UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

ÉCOLE DOCTORALE SCIENCES TECHNOLOGIE SANTE (547)

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Discipline : ***INFORMATIQUE***

Spécialité : Réseaux et Télécommunication

Présentée et soutenue publiquement par

CARLOS JAVIER GONZALEZ

Le 19 décembre 2017

Gestion d'une architecture hétérogène distribuée à l'aide du SDN

Thèse dirigée par **OLIVIER FLAUZAC ET FLORENT NOLOT**

JURY

M. André-Luc BEYLOT,	Professeur,	INPT/ENSEEIHT,	**Président**
M. Florent NOLOT**,**	Maître de Conférences HDR,	Université de Reims Champagne Ardenne,	**Directeur de thèse**
M. Olivier FLAUZAC,	Professeur,	Université de Reims Champagne Ardenne,	**Co-Directeur de thèse**
Mme Nathalie MITTON,	Directrice de Recherche,	INRIA,	**Rapporteur**
M. Isaac WOUNGANG,	Professeur,	Ryerson University, Toronto, Canada,	**Rapporteur**
Mme Leila MERGHEM,	Maître de Conférences HDR,	Université de Technologie de Troyes,	**Examinateur**

“À ma famille qui a toujours été là pour moi"

“To my family, who have always been my strength and inspiration”

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisors Dr. Florent Nolot and Dr. Flauzac Olivier who gave me the opportunity to improve my work through their advices and ideas.

I am also thankful to my colleagues at the CReSTIC Lab, for their support with the simulation developments, suggestion on this manuscript and for the nice time I had during the three years of PhD.

Next, I would like to thank the Secretaria Nacional de Ciencia y Tecnologia (SENACYT) Panama, for funding this research project. Likewise, I would like to thank Société Française d'Exportation des Ressources Éducatives (SFERE) for continuously funding my travels to the Research Conference meetings.

Last but not least, I dedicate this thesis to my family and all of those who supported me. They always give constant love and encouragement during all my studies.

Contents

Table des matières

List of Tables

List of Figures

Introduction

De nos jours, avec la prolifération des dispositifs connectés directement à Internet, la complexité de gérer des réseaux à grande échelle augmente. La plus importante de ces nouvelles technologies, l'Internet des objets ou *Internet of Things* (IoT), apparaît comme un nouveau paradigme du numérique et de la communication où les « objets » sont programmés pour surveiller et contrôler de façon autonome plusieurs aspects de nos vies quotidiennes, et ce dans presque tous les secteurs économiques tels que le transport, les smart cities, l'environnement, l'enseignement, la santé et tant d'autres domaines. Les objets dit intelligents incluent des capteurs domotiques, des dispositifs médicaux, des voitures, des avions, des réacteurs nucléaires, des appareils ménagers pour ne nommer que ceux-là. Il est attendu à ce qu'ils soient embarqués avec une sorte de capacités intelligentes. Ces dispositifs génèrent de grands volumes de données, une forte demande de gestion du flux de ces données et une sécurité à maintenir en permanence. Du fait de son développement, l'hétérogénéité de ce type de réseau devient très difficile à gérer.

Une solution pour surmonter ce défi est fournie par une autre technologie émergente nommé Réseau Défini par Logiciel, *Software Defined Networking* en anglais (SDN). Dans l'architecture du SDN, la fonction de contrôle du réseau n'est pas incluse dans le plan de transfert de données. En effet, les fonctions de contrôle du réseau sont centralisées dans un ou plusieurs contrôleurs. Ce paradigme fait passer le réseau traditionnel bloqué par un fournisseur à un réseau indépendant, offrant ainsi un réseau programmable et personnalisable. Les avantages de contrôle du réseau offerts par le SDN incluent :

- **Expérimentations basées sur les simulations:** Les utilisateurs et chercheurs sont maintenant en mesure d'utiliser et de tester des protocoles expérimentaux dans le réseau de production en utilisant des données utilisateur réelles sans diminuer les performances des services réseau voire de les interrompre. En tant que tel, il est possible de créer un réseau superposé sur le réseau de production;

- **Automatisation:** Cette fonctionnalité fournit une interface de programmation applicative (API) communément utilisée pour l'installation des fonctions du réseau. En outre, le SDN permet l'abstraction de spécifications de couches inférieures du réseau ainsi que l'orchestration de systèmes et d'applications;

- **Développer des applications:** SDN offre la possibilité de développer des nouvelles fonctionnalités et des applications réseau sans avoir besoin de configurer individuellement les périphériques ou d'attendre que les fabricants de périphériques réseau les développent;

- **Réactivité:** Le SDN permet une adaptation dynamique aux changements de topologie et aux pannes des équipements périphériques réseau. En effet, le réseau peut pren-

dre des décisions sans demander certains changements ou configurations effectués par l'administrateur du réseau;

- **Interopérabilité et indépendance**: La gestion et le contrôle des périphériques réseau sont indépendants des fabricants. Avec le SDN, il est possible d'utiliser des équipements auprès de plusieurs fournisseurs sans avoir besoin de reconfigurer des périphériques individuels ou d'attendre les dernières mises à jour des fabricants de équipements réseaux;
- **Réduction des coûts**: Le SDN sépare le plan de contrôle réseau du plan de transfert de paquets, cette opération réduit le prix du matériel réseau tout en poussant les fournisseurs à se concurrencer suivant les performances matérielles du réseau.

Motivations et Objectifs

La communication à travers le réseau est devenue indispensable dans tout type d'organisation ou infrastructure. Actuellement, avec la croissance accélérée des dispositifs numériques et des technologies de communication, gérer l'infrastructure réseau devient plus complexe. Cisco estime qu'aujourd'hui, il y a plus de 25 milliards de périphériques connectés à Internet. Il est prévu qu'il y ai environ 50 milliards de périphériques connectés d'ici 2020 [11]. Avec cette croissance, l'innovation est absolument nécessaire pour optimiser la gestion du réseau d'une manière plus efficace et flexible. La technologie qui devrait atteindre cet objectif, le SDN, attire considérablement l'attention de l'Industrie et de la Recherche.

L'objectif principal de cette étude est de déployer et d'analyser l'architecture du SDN à des niveaux élevés d'évolutivité et permettant la gestion de grands volumes de flux de données. Les nouveaux scénarios émergents de connectivité tels que les appareils *Machine-to-Machine* (M2M), *Device-to-Device* (D2D), *Vehicle-to-Vehicle* (V2V), *Wireless Sensor Network* (WSN), les smartphones et les périphériques embarqués ont besoin d'un système hétérogène pour interconnecter différents objets physiques. Pour résoudre ce problème, le nouveau paradigme de réseau, le SDN offre un mécanisme efficace avec un contrôle centralisé et une vue globale de l'ensemble des équipements du réseau, lequel facilite ses optimisations et sa configuration. Cette étude analyse la mise en place d'un plan de contrôle distribué sur l'architecture du SDN.

Challenges

Les équipements de réseaux traditionnels disposent de systèmes de gestion propriétaire sur le matériel et les logiciels, et les fonctions de contrôle sont codées de manière complexe. Cela rend difficile la conception, le déploiement, et la gestion évolutive du réseau. En pratique, la configuration des paramètres de fonctionnement sur chaque périphérique réseau est effectuée manuellement par un administrateur réseau, à travers l'interface de ligne de commande (CLI), ce qui limite l'innovation et un management flexible. Au cours des dernières décennies, le nombre de dispositifs qui incluent des caractéristiques hétérogènes s'étend rapidement, augmentant la complexité du déploiement et de l'exploitation du réseau et constituant un véritable défi. Pour surmonter ces limitations dans les réseaux traditionnels, un nouveau paradigme sur la gestion du réseau, comme le SDN, permet la manipulation des périphériques réseau via des API, éliminant la dépendance à l'égard des systèmes matériels et logiciels propriétaires.

Au commencement de ce projet de recherche, nous avons rencontré des difficultés techniques liées à l'élaboration de notre propre approche. En effet, la conception d'une telle plateforme demande des ressources assez conséquentes pour son déploiement. Or, les informations techniques sur la façon d'implémenter des systèmes SDN adaptés aux objectifs de notre étude sont difficilement accessibles. Ainsi, la décision fut prise de construire une plateforme d'expérimentation exclusivement dédiée et appropriée à l'évaluation de notre approche. Avec cette plateforme de type testbed développée, il est possible de créer des routes entre les points finaux, de surveiller les requêtes envoyées au contrôleur SDN et de décider des politiques de gestion pour chaque périphérique connecté au réseau, y compris les systèmes à grande échelle et en cluster.

Contributions

La principale contribution de notre projet de recherche porte sur la réalisation d'une architecture hétérogène pour les réseaux ad-hoc et l'IoT, basée sur le nouveau paradigme de réseau, le SDN. La première partie de cette dissertation consiste en une étude théorique approfondie sur des technologies SDN et OpenFlow. Par la suite, l'avantage de cette nouvelle architecture est adapté pour proposer une manière plus flexible de programmer et de gérer des réseaux avec ou sans infrastructure. L'architecture SDN proposée est abordée dans le contexte des réseaux ad-hoc et IoT, y compris l'intégration de périphériques avec une connectivité hétérogène. Dans cette architecture, le contrôleur représente un potentiel risque de noeud critique en raison de l'intelligence centralisée où tous les périphériques réseau dépendent directement d'un seul noeud, ce qui implique une faiblesse du système. Par conséquent, dans le cadre de ce projet, nous avons étudié la mise en place de contrôleurs distribués dans une architecture en grappe de serveurs afin de réduire la latence et augmenter la disponibilité, l'évolutivité et la tolérance de panne dans le déploiement du SDN.

Nous avons développé notre propre plateforme d'essais de systèmes pour permettre la simulation d'un réseau à grande échelle en utilisant des techniques de virtualisation, comme le SDN, le protocole OF et de nombreux autres outils nécessaires pour le développement de ce projet. L'autre partie du projet vise au développement d'une architecture SDN distribuée pour effectuer la gestion du réseau à grande échelle en utilisant le protocole OF, ce qui permet l'évaluation de performance du système dans le cadre de différents environnements de développement. Nous avons examiné les vulnérabilités de l'OF et du SDN, permettant notamment de proposer des mesures et certains mécanismes afin d'éviter ces menaces de sécurité.

Plan de la Thèse

Cette thèse est composée principalement de deux parties. La première partie consiste en une étude approfondie du domaine de la recherche. Cette partie est composée de trois chapitre et organisée de la manière suivante : Le chapitre 1 décrit le développement du réseau défini par logiciel et le protocole OpenFlow. Le chapitre 2 montre les différents outils d'utilisation du SDN. Le chapitre 3 est consacré à la présentation des technologies Internet des Objets et étudie les approches du SDN proposés pour l'IoT. La deuxième partie du manuscrit présente notre contribution, ainsi qu'introduise les différentes architectures proposées. Cette partie est composée de trois chapitre. Le chapitre 4 présente la solution du SDN proposée pour les réseaux

ad-hoc. Au début, nous présentons les protocoles de routage ad-hoc plus utilisés et décrivons notre approche pour contrôler et gérer les réseaux ad-hoc et IoT basés sur le SDN. Le chapitre 5 explique l'architecture SDN distribuée afin de fournir la disponibilité et l'évolutivité du plan de contrôle à l'aide du contrôleur OpenDayLight (ODL). Le chapitre 6 présente les détails de mise en place de notre approche et montre les résultats d'évaluation de notre plateforme d'essais développée. En outre, nous pressentons quelques descriptions des vulnérabilités et nous proposons des solutions et mécanismes de sécurité pour améliorer la protection de l'architecture du SDN. Enfin, le chapitre 7 résume le contenu de la thèse et donne un aperçu des orientations des futurs travaux de recherche.

Liste de Publication

- O. Flauzac, **C. González**, A. Hachani and F. Nolot, **SDN based architecture for IoT and improvement of the security** [12]. In: The IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, 2015, Gwangju, Korea.
- O. Flauzac, **C. González** and F. Nolot, **New Security Architecture for IoT Network** [13]. In: Procedia Computer Science, The International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems (BigD2M 2015), London, United Kingdom.
- O. Flauzac, **C. González** and F. Nolot, **SDN Based Architecture for Clustered WSN** [14]. In: The IEEE 4th International Workshop on Extending Seamlessly to the Internet of Things (esIoT) 2015, San Catarina, Brasil.
- O. Flauzac, **C. González** and F. Nolot, **Original secure architecture for IoT based on SDN** [15]. In: The IEEE 16eme Colloque Francophone sur l'Ingenierie des Protocoles et Conférence sur les Nouvelles Technologies de la Répartition (CFIP & NOTERE) 2015, Paris, France.
- O. Flauzac, **C. González** and F. Nolot, **Developing a distributed software defined networking testbed for IoT** [16]. In: Procedia Computer Science, The 7th International Conference on Ambient Systems, Networks and Technologies (ANT) 2016, Madrid, Spain.
- **C. González**, O. Flauzac, F. Nolot and A. Jara, **A Novel Distributed SDN-Secured Architecture for the IoT** [17]. In: The IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS) 2016, Washington DC, U.S.
- **C. González**, S. M. Charfadine, O. Flauzac, and F. Nolot, **SDN-based security framework for the IoT in distributed grid** [18]. In: The IEEE International Multidisciplinary Conference on Computer and Energy Science (SPLITECH) 2016, Split, Croatia.
- O. Flauzac, **C. González** and F. Nolot, **Software-Defined Clustered Sensor Networks approach to secure IoT networks** [19]. In: ACM, The second International Conference on Internet of Things, Data and Cloud Computing (ICC 2017), United Kingdom

Introduction

Nowadays, with the proliferation of devices that are able to directly connect to the Internet, the complexity to manage large-scale networks is increasing. One of those technologies is the Internet of Things (IoT) which has emerged as a new computing and communication paradigm, where "things" are expected to autonomously monitor and control several aspects of our daily lives in almost all sectors such as transportation, community, home, environment and education, to name a few. The so-called "things" include simple home sensors, medical devices, cars, airplanes, nuclear reactors, smart fridges, just to name a few. They are expected to be embedded with some forms of intelligent capabilities. These devices generate highly amounts of data, heavy flow management demands and multiple security threats. This will make the network progressively difficult to handle.

An alternative solution to overcome this challenge is provided by another emerging technology called the Software Defined Network (SDN). In SDN architecture, the network control function is separated from the data forwarding plane. The network control functions are centralized in one or several controllers. This paradigm shift from traditional vendor-locked networking to vendor-independent networking, offers a programmable network. The network control benefits offered by SDN include:

- **Enabling network experimentation**: Users and researchers are now able to run and test experimental protocols in the production network using real user data without impacting the performance of or interrupting network services. As such, one can create an overlay network on top of the production network;
- **Automation**: This feature provides a common Application Programming Interface (API) needed for implementing the network functions. Furthermore, SDN allows for the abstraction of lower network layer specifications as well as the orchestration of systems and applications;
- **On demand features**: SDN offers the ability to develop new features and network services applications without the need to configure devices individually or to wait until the network device manufacturers develop them;
- **Reactivity**: SDN enables dynamic adaptation to topology changes and network device failures. Indeed, the network can make some decisions without involving some changes or configurations made by the network administrator;
- **Interoperability and Independence**: The management and control of the network devices are independent of the device manufacturers. With SDN, it is possible to use

equipment from multi-vendors without the need to re-configure individual devices or to wait for upcoming vendors' releases;

- **Cost Savings**: As SDN decouples the network control from the packet forwarding, it reduces the hardware prices while pushing the vendors to compete on hardware performance features.

Motivations and objectives

Network communication has become an essential component of any organization or infrastructure. Currently, with the accelerated growth of computing devices and communication technologies, it becomes more complex to handle the network infrastructure. Cisco estimates that today there are more than 25 billion devices connected to the Internet and looking at the future there will be 50 billion of devices by 2020 [11]. It is obvious that innovation is needed to optimize the network management in a more efficient and flexible manner. The most attractive technology for achieving this purpose is the SDN, which is gaining considerably attention of industries and the research community.

The principal goal of this study is to deploy and analyze the SDN architecture performing at high levels of scalability and handling high volumes of traffic data. The newly emerging connectivity scenarios such as Machine-to-Machine (M2M), Device-to-Device (D2D), Vehicle-to-Vehicle (V2V), Wireless Sensor Network (WSN), smartphone and embedded devices need a heterogeneous system to interconnect various physical objects. To address this issue, the SDN is an efficient mechanism with a centralized control and global view of the entire network which facilitates optimization and configuration of network devices. Therefore, the centralized location of the control plane introduces new security challenges and threats. This study analyses the implementation of a distributed control plane on the SDN architecture. It is also important to understand how to set up forwarding decisions on the OpenFlow (OF) protocol for communication between the control and the data plane.

Challenges

Traditional networking devices have proprietary management systems on hardware and software, and the control functions are hard-coded. This make it difficult to design, deploy manage scalable networks. Actually, the configuration of the operating parameters on each network device is done manually by a network administrator, via the Command Line interface (CLI), limiting innovation and flexible management. In the last few decades, the number of devices with heterogeneous characteristics has grown rapidly, increasing the complexity of network design and operation, which is a quite challenge. To overcome these limitations in traditional networking, a new perspective on network management such as the SDN allows the manipulation of network devices via APIs, eliminating the dependency on proprietary hardware and software systems.

At the beginning of this research we encountered technical difficulties for the elaboration of our own framework. In fact, there was a lack of available technical information about how to implement SDN systems that were adaptable to the objectives of our research. It is for that

reason we decided to build our own experimentation platform appropriate for the evaluation of our approach. With the developed testbed platform, it is possible to create routes between end-points, monitor the requests sent to the SDN controller, and to decide the management policies for each devices connected to the network including large-scale and clustered systems.

Contributions

The main contribution of our research is the development of an heterogeneous architecture for ad-hoc networks and the IoT, based on a new networking paradigm, the SDN. The first part of this thesis consists of a detailed theoretical study of SDN and OF technologies, after which we take advantage of this new network architecture to suggest a more flexible way to program and manage networks with or without infrastructure. The proposed SDN-based architecture is addressed in the context of ad-hoc and IoT networks, including the integration of devices with heterogeneous connectivity. In the SDN architecture, the controller represents a potential single point of failure risk due to centralized intelligence where all network devices depend directly on a single node, introducing a system weakness. Therefore, as part of this thesis we have studied the implementation of distributed controllers in a clustered architecture in order to reduce latency and increase availability, scalability and fault tolerance in SDN deployment.

We have developed our own testing platform that can emulate a large-scale network by using virtualization techniques including the SDN, the OF protocol and many other necessary tools for the development of this project. Another part deals with developing a distributed SDN architecture to perform the large-scale OF network management for evaluating the performance of the framework in different development environments. We reviewed the OF and SDN vulnerabilities, providing an approach and some component tool mechanisms addressed to avoiding these security threats.

Thesis Outline

The thesis is organized into two main parts. The first part contains a background about the domain of research. This part consist of three chapters, and the organization is as follows: The chapter 1 describes an overview of Software Defined Networking and the OpenFlow protocol. The chapter 2 introduces related works in the SDN field. Chapter 3 summarizes the development of the Internet of Things technologies and studies the different approaches for the IoT. The second part of this document introduces our contribution and also describes the proposed architectures. Chapter 4 presents the proposed SDN solution addressed to ad-hoc networks. At first, we introduce the most commonly ad-hoc routing protocol and then depict an approach to control and manage ad-hoc and IoT networks based on SDN. Chapter 5 explains the distributed SDN architecture in order to provide control plane availability and scalability using the OpenDayLight (ODL) Controller. Chapter 6 presents implementation details of our approach and shows the evaluation results of our developed framework. Additionally, this chapter includes some descriptions of SDN vulnerabilities and suggests security mechanism solutions to protect the SDN architecture in a better way. Finally, in Chapter 7, we summarize the content of the thesis, and provide an overview of future research directions.

Publications

- O. Flauzac, **C. González**, A. Hachani and F. Nolot, **SDN based architecture for IoT and improvement of the security** [12]. In: The IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, 2015, Gwangju, Korea.
- O. Flauzac, **C. González** and F. Nolot, **New Security Architecture for IoT Network** [13]. In: Procedia Computer Science, The International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems (BigD2M 2015), London, United Kingdom.
- O. Flauzac, **C. González** and F. Nolot, **SDN Based Architecture for Clustered WSN** [14]. In: The IEEE 4th International Workshop on Extending Seamlessly to the Internet of Things (esIoT) 2015, San Catarina, Brasil.
- O. Flauzac, **C. González** and F. Nolot, **Original secure architecture for IoT based on SDN** [15]. In: The IEEE 16eme Colloque Francophone sur l'Ingenierie des Protocoles et Conférence sur les Nouvelles Technologies de la Répartition (CFIP & NOTERE) 2015, Paris, France.
- O. Flauzac, **C. González** and F. Nolot, **Developing a distributed software defined networking testbed for IoT** [16]. In: Procedia Computer Science, The 7th International Conference on Ambient Systems, Networks and Technologies (ANT) 2016, Madrid, Spain.
- **C. González**, O. Flauzac, F. Nolot and A. Jara, **A Novel Distributed SDN-Secured Architecture for the IoT** [17]. In: The IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS) 2016, Washington DC, U.S.
- **C. González**, S. M. Charfadine, O. Flauzac, and F. Nolot, **SDN-based security framework for the IoT in distributed grid** [18]. In: The IEEE International Multidisciplinary Conference on Computer and Energy Science (SPLITECH) 2016, Split, Croatia.
- O. Flauzac, **C. González** and F. Nolot, **Software-Defined Clustered Sensor Networks approach to secure IoT networks** [19]. In: ACM, The second International Conference on Internet of Things, Data and Cloud Computing (ICC 2017), United Kingdom

Part I
State of the Art Review

Chapter 1

Traditional Networking and the SDN

Summary: This chapter provides a background of literature about the traditional network architecture and a new emerging network technology, the SDN. We review both network communication architectures in order to provide a better comprehensive analysis of functionalities and the processing data-path of packets. Moreover, we identify the main components of the Software-Defined Networking technology and the OpenFlow protocol which allows us to build automated and programmable networks.

1.1 Introduction

Nowadays, the Internet has become an integral part of everyday life across different social stratifications. Various types of Internet connections are utilized by users to exchange the information between them. To best understand the intercomputer communications processes between differents networking systems, the Open Systems Interconnection (OSI) model was developed by the International Organization for Standardization (ISO) in 1984 [20] to standardize the architectural design. The OSI model is composed of seven layers each of which defines a specific network funtion. The seven layers are divided into upper layers and lower layers. Upper layers 5-7 consist of application, presentation and session layers which perform applications implemented in software at the end user side. Lower layers 1-4 comprise the transport, network, data link, and physical layers that deal with data transportation issues, implementing hardware and software components.

