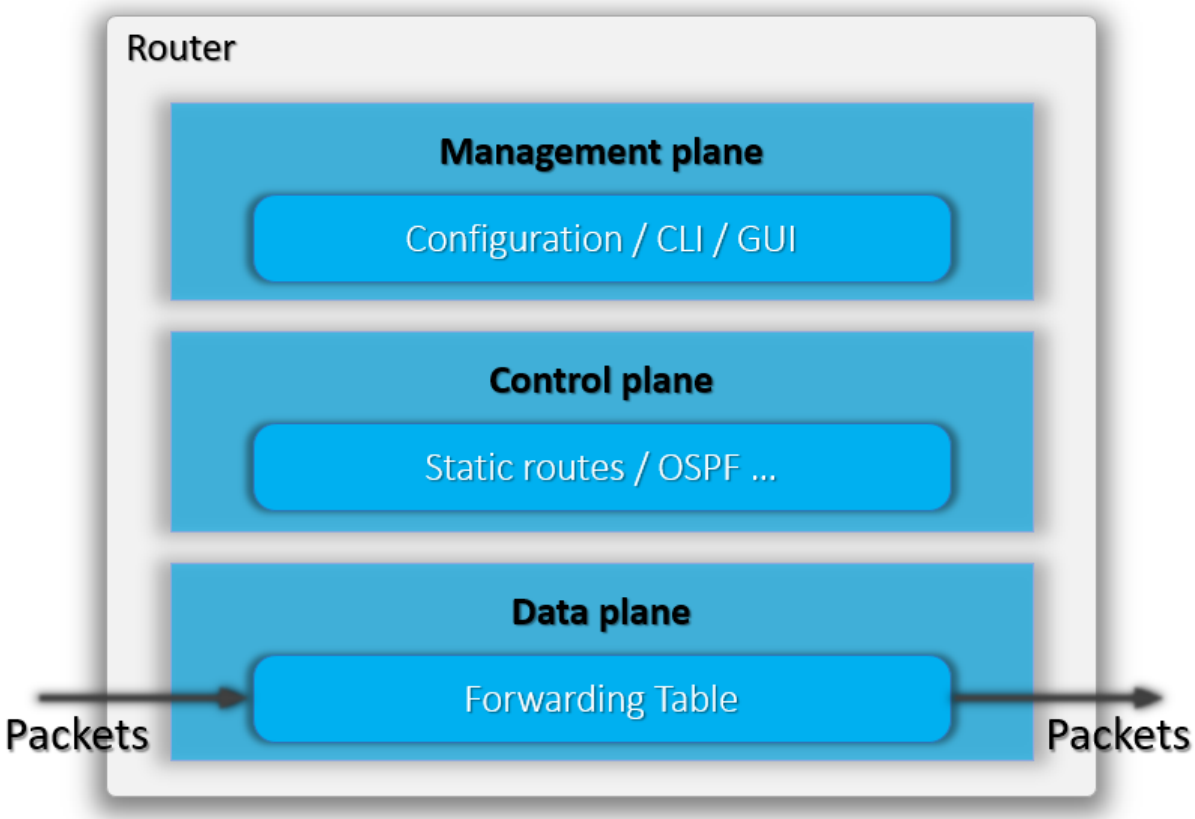

FIGURE 1.1: The default router planes

1.2.2 New approach

The traditional planes architecture that was shown earlier is the main problem preventing the network from reaching new high potentials. The problem is not with the planes themselves, but the fact that they are packed together in a single hardware makes it harder to update individual parts without changing the hardware.

To solve this problem, NV, NFV and SDN appeared. We will examine how they differ and how they guide us down the path of programmable networks, where software can finally be decoupled from the hardware, and no longer constrained by the box that delivers it.

- **SDN** Separates the network's control and data planes, and provides a centralized view of the distributed network for more efficient orchestration and automation of network services.

- **NFV** While SDN separates the control and forwarding planes to offer an easier network manipulation, NFV primarily focuses on optimizing the network services themselves.

  Proposed by the European Telecommunications Standards Institute in 2012, NFV decouples the network functions and services such as DNS, caching, etc., from proprietary and dedicated hardware appliances to generic servers so they can run in software to accelerate service innovation and provisioning, particularly within service provider environments.

- **NV** Using a sort of hypervisor[1] between the physical network and the applications, the network can be treated like virtual machines regardless of the actual hardware available, and just like virtual machines these networks can be created, saved, deleted, restored, etc.

  Thanks to this "Network Hypervisor", the abstraction of the physical network can allow support of multiple logical networks that would appear to applications as a real physical network, connecting physical and virtual machines alike.

All these three solutions are designed to address mobility, agility, and bring new ways to program the network. While SDN changes the actual physical network, NV and NFV add virtual tunnels and functions to the existing network.

Until recently, the conventional wisdom was SDN and NFV were separate topics and didn't need to be formally coordinated. That changed in March 2014 when the Open Networking Foundation (ONF) and the European Telecommunications Standards Institute Industry Specification Group for NFV (ETSI NFV ISG) announced the signing of a Memorandum of Understanding (MOU).

As part of that announcement, the ONF and ETSI stated that "Together the organizations will explore the application of SDN configuration and control protocols as the base for the network infrastructure supporting NFV, and conversely the possibilities that NFV opens for virtualizing the forwarding plane functions" [3].

Even though SDN and NFV are not the same, they're often complementary to one another. Therefore, actually combining the forces of the two approaches can be very beneficial as shown in Fig. 1.2 [4].

[1] A hypervisor or virtual machine monitor (VMM) is computer software, firmware, or hardware, that creates and runs virtual machines.

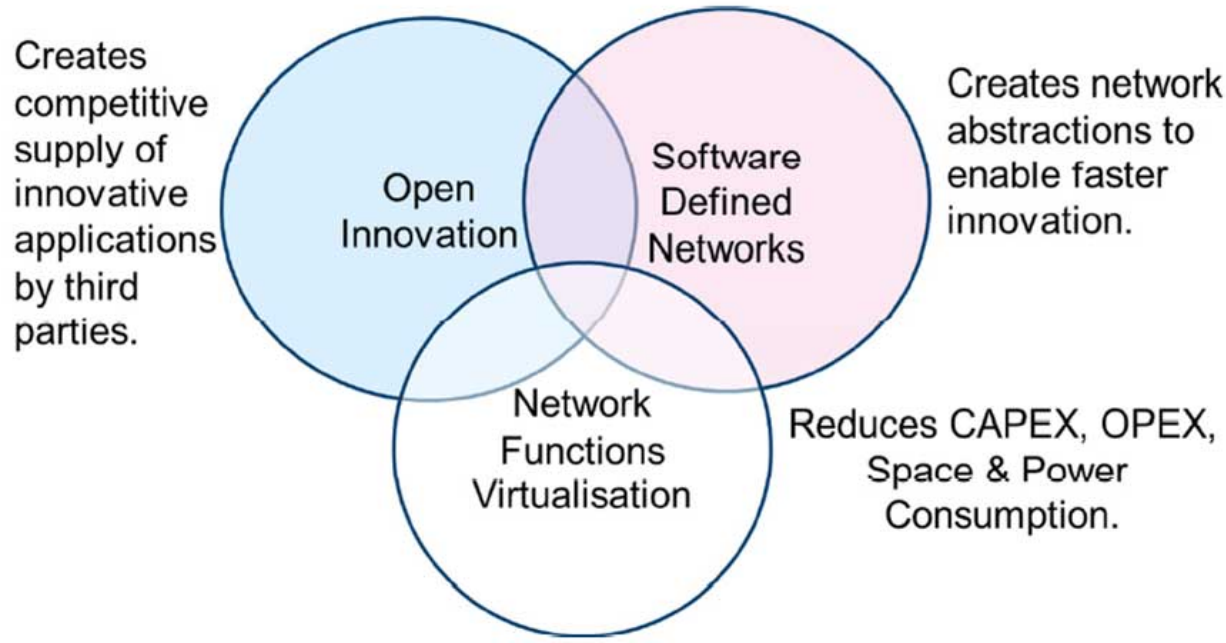

FIGURE 1.2: Benefits of SDN and NFV

SDN and NFV can become a key to building networks thanks to the features that they offer such as [1] :

- Enable innovation: allow organizations to create and test new types of applications, and offer new services quickly.
- Open Source focus: making hardware vendors more open, and allowing the community to drive the network forward.
- Deliver agility and flexibility: thanks to virtualization, organizations can rapidly deploy and maintain new applications, services and infrastructure components to quickly meet their changing requirements.
- Reduce CapEx[2]: allowing network functions to run on off-the-shelf hardware.
- Reduce OpEX[3]: supporting automation and control through increased programmability of network elements to make it simple to design, deploy, manage and scale networks.

[2]Capital expenditure,the funds that a business uses to purchase major physical goods or services to expand the company's abilities to generate profits.

[3]Operating expense,results from the ongoing costs a company pays to run its basic business.

1.3 What is SDN ?

SDN is an approach that allows network administrators to easily manage and control large numbers of network devices and services. [5] defines Software Defined Networking as : "an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services" [6].

Basically SDN is the physical separation of the network control plane from the data (forwarding) plane.

1.4 Why SDN ?

Obviously, the need of an all new network architecture indicates that the previous technologies and standards were not sufficient to meet the current needs.

The problem that we face nowadays is that technology produced some new innovations such as social media, mobile devices, VoIP, and cloud computing, but those innovations are restricted by the limitations of the networks connecting them.

The question is, how can we use and continue developing new technologies and innovations without being restricted by the limitations of the network?

Recently, updating or changing software became easier thanks to various techniques like virtualization and cloud technology, however, this might not be possible in certain cases. Another big issue is the other part of the network -the hardware-, sometimes changing parts of the network or replacing old technologies and hardware might not be feasible without changing or tampering with the infrastructure, which can be expensive and even risky.

This means networks still have their limits especially in areas like upgrade, management and maintenance. As a result, this led to a lack of development and continuity in network technologies, and that is what we observed in the last years.

The network hasn't changed much because every modification needs infrastructure changes and manual interventions, all this requires orchestration between organizations as the network gets larger thus requiring findings as well.

As a potential alternative, Software Defined Networking is becoming a highly discussed topic these last years.

It is the huge impact that SDN has in a variety of fields such as economics, technology and networks overall that makes it so important. It provides new and innovative solutions highly needed by network administrators and service providers.

In summary, it possesses the ability to abstract and reduce the attachment between the application layers and the network resources in a given network. This means that it provides high availability and scalability, making networks dynamic and easily upgradable with little to no costs.

SDN has the potential to solve and avoid the problems mentioned above, it provides a way to virtualize and control different types of network devices easily. SDN can introduce many innovations to the network, like:

- Virtualization: Use network resource from different environments and combine them in a software based network.
- Orchestration: The ability of using, controlling and managing a variety of devices with simple commands.
- Programmable: The ability of changing the behaviour of a device or a selected portion of the network.
- Dynamic Scaling: Changing size, quality, performance...
- Automation: Avoiding human intervention as much as possible for a more efficient system.
- Visibility: Monitor resources, connectivity, network state, network graph and more.
- Performance: Optimizing the network utilization (load balancing, fast failure handling) and the ability to fully use all the resources available on the network.
- Service integration: Services such as firewalls and Intrusion Detection Systems (IDS) can be placed dynamically.

1.5 SDN Architecture

A Software Defined Networking architecture defines how a networking and computing system can be built using a combination of open, software-based technologies and special networking hardware that separate the control plane and the data plane of the networking stack.
The splitting of the control and data forwarding functions is referred to as “disaggregation”, because these pieces can be sourced separately, rather than deployed as one integrated system. This architecture gives the applications more information about the state of the entire network from the controller, as opposed to traditional networks where the network is application aware [7].

The SDN architecture is composed of three main layers or components, these layers are seperated by two interfaces which abstract and help transfer information, data and commands.

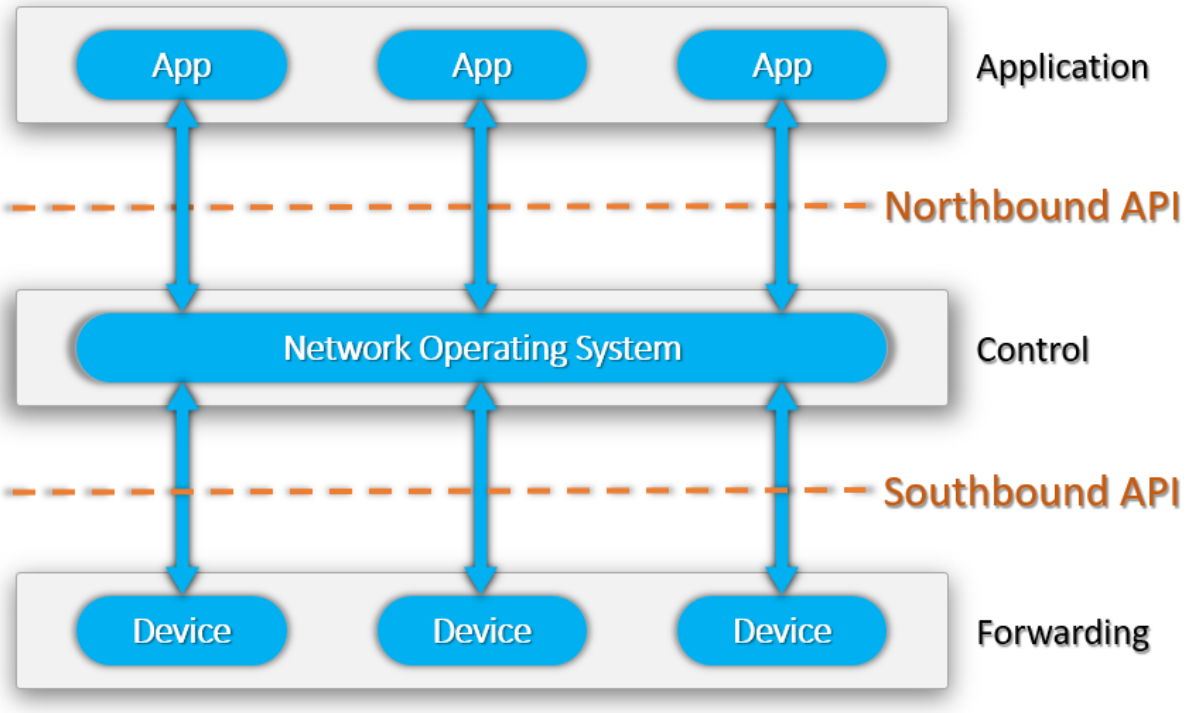

FIGURE 1.3: Standard architecture of Software Defined Networking

In Fig. 1.3 we distinct the following components :

1. SDN Applications : SDN Applications are programs that communicate behaviours and needed resources with the SDN Controller via application programming interfaces (APIs). In addition, the applications can build an abstracted view of the network by collecting information from the controller for decision-making purposes. These applications could include networking management, analytics or business applications used to run large data centers. For example, an analytics application might be built to recognize suspicious network activity for security purposes.
2. SDN Controller : The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and communicates back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening.

3. SDN Networking Devices : The SDN networking devices control the forwarding and data processing capabilities of the network.

4. The SDN architecture APIs are often referred to as northbound and southbound interfaces, defining the communication between the applications, controllers, and networking systems. A Northbound interface is defined as the connection between the controller and applications, whereas the Southbound interface is the connection between the controller and the physically networking hardware. Thanks to SDN's architecture, these elements do not have to be physically located in the same place.

In essence, all SDN models have controllers, Southbound APIs and Northbound APIs. The SDN Controller acts as the brain or the core of the network. It allows SDN users to gain a central look at the entire network and its topology, and empowers network administrators to instruct switches on how the forwarding plane should direct network traffic.

Southbound APIs interact with the hardware components of a network, pushing information to devices below. OpenFlow is the first southbound API and it's a heavily adopted protocol.

Northbound APIs push information above to the applications and business logic, giving network administrators the ability to pragmatically shape traffic and launch services.

The SDN architecture has many features [6] :

- Directly programmable: Decoupling the control from forwarding functions, enables the network to be flexible and programmable via the control plane.
- Centrally managed: The network control can be distributed, but it is logically centralized, it maintains a global view of the network, which simplifies application development, monitoring and management.
- Agile: Thanks to abstractions and the global view over the network, administrators can dynamically adjust network traffic flows to get the most out of their resources.
- Open standards-based and vendor-neutral: SDN simplifies network design and reduces the total network costs because it forces network devices to follow SDN standards, and eliminates vendor-specific devices and protocols.
- Programmatically configured: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN applications, which they can write themselves because applications do not depend on any proprietary software.