Another networking architectural model is TCP/IP which consist of 4 layers: application, transport, Internet and network access. This is a reference model for devices communicating over the Internet. However, these models are only a conceptualization of communication between network components. The communication process takes place through setting up a set of network protocols. These protocols are located on each layer, allowing interaction with the layers above and below them, forming a protocol stack to guide the data trasmission.

As shown in the figure 1.1, the OSI and the TCP/IP model are equivalent communication functions. The application, presentation, and session layers of the OSI model are merged in one application layer in the TCP/IP model. This application layer contains a large number of higher-level protocols including Domain Name Service (DNS), File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP), to name a few, which operate applications and effect translation, encoding, encryption and compression of data. They also handle communication

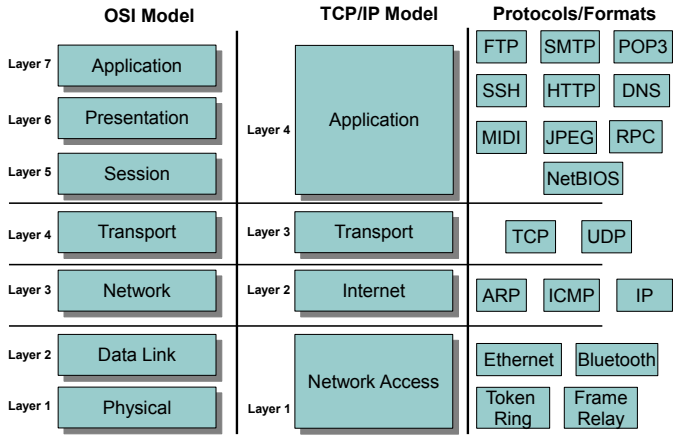

Figure 1.1: Layers and Protocol Network Archicture.

coordination between system endpoints. Once application protocols receive the data, they include header fields before transmitting across the network. Then, the data pass down to the transport layer which uses Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) for the transmission process. At the Internet layer, the received data will include the source and destination logical/IP address, so-called datagrams. In addition, networking devices at this level make the routing decisions. The main protocols of the Internet layer are Internet Protocol (IP), Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP) and Internet Group Management Protocol (IGMP). The physical and data link layers are matched to the network access layer in the TCP/IP model. This layer defines how to establish connectivity between physical components of the network and protocols. Common protocols included in the network access layer are Ethernet, Token Ring, FDDI, X.25, Frame Relay, Wi-Fi, etc.

One the most important aspects of networking is data encapsulation. To establish a process of communication between nodes over the network, a source sends information to a destination. This information sent across the network is referred to as data or data packets, and the process for packaging the information is called encapsulation. The information transferred from a source starts at the application layer of the OSI model, moving the data from upper layers to lower layers. The data passes onto the next layer, receiving at each layer headers, trailers, and other information [2]. Each layer adds a unique protocol data unit (PDU) to identify the information in each layer. On the upper-layers, the alphanumeric characters are converted to data for being transmitted across the network. Then, the data are sent to the lower-layers, in which the adjacent transport layer adds a segment header, including source and destination ports or a sequence number. Once the segment process is accomplished, the transport layer forwards the data segment to the network layer. The network layer adds a datagram or packet that contains a packet header with the source and destination IP addresses to ensure the best path for sending the packet. The packet is converted into a frame at the data link layer. It contains a header and a trailer to allow connection with the physical network. Once the encapsulation process is achieved, the bit stream is transmitted to the destination side. The destination host performs an inverse process called decapsulation, starting on the physical layer which decodes the bits. Then, each layer decapsulates the corresponding header information and sends the

data packet through the upper layers, as shown in the Figure 1.2. If there are no errors found on the entire process, the message is successful transmitted to the destination.

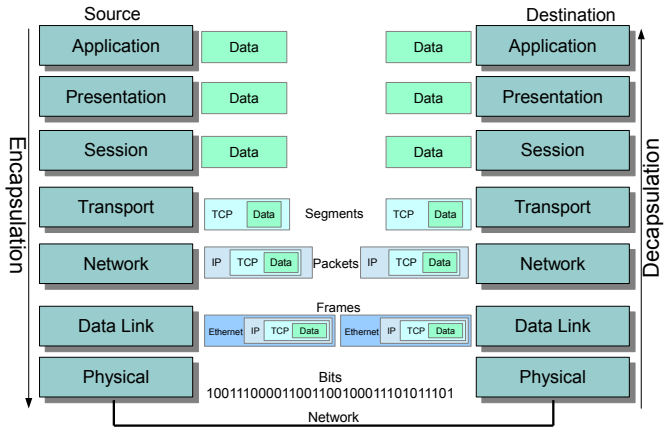

Figure 1.2: Encapsulation and Decapsulation process OSI model [2].

In traditional networking settings, the router knows the right path to the destination using dynamic routing protocols. In an enterprise network, the switch has its own control function for each packet received. Some high-performance network device models are able to recognize the type of application then apply specific rules. To move traffic information across the network, the networking devices perform routing and switching functions. The routing function determines the best path to transfer packets from one host to another host across different networks at the network layer of the OSI Model. This process is performed by a dedicated node called router which maintains a routing table to determine the route to the destination network. On the other hand, the switching function is used to move data packets between devices on the same network at the datalink layer of the OSI Model. A switch device transfers information from source to destination based on a MAC address table and the physical input/output port.

For instance, Cisco network layer switches use the Cisco Express Forwarding (CEF) function [21], an advanced network layer switching technology used to minimize the overhead and packet processing delays. Furthermore, the CEF stores the routes generated by routing protocols in a table called Forwarding Information Base (FIB) as well as the MAC address of each neighbor of a node in an adjacency table. With the CEF technology, the control plane is separated from the data plane, and the data flows are forwarded via the data plane. The control plane builds FIB and adjacency tables based in software, while the data plane forwards IP traffic using hardware. FIB table caches all routing information, even before routing any packet, avoiding the fast-switching process, where the forwarding decisions are made on-demand. Once the information needed is read, the switch can find the information needed to create the packet. However, the rules that are applied on the switch are not dynamically set up to the network packet level. These can be only changed by the network administrator. It takes a considerable amount of time and effort to manage that in large scale environments because they do not dynamically adapt to topology changes. In Software Defined Networking (SDN), there is a paradigm that constantly shifts as to networks should be designed, built and operated. With this emerging technology, it is expected that the network owners and operators will have more control of their infrastructures. It will allow the customization and optimization of network

devices, reducing the overall operational and capital costs. As such, the SDN paradigm can be used to overcome the aforementioned limitations and challenges since it can enable dynamic network configuration and security policies deployment. Moreover, the network devices configuration can be automated and even programmed. The network administrator has the ability to implement some rules and change the network policies in a special device called a controller. By setting up the appropriate instructions to the network devices, SDN can detect and adapt to network changes dynamically in real-time.

1.2 SDN Architecture Overview

Actually, the traditional networking functionalities are mainly set up in one or more switches/routers and application software for the end user's communication process. The implementation guidelines of network functions and protocols are described by the OSI and TCP/IP reference model, providing a comprehensive explanation on how a network device moves the data across the network. Each network device has the control plane to build a forwarding table and the data plane that forwards the traffic based on the forwarding table rules. In this way, a network administrator needs to configure individually each networking device. This represents a hard task when it involves programming a large scalable network. With an emerging technology such as the SDN, the control and the data plane are decoupled, centralizing all network intelligence in a single location, the SDN controller. However, decoupling the control of each packet in a network and the data forwarding in the SDN architecture is not a new concept. For example, this is a key element of the MPLS technology. It is also a feature of many existing Wi-Fi networks. The definition of SDN that is emerging focuses less on decoupling the control plane from the data plane but instead focuses more on the ability to provide the programming interfaces within the network equipment.

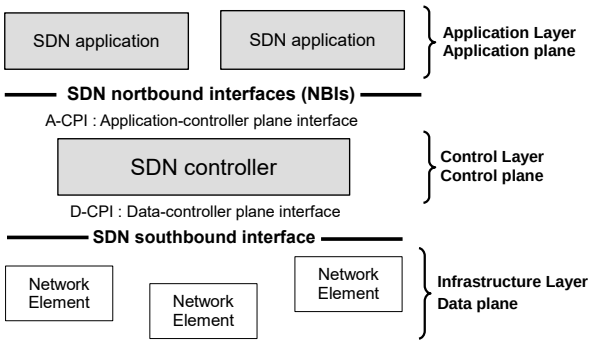

Figure 1.3: SDN architecture layers [3].

The SDN architecture, as shown in Fig. 1.3, is actually composed of three layers: application layer (Application Plane), control layer (Control Plane), and data layer (Data Plane). Each one of them communicates with its adjacent layer using a proper NorthBound Interface (NBI) or Southbound Interface. The NorthBound Interface corresponds to the Application-Controller Plane Interface (A-CPI), and the Southbound Interface corresponds to the Data

Controller Plane Interface (D-CPI) defined in the SDN architecture specified by the Open Networking Foundation [3]. The application layer represents a set of SDN applications that can communicate their network requirements to the SDN control layer using the NBI interface. The control layer is one SDN controller or a group of two or more SDN controllers coordinating with each other, also it communicates with the other SDN layers using specific interfaces and protocols. The SDN controllers are also able to install the routes and policies into the network devices and remotely configure them. Finally, the SDN data plane incorporates the network devices and implements some decisions such as forwarding the traffic or processing it. The most commonly used protocol for communication between the network devices and the controller is OpenFlow [22]. This protocol is standardized by the Open Networking Foundation[1] and is adopted by many IT vendors. In addition, OpenFlow enables the controller to remotely manage, configure and enable compatible OpenFlow switches, in order to read the network device states, collect the traffic informations and statistics. The controller has a global network view. An alternative protocol to OpenFlow, is OpFlex (see Fig. 1.4). In conjunction with industry partners and Open Source communities, Cisco proposes a plugin OpFlex to both OpenDayLight Helium controller and the virtual switch Openvswitch. OpFlex is based on two principle elements, the Policy Authority and the Policy Element. The policies are defined in the Policy Authority. Then they are resolved asynchronously by the OpFlex Policy Element. The End Point Registry contains meta-data information about the endpoint devices which can be organized into endpoint groups. An observer is used by OpFlex to store status information updates of the endpoint devices. The device or host is connected via the Policy Element, empowering the nodes to make more decisions about how to execute the policies. Basically, the policies are defined within a policy repository in the controller, and then the OpFlex protocol is used to communicate and enforce those policies rules within a set of distributed policy element to the network devices.

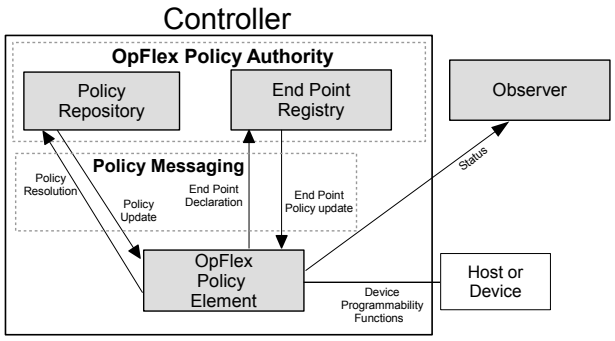

Figure 1.4: Components of OpFlex [4].

Since the emergence of SDN, many works have discussed the SDN architecture [22, 23, 24], with focus on how it can help maintaining and enhancing the network performance, while the network's control functions are hosted in a remote location separated from the forwarding device. On the other hand, using only one controller SDN raises the issue of having a single point of failure due to the centralize control system architecture. In the sequel, we present some works related to the performance of SDN architecture. A discussion is also initiated about the

[1] https://www.opennetworking.org/

challenge to centralize the control plane and to allow the physical devices to be distributed through the network.

1.2.1 Performance of the SDN architecture

In [25], the performance of the SDN architecture was evaluated using a simple OpenFlow model to analyze the forwarding speed and the probability of dropping a packet. This study reveals that the total amount of time that a packet traverses through the system depends on the processing speed. Moreover, the higher probability of getting new flows is, how long the packet's forwarding delay will be. The results lead to many questions such as *how far scalability can be handled with only one controller*, *where must we place the controller*.

In [24], Heller et *al.* have used 100 publicly available wireless access network topologies in order to quantify the effect of the controller placement. The authors proposed a solution to get a better understanding of the fundamental questions by Jarschel et *al.* [25] about the comparison of the controller placement against the latency, fault discovery, and event propagation efficiency. The study shows that there is no specific rule to guide the controller placement or to determine the utilized number of controllers. The answers will depend on the desired reaction bounds, metric choices, and network topology.

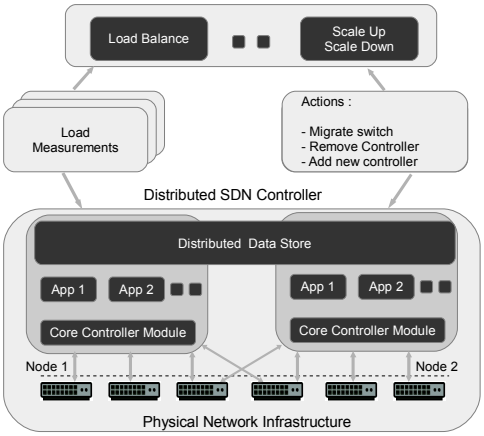

Figure 1.5: ElastiCon distributed SDN architecture [5].

According to Jarschel et *al.* [25], and Heller et *al.* [24], multiple controllers must be used in order to ensure reliability and high availability. As illustrated in Fig. 1.5, an example of such scenario so-called elastic distributed controller architecture (ElastiCon) is proposed in [5]. This architecture has been designed to allow load balancing of network traffic. To achieve this goal, a dynamically expandable pool of controllers is provided, each operating at the same specified baseline load. Furthermore, OpenFlow switches are dynamically associated with the controllers using a migration algorithm based on the existing OpenFlow standard. Another distributed SDN architecture, called Kandoo, has been proposed by Yeganeh and Ganjali in [6]. Kandoo is a framework that uses two layers of controllers. As described in Fig. 1.6, the first or the bottom layer consists of a pool of separated controllers which have only local knowledge of the network state, and run applications over the local connected switches. The second or the top

layer refers to the logically centralized control plane, which controls the pool of local controllers and has a global view of the network state.

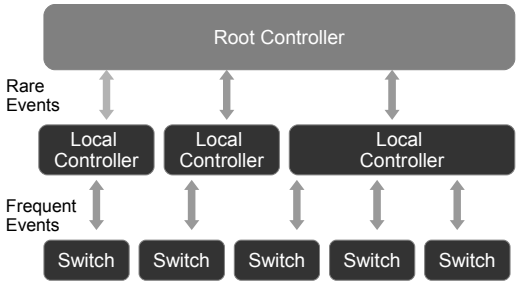

Figure 1.6: Kandoo distributed SDN architecture [6].

HyperFlow [26] is another example of a distributed SDN architecture that is based on distributed events. In order to leverage the load on the logically centralized control plane, HyperFlow keeps a global network state using passive synchronization among the physical distributed controllers. By locally making the decisions and responding to the data plane requests, the controllers are able to reduce the time of response, save some bandwidth, while providing fault tolerance and scalability, keeping the control plane centralized. The two layers-based scheme has been used to design the DIFANE flow management architecture [27], which main tenant is to keep the traffic in the data plane by imposing the necessary rules to the intermediate authority switches. By distributing the flow management process, the authority switches can leverage the traffic load on the control plane and avoid the bottleneck at the controller uplinks while dynamically reacting to events like topology changes, host mobility, the burden of the network traffic and application modifications. The authority rules stored in the intermediate switches also help keeping most of traffic in the data plane for better performance and scalability.

The SDN distributed architectures provide many benefits in terms of performance and scalability, but keeping the control plane logically centralized while having a physically distributed control infrastructure in place raises many challenges. Schmid and Suomela [28] study the interactions between SDN distributed architectures and local algorithms by using different scenarios. They introduces a new model of distributed computing called *supported locality model* that matches the features of SDN networks. Levin et *al.* [29] study the impact of distributed SDN architectures on the performance of the control plane. An existing SDN load balancing application was used for this purpose. Their results show that the performance of the control application significantly decreases when it are utilized by the SDN-distributed architectures. Furthermore two concrete trade-offs were identified, the first is between staleness and optimality, and the second is between application logic complexity and robustness to inconsistency.

In the related literature, there are also few works that have investigated the use of hierarchical architecture for SDN to optimize and distribute control functions. The work of Shuai et *al.* [30], studied the control plane scalability problem from the viewpoint of Information-centric networking (ICN)/SDN integration and proposed an scalable area based hierarchical architecture (SAHA) for controller deployment. Xu et *al.* [31] proposed a network partition into regions and zones, each region is assigned to an SDN controller. In their architecture, slave controllers are in charge of the zones separately, and are managed by a master controller.

Aissioui et *al.* [32] introduced a distribute SDN/OpenFlow control plane on a two-level hierarchical architecture: the first one with a global controller, and the second one composed of several local controllers deployed on-demand. Flauzac et *al.* [12] suggest not to centralize the control functions on multiple controllers, but instead, to distribute the routing functions and security rules on each controller. Moreover they suggest that these controllers have essentially the function to establish the connections between other distributed controllers and assure the integrity of each communication.

1.3 OpenFlow Protocol

OpenFlow is the most commonly used protocol for implementing the SDN scenarios. Often, the terms Openflow and SDN are used continuously in the literature, but it is important to understand the difference between these terms. In principle, the term SDN typically refers to the decoupling of the control plane and the data plane whereas Openflow is a protocol that enables the control of network devices. The main function of OpenFlow is to enable the communication between the SDN controllers and the network nodes. This is probably best known as the southbound interface. Furthermore, OpenFlow is not the only available protocol used for the SDN southbound APIs. The following is a list of other southbound protocols and their descriptions:

- Opflex: This is an open and extensible policy protocol implemented to transfer policy rules from a controller [4, 33].
- Lisp: This is also an open standard protocol, design to support the flow mapping [34].
- The Network Configuration Protocol (NetConf): This uses an Extensible Markup Language (XML) to communicate with the network nodes (routers, switches, etc.), in order to install and make configuration changes [35].
- IEEE P1520: This work as functions as a Programmable Network Interface, and can also help performing the network programming abstractions [36].
- ForCES (Forwarding and Control Element Separation): This protocol proposes a model to manage and separate IP control and data forwarding. Forwarding devices are modeled using logical function blocks (LFB). To the best of our knowledge, this protocol has not yet been widely adopted by major network devices [37].
- SoftRouter: This protocol also allows separation of control plane and data plane functionality. In the Soft Router architecture, there are two main types of network entities the forwarding element (FE) and the control element (CE), which together constitute a network element (NE) router [38].

In traditional network architectures, each networking device must compute independently the path to the destination after exchanging routing information. These processes require many successive operations. For example, each device has to exchange its routing table in order to send a message to any device. In SDN architecture, the routing table is managed only by

the SDN controller (Figure 1.7). Network devices will only handle the packets based on the flow table entries received from the SDN controller via the OpenFlow protocol. Our study is focused on OpenFlow because it is the most popular and thoroughly tested open source protocol on the southbound SDN interface. Moreover, this protocol allows deployment in production networks, providing a centralized control management over the networking devices. Also, it can co-exist with other network protocols. In addition, OpenFlow can be addressed to overcome the limitations of traditional network interconnecting devices that use proprietary closed systems which constrain the innovation. In this section, we will present the OpenFlow protocol and give a non-exhaustive list of OpenFlow-compatible switching platforms.

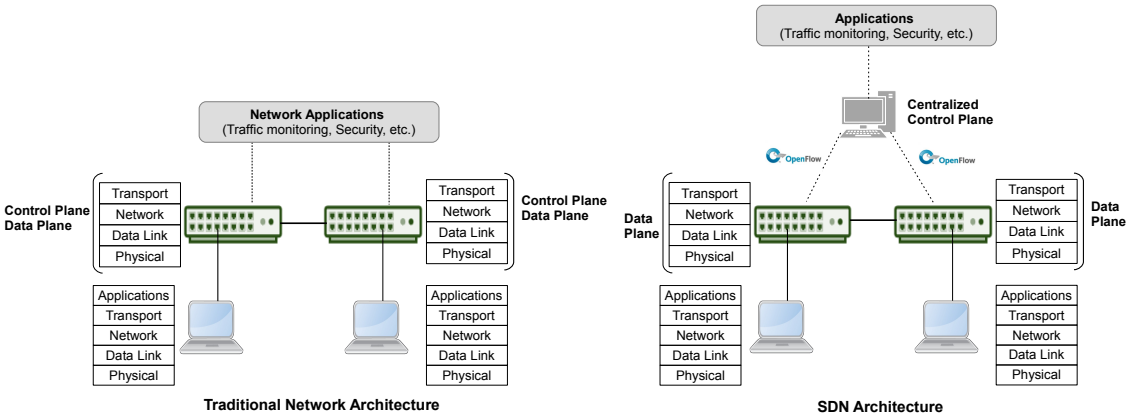

Figure 1.7: Difference between traditional network and SDN Architecture.

OF Message	**Type**
OFPT_HELLO	Symmetric
OFPT_FEATURE_REQUEST	Controller-to-switch
OFPT_FEATURE_REPLY	controller-to-switch
OFPT_BARRIER_REQUEST	Controller-switch
OFPT_BARRIER_REPLY	Controller-switch
OFPT_ROLE_REQUEST	Controller-switch
OFPT_ROLE_REPLY	Controller-switch
OFPT_SET_CONFIG	controller-to-switch
OFPT_ECHO_REQUEST	Symmetric
OFPT_ECHO_REPLY	Symmetric
OFPT_PACKET_IN	Asynchronous
OFPT_FLOW_REMOVED	Asynchronous
OFPT_PORT_STATUS	Asynchronous
OFPT_ERROR	Asynchronous
OFPT_PACKET_OUT	Controller-switch

Table 1.1: OF protocol messages per type [1]

.

In SDN, the communication process between the controller and the networking devices is

established by OpenFlow. There are three types of messages exchanged during this negotiation process: controller-to-switch, asynchronous and symmetric messages [1]. Table 1.1 shows a list of some OpenFlow messages classified by type. In controller-to-switch messages, the request is initiated by the controller to configure, manage or verify the state of the connected networking devices. This type of message is commonly used when the controller and the interconnecting device establish an OpenFlow communication channel. The OF channel may be secure, implementing TLS connection over TCP. The asynchronous messages are initiated by the connected network device to inform the controller about of their state, the incoming packets, the port status or any error events. The symmetric messages are share in both directions controller-to-switch and switch-to-controller. Once the communication channel is established, the controller-switch exchanges HELLO messages to determine the OpenFlow version number supported on each side. Also, it includes ECHO messages which provide the status of the communication channel and measure the latency/bandwidth upon the startup connections. When a networking device joins the OpenFlow network, the connection process starts by sending a TCP sync message to the controller IP address at the default OpenFlow TCP port 6633. Then, the controller and the networking device share TCP sync acknowledgement messages, establishing a handshake of the connection. Following this process, Hello messages are exchanged in both directions to determine the OpenFlow version numbers supported. After the hello messages are exchanged, the controller sends a feature request to the switch in order to identify its capabilities and the available ports. The switch replies with a Features Reply, including a list of available ports, port speeds, OpenFlow tables and actions. Another exchanged message is the role-request used by the controller to communicate a specific role to the switch, and it must return the OpenFlow role reply, with the role set by the controller. The set config messages are used by the controller to set or query configuration parameters of the switch. Finally, both the controller and the switch exchange echo requests and echo replies which contain information related to the latency, bandwidth and connection liveliness. Figure 1.8 illustrates the sequence of processes and a Wireshark capture of OpenFlow packets in Figure 1.9. In section 6, we describe in detail how the connection between two hosts is performed in an OpenFlow network.

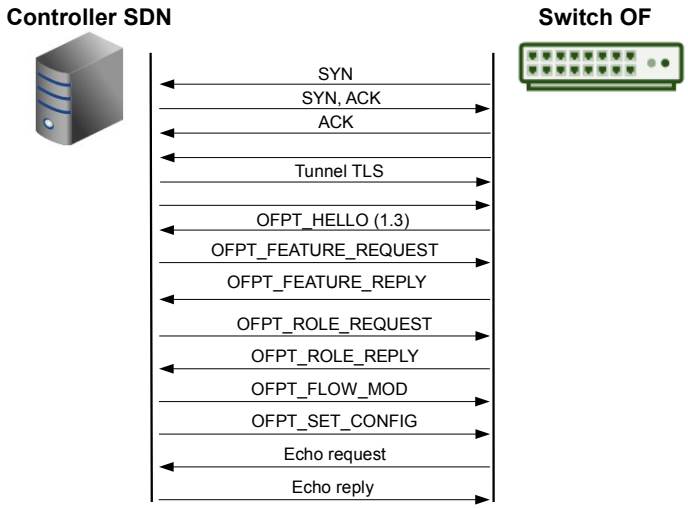

Figure 1.8: Connection establishment between a OF switch and the controller

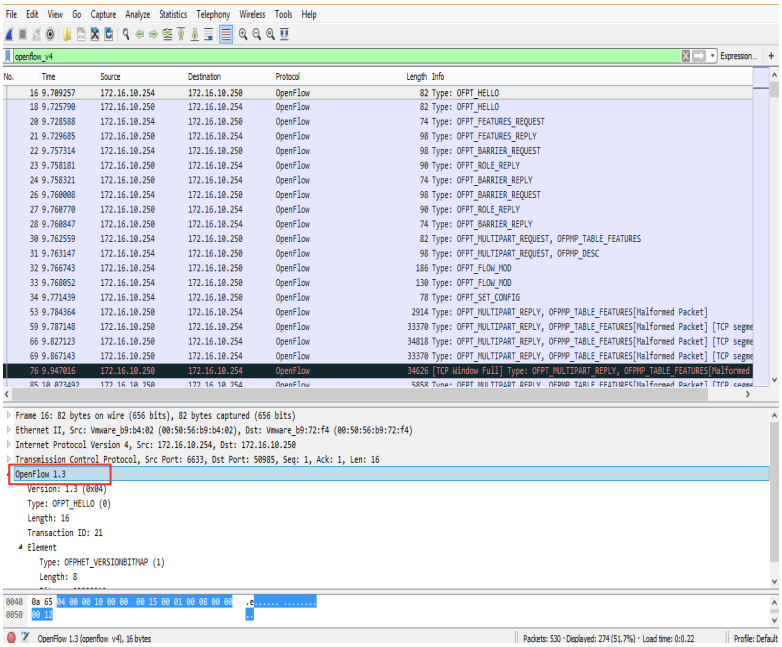

Figure 1.9: Wireshark capture of the connection establishment between an OF switch and the controller

OF Config version	Description and added features
OF Config v1.0	The first version of the OpenFlow switch's configuration and management protocol.
OF Config v1.1	Capability discovery, tunnel configuration, error handling, multiple tables of flow entries.
OF Config v1.2	Allow the assignment of resources to logical switches, simple topology detection, event notification, role election mechanism multicontroller, IPv6 support added.
OF Config v1.3	Event filtering multicontroller, flow meters per application QoS, tunnel-id field metadata, auxiliary connections, refactor capabilities negotiation.
OF Config v1.4	Change default port tcp to 6653, flow monitoring multicontroller, vacancy events control table resource policy, bundles are provided, which can be used to handle a set of simultaneous requests. Also, eviction is available.
OF Config v1.5	Delegation of flow learning to the switch, egress tables, tunnelling using flow entries, port properties of pipeline fields, port properties for recirculation, meter action.

Table 1.2: Main features of the OF-CONF protocol per versions

As explained earlier in section 1.2, basic control operations such as gathering the network information (ARP, MAC learning, routing protocols) or storing the forwarding information (L2/L3 forwarding tables) disappear from the network device and are placed on the controller.

The forwarding rules are now transmitted to the network devices using the OpenFlow protocol which can also collect the information about the state of the network device (such as port state, flow statistics, number of connected devices, etc). The SDN controller configures the OpenFlow capable switches via the OpenFlow Configuration (OF-CONFIG) and Management Protocol. OpenFlow allows the configuration of the switch ports (state up or down) and the discovery of the OpenFlow switch capabilities on the network. It is also involved in the assignment of switches to one or more controllers and the configuration of certificates for secure communications between them. A brief summary of the OF-CONFIG capabilities per version is captured in Table 1.2.

Company or Project name	**Platform**
Arista	Arista 7124FX, Arista 7050
ADVA Optical Networking	FSP 150-GE110Pro series, FSP 150-GE114Pro
Alcatel-Lucent	OmniSwitch 10K, OmniSwitch 6250, 6450, 6860, 6900, Nuage Networks Virtualized Services Platform (VSP)
Big Switch Networks	Big Virtual Switch, Big Tap, Big Network Controller, Switch Light
Brocade	Brocade MLX, Brocade CER, Brocade CES, Brocade ADX Series
Centec Networks	Centec V150, V330, V350 Series Switch
Cisco	Cisco cat6000 series, catalyst 3750, 6500
Dell	S4810-ON, S6000-ON, Dell Force10 Z9000
Juniper	(MX, EX)
HP	HP 2920, 3500, 3500, 3800, 5130, 5400, 5930, 6200, 6600, 8200, 12500, 10500, 11900, and 12900 Switch Series
Huawei	SoftCOM
IBM	RackSwitch G8264 and G8264T
Indigo	Indigo Virtual Switch (IVS)
NEC	PF1000, NEC IP8800, NEC PF5240, NEC PF5820
NetGear	ProSAFE GSM7352Sv2, 7328SO, 7352SO
NoviFlow	Noviflow 2128, 1248, 1132, NoviNID 106 i, Noviware 300, 400
Pantou	OpenWrt router
Pica8	P-3297, P-3930, P-3920, P-3922, P-5401, P-3780, P-3295
Pronto	iW-Pronto 3920, 3295, 3780, iW-SDN8254-S, iW-SDN8952-S
Toroki	Toroki Lightswitch 4810
Open vSwitch	Latest version: 2.4.0
Quanta	LB4G

Table 1.3: OpenFlow Switching Platforms

Table 1.3 presents a non-exhaustive list of software and hardware OpenFlow compatible switch platforms. Note that an OpenFlow compatible switch is intended to be equivalent to an actual physical or virtual network element (e.g. an Ethernet switch) which is hosting one or more OpenFlow data-paths.

1.3.1 Evolution of the OpenFlow specifications

Since its standardization by the Open Networking Foundation, many OpenFlow versions are now available and supported by a variety of vendors devices. Each version is fully detailed in Open Networking Foundation releases. At the time of this report writing and to best of our knowledge, OpenFlow 1.5.0 is the latest version of the protocol. In this version, packets can be matched against fourteen required header match fields (see Table 1.4) which provide a higher level of granularity than the traditional forwarding technique that uses only the layer 2 destination address. Note that not all header fields have to be used in all flow tables. The SDN controller can ask the switch about which match fields are supported in every flow table.

Field	**Description**
OXM_OF_IN_PORT	Physical or logical port
OXM_OF_ACTSET_OUTPUT	Egress port from action set
OXM_OF_ETH_DST	Ethernet destination address
OXM_OF_ETH_SRC	Ethernet source address
OXM_OF_ETH_TYPE	Ethernet type of the OpenFlow packet payload
OXM_OF_IP_PROTO	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	IPv4 source address
OXM_OF_IPV4_DST	IPv4 destination address
OXM_OF_IPV6_SRC	IPv6 source address
OXM_OF_IPV6_DST	IPv6 destination address
OXM_OF_TCP_SRC	TCP source port
OXM_OF_TCP_DST	TCP destination port
OXM_OF_UDP_SRC	UDP source port
OXM_OF_UDP_DST	UDP destination port

Table 1.4: OpenFlow matching fields

In OpenFlow version 1.5, the switches can use multiple flow tables instead of only one to process network traffic. Each flow table contains many flow entries. A pipeline determines the manner in which the packets should interact with those flow tables. For reasons of simplicity and minimization of processing overhead, the use of one flow table is recommended. MPLS and virtual local area network (VLAN) matching are also added to the OpenFlow entries.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Table 1.5: Main components of a flow entry in a flow table

A flow entry (see Table 1.5) contains: a match field to compare the packets header and ingress port, a priority field to match the precedence field of the flow entry. A counter field

updates the packets that match with the entries in the flow table, field used for the dynamic removal of flow entries can be set to expired by the switch. A cookie field that can be used to make flow statistics, flow modification and flow deletion requests is also available as well as flags that can be used to alter the way that flow entries are managed.

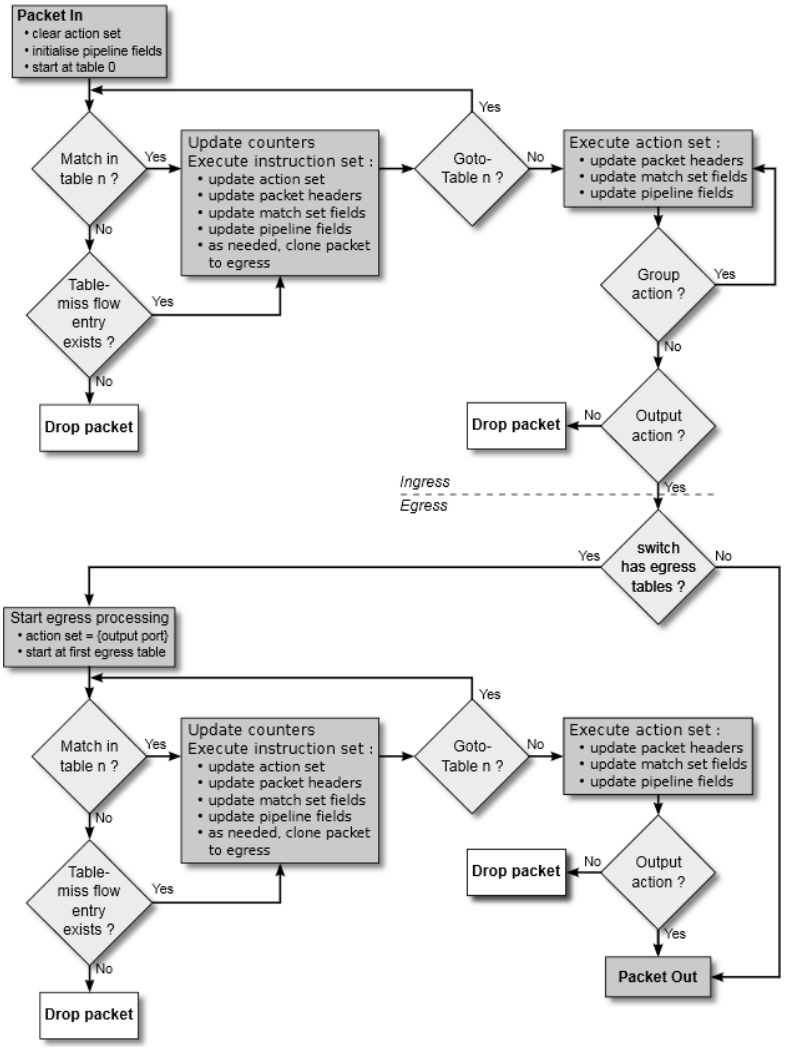

Figure 1.10: Openflow packet processing and forwarding [1].

The pipeline process has two stages, ingress and egress processing. This starts with ingress processing when the packet is received. The OpenFlow switch processes the packet as shown in Figure 1.10 [2]. The switch looks for an entry in its flow table. If it finds an entry, then the counter field is updated and the instructions for this flow entry are added to the action set of the packet but not executed yet. Afterwards, if there is a Goto table instruction in the action set, then the packet is sent to the flow table. When there is no other flow table to go to, the packet exists in the pipeline and the instructions in its action set are executed. If the packet resulting from the ingress processing is forwarded to an output port, the OpenFlow switch

[2] https://www.opennetworking.org/

may start the egress processing. This process is optional. If no egress table is configured, the packet must be processed via an output port. Otherwise, the process will be repeated after the ingress processing step, and the packet will be forwarded out of the switch. In case of no match between the packet header and any flow entries of the table, this flow will be marked as Table-miss and either the flow will be sent to the controller or will be sent to another table. If the table-miss entry is not configured, the packet will be dropped.

OpenFlow versions	**Release date**	**Major features**
OpenFlow v0.2.0	March 28, 2008	Wire protocol: 1
OpenFlow v0.2.1	March 28, 2008	No protocol change
OpenFlow v0.8.0	May 5, 2008	Reorganize OpenFlow message types, add flow priority and error messages
OpenFlow v0.8.1	May 20, 2008	No protocol change.
OpenFlow v0.8.2	Oct. 17, 2008	Add echo request and echo reply messages.
OpenFlow v0.8.9	Dec. 2, 2008	Wire protocol: 0x97
OpenFlow v1.0.0	Dec. 31, 2009	Slicing, flow cookies, match IP fields and IP/-ToS.
OpenFlow v1.1.0	Feb. 28, 2011	Multipath, Tags/tunnels, multiple tables, virtual port, controller connection failure
OpenFlow v1.2	Dec. 5, 2011	Per-Flow rate limiters, multiple controllers, flexible match, IPv6, GRE/L3 tunnel
OpenFlow v1.3.0	June 25, 2012	802.1ah PBB, multiple parallel switch-to-controller channels
OpenFlow v1.3.1	Sept. 6, 2012	Improved version negotiation, other changes
OpenFlow v1.3.2	Apr. 25, 2013	Changes, Clarify padding rules, variable size arrays, meter flags, burst fields
OpenFlow v1.4.0	Oct. 15, 2013	Extensibility, optical port support, flow monitoring, improved management
OpenFlow v1.3.3	Dec. 18, 2013	Changes, Clarify policing of packet-in to controllers, queue relation to ports and packets
OpenFlow v1.3.4	Mar. 27, 2014	Changes, Clarify flow_count for meter stats, action bad argument errors, error code for unsupported actions
OpenFlow v1.5.0	Dec. 19, 2014	Egress table, controller connection status, meter action, scheduled bundles messages
OpenFlow v1.3.5	Mar. 26, 2015	Changes, Clarify he meaning of total_len in packet-in structure, that barrier replies correspond to barrier requests
OpenFlow v1.4.1	Mar. 25, 2015	Changes, Clarify that a table can synchronise on itsel, that Bundle is optional, that Flow monitoring is optional
OpenFlow v1.5.1	Mar. 26, 2015	Changes, Clarify add figure for Meters, spelling, gramar and other typos

Table 1.6: Control Platforms

In this work, we do not focus on the OpenFlow protocol itself, but rather on the way it is used to improve and enhance the network services and performance. For this reason, we have provided only a summary of all OpenFlow specifications along with their corresponding release

dates and new features (see Table 1.6). Since the advent of OpenFlow specification version 1.2, it has become possible for a switch to have an association with more than one controller enabling fault tolerance, fail-over between SDN controllers and high availability by load sharing a switch's control traffic. This OpenFlow version also supports IPv6 header matching. In version 1.3, the options of establishing auxiliary connection and multiple parallel channels between the switch and the controller are added. The Openflow version 1.4 which released in August 2013, added better switch management by the controllers, optical port support (frequency, power), and an improved extensibility. Openflow 1.5 introduced egress tables, PPP packet aware pipeline, TCP flag matching to detect the start and the end of TCP connections, the controller connection status. Some features to monitor the connection between switches and other controllers were also added. There are also other versions of the OpenFlow protocol with special purposes (clarification, fixing bugs of previous version, etc.).

1.4 Control plane

The most important component of the SDN architecture is the controller, a host that centralizes the networking functions and keeps track of the network global status. In this section, we present some projects related to the design and implementation of the SDN control plane and provide a non-exhaustive list of existing SDN controller platforms with a brief description of their implementations and management functions. Below is a summary list of SDN controller platforms that we have been able to find to this point, together with their respective programmatic language and supported OpenFlow version (see Table 1.7).

Note that we have not been able to provide a description for all controller platforms listed above because of the lack of support and documentation related to each platform. We begin with NOX, the first OpenFlow controller [39], a platform for constructing the network control applications developed by Nicira Networks, a company acquired by VMware in 2012. The development of NOX was turned over to the open source community in 2008. Thereafter, that NOX was superseded by POX [3], a variant of it used for Python development, which also supports OpenFlow. It has a high-level SDN API including a query topology graph and support for virtualization. It can be used for SDN debugging. Another control platform is Beacon [40], an open source java based controller created in 2010. The Beacon controller interacts with the OpenFlow using an event based API and it also support threaded operations. The results of different evaluations and benchmarks showed that Beacon can achieve high performance and scalability [40]. Like Beacon, Floodlight [4] is also an open source java based controller. It is easy to set up and offers a module loading system which makes it simple to extend. Floodlight controller supports the OpenFlow standard, a wide range of physical and virtual switches, which can handle mixed networks (OpenFlow and non-OpenFlow) and has support for the OpenStack cloud orchestration platform. Also referred to as the core of commercial controller product from Big Switch Networks, Floodlight is supported by a community of professional developers.

Another control platform is Maestro [41] a multi-threaded java based SDN controller whose functionality structure can be used to maximize the throughput of system by distributing

[3] https://github.com/noxrepo/pox
[4] http://www.projectfloodlight.org/

Controller	Programming language	Data Plane Control Interface
NOX	C++, REST	OpenFlow 1.0, 1.3
POX	Python, REST	OpenFlow 1.0
Becon	Java	OpenFlow 1.0
Floodlight	Java, Python, REST	OpenFlow 1.0, 1.3, 1.4
Flowvisor	C	OpenFlow 1.0
Contrail	Java, Python, REST	XMPP
Open Contrail	Java, Python, REST	XMPP
Opendaylight	Java, Python, REST	OpenFlow 1.0, 1.3
Cisco APIC	Java, Python, REST	OpenFlow 1.0, 1.3 , One PK
Cisco XNC	Java, Python, REST	OpenFlow 1.0, 1.3 , One PK
NEC PFC	Python, REST	OpenFlow 1.0
HP VAN	Controller Java, REST	OpenFlow 1.0 , 1.3
Maestro	Java	OpenFlow 1.0
Trema	Ruby, C	OpenFlow 1.0
NodeFlow	JavaScript	OpenFlow 1.0
Ryu	Python	OpenFlow 1.0, 1.2 , 1.3, 1.4, 1.5
Openvswitch	C, Python	OpenFlow 1.0, 1.3, 1.4, 1.5
FlowER	Erlang	OpenFlow 1.0
Jaxon	Java	OpenFlow 1.0
MUL	C	OpenFlow 1.0, 1.3, 1.4
IRIS	Java	OpenFlow 1.0, 1.3
NEC Programmable Flow	Ruby, C	OpenFlow 1.0, 1.3
ONOS	Java	OpenFlow 1.0, 1.3, 1.4

Table 1.7: Control Platforms

the work evenly among the processor cores. To achive this, Maestro exploits parallelism and minimizes cross-core overhead when moving the data from one processor core to another. Each worker thread is bound to a particular processor core and assures that the same core handles all computations related to a particular flow. Ryu [5] is a Component-based framework that provides many useful components for SDN applications. It is also an open source controller platform that can allow developers to freely add new components. Ryu supports the OpenFlow standard along with many other protocols such as ovsdb, netconf, snmp, xFlow. It can also used to manage OpenFlow as well as non-OpenFlow networks. This means that the Ryu controller can interact with traditional and OpenFlow enabled switches. Another open source SDN control platform is Trema [6], a software platform for developers and researchers with a multi-process modular architecture allowing extensibility by adding new modules. Developers can use C or Ruby to write user modules and use the Trema integrated developing environment to test and debug those developed modules.

Another open source controller that we present in this document is OpenDayLight [7], an open platform for network programmability that enable Network Function Virtualization (NFV). OpenDayLight combines components, interfaces and applications that are clearly defined and documented. This controller allows both users and vendors to customize their SDN and NFV based solutions. OpenDayLight is a reference framework for network control and programmability. It is adopted by a large open community acknowledged by the network industry. Today,

[5] http://osrg.github.io/ryu/
[6] http://trema.github.io/trema/
[7] http://www.opendaylight.org/

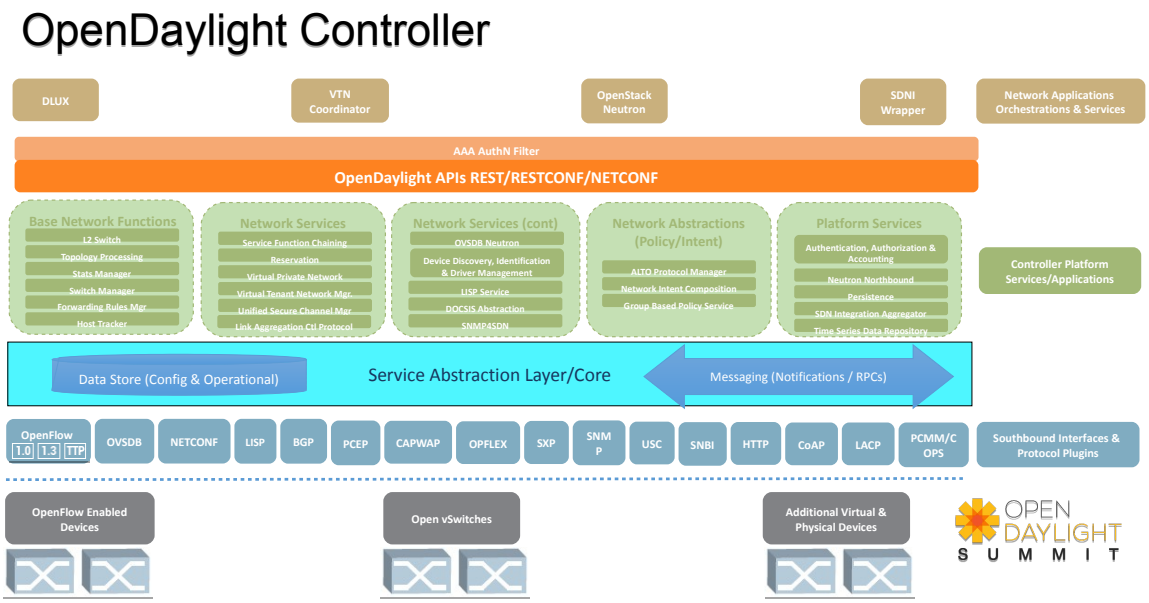

Figure 1.11: OpenDayLight Platform Architecture.