1.6 SDN Protocols

As mentioned previously, SDN's architecture contains a controller and two APIs to communicate with switches and applications. At the southbound API we find the most important part of SDN, the protocols, which helped in shaping and implementing real SDN solutions.

Protocols are responsible for passing information and commands between the controller and the supported hardware devices. The first known protocol used by SDN controllers is OpenFlow [8].

1.6.1 OpenFlow

The OpenFlow protocol is a standardized protocol for interacting with the forwarding behaviours of switches from multiple vendors. This provides a new way to control the behaviour of switches throughout the network dynamically and programmatically, which makes OpenFlow a key protocol in many SDN solutions.

The OpenFlow standard, created in 2008, was recognized as the first SDN protocol that defined how the control and data plane elements would be separated, and it was the first vendor-neutral standard communications interface between the control and forwarding layers of an SDN architecture.

Open Networking Foundation (ONF) is the organization in charge of managing OpenFlow standards, which are open source. However, OpenFlow is not the only protocol that makes up SDN, there are other standards and open-source organizations with SDN resources [7].

Open Networking Foundation

ONF is an organization dedicated to the adaptation, development and implementation of SDN, with the use of standard network protocols such as OpenFlow configuration and management standards protocol.

This organization makes SDN a commercial reality by making it suitable to the costumers' demands and needs [5].

How does OpenFlow work?

In classic switches, the control and the data plane are located on the same device, but in an OpenFlow switch these two functions are separated, the data plane still resides in the device while the control functions are moved to a separate controller.

The OpenFlow switch and the controller communicate via the Open Flow protocol, they use messages such as *Hello*, *FlowMod*, *FlowRemoved*, *PacketIn*, *PacketOut*, etc.

The Open Flow Switch (Fig. 1.4 [9]) can be broken into two parts, the switch agent and the data plane.

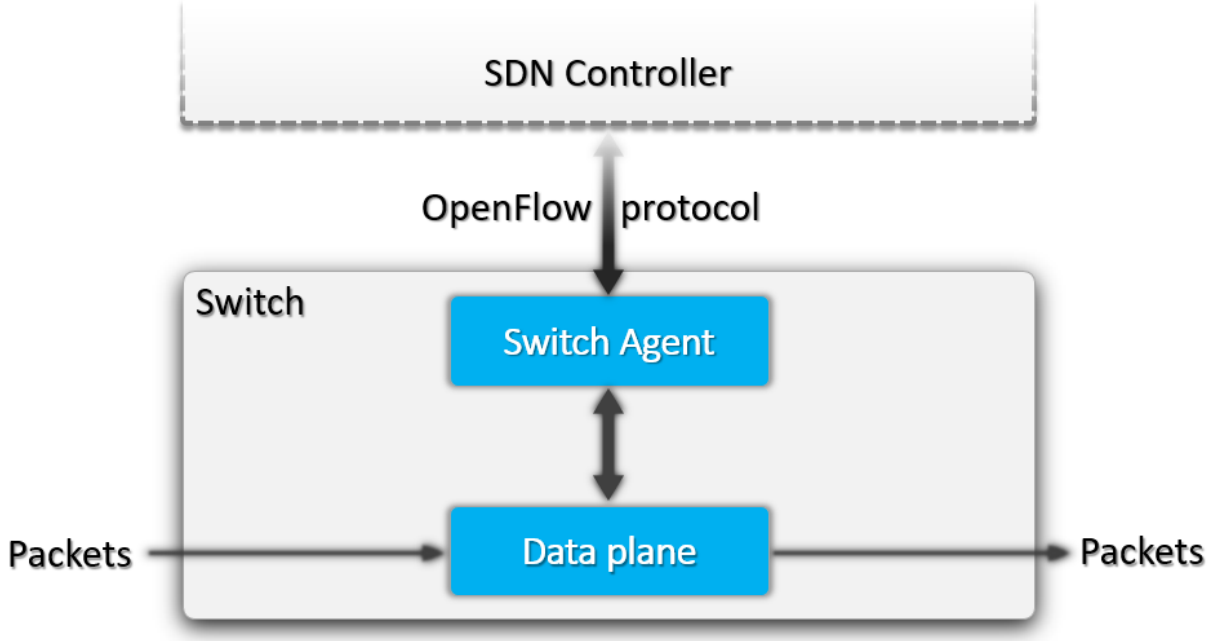

FIGURE 1.4: Standard architecture of an OpenFlow Switch

The first part is the data plane of an OpenFlow Switch (Fig. 1.5 [9]), it presents a clean flow table abstraction, each flow table entry contains a set of packet header fields to match, priority and actions that specify what to do with the packet.

When an Open Flow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends the message *PacketIn* which contains a buffer ID to the controller.

The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry by sending a *PacketOut* message. The message should contain the buffer ID of the packet, the idle and hard timeouts, actions, and the priority of the packet. The new entry shows the switch how to handle similar packets in the future [9].

In case the installed flow rule need to be updated or modified, the controller can send new actions via the *FlowMod* message.

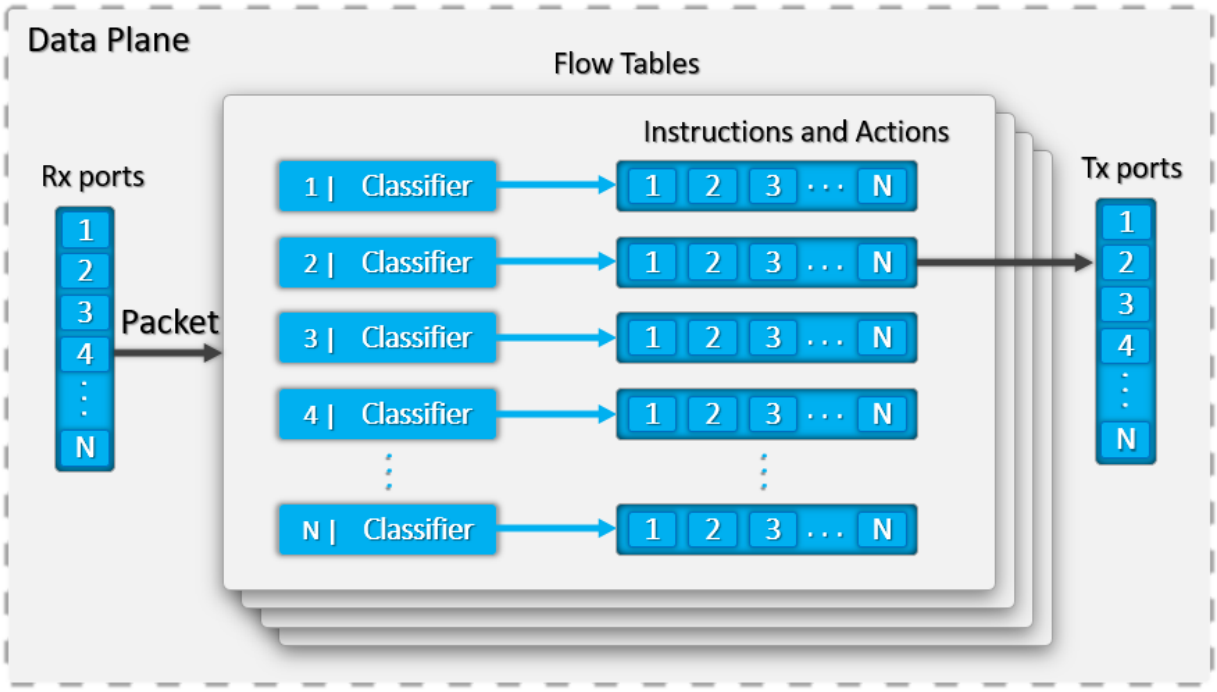

FIGURE 1.5: OpenFlow Data Plane

The second part of the OpenFlow switch is the switch agent (Fig. 1.6 [9]), which contains four components :

- The Open Flow protocol channel: the communication between the controller and the switch uses the secure channel which supported the Secure Sockets Layer (SSL) cryptographic protocol. That support switched to the successor of SSL, Transport Layer Security (TLS) [10].
- Core Logic: this component is responsible for managing the switch and executing commands.
- Data plane Offload: often the control plane will offload some functionality present in Open Flow, but not provided by the existing data plane implementation.
- Data plane protocol: this protocol is used to configure and manipulate the data plane state. The data plane contains ports, flow tables and classifiers. All packets arrive and leave through ports, the classifier matches each packet with its flow in the flow table.

Difference between SDN and OpenFlow

Most of the time, there is a confusion between SDN and OpenFlow, because OpenFlow is the main protocol that helped SDN to become a reality. Therefore, we need to understand and clarify the difference between them.

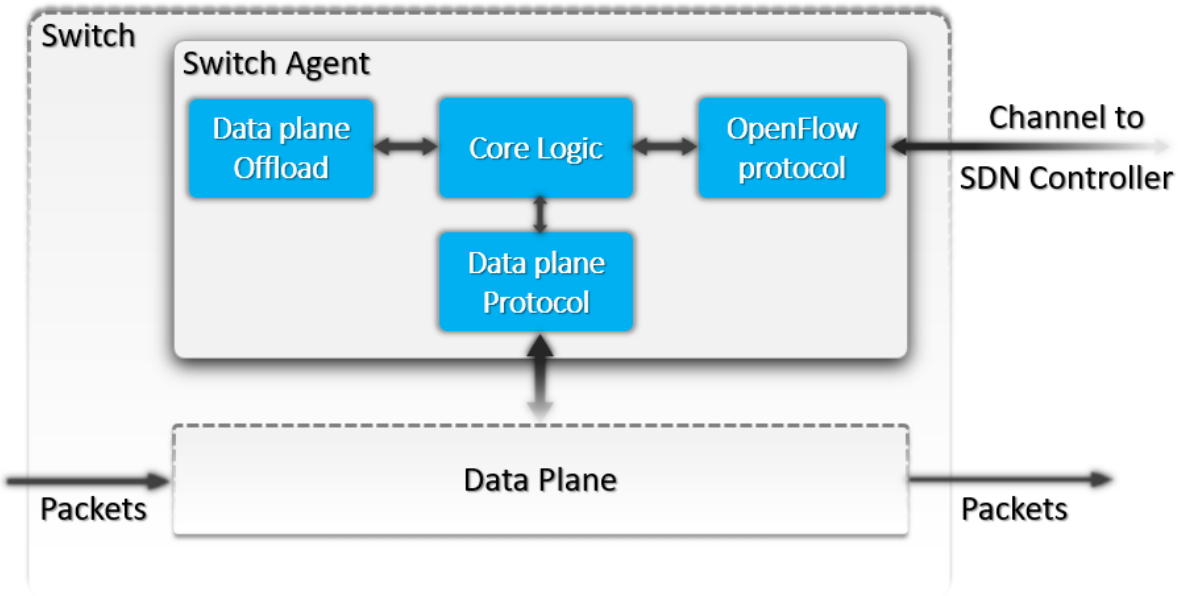

FIGURE 1.6: OpenFlow Switch Agent

We can see in the SDN initial architecture presented in the Fig. 1.7 [6], An SDN Controller is the core of the network, relaying information to the applications 'above' (via northbound APIs) and 'below' (via southbound APIs).

OpenFlow is one of the communication protocols that enables the SDN Controller to directly interact with the forwarding plane of network devices. For example, OpenFlow can provide :

- Transmitting from MAC address X to another MAC address Y.
- Drop all packets from or to an IPv6 address A.

1.6.2 Other Protocols

As mentioned, OpenFlow is the most popular protocol supported by most SDN controllers, that's why we dedicated a part of this chapter to OpenFlow, however there are other protocols out there.

Open vSwitch Database Open vSwitch Database (OVSDB) was created by Nicira team, which was later sold to VMware enterprise, it's a management protocol in SDN environments. It was part of an open source virtual switch designed for Linux named Open vSwitch (OVS). It was created like modern, programmatic APIs for controlling and managing OpenvSwithes.

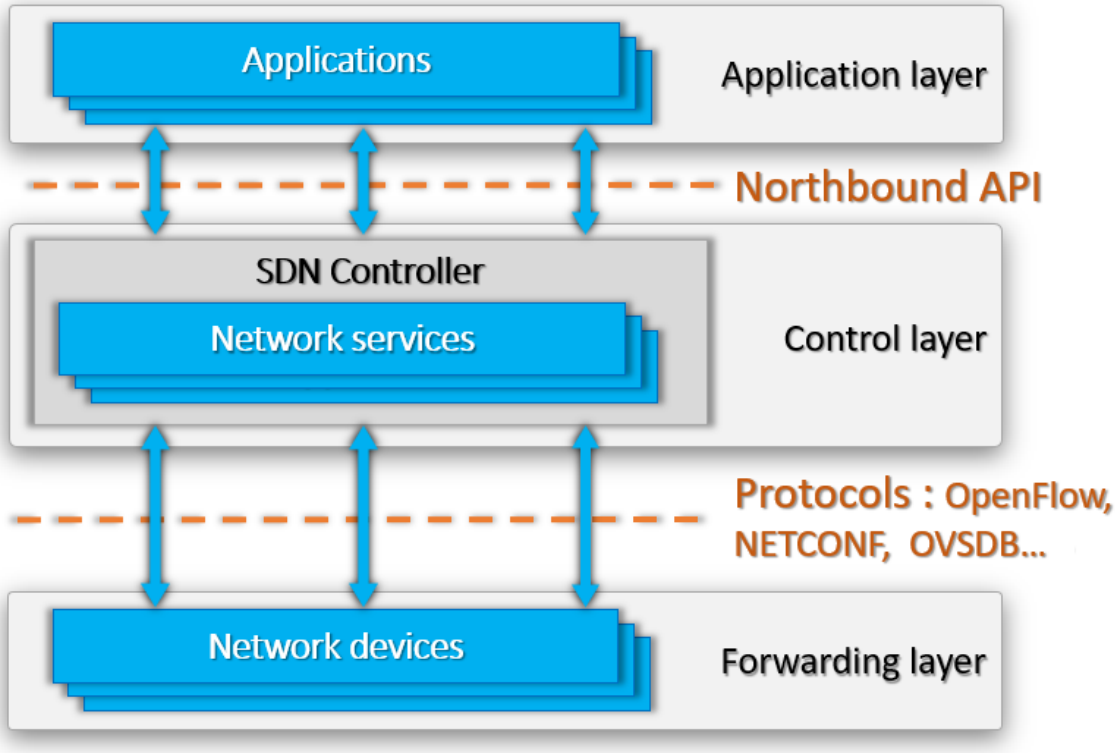

FIGURE 1.7: Basic layers of SDN

For SDN controller deployments of OVS, Open Flow is used to manage the flow entries, but OVSDB is used to configure the OVS, for example creating, modifying, adding, deleting ports, interfaces and more.

OVSDB is supported by a lot of vendors like DELL, Cumulus, Arista and more by integrating it to their platforms with SDN and network virtualization solutions [11].

NETCONF NETCONF, or the network configuration protocol provides operators and application developers with a standard framework and a set of standard Remote Procedure Call (RPC) methods to manipulate the configuration of a network device. It was created by The Internet Engineering Task Force under The NETCONF work group leadership. NETCONF provides a lot of mechanisms to configure network devices, install, manipulate and delete configuration [12, 13].

The interface to the routing System (I2rs) The Interface to The Routing System project was made by The Internet Engineering Task Force group (IETF), it was created in November 2012 to develop and to build an architecture that can support interface to the routing system[4].

[4]The term "routing system" means a hardware device with a virtual router, and any software that can provide routing functions.

While Open Flow is a protocol fully specified for packet forwarding decisions, the I2rs architecture is specified for management, user applications, and traditional routing protocols [14].

1.7 Common SDN controllers

An SDN controller serves as a sort of operating system for the network. By taking the control plane off the network hardware and running it as software instead, the controller facilitates automated network management and makes it easier to integrate and administer business applications.

Even though SDN is relatively a new standard, there are already some SDN controllers available:

NOX The first SDN Controller was NOX, which was initially developed by Nicira Networks, alongside OpenFlow. In 2008, Nicira Networks (acquired by VMware) donated NOX to the SDN community, where it has become the basis for many subsequent SDN Controller solutions [15].