OpenDayLight is considered as a reference model for the controllers architecture since it is adopted and supported by the majority of networks vendors and many open and proprietary control platforms are built upon it. OpenDayLight is available in Helium and Lithium versions, each with its own specific purposes and different modules. Almost all control platforms share the same architecture. Figure 1.11 represents the internal architecture of Lithium, the third release of OpenDayLight. The OpenDayLight community is now developing a fourth release called Beryllium, whose main is on clustering.

This document also includes some proprietary SDN control solutions such as the Cisco Extensible Network Controller [8], and XNC, which to our knowledge, are the first commercial versions of the OpenDayLight controller. XNC supports multiple southbound protocols like OpenFlow and Cisco ONE Platform Kit (onePK) [9]. It has an extensible and modular component architecture provided by the use of the Open Services Gateway initiative (OSGi) [10] framework, which allows the Applications or components on the controller to be dynamically installed, started, stopped, updated and uninstalled without the need to restart the controller. Another example of a proprietary control platform is Contrail [11], the Juniper SDN Control solution which has its open source version. Contrail uses the XMPP protocol for message transmission between the controller and the data plane. It is written in C++ and its user interface is written in Python. It can be combined with OpenStack and CloudStack and it is expected to be integrated with IBM as SmartCloud Orchestrator after a partnership with Juniper is confirmed.

[8] http://www.cisco.com/c/en/us/products/cloud-systems-management/extensible-network-controller-xnc/index.html

[9] http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html

[10] http://www.osgi.org/Main/HomePage

[11] http://www.juniper.net/

Chapter 2

Tools and Services for SDN Deployment

Summary: In this chapter, we describe the most important related works that use SDN approach for network deployment. Several of these research studies are addressed to improving the performance of the SDN concept. Also, as this new emerging networking paradigm is attracting more and more the interest from academic and industrial areas, these approaches provide us an overview of the actual fields of research.

2.1 SDN enhancing network services

Based on the knowledge of the global network state, the programming capability and the automation of the network, SDN can enhance the network services and performance significantly by making them aware of the application's context and being able to properly respond in case of different network events like topology changes, device failure, traffic load changes, etc. In this section, we are interested in some work that discusses SDN and Openflow based network services. These works will be presented according to their type of service application.

2.1.1 Management and monitoring

Having a controller which automatically configures all network devices, supports new application requirements and any other network events for the network engineers allows saving both time and cost. But it will also gives rise to many challenges related to the quality of the implemented code, the interoperability issues that may occur between OpenFlow and non-OpenFlow networks, the failure of nodes controller, some software bugs and possible conflicts resulting from a switch being managed by more than one controllers. In this section, we present works related to the use of OpenFlow for network management.

Motivated by the problems of non-interoperability between legacy networks, OpenFlow networks, and the management of both networks, a management solution called LegacyFlow is proposed in [7]. Figure 2.1 depicts this model which is composed of two layers. The first layer is the datapath layer that consists of one or more OpenFlow controllers. The second layer is the switch controller layer, which is responsible for the translation between the OpenFlow controller and the legacy network. This layer is in charge of converting the OpenFlow instructions into specific vendors configuration commands. It also provides information about the legacy equipment to the OpenFlow controller by reading their configurations and translating them

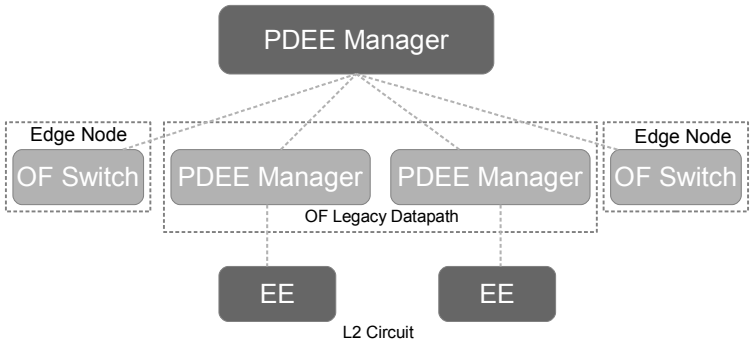

Figure 2.1: LegacyFlow management architecture [7].

into OpenFlow information messages. LegacyFlow can also help managing the migration and deployment of OpenFlow based networks, including hybrid networks.

The SDN network management complexity comes not only from the controller design, but also from the support or not of OpenFlow by the devices. Therefore, the design of a network management solution is a challenge. Various SDN-based network management approaches have been discussed and a distributed management environment, called Programmable Distributed Execution Environment (PDEE) is proposed in [8]. This approach introduces a new node named PDEE Manager which is responsible for PDEE management and is composed of a set of programmable Execution Environments (EE) hosted in all the network nodes. The basic PDEE management operations are accomplished by the EE whereas the complex ones are handled by the PDEE Manager. This approach can help improving the SDN network management, but it requires some modifications to the OpenFlow switch to support the PDEE components, which can be difficult in some cases. With this approach, the control among the network nodes can be redistributed by adding an Execution Environment to them, which is in contradiction with the SDN control and data plane separation. Figure 2.2 shows the PDEE management architecture.

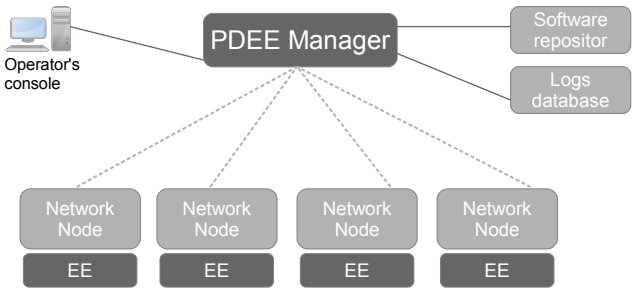

Figure 2.2: Distributed execution environment for SDN management [8].

Another management approach based on autonomic management and network virtualization technologies is the Autonomic Management Architecture (AMA) [42]. The concept explored in this approach is a new management solution of heterogeneous networks with the support of many multi-service applications. The AMA design architecture consists of 5 layers: Basic Virtual Network Functions DE (Decision Elements) that monitor the elements in the switches, Basic Virtual Switch Functions DE that manages and controls the network functions, Network

Virtualization DE that automatically implements the network virtualization, Basic Network Functions DE that automatically manages and controls the elements in the virtual switches, and Basic Switch Functions DEs that manages and controls the virtual network functions. In this framework, the use of autonomic attributes such as self-awareness, self-configuration, self-protection and self-organization, is meant to help relaxing the network control and management complexity. The use of network virtualization technologies also helps in isolating data traffic and network segments and allowing for a better management of the virtual infrastructure.

The heterogeneity of a network is not the only challenge for the management of SDN networks. The control plane heterogeneity itself can also be a challenge. A mashup based approach for virtual SDN management is proposed in [43]. It can allow the administrators to combine SDN control and management solutions by building SDN mashup applications. The proposed approach permits the network administrators to integrate the virtual SDN management in heterogeneous environments and to hide the complexity of virtual SDN networks. The SDN mashup system is based on SDN resources as a service as well as on the end-user centric development of composite applications. It also provides a web user interface, making it an easy-to-use mashup development environment. The OpenFlow MaNagement Infrastructure (OMNI) [44] is another management solution for OpenFlow networks. OMNI performs the administration of SDN networks and can be used to dynamically configure network elements. It offers a web interface which allows the interaction between the user and network control tools. Also, OMNI provides a multi-agent system and an Application Programming Interface (API) to autonomously control the network. Agents can autonomously monitor the network and react to packet loss event in a short period of time which helps to detect the network anomalies and fix them dynamically. Other solution called SDNMP [45] combines the traditional Network Management System (NMS) with the SDN architecture. Its main objective is to allow an hybrid approach to solved the problem of management of SDN. SDNMP enables the management of physical/virtual networks, flow table, and services using the Simple Network Management Protocol (SNMP), by adding data acquisition function, data processing/storage function and view function in the controllers.

2.1.2 Slicing and network resources allocation

OpenFlow was initially created to allow researchers to run experimental protocols and technologies in a production environment. The production network became a testbed without causing negative effects on the performance of the work environment. The physical infrastructure can be used not only for a single experiment but for more than one experiments at the same time by using slicing and network resource allocation techniques. Flow Visor [9] is an OpenFlow controller built and designed for this specific purpose. It receives all OpenFlow messages generated by network devices then redirects them to the appropriate controller that can be another Flow Visor or a simple OpenFlow controller, acting like a proxy. Flow Visor offers a granular network slice control and allows a slice to be customized and defined using a combination of the transport, network and link layer addresses along with the use of union, negation and intersection functions of those attributes.

A Slice Abstraction for SDN is proposed in [46] to isolate the network traffic and physical devices. An abstraction based on VLANs and slice semantics can ensure the isolation of each traffic, allowing the use of the same physical network by many programs at the same time.

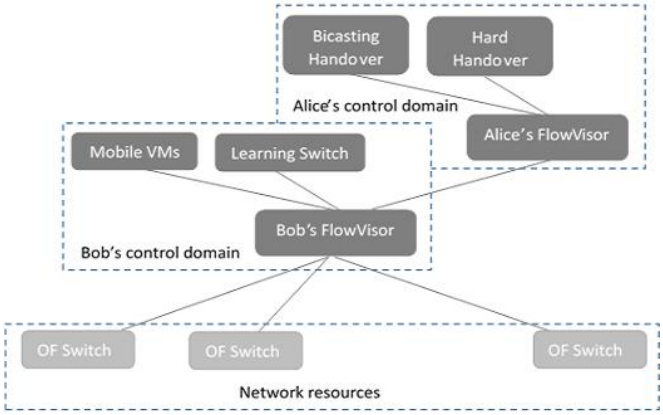

Figure 2.3: Flow Visor architecture [9] .

A prototype compiler is implemented, transforming a collection of defined slices to OpenFlow rules and associating them with their respective programs. The transformation rules generated by the compiler and applied to the traffic of a slice ensure that the slice's traffic will not travel outside the slice or affect the traffic processing of other slices, respecting in this way the non-interference and isolation of the network slices.

Traffic isolation can also be achieved at layer 2 using VLANs, but this solution is limited by the number of available VLANs. VxLAN (Virtual eXtensible LAN) [47], [48], is another layer 2 traffic isolation solution that solves the problem of VLAN limited numbers and uses IP multicast for the isolation of the network. Considering the amount of memory and CPU required for IP multicast management, an OpenFlow based method is proposed by Nakagawa et *al.* [49] to manage the IP multicast in Overlay Networks. The proposed method consists of replacing the Internet Group Management Protocol (IGMP), used for IP multicast management with OpenFlow, reducing the amount of resources required for the IGMP Snooping table and eliminating periodical Join/Leave messages. Basta et *al.* [50] introduced HyperFlex, a virtualization hypervisor that can be used to ensure the isolation of control plane for virtual SDN networks (vS-DNs). An OVS controlled by FlowVisor provides the capability to ingress policing in order to limit the control plane flow towards the hypervisor thereby, realizing the control plane isolation.

2.1.3 Network updates

Another aspect that can help achieving network management efficiency is the consistency of network updates which can be provided by SDN. Based on programmability and centralized control, SDN has the potential to overcome many problems related to the traditional network update mechanisms. An abstraction of network updates is proposed in [51] , [52], distinguishing the update consistency into two types in order to preserve the network stability while transitioning between configurations without interrupt the network functions or changing the network behavior. These two levels are per-packet and per-flow. The per-packet level assumes that the packet reaches its destination completely before or after the update occurs. On the other hands, the per-flow level provides a way for the packets belonging to the same flow to be processed according to the same version of configuration. A system called kinetic that implements this

update abstraction is also introduced in [42], showing that this abstraction brings less overhead and enables simple verification techniques for update configuration.

In [53] McGeer et *al.* proposed an update protocol developed for OpenFlow networks based on packet consistency. The goal of their protocol is to optimize the switch resources, particularly the TCAM (Ternary Content Addressable Memory), where the OpenFlow rule sets are stored, by only installing one set of rules at a time. During the transition period caused by the updates, the affected packets are sent to the controller where they will be processed and routed to their destinations. Once all switches have been updated, they will stop sending the packets to the controller and will use the new set of rules for packet processing. The protocol update introduces a trade off between switch rule space and control bandwidth when the packets are sent to the controller during the update period. Therefore, the choice of using this protocol is dependent upon which resource is available and which one we want to optimize.

The challenging task of updating the network due to virtual machine migration is discussed by Ghorbani and Caesar [54]. A heuristic approach is proposed to provide an optimal solution using the network sequence planning problem where the network initial state along with the source and destination of the migrated VMs are involved. After computing the number of feasible migration sequences and scoring them, the optimal sequence is derived. The evaluation of this approach shows that it can give a better near optimal solution that satisfies the bandwidth and loop-free constraints compared to that obtained using the naive approach.

Since upgrading the network (data plane) can raise many issues, upgrading the network control plane rises even more challenges. This problem is discussed by Vanbever et *al.* [55]. Updating the SDN controllers can be justified for different conditions that may cause the interruption of network services or non-desirable network behavior. To overcome these potential problems, there is an efficient interruption free system called HotSwap, a hypervisor that can be used to maintain a history of the network states and to help new or upgraded controller regaining its functions without interruption or without erasing the rules stored in the switches, control bandwidth, network resources and resulting in less overhead.

The update consistency problem is also discussed by Katta et *al.* [56], Wang et *al.* [57], and the proposed algorithms introduce a tradeoff between update time and rule space overhead [58]. But unlike the work done in [53], the packets do not pass through the controller to be processed. Thus, the control bandwidth and controller resources are not wasted, and the packet processing delay is reduced. The incremental update algorithm consists of managing the rule space by dividing an update for each round while satisfying the consistency priority, then selecting the optimal sequence of new rules to install. The proposed approach indeed saves the switch rule space but it also adds some update time due to incremental transition between the old and new updates.

2.1.4 Load balancing

Load balancing is one of the important forms of high availability that helps maintain the network performance during traffic peaks. Also it can be based on different criteria like Layer 2, Layer 3 and Layer 4 addresses. The network programmability with the automated SDN control can considerably improves the network load balancing function allowing the network to dynamically adapt, scale and handle the traffic load while respecting performance and availability proprieties. SDN load balancer allows one to design and implement a load balancing strategy.

Handigol et *al.*[59] presented a showcase of a load balancing system called Plug-n-Serve using the OpenFlow protocol. The system first gathers periodical information about the network state then handles incoming requests using an integrated optimization algorithm called LOBUS (LOad-Balancing over UnStructured networks). Afterwards, a path to the chosen server is installed in every switch across the route to that server, enabling the network to automatically react to the addition of a new server in an unstructured network and thus minimizing the client's response time. Aster*x [60] is a newer version of the Plug-n-Serve load balancing system which can help maximizing the client response time, not only for unstructured local area networks but also for unstructured wide areas, enabling the operators to add servers in different and arbitrary locations. As pointed out by the authors [60], new load balancing system is yet to be tested using the GENI infrastructure to evaluate Aster*x in a realistic environment of at least 3 university campuses representing a variety of multi-location clients. GENI will also help to evaluate the new integrated load balancing algorithm.

Motivated by the idea of using network wildcard matches provided by OpenFlow in order to minimize the set of load balancing rules, Wang et *al.* [61] discussed the possibility of taking advantage of the OpenFlow wildcard field to regroup the client microflow into a minimal number of load balancing rule sets. The sharing of the traffic is based on the client IP address and a weight assigned to the server replica. Such a solution can significantly reduce the number of load balancing rules but sooner or later this number will begin to increase when dividing those rules into smaller ones to adjust the load between overloaded and non-overloaded servers. The Rule space volume can decrease when the traffic load also decreases, and OpenFlow timeouts can be used to automatically remove the expired flow and then re-install the generic wildcard rules. The Round-Robin load balancing strategy is another method that can be used to manage the network traffic load. In [62], Kaur et *al.* proposed an implementation that does not require a separate load balancer device, allowing some flexibility to the considered network topology. As a result, this solution can be scaled as the number of switches and servers increases, and the client requests can be handled at the line rates. The proposed architecture consists of an OpenFlow switch network with an SDN controller and multiple servers connected to the ports of the OpenFlow switch. The requests from the clients are equally distributed to the servers are assigned on a round robin base. The load balancer selects the server with the minimum load upon receives a new request.

2.2 SDN Security Related Works

In [63], an OpenFlow-based security application for user accounting is proposed which relies on an on-demand service and network protection mechanism. The proposed approach consists of a DDoS defense mechanism, hosted in a controller, which can monitor the network traffic passing across the OpenFlow switches, themselves acting as security devices at the trust boundaries of the network to detect and prevent external attacks. The DDoS defender periodically analyzes the traffic frequency and detects attacks by comparing the monitored traffic rate against a prescribed threshold. Another user accountability approach and DDoS detection mechanism called Content-oriented Networking Architecture (CONA) is proposed in [64]. This approach consists of identifying the content requests by the clients and by keeping track of that information. The CONA scheme is able to detect and react to potential DDoS attacks when the request

frequency for a particular content exceeds a predefined limit. To prevent a DDoS attack, the controller simply limits the request frequency related to the contents targeted by that attack.

The problem of DDoS attack is also addressed in [65]. The proposed solution is based on extracting the information related to DDoS attack detection from the switches' flow table entries, and passing them to an artificial neuronal network classifier hosted in a NOX controller. The classifier then compares this set of extracted flow characteristics against the DDoS attack patterns to determine the legitimacy of the network flows. This lightweight DDoS detection approach introduces less overhead than the traditional approach. The flow will be processed in the controller in the opposite manner of the traditional approach in which the traffic is processed using inline mechanisms such as a software or hardware Intrusion Prevention System (IPS).

Although the denial of service is one of the most severe network attacks, the majority of attacks are sourced from the inside of the network. A network security architecture is proposed in [66] to restrict unauthorized access and communication between network hosts, preventing potential attacks that can be launched from the inside the network. Its design consists of using an OpenFlow switch as an access device, an SDN controller to manage and control the network and a filter responsible for checking the traffic content. Once the client is connected to a switch, it is automatically registered by the controller and has to meet a multilevel security policy in order to be allowed in the network. Furthermore, the content of the accepted flow is checked by a filter for verification and threat detection purposes. In the same vein, an unauthorized access attack can be achieved by bypassing the security rules and policy or by spoofing a legitimate identity that matches those rules and policies. A solution to defend against source address spoofing attacks (so-called Virtual source Address Validation Edge (VAVE)) is introduced by G. Yao in [67]. The proposed approach aims to improve the Source Address Validation Improvements (IETF SAVI) [68] solution, initiated in September 2010. With SAVI, the network access devices (so-called SAVI devices) do not share the binding information (IP and MAC adresses), and each SAVI device works separately. This problem is solved using an OpenFlow-based architecture in which the controller can access binding information on each device and build a global view of the network. Using this technique, a spoofed address can be detected by another device, and this was not possible in SAVI.

Many security problems can be solved by means of network management programming. Network programming features offered by the SDN concept are used to develop new security solutions. As an example, in [69], a security policy enforcement kernel for the NOX controller (so-called FortNOX) is introduced. The proposed software provides a conflict detection mechanism, security constraint enforcement, role based authentication and other security features. By providing some flow rules for verification and policy enforcement, FortNOX introduces a minimal overhead in addition to the simple flow processing overhead. But most important, FortNOX can be used to solve many potential conflicts caused by automatic OpenFlow based application rules and constraints.

An approach for defending against passive attacks such as network scanning is proposed by Jafarian et *al.* in [70]. It consists of hiding real IP address by frequently assigning a random virtual IP address to the hosts while the real addresses remain unchanged. An OpenFlow controller is then responsible for installing the flow rules using the real or random virtual IP address based on whether the packet source is internal or external and if it is authorized or not. This technique called OpenFlow Random Host Mutation (OF-RHM), has been shown [70] to

prevent attacks such as zero-day and network scanning.

OpenFlow also provides many ways to protect and secure a network either by applying a granular and flexible security policy or by dynamically updating the rules and adapting them to network changes. In [71], Hu et *al.* introduced an OpenFlow based firewall called FlowGuard, to address the challenges raised by the frequent flow, data space changes and the dynamicity of the SDN networks. The proposed approach consists of a method for detecting firewall policy violations caused by the creation of new flows. Another verification method called FLOWVER has been proposed by Son et *al.* [72]. It provides a mechanism to whether the newly created OpenFlow rules are following the security policies already in place, and if that is not the case, these rules are modified to conform to the non-bypass security properties.

The increased adoption and deployment of mobile networks and devices reveals other new network security problems. Examples of these problems are discused as follows. Malware attacks can be solved by adopting an OpenFlow based solution as discussed in [73]. The proposed solution consists of a modular and flexible intrusion detection approach that focuses on monitoring the behavior and the frequency of connections initiated by potentially infected hosts or malware. Intrusion detection in the mobile environment is also studied by Skowyra et *al.* [74], where a Learning Intrusion Detection System (L-IDS) is introduced using an OpenFlow architecture to handle the user's mobility, and using the SDN knowledge of the network state to strengthen the network security policies by reconfiguring the network and generating new security rules.

An Identity-based cryptography protocol is proposed to secure the communication in a distributed SDN with multi-domains [75]. The secure SDN communication is established by means of a symmetric session key, Identity-based Authenticated Key Agreement (IB-AKA), which can then be used to apply southbound security communications. The SDN controllers act as a Private Key Generator (PKG) for the switches that are located in their domains. The PKG of IBC generates only the private keys derived from the users or network identities. For the secure east/west-bound communication, the SDN controllers obtain their private keys from different PKGs. In this case, the distributed SDN shares an inter-domain data store that can be located within a different domain with a different PKG. In [76] Lara and Ramamurthy proposed a Policy-based network security management scheme (called OpenSec) using SDN. OpenSec provides a wide set of security services and access control, as well as a reactive component to security alerts. This OpenFlow-based framework can serve as a virtual layer between the OpenFlow controller and the users. In this framework, the security policies are automatically converted into a set of rules that are set up into the network devices level.

2.3 OpenFlow application in Wireless networks

The large adoption and deployment of OpenFlow for wired networks has also motivated many researchers to study the OpenFlow application to improve wireless network management for increasing the wireless network performance. In this section, we present a non-exhaustive list of the main works that have been done so far in this regard, discussing somme novel OpenFlow based approaches for wireless networks. The first deployment of OpenFlow on a wireless platform is called OpenRoad [77]. It have been developed with the objective of providing a test-bed for researchers to evaluate their experimental protocols in a real wireless

environment. OpenRoad is composed of multiple vendors switches including HP and NEC, and different wireless technologies such as WiFi and WiMAX. It also provides a shared environment where many experiments can be run concurrently in the production network.

After its deployment, OpenRoad has shown its usefulness by being a real wireless test environment for some of the first OpenFlow based applications for wireless networks. In [78], a blueprint that describes how to build an open wireless environment that favors research and allows experiments using production infrastructure is presented. In [79], it is demonstrated how multiple wireless technologies can simultaneously be exploited to increase the quality of video streaming. This experiment involves setting a handover between WiFi and WiMAX with n-casting using OpenRoad as test environment, showing the effectiveness and feasibility of using heterogeneous wireless network devices and different technologies to boost the wireless network performance.

Another handover approach is presented in [80]. This solution consists of an architecture that combines the OpenFlow forwarding flexibility and dynamicity with wireless mesh network auto configuration, allowing the host's mobility and the migration of IP addresses without additional tunneling overhead. This approach is implemented using a NOX controller to handle the forwarding rules, the address management and node mobility, as well as to monitor and control the server to build the topology and the association databases. This implementation was tested in the KAUMesh testbed showing interesting results in terms of routing and mobility for small scale networks.

An OpenFlow based architecture for wireless sensors networks is proposed by Mahmud and Rahmani[81] aiming to improve the routing and communication within islands of wireless sensor networks. The proposed architecture uses the OpenFlow protocol for communications between access point, sensors and the controller. This prevents OpenFlow control messages from disturbing the data flows, and thus improves the routing between sensors and gateway, with the option of using alternative paths. The evaluation results show that flow-sensors enable better performance especially in case of medium to large network scales. It also provides robust routing and flow monitoring.

The use of OpenFlow in wireless mesh network (WMNs) applications appears to be more challenging due to the reliability issues that can occur between the deployed OpenFlow controller and the non-directly connected wireless mesh routers [82]. In this regard, Chung et *al.* [82] have proposed two control mechanisms, one based on in-band control and the other based on out-of-band control. Through experiments, they showed that the in-band control method is less suited for WMNs whereas the out-of-band method can provide an acceptable performance level, but requires additional hardware to be effective. In [83], another OpenFlow based control mechanism for WMNs is proposed, where the OpenFlow flexibility and the granular flow management in WMNs are exploited. In this approach, the reliability issues in multi-hop radio networks are considered by using distributed controllers based on OLSR to handle the OpenFlow control and data traffic when a control device fails. In this case, the OpenFlow is also used to manage the rest of the wireless mesh network functions.

In an attempt to introduce programmability in wireless networks, an SDN based framework called Odin is proposed in [84] by Suresh et *al.*, which enables an effective deployment of WLAN features as a service. Fully transparent to the client, Odin is built on top of an OpenFlow controller and it introduces Lighted Virtual Access Point (LVAP) abstraction by virtualizing the association state between the client and the access point. Each client has a different basic

service set identification (BSSID) which logically isolates the clients and enables simple handoff and client mobility management. Later on, Odin was used in [85] to demonstrate the usefulness of the SDN programmability feature to save energy in a wireless network. Cloud MAC is another OpenFlow based wireless architecture that uses the Virtual APs introduced in [86]. In this approach the APs are managed by a central controller and the MAC layer traffic is passed to the virtual machines provided by the cloud for processing. The use of an OpenFlow controller provides the best flexibility in adding new services. At the same time, the use of OpenFlow tunneling to forward the MAC frame between the APs and the VAPs adds an additional overhead which may result in decreasing the performance of the network, especially for probe and association responses.

It should be noted that the SDN opportunities and challenges provided by the application of OpenFlow in wireless environments have been discussed and analyzed in [87], [88], [89], along with some limitations. Examples of such limitations include: the lack of support for duty cycle and data aggregation in order to save energy, the channel isolation and load estimation problems. Those challenges can be subject of future works, and may be solved using the SDN network programmability features in conjunction with the OpenFlow protocol. Some of the advantages of adopting the SDN architecture in wireless networks have also been discussed in [87], [88], [89]. Examples of these include: using SDN architecture for achieving heterogeneous network management, quality of service and accountability, flexible network security, context awareness and other opportunities that can enhance the wireless network performance while resolving many of the existing traditional wireless network challenges.

2.4 OpenFlow based Testbeds

Prototypes and test environments have a crucial importance when it comes to research validation. It is important to run experiments in a real environment similar to the production one. In order for experiments to build their own testbeds, many factors come into consideration such as the cost, size and flexibility of the test environment. But because most network operating systems are propriety-based, most vendor and commercial network devices have specific purpose hardware, build an efficient test-bed is very difficult to achieve. Simulation with Ns2 [1], Opnet [2] and emulation [90] provide an alternative way to perform experiments. At the same time, they do not offer scalability and adaptability to all possible realistic practical scenarios. With the emergence of virtualization, isolation and slicing techniques, building flexible and customized test environments is no longer a challenging task. In this section, OpenFlow based test-beds which can be used for experiments and production networks purpose are presented.

Motivated by the idea of experiments in a production network, researchers from Stanford university proposed a tesbed called OpenRoads [79]. The deployed network allows researchers to use this testbed simultaneously for their experiments and for student to create their own wireless controllers. The network consists of 30 WiFi access points and a WiMAX base stations, and a FlowVisor is used to slice up the wireless network and separate the controllers from each other. As a result, OpenRoad deployment provides a realistic common test platform where researchers can allow other operations to take place while developing and sharing the embedded tools for

[1] Ns2 network simulator. http://www.isi.edu/nsnam/ns/.
[2] Opnet technologies. http://www.opnet.com/

their own experiments.

In [91],the OpenFlow protocol was also used to build a platform called OpenPipes, which can be used for designing, prototyping, and testing high speed networking systems. The proposed platform uses a dynamic modular approach, where modules can be implemented in both hardware and software, providing some flexibility when designing high speed networking system prototypes. In the same platform, the modules can be reused by designers, hence allowing some cost savings in terms of specialized hardware such as FPGA or ASIC. Since OpenPipe has a modular architecture, OpenFlow can be used therein to connect the different modules regardless of their locations in the network as well as their implementations (whether in hardware or software). OpenFlow can also be used to allow the migration of the modules, software, and hardware across the network while the system is running.

Another attempt to use the production network as a test environment for research experiment is presented in [9] using FlowVisor, providing strong isolation of slices by adding a layer between the control and data planes which allows experiments to run on the same deployed hardware without affecting the traffic of other experiments or of the production network. This approach enables the users and researchers to test their protocols and solutions in the same testbed, which also happens to be the production environment itself. Such approach has the advantages of saving the cost of building and maintaining a realistic and scalable test environment without incurring significant changes on the production network.

For the testing of small to moderate size networks, Mininet [92] is a prototyping environment that provides a way to build programmable networks with the support of realistic user traffic. It uses a partial virtualization approach allowing it to run on a single laptop. It also allows to create an interactive and customizable SDN networks which can be shared once the prototype has been developed. But due to its lightweight virtualization approach, Mininet cannot support the OS kernels simultaneously and does not offer the same forwarding performance as a hardware system.

To overcome the inability of establishing virtual topologies (slices) independently of the physical infrastructure (this is considered as a limitation of FlowVisor [85]), a new component called ViRtual TopologIes Generalization in OpenFlow (VeRTIGO), networks [93] has been developed as an extension of FlowVisor, which can allow the instantiation of virtual networks and improve the level of abstraction of the network virtualization layer by slicing network nodes (i.e. switches). Furthermore, VeRTIGO have been deployed within OFELIA [85], another experimental environment for designing and testing SDN networks, proving its effectiveness and flexibility in real and large scale networks setting.

An extensible graphical user Interface-based testbed called OpenGUFI is presented in [94]. It can be used to visualize the network topology, the associations between the mobile clients and the wireless access points traffic flows in real-time. Based on odin, an SDN framework for enterprise WLANs, the control of connected mobile clients and network components is enable through the SDN Controller. The goal is to enable an interaction within a wireless network via the Northbound API. OpenGUFI consists of tree components: GUFI Backend, GUFI Frontend and Odin Master. The GUFI Frontend permits to visualize and interact with the network topology. It also acts as a mediator and processing entity between itself and the REST API server of the Controller. The Odin Master is a component of the Odin framework which controls the mobile clients in a wireless network.

Chapter 3

The Internet of Things

Summary: This chapter describes the features of the Internet of Things technologies including architectures and protocols. It also provides a brief summary of some SDN approaches addressed to the next-generation Internet architectures. These approaches served as an inspiration for building our proposed SDN-based architecture for the IOT and ad-hoc networks.

3.1 Introduction

In recent decades, the Internet of Things (IoT) has been considered as an emerging research area where the objects of everyday life are able to communicate with other objects and users. But, the IoT is not a new concept, because early on it developed by a group of students at Carnegie Mellon University in 1982 [95], where the idea was an internet-connected coke machine. This first smart device was able to report its inventory, which allowed customers to check from their desks if the coke machine was loaded with cold drinks. It was already mentioned by Kevin Ashton during a presentation that he made at Procter & Gamble (P&G) in 1999 [96]. The initial concept envisioned a novel paradigm in which everything can be connected to everything via Internet. For this, the objects required radio-frequency identification (RFID) technology for gathering data such as temperature, air quality, motion detection, tracking objects and low power actuators that can switch things on and off. Ashton used the term Internet of Things to describe a system in which physical devices could be connected to the Internet using sensors.

With the explosive growth of IoT devices (e.g., mobile devices, sensors and wearables) during the last years, this novel paradigm has been attracting the interest of the research communities to build the future Internet even more immersive and pervasive. The interconnection of these devices is enabled by the latest developments in RFID, Internet protocols, communication technologies and smart sensors. Usually, the interconnected IoT devices have short range coverage and low power batteries. Besides, some wireless devices are provided with their own predefined sets of operations.

Nowadays, there are many IoT applications, such as environmental monitoring, home automation, medical and healthcare systems, transportation, energy and infrastructure management and so on. The recent developments of the IoT applications provide us a wide range of opportunities to create value added for the industry sectors and users.

One of the statistics published by Cisco forecasts that 25 billion connected things that were in use worldwide in 2015 will reach 50 billion by 2020. There are projected to be 6.5 devices

and connections per capita by 2020, up from 3.4 per capita in 2015. At the end of 2016, the mobile global data traffic reached 7.2 exabytes per month, increasing from 4.4 exabytes per month at end of 2015. The global mobile data traffic is expected to reach 49 exabytes per month by 2021, and annual traffic will exceed a half zettabyte.

3.2 IoT Architecture and protocols

For the IoT, many architecture models are under development, such as the NIST model for Smart Grid, the ITU-T model, the M2M model from ETSI or the Architectural Reference Model from the EU IoT-A project and related work such as the IETF, W3C, etc. But despite all of that, there is no consensus on a single addressing architecture will be applicable to the whole IoT [96, 97, 10, 98]. Like the Internet, the IoT will grow over the time in evolutionary manner from a variety of separate contributions. As a new phenomenon, the IoT does not have a standard architecture and it is still up for debate.

The most common architectural design to define the main idea of IoT includes three basic layers: perception layer, network layer and application layer (Fig. 3.1) [10].

- Perception layer: The perception layer consists of physical objects and sensor devices. The main task of this layer is to collect information and identify objects with the help of different devices like smart cards, RFID, readers and sensor networks. This task is accomplished depending on the type of objects (devices) used to query location, humidity, temperature, motion, chemical change in air, etc.

- Network layer: Also, called as transport layer, is responsible for the transmission of data gathered from the perception layer to the application layer. The network layer employs different kinds of technologies and protocols to ensure the data transfer, for example 2G, 3G, bluetooth, zigbee, gateway, ad-hoc, wireless/wired network or infrared depending upon the sensor devices. It can be viewed as a combination of a variety of heterogeneous networks.

- Application layer: This layer is at the top of the IoT architecture and includes the IoT applications to acquire, analyze, store and process the received data from different objects of the IoT platform. The application layer interacts directly with the end user. The applications can support all sorts of business services and industries such as: smart homes, smart cities, manufacturing, logistics, retail, environment, public safety, health care, business process modeling, business process execution, device management, authorization and many others. In this layer, all decisions related to control, security and management of the application are made for specific purpose.

All the interconnected devices of the IoT need different protocols to enable the interconnection between physical and digital worlds, allowing the information to be collected and processed in real-time. There are several communications protocols that can be used in IoT environments [96, 98]. The function of these interconnection protocols and technologies of interconnected devices on IoT networks are detailed below:

Many of these IoT communication protocols such as WiFi, LoRaWAN, Bluetooth, ZigBee and 2G/3G/4G cellular are well known, but there are also new emerging networking solutions.

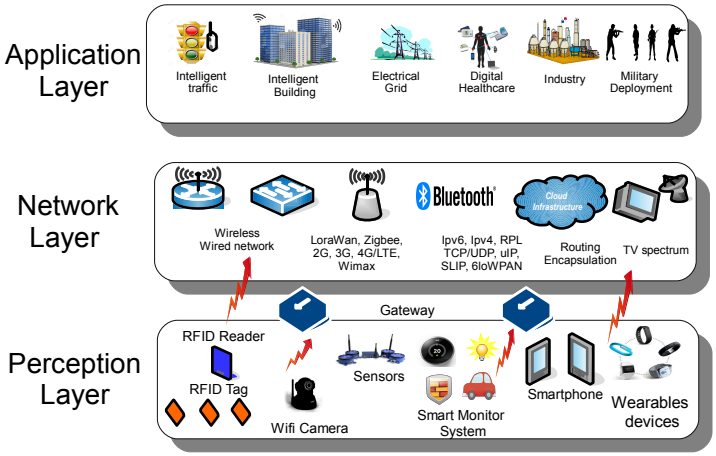

Figure 3.1: Internet of Things architecture [10].

Protocol	Standard	Frequencies	Range	Data Rates
Wifi	802.11	2.4Ghz, 5Ghz	50-125m	150Mbps-1.3 Gbps
Bluetooth LE	4.2-5.0	2.4Ghz	50-240m	1-50Mbps
LoRaWAN	LoRaWAN	various	2-15km	0.3-50kbps
ZigBee	IEEE802.15.4	2.4Ghz	10-100m	250kbps
Z-Wave	ITU-T G.9959	900MHz	30m	9.6/40/100kbit/s
Cellular	GSM/GPRS/ 2G/3G/4G /LTE-M/ Clean Slate	900/1800/ 1900/2100MHz	35km GSM - 200km HSPA	35-170kps (GPRS), 120-384kps (EDGE), 384kps-2Mbps(UMTS), 600kbps-10Mbps (HSPA), 1-10Mbps (LTE)
NFC	ISO/IEC 1800-3	13.56MHz	10cm	100-420kbps
Sigfox	Sigfox	900MHz	10-50km	10-1000bps
Neul	Neul	900MHz (ISM), 458MHz (UK), 470-790MHz (White space)	10km	100kbps
LowPAN	RFC6282	2.4Ghz, ZigBee, low-power (sub-1GHz)	30-100m	20/40/250kbps.
DASH7	ISO 18000-7	433 MHz	1000m	200kbps

Table 3.1: Internet of Things Communications Technologies

As an example, Thread, which is designed especially for home automation with mesh communication. To add a new device to a Thread network, a passphrase is needed by a commissioning device to authenticate and authorize the joining device. There is no single point of failure in a Thread network. A router or border router is designated leader to make decisions. It assigns

and manages the routing addresses. If the routing Leader becomes unreachable, another router leader is automatically elected without user intervention. TV Whitespace technologies (TVWS) offer connectivity over 10 kilometers to rural communities, wireless access across campuses for connecting IoT devices. The frequency range spectrum of TVWS vary by region from 470 MHz to 790 Mhz. To meet the massive IoT requirements, a variant cellular communication technology further optimizes the coverage, device battery life, costs and capacity to support a massive number of connected devices with three technologies: enhanced Machine Type Communication (eMTC), often referred to as LTE-M and NarrowBand-Internet of Things (NB-IoT) and Extended Coverage GSM for IoT (EC-GSM-IoT). The LTE-M features higher bandwidth mobile connections, including voice and greater coverage area for IoT applications. An LTE-M network is still rolling out in North-America, with a maximum data rate of the interconnected devices of 375 kbps. NB-IoT supports ultra-low-end IoT applications with low bandwidth data connections, low power consumption, end-to-end device. It is currently focused in Europe. NB-IoT data rates range from 20 kbps to 65 kbps. EC-GSM-IoT is based in Low Power Wide Area technology (LPWA). It can be deployed via a software upgrade on the existing GSM networks. This technology addresses IoT applications in regions where LTE coverage is insufficient. The data rate of EC-GSM-IoT is 70 kbps. These proposed alternatives share spectrum with existing LTE or GSM networks.

Over time, many other IoT application protocols will emerge for the higher-level Machine-to-Machine (M2M) communication between IoT devices [98, 99]. An important aspect is that these IoT sensing applications allow interoperability among existing Internet applications without translation mechanisms or another application-oriented code. The M2M IoT communications protocols are based on the Internet TCP/UDP protocols, but UDP offers an advantage over TCP due to its low-bandwidth requirements and its easy implementation on resource-constrained devices. Table 3.2 briefly summarizes a comparison of the most frequently used protocols for M2M IoT communications.

The main idea of these IoT applications protocols has been the development of different kinds of technologies to meet specialized requirements for resource-constrained devices with limited memory, small power consumption, low data rate and high latency. Usually, the application layer uses HTTP to provide web service. However, HTTP is not suitable to set up on constrained IoT devices because of high computation complexity and energy consumption. On the other hand, many of such applications support both request/response and publish/subscribe architecture. The request/response is the most commonly used communication model. It consists of a client that utilizes GET, PUT, UPDATE or DELETE message requests to a server, specifying the desired data, and the server sends response to the client. With the publish/subscribe mechanism a client communicates directly with an endpoint in a distributed environment. This is very useful when the devices are dynamically connecting and disconnecting from the network. The client who is sending a message to one or more clients is called publisher, and the one(s) that is/are receiving the message is called a subscriber(s). Hence, the publisher and subscriber communicate between themselves, but, neither knows the existence identity of the other. This model involves a third component called broker, which classifies in topic the receives data from the publisher, then distributes the sensory data to subscribed consumers interested in the topic.

Protocol	Transport	Messaging	Security	QoS	Power Consumption
COAP	UDP	Request/Response	DTLS	✓	Less
MQTT MQTT-SN	TCP/UDP	Request/Response Publisher/Subscriber	SSL	✓	Medium
XMPP	TCP	Request/Response Publisher/Subscriber	SASL/SSL	✗	High
AMQP	TCP	Publisher/Subscriber	SSL/TLS	✓	Medium
DDS	UDP	Request/Response Publisher/Subscriber	DTLS	✓	High
AZURE-IOT	AMPQ or Http-s/TCP	Request/Response Publisher/Subscriber	SSL/TLS	✓	Medium
REST	TCP	Request/Response	SSL/TLS	✗	High
WebSocket	TCP	Request/Response	TLS	✗	Low
ZeroMQ	UDP	Request/Response Publisher/Subscriber	TLS/DTLS	✓	Medium
LwM2M	UDP	Request/Response	DTLS	✓	Medium

Table 3.2: Internet of Things Communications Protocols

3.3 IoT solution based on SDN

Due to the IoT explosive growth, managing a huge amount of data collected and generated by these devices can become a difficult task for the network administrator. Researchers and developers have focused the interest on emerging technologies such as SDN and Network Functions Virtualization (NFV) to help overcome the potential IoT challenges. This section provides an overview of the existing proposed approaches for the IoT, WSN and cellular networks based on SDN.

3.3.1 SDN architectures for the IoT

In [100], authors have suggested a framework to manage IoT devices by using machine-to-machine technology and the benefits of SDN. The proposed framework consists of four components: The Controller, M2M nodes, Gateway and General Nodes. The controller is responsible for learning, building and refining the networking traffic. M2M nodes are mainly support devices and networks using M2M communications. The devices that cannot support M2M communications are connected by a gateway. This gateway uses a Transform Function (TF) to convert the commands in order to operate under its own independent network protocol. Another approach is presented by Jararweh et *al.* [101], which underline the need of heterogeneity between different IoT objects and applications. To adapt both technologies they suggest a high-level architecture which deals with the integration of the cloud with IoT. The core of this framework includes an IoT controller, SDN (SDN-C) controller, SDN storage (SDStore) controller and SDN security (SDSec) controller. Two components are in charge of organization and collection of the data: the sensor network cluster and the database pool cluster. From the network cluster, a set of sensors forwards the information gathered to a sensor board which is an IoT bridge between devices and applications. The collected information is stored and classified

by type into the database pool cluster. Then, the responsibility to grant only authorized access based on the authentication and authorization process is made by the SDSec controller. Once the authentication process is complete, the IoT-C apply communication rules and select the target device. The SDN-C is linked to the IoT-C to calculate the path toward the destination devices. In addition, it makes the forwarding and routing decisions. In this way, all these rules are stored into the SDStore, including other applications used on the framework.

Zhijing et *al.* [102] provide an SDN architecture for IoT, based on four layered architecture: tasks, services, flow&network, communication layer, that collects information from the IoT multi-network environments then stores it into databases. The first operation is to define the task to be realized, then mapping task/service specifies the devices and applications used to complete the task. The network layer makes any routing decisions at application flows and determines which network can be used for this operation. All network decisions are taken from the communication layer which communicates with the SDN Controller in order to optimize the selection of the appropriate IoT devices. Another complementary architecture is presented in [103]. The authors proposed a framework called UbiFlow, which provides the integration of the SDN and the IoT. Ubiflow uses distributed SDN controllers to manage ubiquitous flow control in an urban multi-network environment. This framework divides the IoT network into multiple geographic partitions represented by a cluster. Each partition has its own controller with a view only in its partition. The coordination between distributed controllers is done with a handover process and each IoT device is assigned to the SDN controller using a distributed hashing algorithm.

Nevertheless, the authors in [104] argue that scheduling is the key to making wireless networks deterministic. This work is focused on supporting the deterministic Time Slotted Channel Hopping (TSCH) networks for the best effort traffic, where the centralized scheduling is relying to an SDN controller. Instead of openflow, in this framework they use the Deterministic Networking (DetNet), to allow the controllers to communicate the controllers with the EUs, modifying the standardized SDN principles.