ONIX After NOX, Nicira went on to co-develop ONIX with NTT and Google. ONIX is the base for the Nicira/VMware Controller and rumoured to be the base for the Google WAN Controller. While ONIX was originally supposed to be opened up, the parties later decided not to make it Open Source [16].

POX POX is an SDN controller that supports the Open Flow protocol, it's an open source development platform based on Python, it's becoming more used than the NOX project since it enables fast development and prototyping [17].

Open MUL Open MUL is an SDN controller that supports the Open Flow protocol, this controller is based on the C programming language. It can use multi-level northbound interfaces for the applications. It's also highly flexible, and easy to learn [18].

Ryu Ryu is an SDN controller developed by NTT. Written in Python, the source code of this controller is maintained by the open Ryu community OpenStack, which is a community focused on developing cloud operating systems. Ryu supports a lot of SDN protocols such as Open Flow, NETCONF and more [19].

Beacon Started in early 2010, Beacon is a Java-based OpenFlow Controller licensed under a combination of the GPL v2 license and the Stanford University FOSS License Exception v1.0. Characterized with its high performance, productivity, and having the ability to stop or start a new application at runtime. It has been used for teaching and research [20].

Subsequently, vendors, such as Cisco, HP, IBM, VMWare and Juniper have jumped into the SDN Controller market with their own offerings. The original HP, Cisco, and IBM Controllers were all based off Beacon, and now have moved towards OpenDaylight.

Floodlight Floodlight is an SDN controller based on JAVA and forked from Beacon, it was made available under an Apache 2.0 license and formed the basis of one of the early commercial Controllers from Big Switch Networks. It was supposed to be a part of Cisco's project OpenDaylight, but in June 2013 Big Switch stepped away amid what seems to be conflicting interests with Cisco [21].

Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the Open Flow protocol. It allows developers to easily adapt software and develop applications.

OpenDaylight On April 8, 2013, OpenDaylight was founded by the open-source foundation which is part of the Linux Foundation. This Controller is Java-based, and derived from the original Beacon design. It supports OpenFlow and other southbound APIs (such as Cisco OpFlex) and includes critical features like high-availability and clustering.

An OpenDaylight Controller is implemented solely in software and is kept within its own Java Virtual Machine (JVM), but it can be deployed in a variety of production network environments. In conjunction with its SDN Controller, OpenDaylight Project released its first code, Hydrogen, which offered three different editions for users. In September 2014, OpenDaylight Project unveiled its second code release, Helium. Both code releases are open frameworks for network programmability to enable SDN for any network size.

Companies like Cisco and Brocade are shipping their own OpenDaylight Controllers, including Brocade's Helium-based Vyatta Controlle and Cisco's Hydrogen-based Extensible Network Controller.

OpenDaylight is committed to an open design and development process. The community operates around a time-based release cycle of roughly every six months with frequent development milestones. The naming convention for each OpenDaylight release follows the atomic number of elements in the periodic table such as Boron (september 2016) and Carbon (May 2017) [22].

ONOS Open Network Operating System (ONOS), was initially developed by Open Networking Lab (ON.Lab), and released on December 5, 2014. The next chapter will be fully dedicated to go in depth into this controller's architecture and technology.

1.8 Future of SDN

With everything mentioned, SDN is still relatively a new approach, nevertheless, since its introduction, it has gained a lot of reputation and support from big companies thanks to the benefits SDN can provide for them. Some companies have fully integrated SDN in their business such as Google[5], which uses it to prioritize traffic like Gmail backups, a capability that will be transferred to their cloud services' customers. However, SDN is still avoided since some companies prefer to keep their network infrastructure and avoid the risk and extra cost of deploying a new technology.

From an economical point of view, network maintenance and growth is expensive because the network doesn't scale economically like storage and compute do. As a network grows larger it becomes more expensive to deploy and maintain. On the other hand, using SDN requires changing the infrastructure once, with cheaper hardware, and after that there will be no extra expenses since everything else is done in software. It's only logical to assume that in the near future, it's inevitable for companies to adopt SDN or a similar approach, to make their networks more flexible, and use it to get the most out of their businesses.

There is an idea about making SDN better by taking it to another level. Stanford Prof. Nick McKeown, one of the guys who invented SDN, and a serial entrepreneur in networking technology start-ups, now brings the next transformation: "programmable switching chips". These new switch chips can be programmed so that they could perform different functions such as firewall and load balancing, which currently require specialized networking equipment.

The vision of SDN and that of programmable switching chips is basically one. As Barefoot[6] puts it : *"We envision a world where programmable networks outperform fixed-function networks. We believe that programming the network should be as easy to program as a server. That's a vision worth pursuing"* [24, 25].

SDN and the other network virtualization approaches are taking the networking community by storm and they are here to stay. On October 19, 2016, Open Networking Foundation and Open Networking Lab, the two pioneering organizations dedicated to the adoption of SDN, announced an agreement to become a single organization under the ONF name. Operations began immediately after the agreement, and are expected to be completed in late 2017.

The affiliation between the two organizations will chart the next phase of SDN. According to Allied Market Research [26], the SDN market is expected to reach $132.9 billion by 2022 [27].

[5] Amazingly, Google was one of the first to implement SDN even though it was still in its beginning phases. Google announced at the 2013 Open Networking Summit that by replacing one of their secondary network infrastructure, they were able to successfully implement and test SDN [23].

[6] Barefoot Networks is a computer networking company co-founded by Nick McKeown, headquartered in Palo Alto, California. The company designs and produces programmable network switch silicon, systems and software.

1.9 Conclusion

Software Defined Networking brings radical changes to the network as we know it, SDN is considered as an architecture that allows networks to be programmable, easily managed and opened to all applications. This allows it to be easily exploited and used in different domains like WAN, data centers, campuses, etc.

In this chapter we introduced SDN, the OpenFlow protocol and its importance in SDN implementation, as well as OpenFlow switches and some well-known SDN controllers.

In the next chapter, we will see Open Network Operating System (ONOS) a brand new and one of the most important and interesting SDN controllers available.
In order to understand what is ONOS and what it's used for, we needed first to have an understanding of SDN's concept and architecture.

CHAPTER 2

Open Network Operating System

2.1 Introduction

As we discussed in the previous chapter, SDN is becoming more popular everyday. Therefore we keep seeing many new SDN controllers in the last couple of years. Our focus in this chapter is one particular SDN controller, which is Open Network Operating System (ONOS) that is believed to became the face of the SDN community for many reasons that we will discuss later on.

2.2 Definition

The Open Network Operating System (ONOS) is a software defined networking (SDN) OS for service providers that has scalability, high availability, high performance and abstractions to make it easy to create apps and services. The platform is based on a solid architecture and has quickly matured to be feature rich, and production ready. The community has grown to include over 50 partners and collaborators that contribute to all aspects of the project, including interesting use cases such as CORD [28].

2.3 History

Before going in depth about ONOS and its architecture we should first have a global view on its history, specifically Open Networking Lab (ON.LAB), the makers of ONOS.

2.3.1 ON.LAB

"The Open Networking Lab (ON.LAB) is a non-profit organization founded by SDN inventors and leaders from Stanford University and UC Berkeley to foster an open source community to develop tools and platforms to realize the full potential of SDN. With ON.LAB roots in SDN research, members of its SDN ecosystem can explore new, real-life applications. They can experiment with the latest SDN concepts and thinking from renowned research labs. ON.LAB has a relatively small team of highly motivated and talented individuals, with experience and track records in industry and research labs, who are focused on creating high quality open source tools and platforms that members can use to explore the possibilities of SDN" [29].

Founded in 2012 and situated in Menlo Park, California, this organization specializes in Software Defined Networking, Service providers, and Telecommunications. It's the founding member of the open source project ONOS, which has become a Linux Foundation project.

Other Projects Next to ONOS, the ON.LAB organization has several other projects listed below :

- CORD (Central Office Re-architected as a Datacenter) combines NFV, SDN, and the elasticity of commodity clouds to bring datacenter economics and cloud agility to the Telco Central Office. CORD lets the operator manage their Central Offices using declarative modelling languages for agile, real-time configuration of new customer services. Major service providers like AT&T, SK Telecom, Verizon, China Unicom and NTT Communications are already supporting CORD [30].
- OVX is a network hypervisor that can create multiple virtual and programmable networks on top of a single physical infrastructure. Each tenant can use the full addressing space, specify their own topology, and deploy the network OS of their choice. Networks are reconfigurable at run-time, and OVX can automatically recover from physical network failures [31].
- XOS augments OpenStack and ONOS with explicit support for scalable services, making it possible to create, name, operationalize, manage and compose services as first-class operations. In doing so, XOS provides a framework for service assembly and composition that helps implement scalable services that become part of the cloud. XOS is designed around the organizing principle of Everything-as-a-Service (XaaS),

which unifies SDN, NFV, and the Cloud under a single coherent programming model. It is an open source project on GitHub, and is currently being applied to two use cases: OpenCloud (a joint ON.Lab, PlanetLab, and Internet2 initiative) and CORD (a joint ON.Lab, AT&T initiative) [32].

- Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software Defined Networking. Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a personal computer [33].

2.3.2 History of ONOS

On December 5, 2014 ON.Lab, and a partnership comprised of leading service providers, including AT&T, NTT Communications and key vendors, released ONOS's source code for download to start the open source community [34]. On October 14, 2015, the Linux Foundation announced that ONOS had joined the organization as one of its collaborative projects.

Even with its first release, ONOS provided many new key features such as:

- A clean-slate, clustered, modular architecture with distributed core for high availability, performance and scale-out
- Application Intent Framework, providing a high-level policy driven programmatic abstraction.
- Pluggable southbound for supporting a diversity of devices and protocols.
- OpenFlow 1.3 and 1.0 support.
- GUI for visualization, visibility and configuration.
- Apache Karaf for modularity, customization and extensibility.
- Service provider use cases to demonstrate ONOS's capabilities.

Releases After ON.Lab was founded, they started implementing ONOS. In order to overcome all the new faced challenges, they started with a few prototypes before releasing ONOS's first version (Avocet 1.0.0).

The first prototype was all about basic functionality, scale-out, high-availability, fault tolerance, network view and supported only OpenFlow 1.0.

With the second prototype ON.Lab were mainly focused on improving performance using lightweight components such as RAMCloud instead of Cassandra. Doing this introduced noticeable improvements, all while relying on Open Source components.

This led to the introduction of the first version of ONOS (Avocet on Dec 5, 2014), and there have been many releases ever since. ONOS follows a quarterly release pattern at the end of February, May, August and November. The latest release, codenamed Kingfisher is set for release on may 2017 [35].
ONOS is being supported by many collaborators and partners like AT&T, Google, Verizon, Huawei, Cisco and many others. This can be observed clearly in Fig. 2.1 [36] which is a chart representing ONOS's active member growth from January 2015 to October 2016.

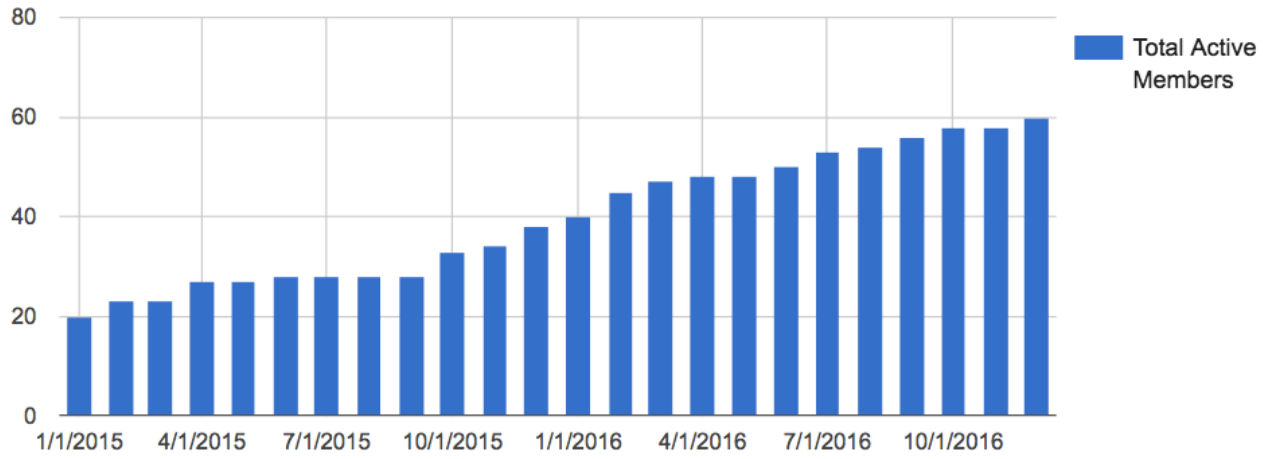

FIGURE 2.1: ONOS Partners and Member growth chart (Jan-2015 - Oct-2016)

2.4 Architecture overview of ONOS

Any SDN controller must follow the standards that make up SDN, but this doesn't necessarily mean that all controllers are the same.

SDN controllers are diverse, with different priorities, goals and architectures. Each controller tries to provide as many features and benefits as possible, some controllers try to provide specific features to target specific clients and use-cases.

2.4.1 Features

Since ONOS is targeted towards Service Providers and large WAN networks, it was developed from the ground up with a few goals in mind :

Distributed Core : One of the main goals that ONOS was architected to achieve is having a distributed core, it is responsible for presenting a logically centralized view of network state and manages the state across instances. The distributed core is the key component enabling scale, clustering and high availability.

Abstractions Northbound abstractions and APIs ease application programming and provide easier ways for applications to talk with the core of ONOS.
Southbound abstractions allow the OpenFlow protocol (early releases of ONOS) or other protocols (supported in later versions) to interact with network elements.

Open Source Like most SDN controllers available, another feature in ONOS is that it's built using open source components which makes ONOS free as well as adoptable, modular and easily upgradable.

Software Modularity probably the most needed feature since it provides many benefits :

- Architectural integrity and coherence.
- Simplified test structure, allowing more comprehensive testing.
- Easier maintenance with fewer side effects of changes.
- Extensibility and customization of components.
- Avoidance of cyclic dependencies.

Performance The support for software modularity and abstractions help with developing applications easily which is critical for ONOS to keep evolving. And thanks to the distributed core, the probability of the network being paralyzed is reduced significantly. It also provides an easy way to scale by adding or removing instances, all without sacrificing performance.

All these features and more, provide a highly noticeable performance gain over traditional networks.

2.4.2 Tiers of ONOS

Since ONOS is an SDN controller, it's based on SDN architecture layers, and is comprised of several unique set of layers or tiers that make it different even from other SDN controllers.

As shown in Fig. 2.2 [37], the ONOS architecture consists of three main tiers (Apps, Core and Providers), the other tiers act as bridges or intermediates.

1. **Apps :** Applications consume and manipulate information aggregated by the managers via the AdminService and Service interfaces. Applications have a wide range of functionality, ranging from displaying network topologies in a web browser to setting up paths for network traffic.

   Each application is associated with a unique ApplicationId. This identifier is used by ONOS to track context associated with an application, such as intents.