3.3.2 SDN for cellular networks

Based on SDN, Erran et *al.* [105], proposed an architecture called CellSDN to simplify the management on cellular data networks. Using SDN applications, specific attributed policies are applied over each User Equipment (UE) providing fine grain control over the LTE network. The switches support a local control agent for deep packet inspection based on application, such as video, peer-to-peer, web and VoIP traffic. Performing these operations with local agents increases the scalability and reduces excessive load on the controller, providing fast response to critical events. This work was extended in SoftCell [106], a proposed architecture based in four main components: the controller, access switches, core switches and middle-boxes. The policy rules such as firewalls or transcoder for multimedia transfer defined on the controller at the switch-level are implemented through middle-boxes. Traffic from UEs is fine-grained classified for every access switch which is co-located in the BS. Over the access switch also runs a local agent that caches specific packet classifiers from EUs to process flows locally, minimizing the overload on the controller. Core switches consist of gateway switches connected to the Internet. From there it can forward traffic through the appropriate middle-boxes. With this architecture, the use of specialized network elements such as Serving Gateways (S-GWs) and Packet data

network Gateways (P-GWs) are not necessary on LTE networks.

An approach that underlines an SDN-based cluster architecture can be found in [107]. This solution is proposed for a cellular network where the cellular area is divided into clusters. Each cluster is managed by an SDN Controller. A cluster head controller is responsible to manage the radio access that established the UE session and makes the load monitoring flexible. For the coordination between cluster heads, the conceptual details are not clear. Other framework called SoftAir [108], is proposed to support wireless networks such as 5G cellular systems. The main contribution of the proposed architecture is resource-efficient performance through network virtualization, mobility management, including a collaborative and distributed traffic balancing in a cloud environment. There, the data plane consists of a software-defined radio access network (SD-RAN) and software-defined core network (SD-CN), in which SD-RAN contains a set of Software-Defined Base Stations (SD-BSs), and the SD-CN is composed of several SD-switches. On the other hand, the control plane incorporates the network servers, the network management and customized optimization tools.

An alternative hybrid SDN/SDR architecture is proposed for 5G networks by Cho et *al.* [109]. With the combination of a cross layer between SDN and SDR this architecture can be used to exploit frequency spectrum and link information in a 5G network. The SNR layer provides the connection of the devices to support access to radio spectrum usage. All data sent and received will be stored in the SDN layer, which includes a heterogeneous network. Then it is enabled to forward the information to a controller. The cross-layer controller is a component that receives information from both layers. It is used to response user requests for access to spectrum resources and has an available band to access. Also, the traffic flow information of the band is monitored in the cross-layer controller, providing a better management of users asking for authorization access. This framework lacks 5G testing simulation tools because at the present there does not exist a developed open-source test package for this kind of technologies. Another relevant approach that merges SDN/SDR technologies for IoT networks is given in [110]. SDN follows its architectural principles by decoupling the data from the control plane and SDR is implement in a base BS, maintaining the radio status information on the control plane. Openflow is deployed on the control plane to perform the radio control on the BS and cognitive edges (CE) which have a global view of the radio spectrum.

3.3.3 SDN architectures for WSN

An idea for managing WSN with SDN is proposed by Luo et *al.* [111], with the concept of Software-Defined Wireless Sensor Networks SD-WSN. This study is based on the idea that each sensor node supports OpenFlow, and sensors should be able to recognize the flow table entries. This architecture proposes a separation between data and the control plane. The Sensor OpenFlow (SOF) is a communication protocol between the control plane and data plane. The data plane contains sensor flow packet forwarding, and the control plane is a controller for performing routing and QoS network control. Another work by Zeng et *al.* in [112] has studied the evolution of Software-Defined Sensor Networks, integrating sensor nodes into cloud computing using SaaS. This model is named Sensing-as-a-service and combines the sensing data with existing cloud services such as mashup services. The controller is a sensor controller with SDN functionalities, and it is provided with a local database to store sensed data. In this work, the authors assume that every sensor node is able to deliver packets to a sensor control server.

Gante et *al.* [113] presented a Base Station (BS) architecture for WSN based on SDN with a review of benefits of this technology. The authors mentioned the use of an SDN controller as a BS in WSNs, but did not present any detail about communication between sensor nodes. The controller can determine the best routing, forwarding decisions and insertion of these decisions into sensor node flow tables. In others words, the sensor nodes do not make routing decisions, they only forward and drop packets according to the rules set by the controller BS. Contanzo et *al.* [87] analyze the opportunities and challenges of SDN in IEEE 802.15.4 networks, implementing a scenario called SDWN. The controller is executed into sink nodes, and, in order to communicate, each node must learn a path (as convenient as possible) to reach the controller. The controller periodically generates a beacon packet to send to the nodes. Also, the nodes store the list of nodes from which they have received a beacon packet. All discovered neighbors will be linked frequently to the sink node using a packet called a report packet.

TinySDN [114] is another proposed approach which incorporate multiple controllers on the WSN and locate one of them as close as possible to the end nodes. The framework has two components:

- the SDN-enabled sensor node, which includes an SDN switch and an SDN end device where the data plane is programmed
- the SDN controller node, where the control plane is programmed.

The main focus is to reduce the latency and energy consumption of WSN. But in addition, this solution uses a modified version of the OpenFlow protocol to support customizable flow tables on each sensor node.

The traditional network protocols and equipment are not designed to support a high level of scalability or high amounts of traffic and mobility. In this section, we provide an overview of some related works in this research area such as: SDN architectures for the IoT in [100, 101, 102, 103, 104], the SDN for cellular network in [106, 107, 108, 109, 110], or a combination with WSN field [87, 111, 112, 113, 114]. A representative summary of existing SDN based framework Solutions for IoT is presented in Table 3.3.

Our approach consists of Software Defined Clustered Sensor Network (SDCSN). Clustering consists of organizing the network into groups of nodes, in a hierarchical structure. Each cluster is managed by a cluster head. To develop this architecture, the SDN controller is placed on the cluster head [17], which is the most powerful node of the cluster. The proposed architecture can be used in a context of wired, wireless network (including IoT devices) and ad-hoc nodes. The following part describes our approaches and details the experimental results.

Framework	Target Network	Experiment Tools	Summary
SDN M2M [100]	IoT	-	Integration solution between SDN and IoT for M2M communication.
DetNet SDN [104]	IoT	PCE 6TiSCH DetNet	An SDN-IoT deterministic networks based on DetNet and 6TiSCH
SDIoT [101]	IoT	-	A middleware based on Software-defined network, storage, security to simplify the IoT management.
UbiFlow [103]	IoT	Floodlight OpenvSwitch GENI ORBIT OMNeT++	A software-defined IoT framework for management in urban multinetworks.
SDN LTE [107]	Cellular	Mininet	Divide cellular areas into clusters, each cluster is managed by an SDN controller.
SoftAir [108]	Cellular 5G	Openflow CPRI	Flexible Architecture for 5G wireless system exploiting cloudification and network virtualization.
CellSDN [105]	Cellular LTE	OpenRoad FlowVisor	Scalability and flexible slicing with SDN, to manage cellular data networks.
SoftCell [106]	Cellular LTE	Floodlight Openflow Cbench	Fine-grained traffic policies for LTE.
SDN/SDR [109]	Cellular 5G WiMAX LTE WIFI	MATLAB	Combination of SDN/SDR in a cross layer for 5G network.
SDN D2D [110]	Cellular 5G	NS3	Hybrid D2D architecture to integrate smart city applications with 5G network.
SDWN [87]	WSN	NOS	Apply SDN into IEEE 802.15.4 networks
SD-WSN [111]	WSN	Openflow OTA	Make compatible OpenFlow with sensors node.
SD-SN [113]	WSN	Openflow	SDN controller is placed at the base station.
TinySDN [114]	WSN	COOJA nesC TinyOS	Multiple TinyOS considered such as sensor nodes.

Table 3.3: Summary of SDN-based frameworks

Conclusion of Part I

This part provides an overview of programmable networks. In this context, we have explored different SDN architectures, focusing on aspects such as scalability, performance, security and dependability. A discussion on how the SDN paradigm has evolved and has been used in the development of new networking applications, is also provided. The IoT will continue to grow exponentially, with a degree of complexity to manage multi-networks. For that, several research efforts are trying to provide a fusion between IoT, SDN and network virtualization. Specifically, some major components that must be investigated for the design of a proper context SDN-based network management system. The emerging concept of SDN and network virtualization becomes increasingly attractive to the researches community, it is considered a promising solution addressed toward the complex management configuration of heterogeneous multi-networks. The main idea behind SDN is to separate the control of network behavior (control plane) outside the networking devices from the forwarding network traffic (data plane), through a component software called a controller. This allows an abstraction of the network interconnecting devices, simplifying the function of forwarding devices with an API that enables flexible programmability over the entire system. It avoids the dependency on vendor-proprietary hardware with an open method to control the switches/routers in real-time through a logical centralized controller. By sharing the network infrastructure with network virtualization techniques, it is possible to reduce Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) costs.

Résumé Partie I : L'Architecture du SDN et l'IoT

Version Française

Introduction

Depuis quelques années, le concept prometteur du SDN a énormément fait évoluer le monde du réseau. En effet, son arrivée a bousculé la façon de voir et de concevoir un réseau informatique grâce à la programmabilité de ses fonctions réseaux. Ses applications permettent alors de réduire les coûts d'exploitation par sa gestion simplifiée au niveau matériel et logiciel. Dans ce chapitre, nous approfondissons les connaissances sur l'architecture du SDN, ainsi que sur les détails techniques afin de déployer le protocole OpenFlow. Dans la suite de cette partie, la façon dont le paradigme SDN a évolué et peut être utilisé dans le développement de nouvelles applications de réseau a été discutée. Une de ses application majeures concerne l'Internet des Objets qui croît de façon exponentielle au cours du temps. Ce phénomène amène une certaine complexité dans la gestion des systèmes des réseaux hétérogènes. Pour pallier à cela, nous avons fait de la fusion de l'IoT, du SDN et des techniques de virtualisation des systèmes du réseau nos objectifs de recherche.

Architecture du SDN

Le découplage du plan de contrôle de chaque paquet dans un réseau et le flux de données dans l'architecture SDN n'est pas un nouveau concept. Par exemple, c'est un élément déjà utilisé sur la technologie MPLS. Il constitue également une caractéristique intégrée sur de nombreux réseaux Wi-Fi existants. Le concept émergeant du SDN à travers son développement renvoie de moins en moins au découplage du plan de contrôle du plan de données. Au lieu de cela, il se concentre davantage sur la capacité à fournir des interfaces de programmation au sein des équipements des réseaux.

L'architecture du SDN, représentée dans la Figure 3.2, est en fait composée de trois couches: la couche applicative, la couche de contrôle et la couche infrastructure. Chacun d'entre eux communique avec une couche adjacente à l'aide des interfaces de programmation *NorthBound* (NBI) ou *Southbound*. L'interface *NorthBound* correspond à Application-controller plane interface (A-CPI), et l'Interface *Southbound* correspond à Data-controller plane interface (D-CPI) définie dans l'architecture SDN et en même temps normalisé par le *Open Networking Foundation* [3]. La couche d'application représente un ensemble d'applications du SDN qui permettent la programmabilité de la couche "plan de contrôle" en utilisant l'interface NBI. La couche du plan de contrôle est constituée d'un ou plusieurs contrôleurs du SDN qui se coordonnent entre

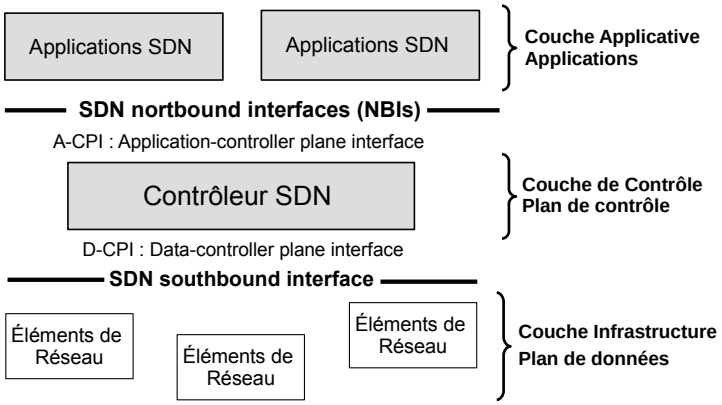

Figure 3.2: Couches de l'architecture du SDN [3].

eux, et avec les autres couches du SDN à l'aide de différentes interfaces et protocoles spécifiques. Les contrôleurs SDN sont également en mesure d'installer le routage et les règles de sécurité dans les périphériques réseaux et de les configurer depuis un contrôleur distant. Enfin, le plan de données SDN définit les équipements du réseau et implémente certaines décisions telles que la transmission et le traitement du trafic. Le protocole le plus couramment utilisé pour la communication entre les périphériques réseaux et le contrôleur est OpenFlow [22]. Ce protocole est normalisé par la *Open Networking Foundation*[1] et il est adopté par de nombreux fournisseurs TIC. En outre, OpenFlow permet au contrôleur de gérer à distance, de configurer et d'activer les commutateurs OpenFlow compatibles, afin de lire les états des équipements du réseau, ainsi que collecter les informations et les statistiques du trafic. Avec ces informations, le contrôleur a une vue globale du réseau. Un autre protocole alternatif à OpenFlow proposé par Cisco est OpFlex. En collaboration avec des partenaires de l'industrie et les communautés Open Source, Cisco offre un plugin compatible avec deux composants SDN Open Source: le contrôleur OpenDaylight Helium et le commutateur virtuel Openvswitch. Ce protocole est basé sur deux éléments clés, le *Policy Repository* et *Policy Element*. Les politiques sont définies dans le *Policy Repository*. Ensuite, elles sont résolues de manière asynchrone par OpFlex *Policy Element*. En outre, le Endpoint Registry (ER) permet d'enregistrer les informations sur l'état opérationnel des dispositifs finaux dans le réseau. Les statistiques et les événements sont remontés via l'entité logique Observer. Le périphérique ou l'hôte est connecté via le *Policy Element*, habilitant les noeuds à prendre plus de décisions sur la façon d'exécuter les politiques de sécurité. En général, les politiques sont définies dans un référentiel de politique dans le contrôleur, puis le protocole OpFlex est utilisé pour communiquer et appliquer ces règles et politiques à l'ensemble des éléments distribués aux périphériques réseau.

Depuis l'émergence du SDN, de nombreux travaux au sein de la communauté scientifique ont étudié l'architecture SDN [22, 23, 24], en se concentrant sur la façon dont il peut aider à maintenir et à améliorer les performances du réseau, tant que les fonctions de contrôle du réseau sont hébergées dans un emplacement distant séparé du dispositif qui effectue des fonctions de transfert. D'autre part, l'utilisation d'un seul contrôleur SDN représente un point critique du

[1] https://www.opennetworking.org/

réseau. Un risque potentiel est lié au déni de service (DoS). Par la suite, nous présentons quelques travaux liés aux performances de l'architecture SDN. En outre, nous discutons le défi de centraliser le plan de contrôle et de permettre aux périphériques physiques d'être distribués à travers le réseau.

Dans [25], la performance de l'architecture SDN a été évaluée à l'aide d'un modèle OpenFlow simple pour analyser la vitesse de transfert et la probabilité de supprimer un paquet. Cette étude montre que le temps total d'acheminement d'un paquet dépend de la vitesse de traitement. En outre, la probabilité la plus élevée d'obtenir de nouveaux flux dépend du délai de transfert du paquet. Les résultats obtenus suscitent deux questions : *à quel point l'évolutivité peut être manipulée avec un seul contrôleur*, *où dois être placé le contrôleur*?

Heller et *al.* [24], ont utilisé 100 topologies de réseau d'accès sans fil accessibles au public afin de quantifier l'effet du placement du contrôleur. Les auteurs ont proposé une solution pour mieux comprendre les questions fondamentales, posées par Jarschel et *al.* [25], sur la comparaison du positionnement du contrôleur par rapport à la latence, aux défauts de découverte et a l'efficacité de propagation d'événement. L'étude montre qu'il n'existe pas de règle spécifique pour guider le placement du contrôleur ou pour déterminer le nombre de contrôleurs utilisés. Les réponses dépendent des limites de réaction souhaitées, des choix des métriques et de la topologie du réseau.

OpenFlow est le protocole le plus couramment utilisé pour la mise en place du réseau SDN. Souvent, les termes Openflow et SDN sont utilisés de manière interchangeable dans la littérature, mais il est important de comprendre la différence entre ces termes. En principe, le terme SDN est désigné pour le découplage du plan de contrôle et du plan de données tandis qu'Openflow est un protocole qui permet de contrôler les périphériques réseau. La fonction principale d'OpenFlow est de gérer la communication entre les contrôleurs SDN et les noeuds du réseau. Ceci est probablement le protocole le plus connu sur l'interface sud (*southbound*). Cependant, OpenFlow n'est pas le seul protocole disponible et utilisé pour les southbound API du SDN. Il existe d'autres interfaces de programmation telles que : Opflex, Lisp, Netconf, IEEE P1520, ForCES, SoftRouter.

Architecture de l'Internet des objets

Au cours des dernières années, avec l'accroissement du nombre d'objets connectés, l'Internet des objets (IoT) a été considéré comme un domaine de recherche émergeant où les objets utilisés dans la vie quotidienne peuvent communiquer avec d'autres objets et leurs utilisateurs. Ce nouveau paradigme a attiré l'intérêt de la communauté scientifique pour construire le future de l'Internet encore plus immersif et pervasif. L'interconnexion de ces dispositifs se fait grâce aux derniers développements de RFID, des protocoles Internet, des technologies de communication et des capteurs intelligents. Habituellement, les périphériques IoT interconnectés ont une couverture à courte portée et des batteries à faible puissance. En outre, certains périphériques sans fil sont équipés de leurs propres ensembles d'opérations prédéfinis.

À l'heure actuelle, il existe de nombreuses applications de l'IoT, telles que la surveillance de l'environnement, le domaine médical, les maisons et villes intelligentes ou encore le suivi, la logistique et la gestion de l'énergie et de l'infrastructure industriel, l'informatique décisionnelle (business intelligence), etc. Les récents développements des applications de l'IoT nous offrent un large éventail de possibilités afin de créer de la valeur ajoutée pour les secteurs industriels

et les utilisateurs finaux.

Pour l'IoT, de nombreux modèles d'architecture sont en cours de développement, par exemple : le modèle NIST pour Smart Grid, le modèle ITU-T, le modèle M2M d'ETSI ou le modèle de référence architecturale du projet IoT-A de l'UE et des autres travaux tels que l'IETF, W3C, etc. Mais, malgré tous ces modèles en développement, il n'existe pas encore de consensus sur une seule architecture applicable à l'ensemble IoT [96, 97, 10, 98]. Comme l'Internet, l'IoT sera développé au fil du temps de manière évolutive à partir d'une variété de différentes contributions. En tant que nouveau phénomène, l'IoT n'a pas une architecture standardisée, car il est en cours de discussion.

La conception architecturale la plus couramment utilisée pour définir l'idée principale de l'IoT comprend trois couches de base: la couche perception, la couche réseaux et la couche application (Fig. 3.3) [10].

- Couche Perception: La couche de perception est composée d'objets physiques et de capteurs. La tâche principale de cette couche est de collecter des informations et d'identifier des objets à l'aide de différents dispositifs comme les cartes à puce, les RFID, les lecteurs et les réseaux de capteurs. Cette tâche est effectuée en fonction du type d'objets (périphériques) utilisé pour mesurer l'emplacement, l'humidité, la température, le mouvement, les changements chimiques dans l'air, entre autres.

- Couche réseaux: Aussi appelée la couche de transport, elle est responsable de la transmission des données récupérées de la couche perception vers la couche application. La couche réseaux utilise différents types de technologies et protocoles pour assurer le transfert de données, par exemple 2G, 3G, bluetooth, zigbee, passerelle, ad-hoc, réseau sans fil/filaire ou infrarouge en fonction des capteurs. Il peut être considéré comme une combinaison d'une variété de réseaux hétérogènes.

- Couche application: Cette couche est placée au niveau le plus haut de l'architecture IoT et comprend les applications IoT pour obtenir, analyser, stocker et traiter les données reçues à partir de différents objets de la plateforme IoT. La couche d'application interagit directement avec l'utilisateur final. Les applications peuvent supporter toutes sortes de services et d'industries commerciales tels que: les maisons intelligentes, les villes intelligentes, la logistique, le commerce, l'environnement, les systèmes publics, la santé, les appareils électroménagers, la gestion des périphériques, les autorisations et bien d'autres. Dans cette couche, toutes les décisions relatives au contrôle, à la sécurité et à la gestion de l'application sont faites de manière spécifique.

L'ensemble des périphériques IoT interconnectés ont besoin de différents types de protocoles pour permettre la communication entre les mondes physique et numérique, ce qui permet de recueillir et de traiter des informations en temps réel. Il existe plusieurs protocoles de communication qui peuvent être utilisés dans les environnements de l'IoT [96, 98]. Les principales fonctions de ces protocoles d'interconnexion et ses technologies pour interconnecter les réseaux IoT sont détaillées ci-dessous:

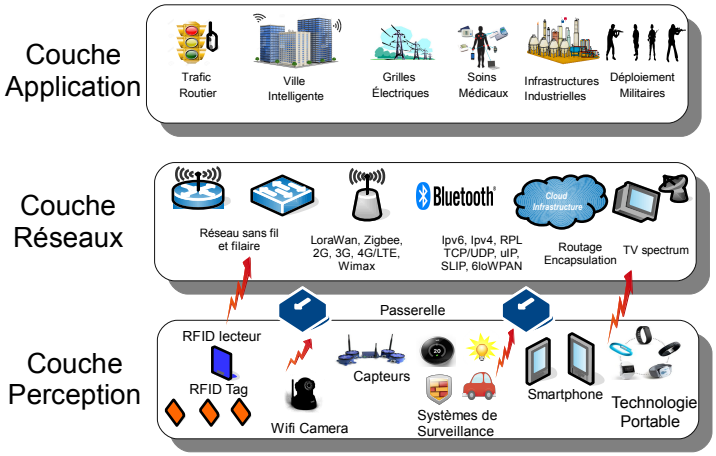

Figure 3.3: Internet of Things architecture [10].

Protocole	**Standard**	**Fréquences**	**Couverture**	**Débit**
Wifi	802.11	2.4Ghz, 5Ghz	50-125m	150Mbps-1.3 Gbps
Bluetooth LE	4.2-5.0	2.4Ghz	50-240m	1-50Mbps
LoRaWAN	LoRaWAN	various	2-15km	0.3-50kbps
ZigBee	IEEE802.15.4	2.4Ghz	10-100m	250kbps
Z-Wave	ITU-T G.9959	900MHz	30m	9.6/40/100kbit/s
Cellular	GSM/GPRS/ 2G/3G/4G /LTE-M/ Clean Slate	900/1800/ 1900/2100MHz	35km GSM - 200km HSPA	35-170kps (GPRS), 120-384kps (EDGE), 384kps-2Mbps(UMTS), 600kbps-10Mbps (HSPA), 1-10Mbps (LTE)
NFC	ISO/IEC 1800-3	13.56MHz	10cm	100-420kbps
Sigfox	Sigfox	900MHz	10-50km	10-1000bps
Neul	Neul	900MHz (ISM), 458MHz (UK), 470-790MHz (White space)	10km	100kbps
LowPAN	RFC6282	2.4Ghz, ZigBee, low-power (sub-1GHz)	30-100m	20/40/250kbps.
DASH7	ISO 18000-7	433 MHz	1000m	200kbps

Table 3.4: Protocoles de communication pour l'Internet des objets

Des solutions pour l'IoT basée sur le SDN

En raison de la croissance exponentielle de l'IoT, la gestion de grands volumes de données collectées et générées par ces appareils peut devenir une tâche difficile pour l'administrateur du

réseau. Les chercheurs et les développeurs ont concentré leurs intérêts pour des technologies émergentes telles que le SDN et la virtualisation des fonctions réseau (NFV) pour aider à mieux traiter les potentiels challenges de l'IoT. Ces technologies évoluent à travers le temps, et avec son évolution, différentes approches et techniques ont été proposées pour les réseaux IoT basés sur le SDN. L'émergence des technologies réseau permet d'exploiter de nombreuses approches qui peuvent être prises en considération pour le développement de la nouvelle génération d'Internet. Les différentes plateforme SDN proposée pour l'IoT sont résumées dans le tableau 3.5.

Les protocoles et l'équipement de réseau traditionnels ne sont pas conçus pour supporter un passage à l'échelle avec un grand volume du trafic ou une forte mobilité. Dans cette section, nous présentons un aperçu de certains travaux connexes à ce domaine de recherche tels que: Des architectures du type SDN pour l'IoT dans [100, 101, 102, 103, 104], le SDN pour le réseau cellulaire dans [106, 107, 108, 109, 110], ou une combinaison avec WSN field [87, 111, 112, 113, 114]. Notre approche consiste en un cluster défini par logiciel pour les réseaux de capteurs (SDCSN). Le regroupement consiste à organiser le réseau en groupes de noeuds, dans une structure hiérarchique. Chaque cluster est géré par un cluster head. Pour développer cette architecture, le contrôleur SDN est placé sur le cluster head [17], car il est le noeud contenant le plus ressources du cluster. L'architecture proposée peut être utilisée dans un contexte de réseau filaire, sans fil (incluant les appareils IoT) et des noeuds ad-hoc.

Plateforme	**Réseau Ciblé**	**Simulation Outils**	**Résumé**
SD-WSN [111]	WSN	Openflow OTA	Étudie la compatibilité du OpenFlow avec des capteurs.
SD-SN [113]	WSN	Openflow	Met le Contrôleur du SDN à la place d'une base station.
SDWN [87]	LR-WPAN	NOS	Applique SDN dans les réseaux IEEE 802.15.4
TinySDN [114]	WSN	COOJA nesC TinyOS	Multiple TinyOS considérés comme des capteurs.
SDN M2M [100]	M2M	-	Solution d'intégration entre SDN et IoT pour communication M2M.
DetNet SDN [104]	Generic IoT	PCE 6TiSCH DetNet	Un réseau SDN-IoT basée sur DetNet et 6TiSCH
SDIoT [101]	WSN	-	Un middleware basée sur le SDN, stockage, sécurité pour simplifier le management de l'IoT.
UbiFlow [103]	Cellular Wifi WiMAX	Floodlight OpenvSwitch GENI ORBIT OMNeT++	Un plateforme software-defined IoT pour le management des multi-réseaux urbaine.
SDN LTE [107]	Cellular	Mininet	Divise cellular zone en clusters, chaque cluster est géré par an contrôleur du SDN.
SoftAir [108]	Cellular 5G	Openflow CPRI	Architecture Flexible pour les systèmes wireless 5G qui exploite le cloud computing techniques et la virtualisation du réseau.
CellSDN [105]	Cellular LTE	OpenRoad FlowVisor	Scalabilité et automatisation flexible avec le SDN, pour gérer les données dans le réseau cellulaire .
SoftCell [106]	Cellular LTE	Floodlight Openflow Cbench	Gestion à grain fin défini par politiques pour le réseau LTE.
SDN/SDR [109]	Cellular 5G WiMAX LTE WIFI	MATLAB	Propose une combinaison de SDN et SDR pour le réseau 5G.
SDN D2D [110]	Cellular 5G	NS3	Hybride D2D architecture pour intégrer les applications de ville intelligentes avec 5G.

Table 3.5: Résume de plateformes basées sur le SDN pour l'IoT

Part II
New Architectures for Ad-hoc Networks and IoT

Chapter 4

SDN-Based Architecture For Ad-hoc Network

Summary: In this section, we will present a brief summary of ad-hoc networks and describe the suggested SDN-based architecture to address the flexible routing management of embedded devices. The goal of the implementation is to allow the development of one of the first SDN-based solution approaches for ad-hoc networks. In addition, we extend this approach to the heterogeneous network concept, including devices with and without SDN capabilities which are organized into domains. The content of this chapter corresponds to our publications in [12, 13, 15].

4.1 Introduction

An ad-hoc network is a group of mobile devices that communicate directly with each other, using a common wireless channel without any fixed infrastructure support. This allows to setup a temporary network via peer-to-peer communication without the need of access points or routers. This type of network is also known as mobile ad-hoc network (MANET). In a MANET, the nodes or devices are responsible to organize themselves dynamically. Due to the absence of a fixed infrastructure, it discovers the best communication path between devices, providing the necessary network functionality. In such environments, mobile devices act as routing nodes where the packets are routed from the source to destination via another device in the network. All network intelligence is placed into the mobile user devices. This is in contrast to the networks with infrastructure, wherein the devices on the network communicate through a single access point, most commonly a wireless router. In cellular networks the wireless communication is made by installing base stations as access points. The following figure 4.1 shows some nodes forming ad-hoc networks and an infrastructure network.

In the past few years, ad-hoc networks have been the focus of much research and development thanks to the wide range of applications that this type of network provides. These networks are suited for many applications in which it is impossible to build a network with infrastructure supporting mobility and higher self-organizing. Some applications of ad-hoc networks such as collaborative work, military environments, personal area networking, rescue operations, emergency, industry, conference are the most popular examples of MANET. Compared to traditional computer networks, MANET has many constraints which still need to be investigated more. Some of these system constraints include limited power, memory, wireless communication bandwidth, storage space, computational power and security.

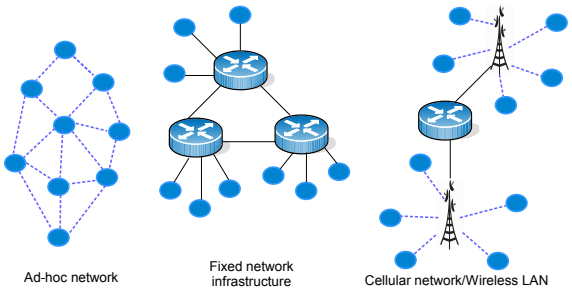

Figure 4.1: Fixed and Wireless Networks

4.1.1 Routing in Ad-hoc Network

In an ad-hoc network, the routing process between nodes depends on many factors. It includes challenging tasks, including topology changes, request initiation and calculating the best path for determining a packet route quickly and efficiently within the network. Furthermore, it is frequently the case that the route to destination with a large number of nodes connected on the network requires a high amount of traffic. Also, it can move randomly. In addition to uncertain path selection, signal interference over the wireless channel can cause a path to be disrupted.

Routing in a MANET is different from the traditional routing found in mesh networks, also known as wireless sensor networks (WSN). Usually, mesh WSN consists of one sink or base station able to manage and optimize the communications between nodes. MANET consists of a group of nodes continuously moving in any direction without a fixed central controller. Consequently, they repeatedly reconfigure their routes. Ad-hoc Network routing protocols can be classified in three categories: proactive, reactive and hybrid.

- Proactive (Table-driven): All nodes keep track of routes to all possible destination, including nodes in which no packets are need to be forwarded. In this way, when a packet needs to be forwarded, a route for transferring it is already known. In case of network topology changes, all nodes react updating their own routing tables on the status path. However, the exchange of these control messages periodically reduces the node power and increases the bandwidth consumption. An advantage of a proactive is minimal response delay whenever a route is needed.

- Reactive (On-demand): An inverse approach to proactive protocols, in which the nodes only discover the routes on demand. To send a packet, the node begins by calling a path discovery process, then store the register of this path. Thus, the nodes do not require to maintain the route to destination, as each one independently carries the information to an adjacent node or to other nodes in the network. The main benefit of reactive protocols is the small amount of data needed for information transfer. Also it does not need to save the entire network routing table. Nevertheless, a drawback is that the procedure to find the path can cause delays and the average delay time may be longer.

- Hybrid: It combines the elements of table-driven and on-demand routing protocols. Every network node is proactive and maintains the routing information about its local neighborhood routing zone, while reactive acquiring routes to destinations beyond the routing

zone. By combining these approaches appropriately, on-demand protocols facilitate the achievement of higher performance.

4.1.2 Ad-hoc Protocols

A wide range of routing protocols for ad-hoc networks has been proposed by the researcher community. This section describes briefly some of the most used ad-hoc routing protocols.

- Destination-Sequenced Distance-Vector Routing (DSDV) [115]: Based on the Bellman-Ford algorithms, each node stores a routing table with the number of hops to reach a possible destination. Also, DSDV protocol records have a sequence numbers in order to maintain consistency within the routing table. Periodically, routing updates are transmitted throughout the network by two methods: full dump, in which nodes send the entire routing table, and incremental dump exchanging only a part of updates to neighbors nodes.

- Ad-hoc On-Demand Distance Vector Routing (AODV) [116]: It reduces the number of broadcast messages considering that, AODV does not need to maintain a routing table. When a node wants to communicate with another node, only then does it discover routes on-demand and build a routing table. If the desired route cannot be found, the source node broadcasts a route request packet (RREQ) to its neighbors. The RREQ contains the IP address and the sequence number of the source/destination node.

- Dynamic Source Routing (DSR) [117]: Similar to AODV, DSR obtains the route on-demand when the transmitting node sends a packet request to its neighbors. However, instead of relying on a routing table, DSR uses source routing in which the source knows approximately the hop sequence to destination. Discovering the path information, the nodes cache routes during the route discovery.

- Temporally ordered routing algorithm (TORA) [118]: It propagates control messages only around the adjacent nodes when a link fails. To accomplish this task, the nodes maintain one-hop routes to any adjacent nodes. The limitation of TORA protocol is that it may create improper routes.

- Zone Routing Protocol (ZRP) [119]: Combines the best features of proactive and reactive routing protocols to create a hybrid approach. ZRP employs a proactive routing scheme in local neighborhoods, using Intra-zone Routing Protocol (IARP) as well as Inter-zone Routing Protocol (IERP) for communication beyond the neighborhood. The main drawback of ZRP is the large overlap of neighboring node routing zones.

The main functions of these routing protocols are characterized by addressing constrained energy, bandwidth, memory, constant mobility and by reacting quickly to network topology changes.

4.2 SDN architecture for ad-hoc Network

Routing in ad-hoc network with the SDN emerge as a new concept that allows a better management to select best routes, and also a flexible programmability on the control plane. Some approaches for routing in mesh networks by using the SDN principles have been investigated. In [80], the authors proposed an architecture which consists in the mesh OpenFlow nodes, a Monitoring/Control Server (MCS) and the controller. Each node is equipped of multiple physical wireless cards divided into two virtual interfaces. A virtual interface to control the traffic between nodes with the OLSR mesh routing protocol and a data traffic interface used for communication with the MCS and the SDN controller in the core network. MCS gets information from the mesh gateway nodes to build a topology database. Then, the controller performs the network management operations. Another solution is to use BATMAN (Better Approach To Mobile Ad-hoc Network) routing protocol for the mesh ad-hoc network combined with OpenFlow [120]. Each ad-hoc node is a raspberry pi 2 with a BATMAN interface and an embed OVS which support OF command. The path to forward the data is calculated by Dijkstra algorithm on the controller side. This approach does not provide information on how the raspberry pi establishes the communication with the controller.

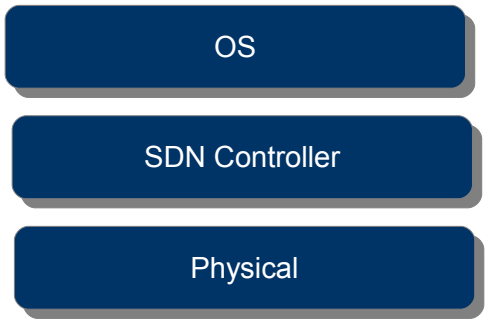

Figure 4.2: A node in ad-hoc network.

In order to enable a framework addressed to the limitations of traditional routing ad-hoc networks, we describe a policy with a rule-based approach which takes advantage of the SDN concept. The scope of ad-hoc network management is very broad, but extending the SDN can increase the possibilities of taking the control, managing behavior and maintaining a seamless connectivity of the ad-hoc nodes. In an SDN based architecture for ad-hoc Networks a node can be viewed (Fig. 4.2) as a combination of:

- Legacy interfaces: The physical layer which includes the physical interfaces or devices connected in an ad-hoc network. This is the lowest layer with hardware resources such as phones, tablets, and other physical computing elements. Every device must be an the entity providing virtual abstractions for computation.
- Programmable layer: This consists in an SDN-compatible virtual switch and an SDN controller. These virtual switches then allow the ad-hoc users to communicate with each other on the same network. The controller also bridges the connecting virtual switch to the physical devices for communication inside and outside the network. Each virtual

switch has a set of rules known as flows that are used by the SDN controller to manipulate the direct incoming and outgoing network traffic.

- Operating system and their applications: These runs applications on top of the operating system (OS), performing management tasks by utilizing the OS interface. At a high level, the OS manages network management interface applications that are needed by users of the ad-hoc network.

In this model, all legacy interfaces are connected to a virtual switch and this switch is controlled by an SDN controller, integrated with each node. On every node the controllers operate in equal interaction. They are not concerned with nodes liability for misbehaving users connecting through them, as seen in [22]. Ad-hoc nodes connect with each other through their embedded SDN compatible switch. At the same time, the SDN controller in equal interaction can enhance the security and connectivity between nodes.

The scenario depicted in Figure 4.3 illustrates the communication process between two ad-hoc nodes. We assume that when an ad-hoc node wants to send data to its neighbor, the whole communication process is managed by the embedded OVS and the controller. The physical interface is bridged to a virtual switch that which at the same time establishes a point-to-point Gre tunnel connection between both nodes. For example, when node A wishes to send data to node B, node A needs an ARP reply from node B to establish a stable next-hop link. The OpenFlow specification [1] does not describes message exchanges between nodes connected on an OpenFlow network. By identifying and evaluating a wireshark capture 4.4, we provide an overview of the communication path in an ad-hoc OpenFlow network. The process can be expressed as follows:

1. The controller may be launched in the console or the command prompt. The process start when node A sends ARP request to its embeded OVS asking for a matching OpenFlow rule at by OS level.

2. In the first instance, if the OVS does not have a flow rule, it forwards the data to the SDN controller,

3. The controller replies to the OVS, even if it does or does not have a matching rule. As both ad-hoc nodes are in equal role set by OpenFlow, and the physical interfaces are interconnected with a GRE tunnel, they can exchange information about match rules,

4. The ARP request arrives at the physical interface of Node B. Then, the request is sent through the OVS,

5. The OVS forwards the data to the SDN controller. If there is not a match flow entry installed, the controller can decide how to forward the data based on the MAC address of the incoming packet from Node A. Otherwise, the ARP request passes to the host B through the OVS,

6. When the controller has received forwarding decision, it sends the information received to the OVS,

7. Upon receiving the data, the OVS connected to node B communicates with the OS for data processing at the application level. The applications perform deep packet inspection to filter network traffic, then return the allowed flow traffic through the OVS,

8. At this stage, the OpenFlow switches establish a route link to forward data,

9. Once the OpenFlow switches have the appropriate matching rules installed, the ARP requests do not reach the controller for forwarding decisions. All received data will be processed only by the OpenFlow switches,

10. The communication path is established between node A and node B.

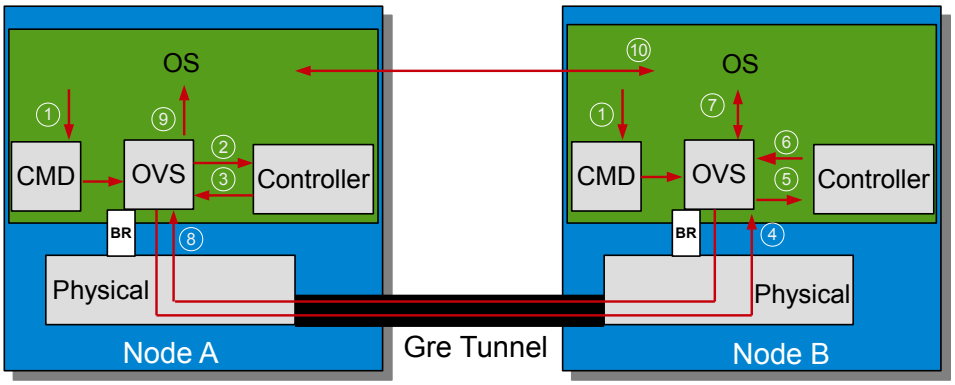

Figure 4.3: Communication path between ad-hoc nodes.

Following the experiments, we explain the procedure to enable ad-hoc OpenFlow communication. The OpenFlow messages exchanged between the controller and a networking devices have been described in detail in Section 1.3. Thus, we take into consideration the OpenFlow role Controller-switch message that can be set up to establish communication on an ad-hoc network by establishing the default role *OFPCR_ROLE_EQUAL* operation mode. Then, with OpenFlow within and among all participating devices in an SDN-based ad-hoc network, gives full access and control through the switch-interconnected resources. In the same *OFPCR_ROLE_EQUAL* role, the ad-hoc node synchronizes the management of each asynchronous message from the virtual switch. Commands such as, (Hello, PacketIn, PacketOut, PortStatus, PortMod, FlowRemoved) may be sent and modified within this role during connection setup of devices-to-switch or devices-to-devices.

However, to support redundancy, availability, and load balancing between devices, OpenFlow not only uses *OFPCR_ROLE_EQUAL* role, it also allows *OFPCR_ROLE_MASTER* and *OFPCR_ROLE_SLAVE* roles to establish communication with each of them. In *OFPCR_ROLE_MASTER*, only one controller can be master at one time. The other one has the slave role. It has full access to handle the switch resources through which all synchronous and asynchronous messages are carried to the master controller node. The *OFPCR_ROLE_SLAVE* role can only read the state of the switch resources and cannot process switch asynchronous messages except the port-status messages. Building a distributed SDN-based ad-hoc network using *OFPCR_ROLE_EQUAL* role, any packet can be examined on the control plane of the ad-hoc

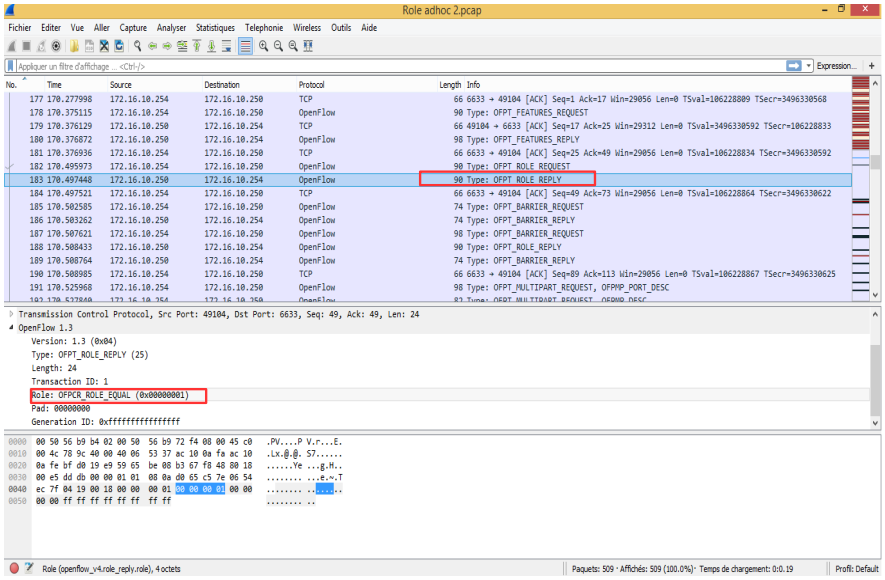

Figure 4.4: Wireshark capturing OF traffic.

node that decides whether to drop or forward the packet to the next host. The IP table entry can be also updated in real-time. The controller sends an *OFPT_ROLE_REQUEST* messages to change or query the role in the virtual switch, then it returns with *OFPT_ROLE_REPLY* messages for the current role to the controller. Figure 4.4 shows *OFPT_ROLE_REQUEST* message replies sent to the SDN controller with their corresponding roles.

Via persistent point-to-point or point-to-multipoint TCP/IP connections, the enabled OF Switches connect two or more instances among ad-hoc hosts. On the northbound side the interaction between ad-hoc hosts and the applications is done by using REST API, YANG user interface or *ovs add-flow* command line. For all type of Request-Response interaction between source/destination, when the hosts send traffic to each other, the switch asks their own controller how to handle this traffic. The controller is able to discover the topology and gives input to the routing module for known destinations and computes paths for unknown destination. Figure 4.5 shows a schematic of the simulated ad-hoc network.

Once the SDN controller learns the network topology, it can make optimal path decisions for each learned flow. Each SDN ad-hoc node exchanges OF messages to learn information about its immediate neighbors. For example the *OFPCR_ROLE_EQUAL* is set up within OF messages, and identifies automatically the appropriate entry in the flow table.

Parameters	**2 nodes**	**3 nodes**	**4 nodes**
Number of Packets OF	274 packets	376 packets	479 packets

Table 4.1: Flows Messages Exchanges

The Table 4.1 shows the number of OpenFlow packets exchanged during a topology discovery between the controller, OF switch, and OF nodes links. With an initial protocol handshake OF use the *OFPT_FEATURES_REQUEST* message for the discovery process of these links. Then, the SDN controller knows the number of active ports. Also it will be able to install the corresponding flow entries in the switch to forward the packet through the appropriate

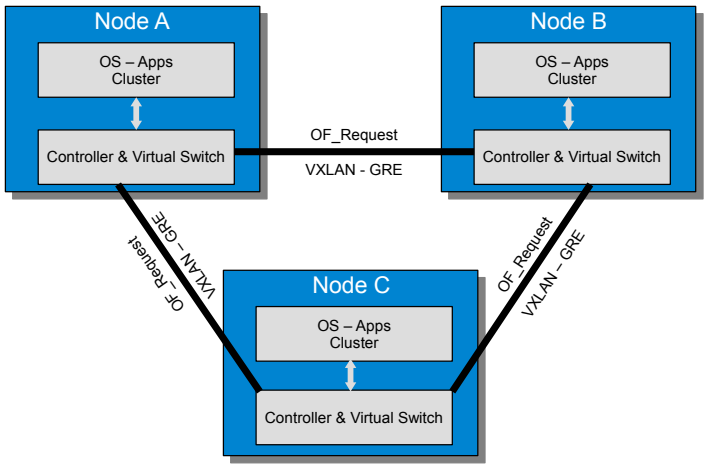

Figure 4.5: Communication patch between OF ad-hoc nodes.

in-output port. To evaluate the OF discovery process, we analyze the packet message traffic generated by nodes connected to a Virtual Switch. In a first attempt, the number of OpenFlow packet messages were 274 with 2 node interactions, 376 packets with 3 nodes and 479 with 4 nodes, only for the topology discovery process without any transmitted information or flow rules installation. The SDN controller continuously sends multiple OF packets to each switch, which could decrease the performance of the data plane. Moreover, when a set of multi-controllers is deployed in an SDN network, the discovery cost increases linearly every time that a controller is added.