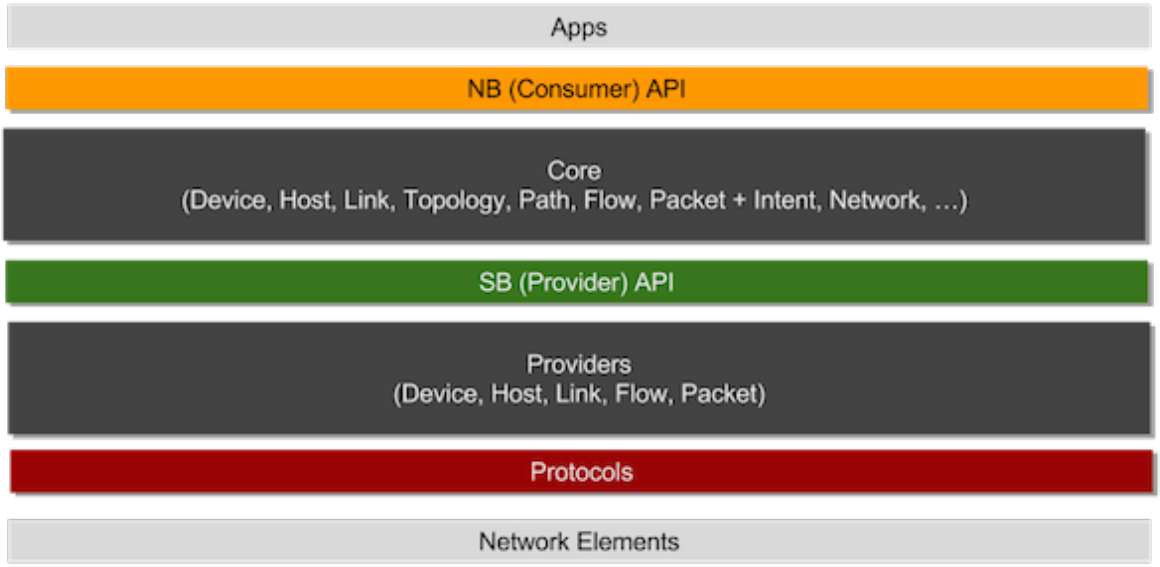

FIGURE 2.2: Tiers of ONOS

To obtain a valid ID, applications register with the CoreService, providing their name which is expected to follow the reverse DNS notation, e.g. org.onlab.onos.fwd [37].

2. **Northbound Abstractions :** (The consumer) Northbound Abstractions and APIs ease application programming and provide a way for applications to talk with ONOS.

   There are two powerful Northbound abstractions: the Intent Framework and the Global Network View.

   The Intent Framework allows an application to request a service from the network without having to know details of how the service will be performed. This allows network operators and application developers to program the network at a high level, they can simply specify their intent: a policy statement or connectivity requirement.

   Some example intents:

   - Set up a connection between Host A and Host B
   - Set up an Optical Path from Switch X to Switch Y with Z amount of bandwidth
   - Don't allow host A to talk to host B

   The Global Network View provides the application with a view of the Network like hosts, switches, links, and any other state associated with the network such as utilization. An application can program this network view through APIs [38].

3. **Core :** The distributed core manages the state across instances and is the key component enabling scale, clustering and high availability. The core will be described in more detail in the following section.

4. **Southbound API :** (The provider) A pluggable Southbound allows OpenFlow and other protocols and components to interact with the core. Through this abstraction, the distributed core can maintain the state of network elements without having to know any unnecessary specifics.
5. **Providers :** Providers[1] interact with the network via protocol-specific libraries, and with the core via the ProviderService interface. The protocol-aware providers are responsible for interacting with the network environment using various control and configuration protocols, and supplying service-specific sensory data to the core. Some providers may also need to accept control edicts from the core and apply them to the network using the appropriate protocol-specific means. These are fed into the Provider via the Provider interface [37].
6. **Protocols :** Contains implementation of specific control and management protocols needed by ONOS to communicate with real devices like OpenFlow, NETCONF and more.
7. **Network elements :** The actual physical devices that create the network, these devices must be compatible with the southbound protocols supported by ONOS like OpenFlow.

2.5 Core technology of ONOS

One of the key advantages of ONOS is its distributed core, it simplifies scale-out and provides very high availability. Thanks to this core, multiple instances can behave as a single logical entity, and appear to applications and network devices as a single instance.

These instances can quickly inform each other using speed messaging in a publish/subscribe model. To ensures that the state evolves in consistent fashion, the logical clock in each instance timestamps events observed in the underlying network [39].

The core itself consists of many subsystems that are designed to be extended and tailored when required, this allows the core to be modular and flexible.

2.5.1 Subsystems

A subsystem or service is a unit of functionality that is comprised of multiple components that create a vertical slice through the tiers as a software stack [37, 40].

[1]The providers were referred to as 'Adapters' in earlier versions and ONOS architecture diagrams and the two terms can be used interchangeably.

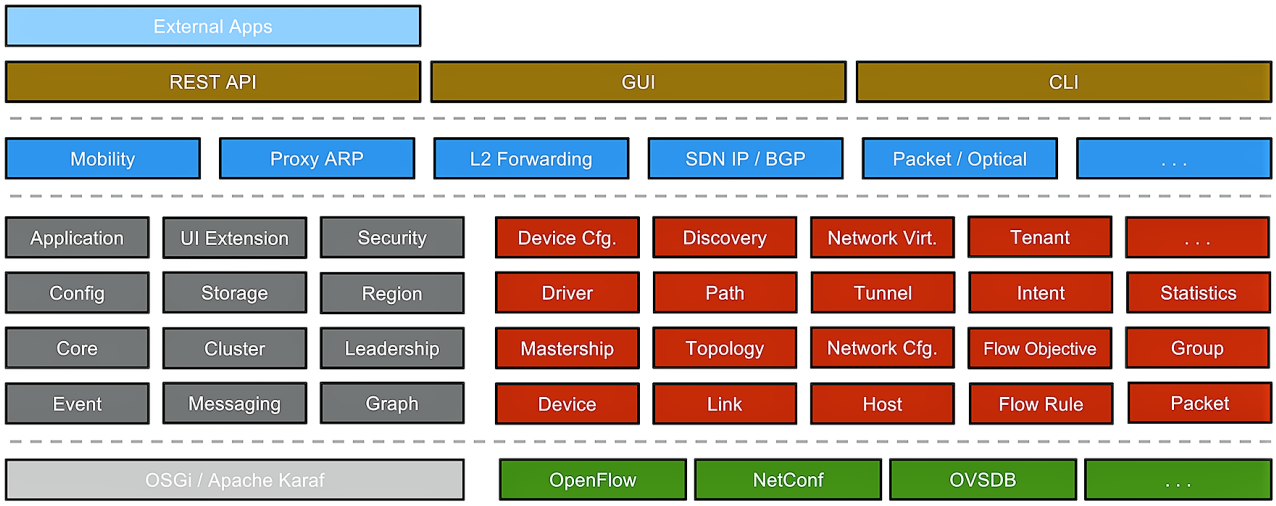

FIGURE 2.3: ONOS subsystems

There are many subsystems that make up ONOS as presented in Fig. 2.3 [37]. Some of the primary subsystems include :

- Device Subsystem : Manages the inventory of infrastructure devices.
- Link Subsystem : Manages the inventory of infrastructure links.
- Host Subsystem : Manages the inventory of end-station hosts and their locations on the network.
- Topology Subsystem - Manages time-ordered snapshots of network graph views.
- PathService : Computes/finds paths between infrastructure devices or between end-station hosts using the most recent topology graph snapshot.
- FlowRule Subsystem : Manages the inventory of flow rules installed on the infrastructure devices and provides flow metrics.
- Packet Subsystem : Allows applications to listen for data packets received from network devices, and to emit data packets out onto the network via one or more network devices.

Each of the subsystem's components resides in one of the three main tiers, and can be identified by one or more Java Interfaces that they implement. The relationships between various subsystem components are summarized in Fig. 2.4 [37]. The top and bottom dotted lines in the figure represent the inter-tier boundaries created by the northbound and southbound APIs, respectively [37].

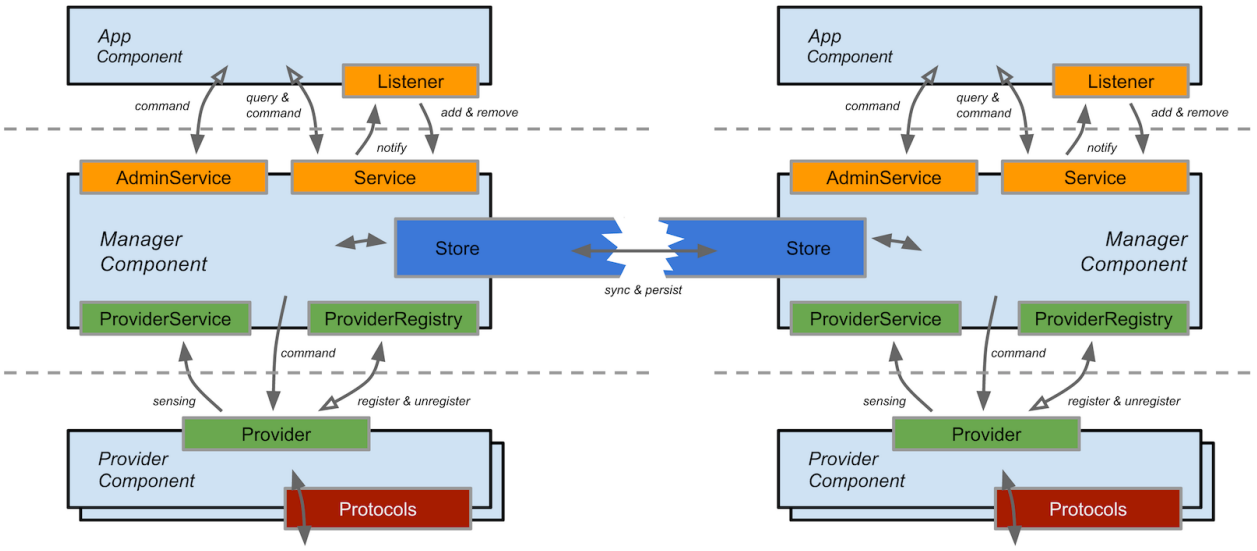

FIGURE 2.4: ONOS subsystem components relationship

2.5.2 Intents

In order to simplify application developing and reduce complexity, ONOS uses a concept called intents for applications to describe and pass their requests to the ONOS core in order to alter the network's behaviour.

Thanks to these intents, apps can focus on what should be done, rather than how it should be done. Then the core takes care of compiling intents into actionable objectives which are installed into the infrastructure.

Intents are : immutable, contain network resources and constraints, have a priority, and are always identified by the ApplicationId of the application that submitted them, and a unique IntentId, generated at creation. Some of the built-in intents include : *HostToHost*, *PointToPoint*, *SinglePointToMultiPoint*, *MultiPointToSinglePoint*, *TwoWayP2PIntent*. Developers can extend built-in intents to new intents for special purposes [41, 42].

A good example of an intent use-case can be the *HostToHost* intent, where one host needs to talk to another, ONOS calculates the best path and installs flows. Resources required by objectives are then monitored and if there is any failure in the path between the two hosts, ONOS automatically reacts by recompiling the intent and re-installing the flows. In case a new path cannot be found, the particular intent goes to failure state.

2.5.3 Technologies and implementation of ONOS

The core of ONOS is implemented using a number of APIs and programs. The platform is written in Java and uses OSGi for functionality management.

The individual features are loaded using the OSGi runtime called Karaf. The following items are the main components in ONOS :

- Java 8 Java 8 is a revolutionary release of one of the most used development and programming platforms in the world. It was released on March 18, 2014. This version includes many upgrades for the Java language and all the libraries. Java 8 includes features for productivity, ease of use, security and improved performance [43].

- Karaf Karaf is an open source project for The Apache Software foundation, under the Apache v2 license. Written in Java, it provides a distributed environment OSGI to create and implement servers, it can be used in every operating system with the Java package [44].

- Raft Raft is a consensus[2] algorithm, it is used in the development of the ONOS controller to manage the consistency (ConsistentMap) [46].

- AngularJS AngularJS is an open source JavaScript framework to develop web applications created by Google, and launched on October 20, 2010. The idea behind this project is to create a platform for developing user interfaces and to connect it with software packages and business applications easily [47].

- Kryo Kryo is a Java framework for serialization graphs, it was created to be fast and easy to use, it can be used for files, databases or networks [48].

- Atomix Atomix is a framework for developing web applications. Written in PHP, this framework is simple and easy to use. It was mainly used for the distributed mechanisms and store management in making ONOS's distributed core [49].

- Copycat Copycat is a framework that was built on the Raft algorithm. The objective of this framework is to deal with all repetition in outputs and inputs so it can be easy for developers to manage and build distributed systems [50].

- Jersey Jersey is an open source framework based on Java for developing RESTful web services. This framework provides the JAX-RS API an important element for developing RESTful applications, it's an API with mailing lists for communication with the expert group [51].

[2] Consensus is a fundamental problem in fault-tolerant distributed systems. Consensus involves multiple servers agreeing on values. Once they reach a decision on a value, that decision is final [45].

- Guava-libraries Guava is an open source set of java libraries. This utilities contains a lot of the most used libraries in Java programming [52].

2.6 SDN-IP

The problem that will arise as more SDN networks are deployed, is how to be compatible with traditional networks. In Internet today, Autonomous Systems (AS) use Border Getaway Protocol (BGP) to communicate with each other, and to exchange information between their networks.

SDN-IP is an ONOS application that allows SDN networks to scale and connect to Internet and other external networks. It acts as a regular BGP speaker and allows the SDN network to appear as a single AS. SDN-IP provides :

- Compatibility: SDN-IP can be used with networks that already have BGP.
- Operational flexibility: SDN-IP can run on one or multiple ONOS instances.
- High availability: SDN-IP exploits ONOS's distributed architecture to provide high availability, which means the SDN-IP service keeps working as long as there is an instance with SDN-IP running.
- Scalability: with the use of the BGP and the ONOS clusters, SDN-IP can control Software Defined networks with Large-scale.
- Protocol compatibility and vendor independence: SDN-IP is an application based on BGP and does not need any specific protocol or vendor extensions.

SDN-IP architecture In a typical scenario, an SDN network is composed of different OpenFlow switches, controlled by a set of ONOS instances.

As shown in Fig. 2.5 [53], SDN-IP runs on top of ONOS instances and is connected to each BGP speaker, only one of the SDN-IP instances is active at the time, leaving others as a backup that can take over if the main SDN-IP instance fails.

When the SDN-IP application starts working, it needs to discover the related external networks that use BGP, and let them connect with the SDN network. To achieve that, the external routers advertise their routes and reach the BGP speakers which makes routes and sends them to the running SDN-IP.

Since SDN-IP is an application it can create intents from the received BGP routes, then ONOS compiles them into flow rules that get installed on the data plane [53, 54].

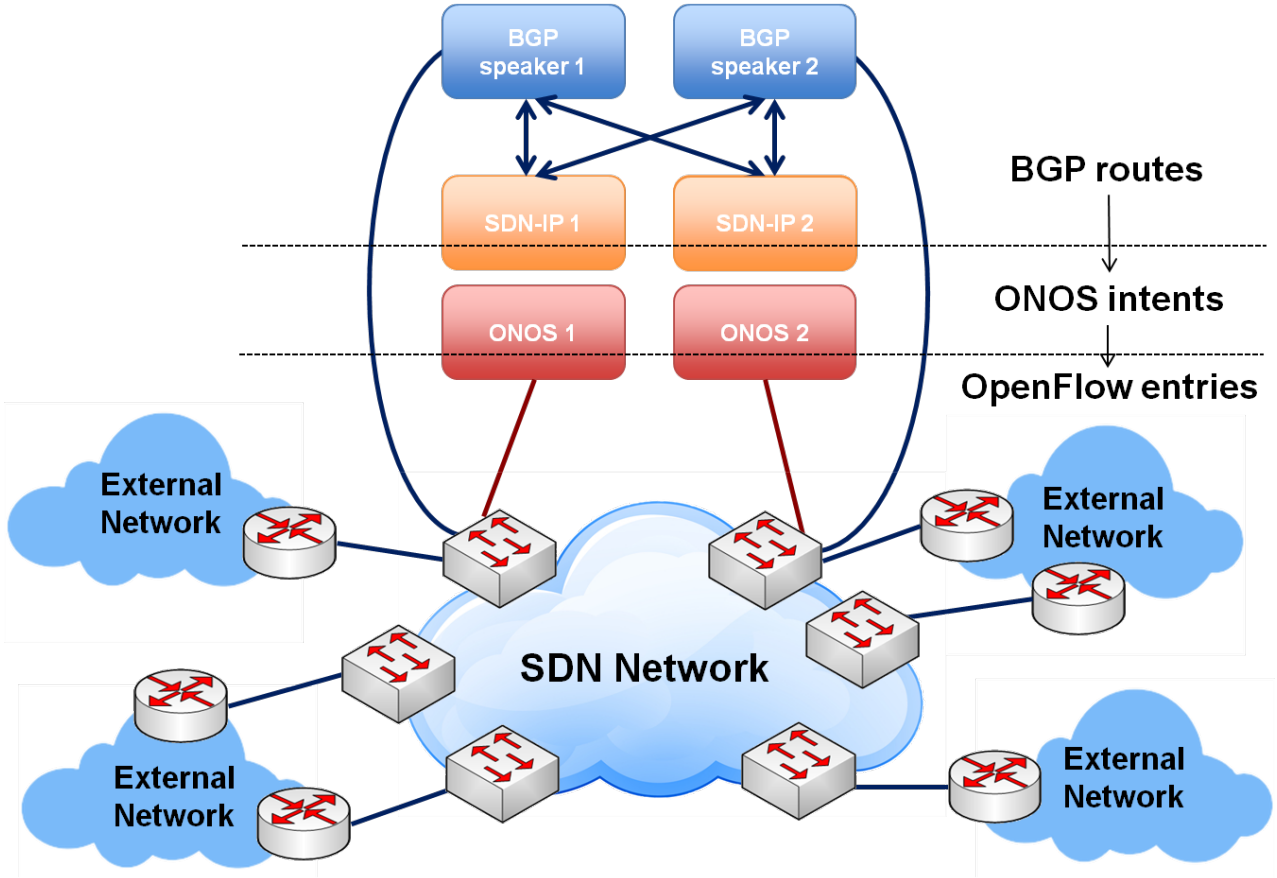

FIGURE 2.5: SDN-IP architecture

2.7 Conclusion

ONOS is a very reliable SDN controller with a lot of features to offer. In this chapter, we presented the features of ONOS and its architecture, which enables high availability, scalability and fault tolerance. The architecture is also modular to encourage and simplify future improvements.

ON.Lab released ONOS as an open source project to enable the community and the developers to change or create their own apps and help improve the project. The ONOS project also provides tutorials for contributors and help them understand the controller and be able to contribute, improve and be a part of this project, and that's what makes it one of the best SDN controllers.

CHAPTER 3

Testing and discussion

3.1 Introduction

After understanding the logic of SDN controllers and how they work, and introducing the ONOS controller, we can now have a practical approach and hands-on experience. This chapter will be dedicated to cover all the basic requirements needed for a proper initial start-up to work with SDN controllers and ONOS specifically.

The infrastructure of SDN is different, but we can simulate an SDN network using mininet. This will help us test ONOS freely and gather results in a controlled environment, we can work with ONOS and see some of its features and hopefully give a good overview of the use cases that ONOS is built to manage in reality.

3.2 Mininet

Mininet is an indispensable tool for anyone who wishes to work with SDN. The importance of mininet lies in its ability to provide any desired network topology, developers can create their custom topologies with hosts, OpenFlow switches and most importantly, SDN controllers. All this is done in a virtual environment which means network devices can be controlled to simulate real life situations without any physical resources needed other than the terminal running mininet. This makes it useful for developers, teachers, and researchers [55, 56].

Using mininet python API we can create custom topologies to be tested and monitored. Bandwidth, delay and other network proprieties can also be defined.

In order to use mininet we can either download and install it, or download the independent virtual machine instead. Either way there are no downsides, since we can always connect mininet with the controller by providing its IP address.

Due to mininet's significance, a good part of this chapter is dedicated to show some of its abilities.

3.2.1 Working with mininet

Before we start our tests we should at least know the basics of mininet and its CLI, these are its most needed commands:

- `$ sudo mn` start mininet and access the CLI. (mininet must run as root)
- `mininet> exit` to remove the network and close mininet's CLI.
- `$ sudo mn -c` is very recommended after closing mininet to perform a clean-up and remove any potential unwanted remains.
- To start a network topology we can have multiple options added directly to `$ mn`
  - `--topo` create single, minimal, linear or tree topologies and specify the diameter.
  - `--custom` users can also run custom topologies by either downloading or creating a python topology file (*.py) and specifying its path.
  - `--mac` create MAC addresses similar to IP addresses thus making them easier to read.
  - `--switch` specify the switch type, usually open vswitch.
  - `--controller` specify the SDN controller to use such as Ryu or NOX, we put "remote" for other unidentified controllers like ONOS.
  - `ip` in case the controller is not running locally we must provide its IP address.
- `mininet> nodes` lists all network nodes (controllers, hosts, switches).
- `mininet> net` lists the links between nodes of the network.
- `mininet> pingall` ping all hosts, this allows the controller to discover the hosts in the network.
- We can execute commands from the hosts' terminals by specifying the name of the host which will be replaced by its IP address for the command to work properly, some of the most used commands are :

  - `mininet> h1 ping h2` ping the host `h2` from `h1` and prints the results in the terminal.
  - `mininet> h1 ifconfig` display the network proprieties of `h1` .
  - `mininet> xterm h1 h2 h3` in case the hosts' terminals are often needed they can be opened in separate dedicated windows to execute linux commands directly from the host terminals.

- `mininet> link s1 s2 down` and `mininet> link s1 s2 up` are very useful commands for fault tolerance testing, the two commands bring the link between s1 and s2 down then up respectively.
- `mininet> source myscript` To avoid repetitive CLI typing, especially in cases of testing and simulations, it can be useful to create then run a file with multiple mininet CLI commands.
- `mininet> py` is another useful command to execute python code while running the simulation, it can be useful to add switches, hosts and links to the topology on the go. For example to create a host h1 and add it correctly to the switch s1 :
  - `mininet> py net.addHost('h1')` creates the host 'h1'.
  - `mininet> py net.addLink(s1,h1)` creates the link between 's1' and 'h1'.
  - `mininet> py s1.attach('s1-eth1')` attach the virtual link to the switch.
  - `mininet> py net.get('h1').setIP('10.0.0.1')` create an IP address for the host.

3.2.2 Custom topology

Mininet supports parametrized topologies. With a few lines of Python code, we can create a topology, specify the number of hosts, switches, controllers, links and link proprieties.

Another way is to run the MiniEdit script located in mininet's examples folder `$ sudo ~/mininet/examples/miniedit.py` , create a topology with the GUI shown in Fig. 3.1 and run or save the file.

The interface is basic and simple, it contains a menu bar and a raw of components at the left side. The components are legacy or OpenFlow switches, routers, hosts, controllers and links to connect the components.

The example seen in Fig. 3.1 is a topology with three controllers, nine switches and fourteen hosts.

In case of creating an SDN network,the main things that should be taken into consideration are the switches and the controllers. For the SDN network to run properly, we must choose

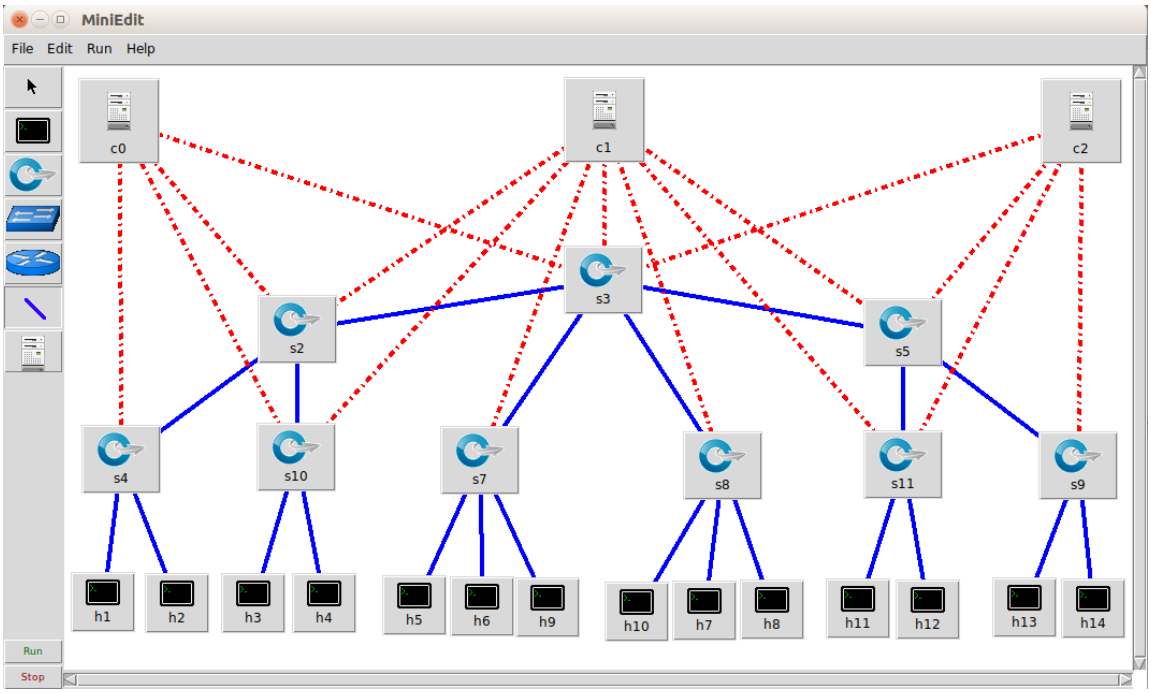

FIGURE 3.1: MiniEdit's Graphical User Interface

OpenFlow, not legacy switches. And the switches must be controlled by at least one SDN controller.

3.2.3 Python API topology

The best way to customize and simulate a network is to create a topology file. Mininet offers support of custom topologies and has specific classes and commands to make it even more flexible.

The text in Fig. 3.2 represents the file `topo-2sw-2host.py` included with mininet under `~/mininet/custom`. This file shows the most basic features in mininet, including a simple topology with only two switches and two hosts.

Some of the most needed classes and functions include :

- mininet: main class to create and manage a network, all other classes are inherited from it.
- topo: the base class for mininet topologies.
- node: nodes are what composes the topology, the nodes can be hosts, switches and controllers [57, 58].
- addSwitch(), addHost(): adds a switch / host respectively, the developer can always use loops for more complex creations.

```
1 """Custom topology example
2 Two directly connected switches plus a host for each switch:
3   host --- switch --- switch --- host
4 Adding the 'topos' dict with a key/value pair to generate our newly
5 defined topology enables one to pass in '--topo=mytopo' from the
6 command line."""

7 from mininet.topo import Topo
8 class MyTopo( Topo ):
9     "Simple topology example."

10    def __init__( self ):
11        "Create custom topo."
12        # Initialize topology
13        Topo.__init__( self )

14        # Add hosts and switches
15        leftHost = self.addHost( 'h1' )
16        rightHost = self.addHost( 'h2' )
17        leftSwitch = self.addSwitch( 'sw' , dpid='6')
18        rightSwitch = self.addSwitch( 'se' , dpid='7')

19        # Add links
20        self.addLink( leftHost, leftSwitch )
21        self.addLink( leftSwitch, rightSwitch )
22        self.addLink( rightSwitch, rightHost )

23 topos = { 'mytopo': ( lambda: MyTopo() ) }
```

FIGURE 3.2: Representation of the file ∼/mininet/custom/topo-2sw-2host.py

- addLink(): adds a bidirectional link to a topology.

For more realistic simulations, we can add performance parameters to links and hosts: self.addHost('h1', cpu=f): This allows us to allocate a fraction of overall system CPU resources to the virtual host h1.

self.addLink(s1, s2, bw=10, delay='5ms', max_queue_size=1000, loss=1, use_htb=True): adds a bidirectional link between s1 and s2 with these characteristics :

- **bw** is bandwidth expressed in Mb/s.
- **delay** is expressed as a string with units in place (e.g. '5ms', '100us', '1s').
- **max_queue_size** is expressed in packets.
- **loss** is expressed as a percentage (between 0 and 100).
- **use_htb** is a boolean value for using the Hierarchical Token Bucket rate limiter [59].

3.3 ONOS

We already introduced ONOS and it's technology in the second chapter, now we get to work with ONOS, get a hands-on experience, and test some of its advertised features.

3.3.1 Setting up ONOS

In order to get ONOS up and running, the user must choose either :

1. To download the virtual image which is an Ubuntu image with ONOS already installed and ready to be tested. ONOS provides many tutorial images to choose from[1].
2. Or install it stand-alone on a Linux OS manually by installing git, downloading then extracting Karaf and Maven binaries, installing Oracle Java 8 , setting up the necessary environment variables, then finishing up by building ONOS either by using maven or buck, depending on the downloaded version.

3.3.2 Basics

Typing this command : `$ ok clean` into the terminal starts the ONOS instance, in Fig. 3.3 we see ONOS starting up and giving access to the ONOS CLI.[2]

After getting access to ONOS's CLI, there are many commands to use such as :

- `onos> summary` provides some basic information :
    1. node : the IP address of the accessed instance, in this case ONOS is running locally (127.0.0.1)
    2. version : the installed version of ONOS, in this case Falcon (1.5.2)
    3. nodes : the number of instances in the ONOS cluster, in this case only one instance of ONOS is running.
    4. devices, links, hosts, Strongly Connected Components (SCCs), flows and intents are not available since ONOS is not connected to any network yet.

[1]Sometimes with the tutorial images, some ONOS features may not work out of the box, and may need further environment setup. images can be found at wiki.onosproject.org/display/ONOS/Downloads

[2]It is important to mention that building and running ONOS changed after version 1.7.0 to using BUCK which enables faster building of ONOS and makes adding custom applications easier. In versions 1.7.0 and up, `$ ok clean` is replaced with `$ buck run onos-local -- clean debug`, and launching ONOS must be from the ONOS path (`~/onos`)

```
tarek@ONOS-machine:~$ ok clean
Creating local cluster configs for IP 127.0.0.1...
Staging builtin apps...
Customizing apps to be auto-activated: drivers,openflow,fwd,proxyarp,mobility...
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos>
```

FIGURE 3.3: ONOS instance start

- `onos> devices`, `onos> links` and `onos> hosts` show detailed information about the connected switches, links and hosts respectively.
- `onos> apps -s` lists applications along with their ID, version and description. Adding `-a` will show only active applications.
- `onos> app activate/deactivate/uninstall app-name` activate, deactivate or uninstal an application.
- `onos> device-remove URI` removes an infrastructure device identified by its Uniform Resource Identifier.
- `onos> paths src dst` displays the shortest path between two devices and displays the number of hops.
- `onos> wipe-out` clean the entire network information base.
- In case we use a cluster, these commands are the most useful :
  - `$ onos IP` this command allows access from remote machines to the target ONOS instance with the provided IP address.
  - `onos> nodes` displays all the instances of the cluster.
  - `onos> roles` lists the devices in the network with their corresponding current and standby masters.
  - `onos> masters` lists the instances in the network with the corresponding devices they control.

- `onos> balance-masters` forces device mastership rebalancing between the instances.
- `onos> device-role URI node master/standby` choose individual device mastership by providing the device URI and the node IP.

Now that ONOS is running we can access its web user interface. To do so we need the address of the ONOS instance we wish to access, the web UI can be accessed through the port 8181 followed by /onos/ui. The user will be prompted to enter the username and password, then allowed access directly to the topology view.
With this link: `localhost:8181/onos/ui/index.html#/topo` we get the web UI shown in Fig. 3.4.

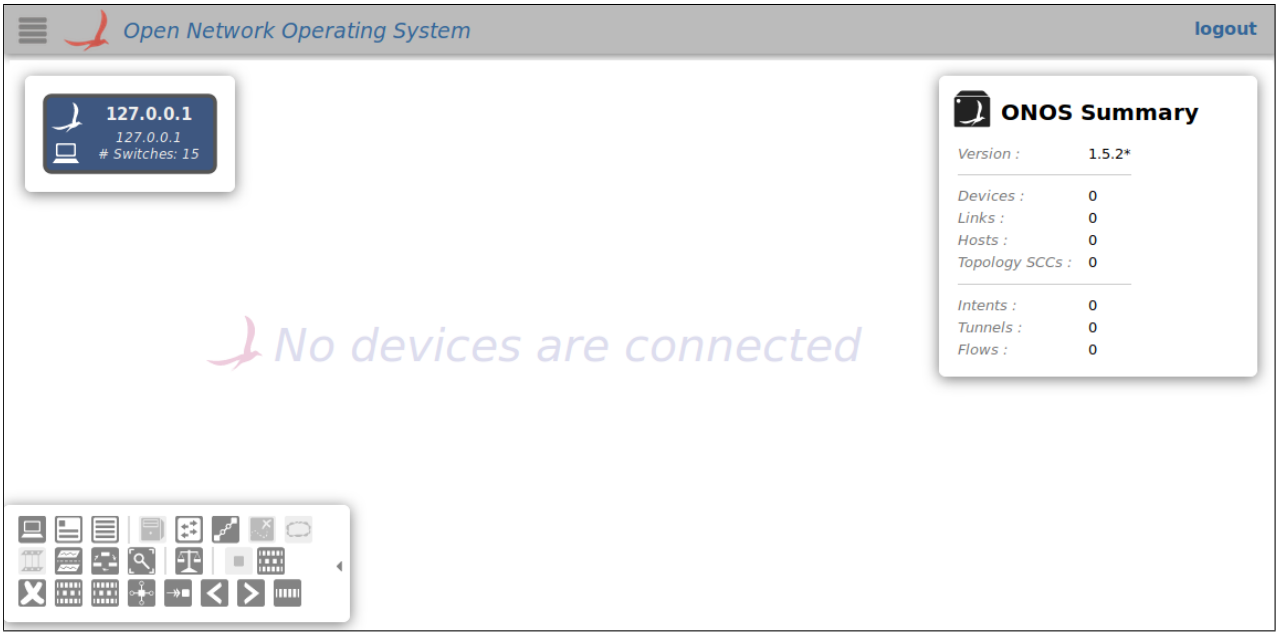

FIGURE 3.4: ONOS web UI

We notice "No devices are connected", again just as the CLI stated in the summary, there are no hosts, links, switches, etc., in the ONOS summary panel at the top right.

On the top-left side we see how many instances are in the ONOS cluster, their IP addresses and how many switches they control. While the lower-left panel provides buttons related to the topology view.

After starting both the CLI and web UI of ONOS, for the rest of this chapter, we will address both these aspects of ONOS - CLI and web UI - together.

3.3.3 ONOS with a topology

First we create a virtual network topology with mininet :
`$ sudo mn --custom tower.py --topo=tower --controller remote`

This command will create a virtual network from the file tower.py and connects to the local ONOS instance. The Fig. 3.5 shows the topology as viewed in ONOS's web UI.

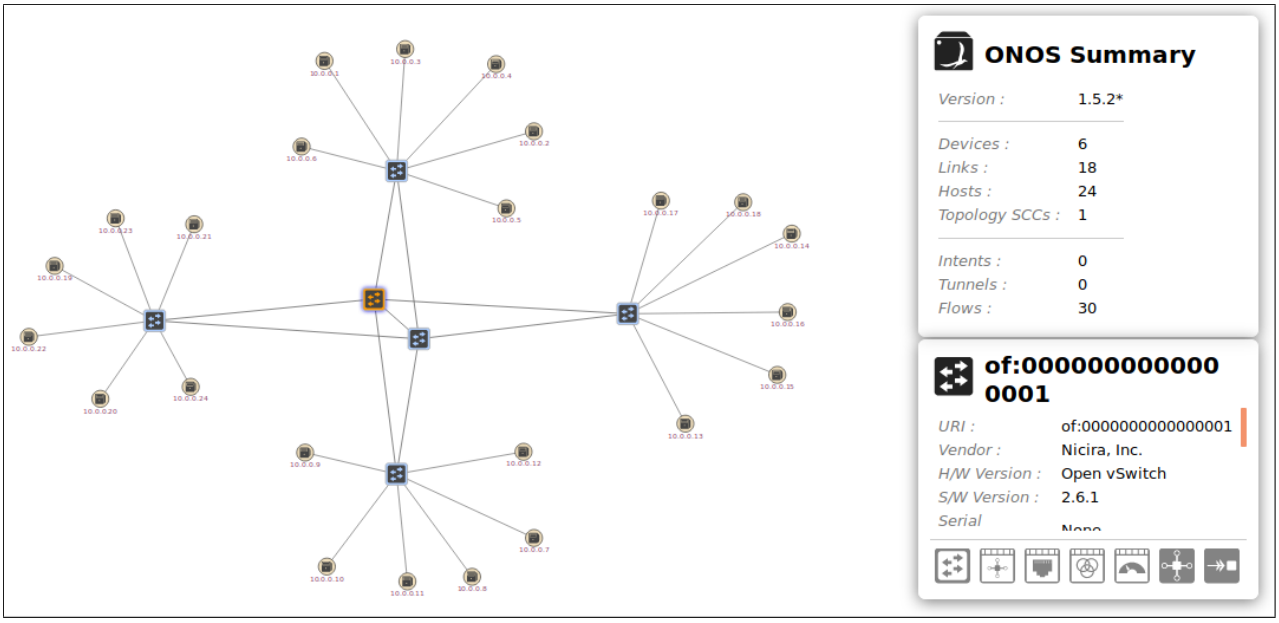

FIGURE 3.5: A topology run in ONOS

In the web UI, clicking on a switch, host or a link brings up a panel in the lower-right that provides detailed information about the selected item.

Now we can observe the summary panel, ONOS discovered six switches, eighteen links and twenty-four hosts. A host can not be discovered until it interacts with the network, to see all hosts in the topology we usually use `mininet> pingall`.

ONOS displays the number of links between the switches only. An important thing to remember is: each link is considered unidirectional, meaning two links are needed for each connection. Which explains why we see eighteen links instead of nine.

3.3.4 Other features

The web UI provides more than the topology view, it is the easiest way to interact with ONOS, most CLI commands and functions can also be done through the interface.

Most of the features are accessed through the dropdown menu (Fig. 3.6) located on the top-left corner.

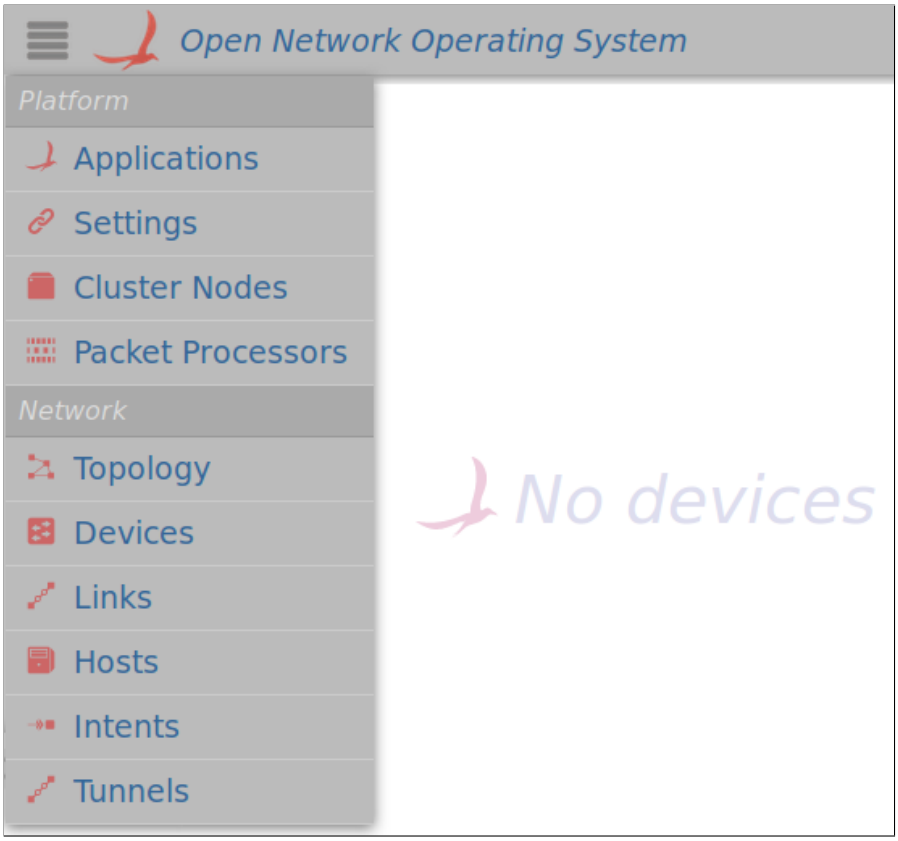

FIGURE 3.6: The navigation menu of ONOS

1. Applications : gives a table view on all applications in ONOS, and provides controls such as deactivating and uninstalling apps.
2. Settings : provides information about all configurable settings in the system.
3. Cluster Nodes : lists the cluster instances, can be done through the CLI with `onos> nodes`
4. Packet Processors : shows the components that participate in the processing of packets sent to the controller.
5. Topology : the default view when accessing the web UI. It gives an interactive visualization of the network topology like in Fig. 3.4 and Fig. 3.5.
6. Devices : the main view on all the network's devices, it is accessed to control and monitor the devices (Fig. 3.7).
7. Links : lists the links in the topology.
8. Hosts : lists the discovered hosts of the network.

9. Intents : lists all the installed intents, intents can be created from the CLI or the topology view.
10. Tunnels : lists the tunnels defined in the network.

After clicking on a device in the topology view we can get access to the devices view and see all devices of the network as in Fig. 3.7. Selecting a device provides more detailed information such as the device flows and ports.

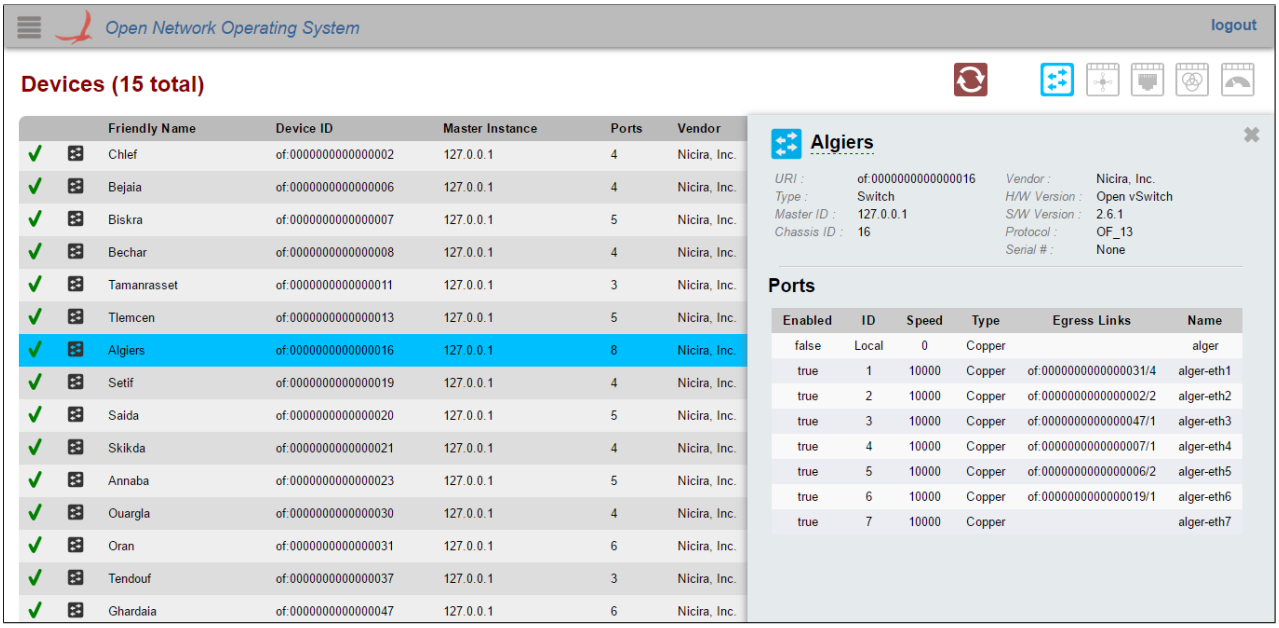

FIGURE 3.7: The device view

3.3.5 Distributed Clustering

Thanks to the distributed core architecture, ONOS supports running multiple controller instances in a clustered mode where they share state among each other and act together as a unified, coherent distributed system.

The underlying OpenFlow switches have L3 connectivity via TCP/IP to one or more ONOS instances, provided that only one instance will act as the master of the switch and others will be in standby mode in case the connection of the master fails. This is very useful for fault tolerance and high availability purposes.

Once ONOS is running on multiple independent machines, we can form a cluster. The cluster can be created from any of these machines.

In `~/onos/tools/package/bin` there is the file `onos-form-cluster` included with ONOS. The command `$ ./onos-form-cluster -u onos -p rocks IP1 IP2 IPn`

uses the file `onos-form-cluster` to create a cluster from the provided IP addresses that belong to the machines running ONOS.

If all addresses are reachable and ONOS is running, there should be no errors and the cluster will be formed :

```
./onos-form-cluster -u onos -p rocks  192.168.1.4  192.168.1.8  192.168.1.9
Forming cluster on 192.168.1.4...
Forming cluster on 192.168.1.8...
Forming cluster on 192.168.1.9...
```

After this, all ONOS instances will restart and come back up as a cluster. Accessing the web UI of any instance will provide the same information about the cluster.

For this example, we used mininet to create a custom topology with fifteen switches, each switch has one host connected to it. The output in Fig. 3.8 shows the creation of the virtual network in mininet.

```
$ sudo mn --custom ~/Desktop/mytopo.py --topo=mytopo --controller remote

*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.1.8:6653
*** Adding hosts:
h01 h11 h21 h31 h41 h51 h61 h71 h81 h91 h101 h111 h121 h131 h141
*** Adding switches:
alger annaba bechar bejaia biskra chlef ghardaia oran ouargla saida setif
skikda tamnrst tendouf tlemcen
*** Adding links:
(alger, biskra) (alger, chlef) (alger, setif) (annaba, setif) (bejaia, alger)
(bejaia, skikda) (biskra, annaba) (ghardaia, alger) (ghardaia, tamnrst)
(ghardaia, tlemcen) (h01, chlef) (h11, oran) (h21, alger) (h31, bejaia)
(h41, annaba) (h51, setif) (h61, bechar) (h71, ouargla) (h81, skikda)
(h91, ghardaia) (h101, tendouf) (h111, biskra) (h121, saida) (h131, tlemcen)
(h141, tamnrst) (oran, alger) (oran, chlef) (oran, tlemcen) (ouargla, biskra)
(ouargla, ghardaia) (saida, bechar) (saida, oran) (saida, tlemcen) (skikda,
annaba) (tendouf, bechar)

*** Configuring hosts
h01 h11 h21 h31 h41 h51 h61 h71 h81 h91 h101 h111 h121 h131 h141
*** Starting controller
c0
*** Starting 15 switches
alger annaba bechar bejaia biskra chlef ghardaia oran ouargla saida setif
skikda tamnrst tendouf tlemcen ...
*** Starting CLI:
mininet>
```

FIGURE 3.8: Output of mininet initializing the virtual network

The command :
`mininet> sh ovs-vsctl set-controller S tcp:IP1:port1 tcp:IP2:port2`
can be used to indicate the masters of the switch 'S', in our example, each switch can be controlled by all three instances.

For now, the switches are still controlled by one instance, we could use either the button in the topology view of ONOS, or the command `onos> balance-masters` to balance the mastership between instances.

It was already mentioned that accessing the web UI of any instance will provide almost the same information. We chose to access the web UI of `192.168.1.4` as seen in Fig. 3.9.

We can see the three instances with their IP address and number of devices they control. The lower-left white icon in the `192.168.1.4` instance indicates the current accessed instance.

After balancing the masters, we can see in Fig. 3.9 that every instance controls five switches, the switches are color coded to match their masters.

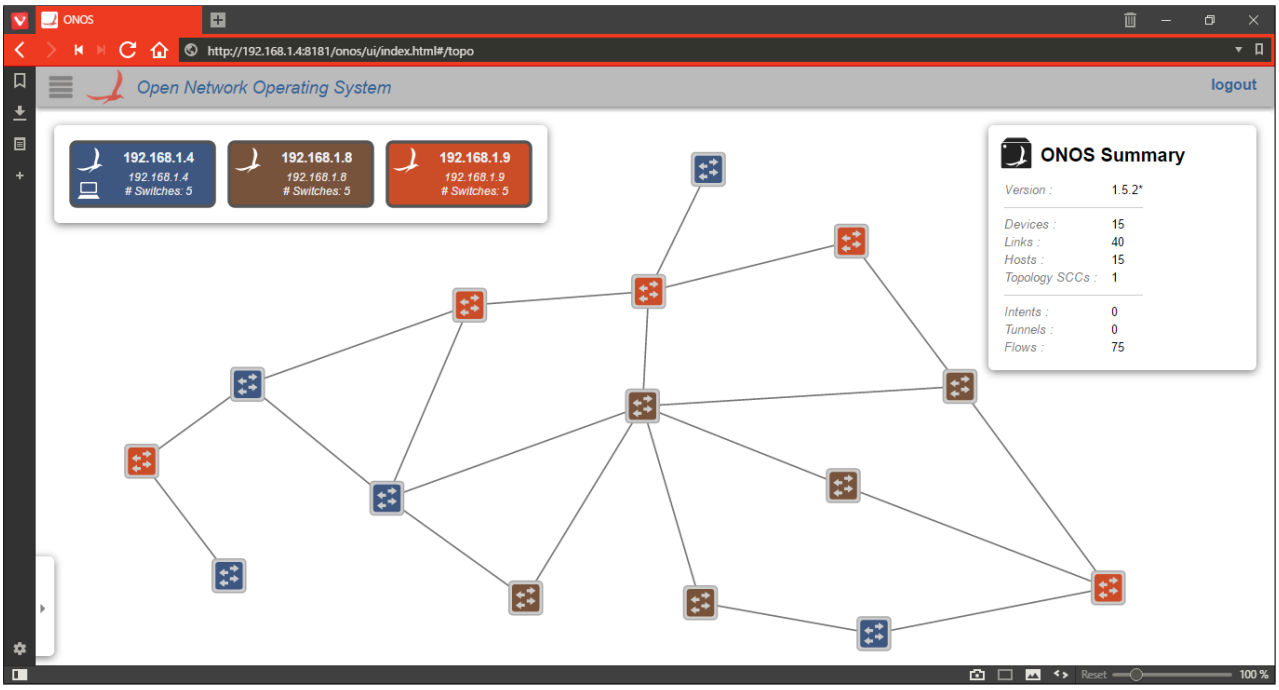

FIGURE 3.9: Network topology view with three node cluster

To get even more information about the masters, we used `onos> roles` which will show the current and the standby masters of each switch, as presented in Fig. 3.10.

```
onos> roles
of:0000000000000002: master=192.168.1.8, standbys=[ 192.168.1.9 192.168.1.4 ]
of:0000000000000006: master=192.168.1.8, standbys=[ 192.168.1.9 192.168.1.4 ]
of:0000000000000007: master=192.168.1.8, standbys=[ 192.168.1.4 192.168.1.9 ]
of:0000000000000008: master=192.168.1.9, standbys=[ 192.168.1.4 192.168.1.8 ]
of:0000000000000011: master=192.168.1.4, standbys=[ 192.168.1.8 192.168.1.9 ]
of:0000000000000013: master=192.168.1.9, standbys=[ 192.168.1.4 192.168.1.8 ]
of:0000000000000016: master=192.168.1.8, standbys=[ 192.168.1.9 192.168.1.4 ]
of:0000000000000019: master=192.168.1.8, standbys=[ 192.168.1.9 192.168.1.4 ]
of:0000000000000020: master=192.168.1.4, standbys=[ 192.168.1.8 192.168.1.9 ]
of:0000000000000021: master=192.168.1.4, standbys=[ 192.168.1.9 192.168.1.8 ]
of:0000000000000023: master=192.168.1.9, standbys=[ 192.168.1.4 192.168.1.8 ]
of:0000000000000030: master=192.168.1.9, standbys=[ 192.168.1.4 192.168.1.8 ]
of:0000000000000031: master=192.168.1.4, standbys=[ 192.168.1.8 192.168.1.9 ]
of:0000000000000037: master=192.168.1.4, standbys=[ 192.168.1.8 192.168.1.9 ]
of:0000000000000047: master=192.168.1.9, standbys=[ 192.168.1.4 192.168.1.8 ]
```

FIGURE 3.10: Output of the command onos> roles

Testing high availability

The main benefit of using the distributed cluster is fault tolerance. After starting a cluster of three nodes, we tried taking one instance down. The snippet below shows the new cluster status :

```
onos> nodes
id=192.168.1.4, address=192.168.1.5:9876, state=ACTIVE, updated=24m ago *
id=192.168.1.8, address=192.168.1.8:9876, state=ACTIVE, updated=13m ago
id=192.168.1.9, address=192.168.1.9:9876, state=INACTIVE, updated=just now
```

We see that the instance at `192.168.1.9` is now inactive. ONOS automatically assigns the switches controlled by `192.168.1.9` to the next standby master (`192.168.1.4` in this case). As a result `192.168.1.4` is now the master of ten devices and the network is still running (Fig. 3.11) thanks to the distributed core of ONOS.

3.4 How to contribute and get involved

ONOS is a controller that brings a lot of features to SDN. The most interesting thing about ONOS is that it is an open source project, open for everybody to contribute to.

Developing apps and writing code is not the only way to contribute to the ONOS project, there are many ways where contributors can help including :

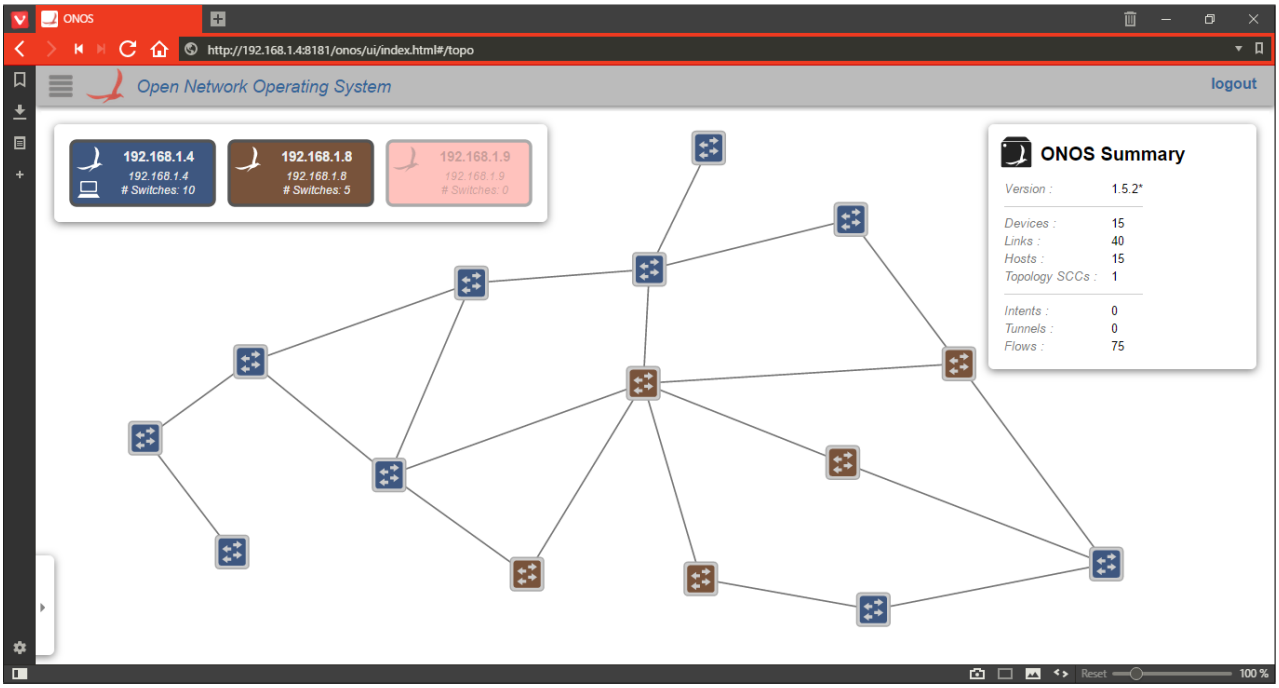

FIGURE 3.11: High availability test with one instance offline

- Development : from coding or developing new features and making applications to testing and fixing bugs, these all are possibilities to contribute in the development of the project [60].
- Deployment: contribute in the deployment of the project, by deploying ONOS in real networks, this helps ONOS to demonstrate how it performs in real use cases and prove its performance, scalability and high availability [61].
- Documentation: add some clarification or new definitions and fix what is unclear or false to improve the documentation of ONOS [62].
- User Interface: contribute in improving ONOS's user experience by customizing and extending the UI [63].
- Quality: check the integrity of the documentation and the quality of the product by using ONOS and its applications and testing their functionality. This contribution allows ONOS to become the best and most reliable controller [64].

There are other ways to contribute to ONOS, an account registration is required to access the contributor services.

ONOS uses Google Groups for its mailing lists that the contributors can access and use. There are several other mailing lists like ONOS-dev for ONOS developers and ONOS-discuss for general questions and discussions.

ONOS uses Slack for real-time conversations about ONOS. There is a general channel as well as channels dedicated to specific topics [65].

For example, if a developer has written a new application or a provider for a new protocol, and wishes to contribute in ONOS, he needs to approach the mailing lists with project ideas and proposals, so the ONOS foundation can discuss it.

Deploying applications

The most basic form to contribute to this project is through developing applications. To help out, we would like to present a simple way to add custom applications to ONOS, anyone who developed an application and wishes to add and test it in ONOS can follow the steps described in this section.

The command `$ onos-create-app` will create a minimal empty application, the developer will be prompted to provide some information about the app. The code snippet in Fig. 3.12 shows the test example we are creating.

```
$ onos-create-app

[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
   .
   .
   .
Define value for property 'groupId': info
Define value for property 'artifactId': myapp
Define value for property 'version' 1.0-SNAPSHOT: : 0.1
Define value for property 'package' info: : lagh
Confirm properties configuration:
groupId: info
artifactId: myapp
version: 0.1
package: lagh
 Y: : Y
   .
   .
   .
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

FIGURE 3.12: Snippet of the output after the command $ onos-create-app

Now that the project is generated, we should find a newly created folder with the name given to the 'artifactId' field, in our case the folder is 'myapp'. Inside we find the pom.xml file and the folder 'src', which contains the source files for the application.

We are only interested in deploying and installing the application, and to do so, we need to edit the pom.xml file. There are more properties for the developer to edit as shown in Fig. 3.13, we put our own information, next we should uncomment the section and save the changes.

```
<groupId>info</groupId>
<artifactId>myapp</artifactId>
<version>0.1</version>
<packaging>bundle</packaging>

<description>our custom dummy application</description>
<url>http://www.lagh-univ.dz</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <onos.version>1.5.2-SNAPSHOT</onos.version>

    <!-- Uncomment to generate ONOS app from this module.
    <onos.app.name>org.myapp.app</onos.app.name>
    <onos.app.title>MyApp</onos.app.title>
    <onos.app.origin>Lagh univ</onos.app.origin>
    <onos.app.category>default</onos.app.category>
    <onos.app.url>http://www.lagh-univ.dz</onos.app.url>
    <onos.app.readme>dummy application example</onos.app.readme>
    -->
</properties>
```

FIGURE 3.13: Snippet of the file pom.xml

Uncommenting the proprieties and running `$ mvn clean install` will create the ONOS App aRchive (.oar) file. Adding the `-DskipTests` argument will skip tests and reduce building time.

The .oar file is created inside `myapp/target`, to install the application into a running ONOS instance, we will use `$ onos-app 192.168.1.6 install! myapp-0.1.oar`

- The command will install the app from the provided .oar file.
- Using '!' after `install` will automatically activate the application after installation.
- The application can be installed on the local ONOS instance or a remote node, or even a cluster. Providing the IP address of only one node in the cluster will suffice, and the application will then be deployed across all instances.

We created the dummy application using ONOS 1.5 (2016 release), and we installed it in a remote machine running ONOS 1.9 (february 2017). The application installed and ran properly proving that compatibility is not an issue across ONOS versions.

After accessing the ONOS instance at `192.168.1.6` we listed the running applications (Fig. 3.14) to check if our app installed.

```
onos> apps -s -a
*  15 org.onosproject.optical-model       1.9.0   Optical information model
*  22 org.onosproject.drivers             1.9.0   Default device drivers
*  30 org.onosproject.proxyarp            1.9.0   Proxy ARP/NDP App
*  33 org.onosproject.hostprovider        1.9.0   Host Location Provider
*  34 org.onosproject.lldpprovider        1.9.0   LLDP Link Provider
*  35 org.onosproject.openflow-base       1.9.0   OpenFlow Provider
*  36 org.onosproject.openflow            1.9.0   OpenFlow Meta App
*  47 org.onosproject.mobility            1.9.0   Host Mobility App
*  53 org.onosproject.fwd                 1.9.0   Reactive Forwarding App
* 101 org.myapp.app                       0.1     our custom dummy application
```

FIGURE 3.14: Output of the command onos> apps -s -a

Just like other apps we can now deactivate, activate or uninstall our app. After this, developers can just keep improving their code and rebuilding their application to test it.

3.5 Economical benefits of SDN

The last point that we wanted to address is how different is an SDN infrastructure, and would it be beneficial from an economical perspective to choose building an SDN network over a traditional one.

The way we wanted to address this, is through a simple financial comparison between a traditional network and an SDN network. This will help us gather some data, present some statistics and extract results. Based on these results we can demonstrate the economic advantage of SDN.

3.5.1 Example

The basis of this test is an example of a company wishing to create and deploy a new network. The company is given a choice to either use legacy switches or OpenFlow switches, which means choosing between creating a traditional IP network, or an SDN network.
We will calculate the cost of hardware only, for each approach, and hopefully, we can observe the difference and decide which one is best.

To provide more thorough results, we decided to make a comprehensive test by trying out multiple use cases. We will present five sizes for each network type to get more statistics.

Only different aspects of the network are targeted in this example, the cost of cables, deployment, etc., will remain the same for both approaches. Eliminating the same components from our example will leave us only with switches, and the controller of the SDN network. These components are the only ones that will make a difference to the total cost.

For both network types, we will calculate the total cost of each of these five suggested topologies :

1. tiny network : 5 switches, 1 SDN controller.
2. small network : 10 switches, 1 SDN controller.
3. normal network : 100 switches, 2 SDN controllers.
4. big network : 500 switches, 7 SDN controllers.
5. large network : 1000 switches, 14 SDN controllers.

For the actual hardware, we chose what we believed are the best switches, the switches are almost identical in terms of characteristics :

- The switch for the SDN network is an OpenFlow switch : Pica8 P-3922 (48 port with 10GbE), price : almost 7,000.00 $ [66].
- SDN switches require SDN controllers, each instance of the controller is installed on a server : Dell PowerEdge T430 Tower Server, price : almost 1,500.00 $ [67].
- For the regular network we only need legacy switches, which are more expensive, but do not require a remote controller. the Switch is Cisco Nexus 9372PX-E (48 port with 10 GbE and includes Cisco NX-OS software), price : almost 18,000.00 $ [68].

3.5.2 Results

	Regular network	**SDN network**	**Gain**
Tiny network	90,000.00 $	36,500.00 $	53,500.00 $
Small network	180,000.00 $	71,500.00 $	108,500.00 $
Normal network	1,800,000.00 $	703,000.00 $	1,097,000.00 $
Big network	9,000,000.00 $	3,510,500.00 $	5,489,500.00 $
Large network	18,000,000.00 $	7,021,000.00 $	10,979,000.00 $

TABLE 3.1: Results

The total costs of each approach, and the gain of choosing SDN, are listed in Table 3.1. The costs are also represented in Fig. 3.15.

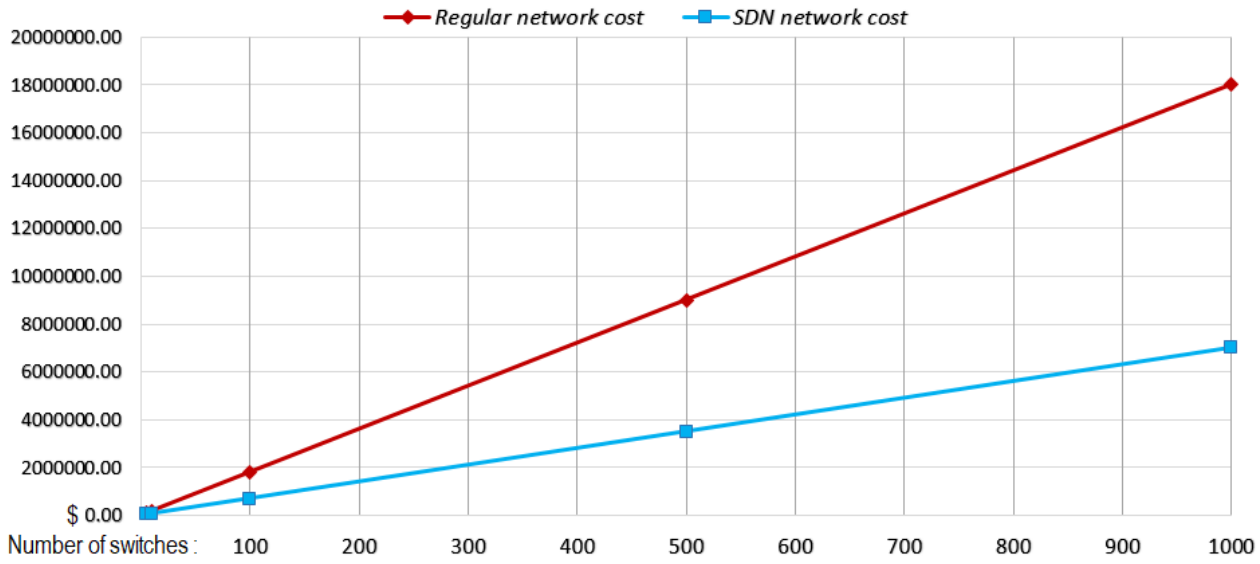

FIGURE 3.15: Graph representing the cost of different network scenarios

3.5.3 Analysis

As we notice in Fig. 3.15, deploying an SDN network is cheaper than traditional network, throughout the use cases, as the network grew larger, SDN remained cheaper by almost 60%.

With SDN, we can get standard and cheap switches rather than proprietary, costly ones from vendors like Cisco and Juniper. The company can save a lot of money by choosing the SDN network, it is clear how SDN reduces CapEx.

Even with the cost of servers for the controllers, SDN is still cheaper, but the controllers will provide even more benefits in the long run. Operating traditional networks sometimes mean changing the outdated hardware, and employing specialists in managing the proprietary equipment, meaning more expenses in the future. However, with SDN, orchestrating the network can be managed easily thanks to the centralized controller, thereby reducing OpEx.

In conclusion, this example shows just the initial benefits from choosing SDN, in the long run it would prove to be a better investment thanks to its scalability, fault tolerance and its open source nature.

3.6 Challenges and problems with SDN

Now that we worked with ONOS and SDN and saw the features and benefits that they offer, in this section, we wanted to address the current issues and obstacles faced by SDN. SDN has a few issues that it needs to overcome before it can be trusted and deployed.

Security is by far, the most important issue that we have to discuss, the problem is not with SDN or its controllers, but the fact that SDN architecture centralizes the control over the network, makes it vulnerable to potential unwanted takeover.
SDN offers many features and innovative concepts, but the centralized controller can be Achilles' heel in an SDN infrastructure. Therefore, most companies would not risk exposing their networks to this vulnerability.

The next obstacle for SDN is deployment, since it's a new architecture, service providers and network administrators can be skeptical, and prefer to keep their networks unchanged. Companies still need to be introduced to SDN, get familiar with it, and see how it improves upon their networks. Most importantly, they need to see proof of its performance, this will not be possible unless SDN became largely deployed and used. Only then will companies become more comfortable with upgrading their networks.

Next to deployment, training and education can also be an issue, there is a lack of SDN expertise. professionals are needed, since not everyone knows how to correctly deploy or use SDN and get the most out of it.

Serious financial issues also prevent the rise of SDN, service providers will not easily invest in a new solution, especially one that requires big finances, changing their infrastructure and replacing their hardware.

"SDN opens lots of questions for the future of advance networking and computing technologies. We need more responsive technology but not at the cost of security and control. In the long run SDN may be deployed in provider networks but individual corporations may find it just too much to deploy. SDN is in need of a lot more development and proof of being a secure and deployable technology" [69].

3.7 Discussion

We tried through the course of this thesis and especially this chapter, to provide a global overview on ONOS and SDN, and in this section we will share our opinions on the experience. Mininet was the perfect tool for our use case, it offered us many features to test the abilities of ONOS. Mininet deserves to be the standard simulation and testing tool for anyone working with SDN networks.

We had a good experience installing and running ONOS, although, it can be a long process due to the number of components in ONOS. But the experience became smoother after multiple tries, especially installing recent versions of ONOS.

The documentation in the ONOS wiki is very good, helpful and constantly updated. We discovered many wiki drafts on several occasions, which proves further that there are always new pages and documentations in progress.

ON.Lab also takes a lot of time to address any bugs or issues reported on their groups and videos. It is noticeable that ON.Lab is trying to ease the experience with ONOS and make it adoptable and deployable.

The distributed core is one of the most useful features in ONOS, we saw clearly its benefits, how it works and how easily deployable it is. Clustering and high availability are obviously built from the ground-up in ONOS, and it proved to be very reliable and scalable.

The ONOS CLI is a powerful administrative tool, it is also a relatively easy CLI to learn and familiarize with. The CLI provides the manual of any command if followed by `--help`, another useful tip is using 'tab' to show suggestions from a partially written command.

The web UI proved to be useful and simple, basically anyone can start using it without any prior knowledge. The web UI provides an extremely easy way to interact with ONOS, it offers most of the needed functions of the CLI while eliminating the complexity.

The ONOS core and GUI are light-weight, which makes ONOS faster to build and run. This is because throughout the development cycles of ONOS, ON.Lab tried to keep the number of modules included to a minimum.

Next to being open source, ONOS is also modular, the modules can be removed, modified or developers can create and add their own. Which is why we encourage developers and colleagues to get involved in SDN and ONOS in particular.

Cost reduction and financial benefits has been the major driver behind the move toward SDN. That is why we dedicated a section that proved how SDN reduces the cost of network equipments, and the total cost of creating and managing the network.

SDN has a lot of advantages including financial benefits, as mentioned earlier. But of course, no technology is flawless, SDN faces some issues as well, ONOS did eliminate some obstacles, and a few remained that need to be addressed in order for SDN to be deployed worldwide.

3.8 Conclusion

In this chapter, we talked about mininet, and how to use it to create custom virtual networks that are needed to test SDN controllers. We also presented ONOS, with both its CLI, and web UI, and how to use them to interact with ONOS.

We also introduced the different ways that are available to contribute to the ONOS project, with a focus on applications and how to deploy them.

Economics is the main power moving our world, that is why we made a simple example to show how SDN is performing in this domain. Lastly we listed some major obstacles that SDN is facing, that must be addressed in the future.

Thanks to the standard architecture of SDN, all controllers will feel the same to an extent. Getting familiar with any SDN controller will grant us enough experience to quickly adapt to others.
We chose ONOS, we shared our work and experience with it, and presented and discussed our thoughts, with the hope that it should be enough to get the reader started and invested in SDN.

General conclusion and future work

SDN is the new path that can bring a revolution in networking architectures and make networks better for everyone. We hope to see this architecture profoundly developed to provide new functions and innovations in the near future.

Thanks to SDN, networks can use white box solutions and be open, programmable, customizable, and easy to virtualize and orchestrate. This makes the providers' jobs easier, therefore allowing them to focus more on improving services offered to the consumer.

This new revolution targets both software and hardware aspects of the network while keeping them undependable of each other. SDN forces hardware vendors to respect the standardized protocols, which will help introducing simpler and cheaper hardware.

As we mentioned in the second chapter, and proved in the third, ONOS is an SDN controller that takes full advantage of the SDN architecture, improving it, adding more features to it, and taking it to the next level. We saw the true power and full potential of SDN through ONOS, fault tolerance, and scalability are just a few benefits of choosing this controller. But contribution is one of its most interesting features, it is what made ONOS rise above other big controllers in such a short period of time, despite being the newest one.

Unfortunately, we didn't have enough time to contribute and be a part of this project, instead, we aimed to introduce this new revolution and to draw the first guidelines for future developments. the ONOS community is always open to ideas and contributions, therefore we encourage our colleagues to suggest and develop some applications of their own to contribute to ONOS in the future.

Although we might not have covered everything, we tried to provide all the basic, necessary and most needed information related to this topic. We hope this thesis encourages and helps anyone interested, to get to know and pursue this new innovation further.

References

[1] SDxCentral. Why sdn or nfv now? https://www.sdxcentral.com/sdn/definitions/why-sdn-software-defined-networking-or-nfv-network-functions-virtualization-now/.

[2] Ivan Pepelnjak. Control and data plane. http://wiki.nil.com/Control_and_Data_plane.

[3] Jim Metzler; Network World. 2016. Survey shows growing interest in sdn. http://www.networkworld.com/article/3019863/software-defined-networking/survey-shows-growing-interest-in-sdn-where-and-how-companies-might-deploy-the-tech.html.

[4] Prayson Pate; SDxCentral. Nfv and sdn: What's the difference? https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/.

[5] Open Networking Foundation. About the open networking foundation. https://www.opennetworking.org.

[6] Open Networking Foundation. Software-defined networking (sdn) definition. https://www.opennetworking.org/sdn-resources/sdn-definition.

[7] SDxCentral. Understanding the sdn architecture. https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/.

[8] Open Networking Foundation. About the openflow. https://www.sdxcentral.com/sdn/definitions/what-is-openflow/.

[9] Flowgrammable. Openflow switch. http://flowgrammable.org/sdn/openflow.

[10] Open Networking Foundation. Openflow switch specification version 1.3.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf.

[11] Margaret Rouse; SearchSDN. Open vswitch database management protocol. http://searchsdn.techtarget.com/definition/OVSDB-Open-vSwitch-Database-Management-Protocol.

[12] R. Enns; M. Bjorklund; J. Schoenwaelder; A. Bierman; [RFC 6241]. Network configuration protocol (netconf). https://tools.ietf.org/html/rfc6241.

[13] wikipedia March 2017. Netconf configuration protocol. https://en.wikipedia.org/wiki/NETCONF.

[14] A. Atlas; J. Halpern; S. Hares; D. Ward; T. Nadeau; [RFC 7921]. An architecture for the interface to the routing system. https://tools.ietf.org/html/rfc7921.

[15] wikipedia September 2015. About the nox platform. https://en.wikipedia.org/wiki/Nox_(platform).

[16] Teemu Koponen; Martin Casado; Natasha Gude; Jeremy Stribling. Onix: A distributed control platform for large-scale production networks. http://yuba.stanford.edu/~casado/onix-osdi.pdf.

[17] Margaret Rouse; SearchSDN. About pox. http://searchsdn.techtarget.com/definition/POX.

[18] OpenMUL. About open mul. http://www.openmul.org/.

[19] SDxCentral. About ryu. https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/what-is-ryu-controller/.

[20] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.

[21] SDxCentral. About floodlight. https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/what-is-floodlight-controller/.

[22] Opendaylight: Open source sdn platform. https://wiki.opendaylight.org/view/Release_Plan and https://www.opendaylight.org/.

[23] Brent Salisbury; Network Computing. Inside google's software-defined network. http://www.networkcomputing.com/networking/inside-googles-software-defined-network/512240144.

[24] Jay Turner; InfoWorld. The future of sdn january 19 2017. http://www.infoworld.com/article/3159067/networking/the-future-of-sdn.html.

[25] Dotan Horovits. Programmable networks - july 17 2016. https://www.linkedin.com/pulse/programmable-networks-dream-finally-coming-true-dotan-horovits.

[26] Dhananjay Potle; Allied Market Research. Software defined networking (sdn) market 2016-2022 - june 28 2016. http://www.prnewswire.com/news-releases/software-defined-networking-sdn-market-is-expected-to-reach-1329-billion-by-2022-584680731.html.

[27] Open Networking Foundation. Open networking foundation and on.lab to merge to accelerate adoption of sdn press release - october 19 2016. https://www.opennetworking.org/news-and-events/press-releases/3194-open-networking-foundation-and-on-lab-to-merge-to-accelerate-adoption-of-sdn.

[28] ONOSProject. About open network operating system. http://onosproject.org/.

[29] Open networking lab linkedin overview. https://www.linkedin.com/company-beta/3232124/?pathWildcard=3232124.

[30] ON.LAB. About cord project. http://opencord.org/.

[31] ON.LAB. About ovx project. http://ovx.onlab.us/.

[32] ON.LAB. About xos project. http://xosproject.org/.

[33] About mininet. http://mininet.org/overview/.

[34] On.lab delivers software for new open source sdn network os. http://www.prnewswire.com/news-releases/onlab-delivers-software-for-new-open-source-sdn-network-operating-system--onos-300004797.html.

[35] Luca Prete; Uyen Chau; May 04 2017. Onos release model. https://wiki.onosproject.org/display/ONOS/Release+Model#ReleaseModel-ReleaseNaming.

[36] January 2017 board meeting, onos and cord community update. http://onosproject.org/2017/01/11/onos-and-cord-community-update-january-2017-board-meeting/.

[37] Ayaka Koshibe; You Wang. System components and architecture of onos. https://wiki.onosproject.org/display/ONOS/System+Components.

[38] ON.Lab. Introducing onos - a sdn network operating system for service providers. http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf.

[39] Ali Al-Shabibi. Onos architecture overview. In *OpenDaylight Meetup*. https://www.slideshare.net/opendaylight/onos-platform-architecture.

[40] D. Dimitrova. Onos overview - architecture, abstractions & application. In *SDN Seminar*, Zurich University March 2017. https://www.systems.ethz.ch/sites/default/files/slides_onos.pdf.

[41] Yi Tseng. Onos intent framework dec-29-2016. Taiwan, NCTU. https://www.slideshare.net/YiTseng/onos-intent-introduction.

[42] Ayaka Koshibe; Jonathan Hart. Onos's intent framework. https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[43] java 8. http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html.

[44] Apache karaf. https://fr.wikipedia.org/wiki/Apache_Karaf.

[45] About raft. https://raft.github.io/.

[46] Madan Jampani. Raft for onos. https://wiki.onosproject.org/display/ONOS/Distributed+Primitives.

[47] Angularjs. https://en.wikipedia.org/wiki/AngularJS.

[48] Kryo. https://github.com/EsotericSoftware/kryo.

[49] Ayaka Koshibe; Elena Olkhovskaya; Nov 18 2016. Atomix for onos cluster management. https://wiki.onosproject.org/display/ONOS/Cluster+Coordination#ClusterCoordination-ClusterManagement.

[50] Copycat, novel implementation of the raft consensus algorithm for java 8. http://atomix.io/copycat/.

[51] About jersey. https://github.com/jersey/jersey.

[52] Guava-libraries. https://en.wikipedia.org/wiki/Google_Guava.

[53] Sdn-ip. https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture.

[54] Sdn ip demo using onos avocet video. https://www.youtube.com/watch?v=U77Md2ghIRY.

[55] Mininet's web page. http://mininet.org/.

[56] Introduction to mininet. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

[57] Mininet class list - mininet python api reference manual. http://mininet.org/api/annotated.html.

[58] Mininet class hierarchy - mininet python api reference manual. http://mininet.org/api/hierarchy.html.

[59] Felipe Estrada-Solano; Princeton University. Using mininet and mininet python api. https://github.com/PrincetonUniversity/Coursera-SDN/tree/master/assignments/mininet-topology.

[60] Ayaka Koshibe; David Boswell. Contributing to the onos codebase. https://wiki.onosproject.org/display/ONOS/Contributing+to+the+ONOS+Codebase.

[61] Luca Prete. Contribute with deployments. https://wiki.onosproject.org/display/ONOS/Deployments.

[62] Ayaka Koshibe. Contributing to onos documentation. https://wiki.onosproject.org/display/ONOS/Contributing+to+ONOS+Documentation.

[63] Simon Hunt. Web ui tutorials. https://wiki.onosproject.org/display/ONOS/Web+UI+Tutorials.

[64] How to contribute to system test apr 20, 2016. https://wiki.onosproject.org/display/ONOS/How+to+Contribute+to+System+Test.

[65] Ayaka Koshibe; David Boswell. Onos's guide to contribution. https://wiki.onosproject.org/display/ONOS/A+Beginner%27s+Guide+to+Contribution.

[66] Pica8 p-3922 openflow switch. http://www.colfaxdirect.com/store/pc/viewPrd.asp?idproduct=1813&idcategory=7.

[67] Dell poweredge t430 tower server. http://www.dell.com/en-us/work/shop/povw/poweredge-t430.

[68] Cisco nexus 9372px-e switch. https://www.compufirst.com/cisco-nexus-9372pxe-commutateur-48-ports-gere-montable-sur-rack/fiche_prod.do?prodId=1512738#prod-spec.

[69] Tim O'Neill. What is the future of sdn? https://www.garlandtechnology.com/blog/what-is-the-future-of-sdn.