With this approach, it is possible to route packets in an ad-hoc network without the use of traditional routing protocols. Via the OVS, each node maintains its own network information which is periodically shared in the form of routing tables. The nodes can perform computationally complex tasks, including energy savings, optimal bandwidth, better quality link and successful transmission. Whenever an ad-hoc node joins the network, the appropriate path is found by exchanging the OF role with the neighbor node.

4.3 SDN Extended Domain for Ad-hoc Network

One of the advantages of this new SDN based ad-hoc network architecture is its compatibility with SDN legacy network. Here, each node in the Ad-Hoc network has an embedded SDN compatible switch and an SDN controller, which can interconnect the ad-hoc network to the legacy network to build an extended SDN domain, see Figure 4.6. Moreover, by setting multi-controllers in equal interaction through the extended SDN domain, the flow tables and rules will be synchronized in the whole SDN network.

In a work like [121], the SDN domain is limited to the network with infrastructure. In this configuration, ad-hoc users have to connect through other nodes directly connected to the SDN domain. In our proposed architecture, the SDN domain is extended in order to include all ad-hoc devices. The proposed solution consists of deploying an OpenFlow software switch, such as Open vSwitch [122] in each ad-hoc node. This configuration enables ad-hoc nodes to

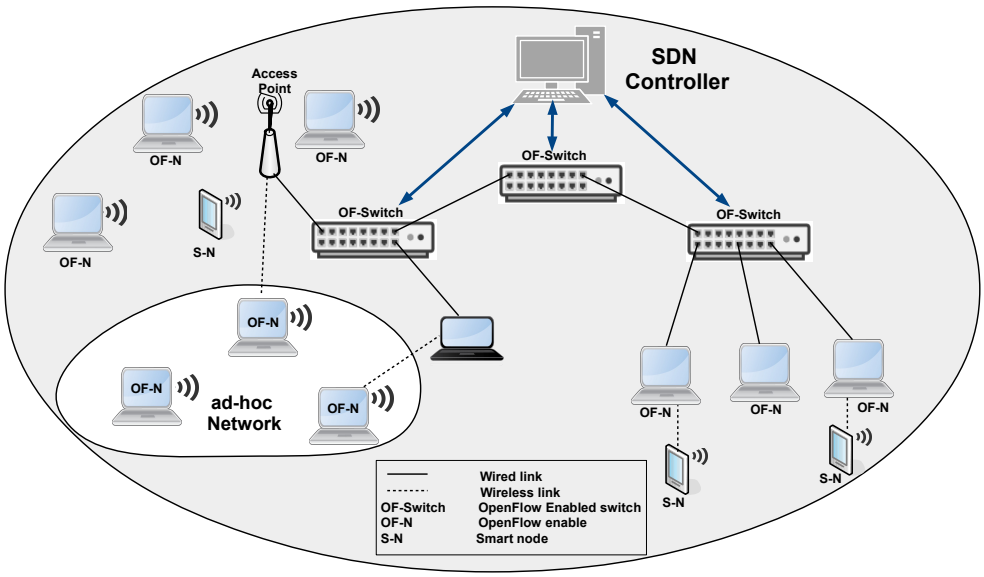

Figure 4.6: Extended SDN Domain.

connect the network as part of the SDN domain. Thus the same security rules can be applied for the SDN domain users. As shown in Figure 4.6, the proposed architecture supports both networks, with or without infrastructure.

The suggested SDN domain includes wired and wireless communication links. In this heterogeneous network, we have two types of nodes in a domain. If a node has enough resources with SDN capability it is called an OpenFlow Node (OF-N). The nodes without SDN capability are called smart nodes or sensor nodes (S-N). For IoT devices or sensor networks, every device cannot have an embedded SDN-compatible switch and an SDN controller as we have proposed in Section 4.2. But we assume that each device with low resource can be associated with one neighbor node which has the SDN capability. Each domain has a border SDN controller which controls all traffic only in its domain.

As each ad-hoc node has its own SDN controller, the SDN control plane manages the evolution of each SDN virtual switch on each ad-hoc device. When a new ad-hoc device connects itself or leaves the network, there are many OF messages exchanged. It allows synchronization a set of rules by adding new flows into the flow table pipeline. In order to ensure scalability and fault tolerance, a distributed SDN architecture with multiple controllers is preferred, as in [22]. To ensure that, we dynamically add Extension Controller (EC) to the ad-hoc network area and authorize special nodes to run control operations, as illustrated in Figure 4.7. The EC controllers share the same network global view with the ad-hoc nodes based on the equal interaction role. However, their SDN management into the domain will be limited to a small ad-hoc area. The main functions of the EC are to control the communications between ad-hoc nodes and also, delete or authorize the network traffic based on the forwarding OF rules. Furthermore, these controllers will be responsible for monitoring the behavior of their software switches because they are deployed at the user side.

A border controller is placed on the top of this framework, which is responsible to monitor, collect information, resource allocation and make decisions based on the global network view

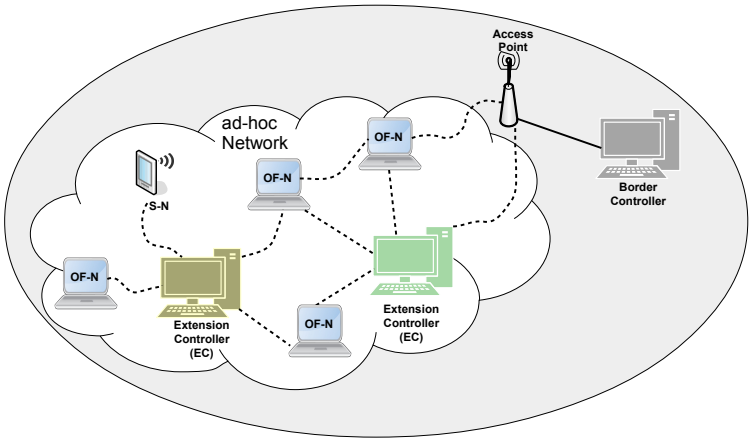

Figure 4.7: Distributed ad-hoc Control Plane.

of the domain. In this way, after the overall information gathered from the domain, the border controller can apply security rules, software verification and control access authorization to the whole devices connected in it domain. This coordinate controller has also the ability to implement orchestration across multiple SDN domain, as show in section 5. These controllers play a significant role to establish connections and exchange information with the other SDN edge controllers. Consequently, in this distributed multi-controller architecture the routing functions and security rules can be setting up on every border controller.

A distributed network access control installed on the border controller enables faster responses to network changes. Moreover, it reacts to attacks occurring in the SDN extended domain, while sharing the traffic load management with the border controller. As mentioned earlier, control functions of ad-hoc controllers will be limited and adapted to the available resources of the hosting ad-hoc device. We intend to extend the SDN domain even more, to include smart objects such as tablets, smart phones, mobile vehicles, etc. by developing a framework that integrates OpenFlow software switch into those devices.

In a wireless sensor network (WSN), one of the most important applications is collecting sensed data from one or more sensors, then sending the data to a base station called a sink node. In our proposed system, instead of a sink node that receives the captured data from the sensor nodes, an SDN controller is able to handle all information gathered from these devices. In Chapter 5, we describe an SDN-based cluster architecture for WSN.

Chapter 5

SDN Cluster Architecture

Summary: In the previous chapter, we define an SDN-based architecture for ad-hoc and IoT networks. We also identify the number of processed OpenFlow packets by the SDN controllers during the network topology discovery process in a single domain. This chapter presents an approach that consists of distributing multiple SDN controllers into clusters. It further describes distribution of routing functions and security rules among controllers. We study a cluster-based high availability model over the Application-Driven (AD-SAL) and Model-Driven (MD-SAL) Service Abstraction Layer provided by the OpenDayLight controller (ODL) [123]. The content of this chapter corresponds to our publications in [14, 16].

5.1 The Software Defined Clustered Networks Architecture

In this section, we introduce the concept of Software Defined Clustered Sensor Networks (SD-CSN) assuming that the wireless sensor network may contain hundreds or thousands of sensor nodes. Normally, a large network cannot operate efficiently without some organized structure. For this reason, we propose to cluster the network assuming that each cluster head is an SDN controller. In each SDN cluster, the Software-Defined Network cluster head (SDNCH) is in charge of managing the operation of the sensor nodes (Figure 5.1). With this clustering approach, the collected information about the environment is processed in the cluster by nodes and will be routed to the SDNCH. Moreover, the controller is the most powerful node on the cluster. By using the active/active interaction between SDNCH, it will have full access to the switch and the same flow rules. Based on this approach, we have been able to set the configuration parameters, store the data and aggregate the collected information in the cluster or send the information to the sink or some other SDNCH.

In [124] Diogo et *al.* proposed a new architecture model for IoT. They also reported on the possibility of exploiting the European Telecommunications Standards Institute-Machine to Machine (ETSI-M2M) architecture by allowing a device to negotiate the quality of service (QoS) and security parameters with the gateway. The idea of real-time configuration of a cloud service connectivity that can provide the information about the device connected to it is also discussed, the goal being to design an IoT-based system that can achieve interoperability, scalability and adaptability. Various other proposals have been introduced that deal with SDN-approach for IoT environment [102], [103], [125], most of which have focus on investigating the integration

of SDN and IoT into a single framework, but not on security-related challenges. Based on the SDN architecture, we have recently proposed a secure SDN-based architecture for IoT [12], assuming that each device in the network that has low resources can be associated with one neighbor node that has the SDN capability (meaning more resources that can be shared).

In [113], Gante et *al.* proposed a base station architecture where an SDN controller is combined with a wireless sensor network and the only controller requirement is to have a full knowledge of the network topology. In this system, the SDN component is used to determine the sensor nodes' forwarding decisions based on rules imposed by the controller, thereby allowing a better cooperation among the SDNCH and the sensor nodes. Moreover, the SDN controllers can help to reduce the energy consumption of the sensor nodes by making the best routing decisions and by injecting these decisions in the nodes' flow tables. With such control of the network management by the SDN, the routing decisions and policies yield a low convergence time compared to that generated by traditional routing protocols for wireless sensor networks.

We employ a Cluster-based High Availability model provided by the OpenDayLight controller due to fact that better network performance, trustworthiness and fault tolerance are obtained when using multiple controllers. Because each controller has a partial view of the network, the controllers have to collaborate and exchange information with each other. If one of the controllers goes down, another SDN controller can take control to avoid system failure. This cluster model provides two possible operation modes: Active-Active or Active-Passive Deployment.

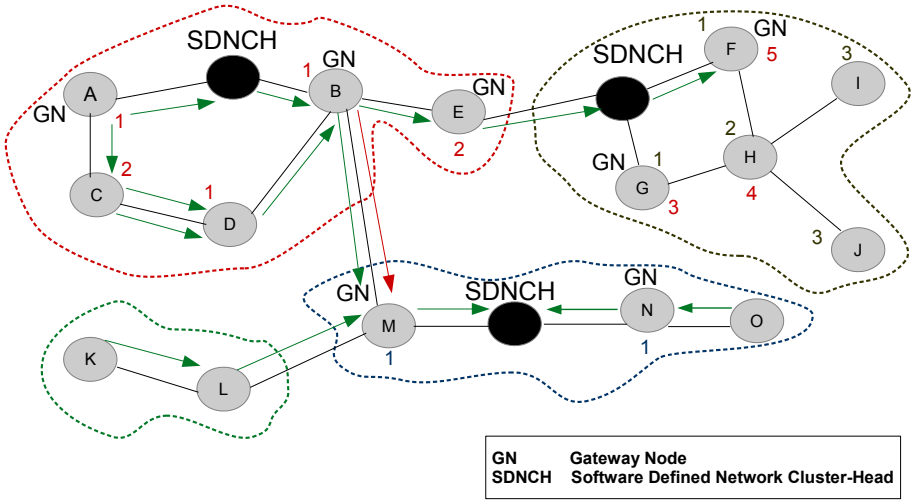

Figure 5.1: Data Communication of the Software-defined wireless sensor networks.

Each node exchanges information with its immediate neighbors or 2-hop neighbors. Under the assumption of graph connectivity, the information generated at one sensor can reach the SDNCH by routing it through the network. The gateway is engaged in aggregating and transmitting the data from the entire sensor node cluster to the other clusters. Thus, when a gateway goes down, the communication between clusters will be disconnected and the associated sensors to SDNCH are isolated. The SDNCH plays the role of coordinator in every cluster. Every SDNCH acts as a temporary base station within its cluster, and it is capable of

communicating with other SDNCHs. A cluster is therefore composed of an SDNCH, gateways and sensor nodes.

- The SDN Cluster Head (SDNCH): This is the coordinator of the cluster.
- Gateway: This is a bridge node between Sensor Nodes and SDNCH.
- Sensor Nodes: These are groups of nodes in a cluster together with their gateway nodes.

Figure 5.1 illustrates the schematic setting to build a data flow path between sensor nodes. In our scenario, the sensor nodes comunicate by using the reactive protocol AODV [116]. Communication between nodes occurs only when the source node invokes discovery of an optimal route for data transmission to the destination node. The route discovery process starts at the source node by broadcasting RREP Messages to its neighbors' nodes. Upon receiving the RREQ Messages, the destination node sends back an RREP message to the source node through the same route traced by the RREQ Messages. Intermediate nodes can be reach in order to find new or alternate routes for multi-hop destinations. The gateway node is equipped with two physical interfaces. One interface achieves connectivity with the sensor nodes using AODV routing protocol, and the other interface is bridged to an OVS with OF capabilities. The OVS establishes a connection to the controller, allowing data inspection and process the flow rules.

Nodes in the same cluster can exchange information about the next link (first-hop) in order to find the best route through to the destination node. For example, following the data-path communication of Figure 5.1, when node A needs to send data to the node D, it broadcasts RREQ Messages to the neighbor node B. As the destination node D is in the same cluster, the intermediate node B can reach the node D by broadcasting the RREQ Messages to the first-hop. Otherwise, if the destination node is in another cluster, the communication path may be found via a gateway node connected directly to the SDNCH. In this case, if node B wants to reach node H, the discovery of route is started by transmitting RREQ packet to its neighbors' nodes D and E. Node B obtains a RREP Messages from the gateway node E that knows the complete hop sequence to reach node H. Then, as node E communicates with the SDNCH of the other cluster, the SDNCH forwards the request to its gateway node until it reaches node H. Each node initiates the path discovery process hop-by-hop, and the SDNCHs are responsible for the routing decisions and the synchronization of the rest of the processes.

Based on the up-to-date global view of the data plane, every node knows about its distance and topology changes in the network. The SDNCH participates by actively monitoring multiple devices, and maintaining its routing tables. For the clustering procedure, the devices transmit the gathered information to the mobile gateway candidates. Then, the gateway sends received data to the SDNCH, which is responsible for making forwarding decisions between devices and throughout the other SDNCHs.

In order to secure network access and network resources, the SDN controllers begin by authenticating the network devices. Once the OpenFlow secure connection between the switch and the controller is established, the controller blocks the switch ports directly connected to the users. Afterwards, the controller authorizes only the authentication of the user's traffic. Once a user is authenticated, and based on the authorization level of the user, the controller will push the appropriate flow entries to the software or the hardware access switch. For IoT, we plan to extend this authentication process to devices. Each device will need an association

with an OpenFlow enabled node connected to one controller in its cluster. This architecture can further be exploited to prevent the users from accessing unauthorized new services, i.e. only services authorized by controller will be exploited by the endpoint devices.

5.2 The SDCSN Domain

To avoid the management of many clusters with few controllers, we build a new group of clusters called domain. A domain is a set of clusters managed by an SDN controller called a border controller (see Fig. 5.2). With this concept, we extend the concept of an SDCSN domain. In each domain, all controllers are controlled by a border SDN controller which has its own security policies and management strategy to distribute routing functions and security rules to other border controllers. We adopt this architecture to guarantee security with the concept of a grid of security [126] embedded in each controller to prevent attacks. When a sensor node needs to transmit the collected data to another domain, the flow must be forward to the security border controller on the domain.

In the proposed SDN-based architecture for IOT [12], the idea is to extend the SDN domain architecture to multiple domains, each having one or multiple SDN controllers that control all traffic and manage only the devices in its domain, where a domain represents an enterprise network or a datacenter. We intend to extend the SDN domain to include smart objects such as tablets, smart phones, and mobile vehicles, to name a few. This can be achieved by developing a framework that integrates the OpenFlow software switches into those devices. Our approach is not to distribute the control functions on multiple controllers, but rather to distribute the routing functions and security rules on each border controller. These controllers will be responsible for establishing the connections and exchanging information with other SDN neighbor border controllers.

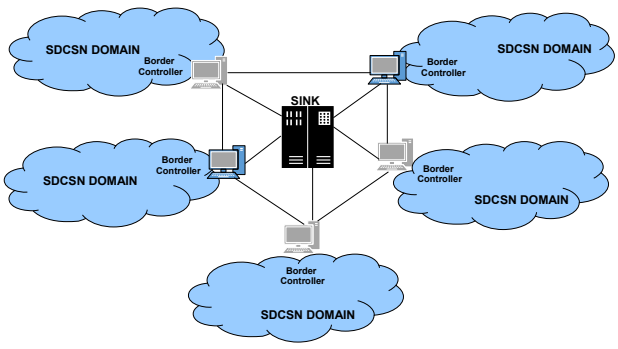

Figure 5.2: Domain Software-defined wireless sensor networks.

In a hierarchical SDN domain, the network is divided into several domains, wherein each domain is managed by a border controller. The sensor nodes transmit data to their SDNCH, which transmits the aggregated data to the base station. In each SDCSN Domain, the sensor nodes can receive policies and routing decisions from one or more SDNCH. Then, the border controller acts as the local base station for the domain, receiving information from the SDNCHs and transmitting this information to the other domains.

SDN inter-domain routing is required to exchange data among controllers. Each border controller builds a local network view, then it may exchange its local domain view with other border controller through a WE-Bridge [127]. This WE-Bridge provides a mechanism for different SDN administrative domains to peer and collaborate with each other. In [128], the authors propose a distributed SDN control plane, DISCO for WAN and overlay networks. It relies on the domains, and each controller oversees an SDN domain. Every neighbor domain can exchange aggregated network-wide information.

The main issue is the gateway node between clusters. In an IoT network [124], a thing may not have routing capabilities, so we can not have a thing which is a node with few resources as a gateway. For the clustering architecture, we propose an SDN controller in each domain. This controller manages and controls all traffic of nodes connected only in its domain, so-called intra-domain. In this environment, the controllers communicate with others via an inter-domain link. Previous work [30, 31] proposes hierarchical architecture for SDN to optimize and distribute control functions. We propose not to distribute control functions on multiple controllers but to distribute routing functions on each border controller.

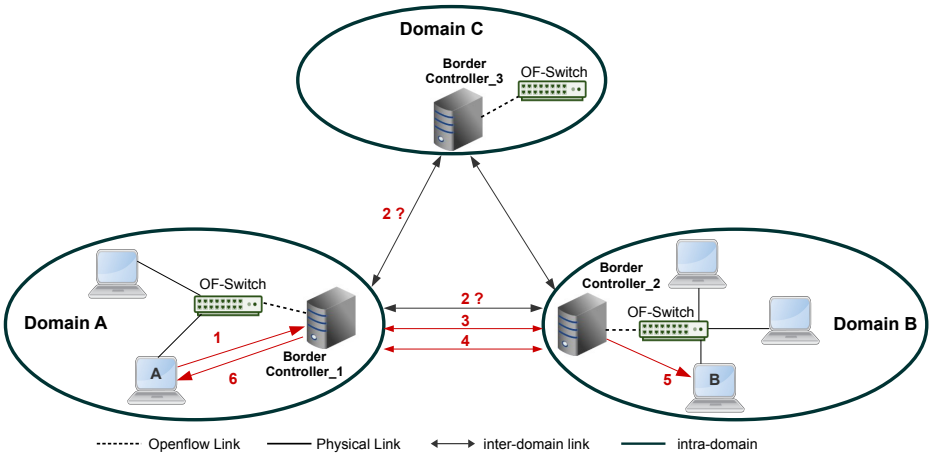

Figure 5.3: Distributed Routing Cluster for SDN.

Software defined information centric networking (SD-ICN) has already been proposed [30]. The scalable area-based hierarchical architecture (SAHA) has been used for the controller deployment in SD-ICN, in which a centralized root-controller has a global network view and the area controllers only know their local area view. The problem of zoning for distributed network optimization has been discussed in [31] and the proposed approach consists of a number of slave controllers to be in charge of the zones separately, under the coordination of a master controller. Its design consists of using the clustering heuristic, the partitioning heuristic, and the assignment heuristic solutions which allow the communication and load processing between controllers.

The deployment of our architecture is based on the perspective of an OpenDayLight MD-SAL AKKA-based clustering solution. To select an appropriate path between nodes connected on the cluster, the process of routing flows has been indicated in (Fig. 5.3).

The main process flow is as follows:

1. Node A joins node B, and node A sends a request to the border controller 1,

2. SDNCH1 sends the same request to the neighbor controllers connected on the network,
3. The ones which know node B, send replies to border controller 1, for this example border controller,
4. The flow may be installed on the border controller 1. Node A can use the communication path via border controller 1 for taking a packet out of cluster,
5. The routing information between border controller 1 and border controller being set up on the inter-domain path,
6. The messages will be exchanged between both nodes.

We focus in the proposal of a new Routing Protocol for a Distributed Cluster SDN which can be used to support SDN-based inter-domain collaboration. The goal is to allow an automated routing setup of inter-domain clusters. As a result, a node can interact with other nodes located in different clusters through an inter-cluster routing protocol. Our goal from this point of view is to design a cluster structuration by grouping network nodes into a domain. In the next section we have implemented OpenDayLight clustering solution which is based in provide a high availability model. In our study we evaluate the performance of the OpenDayLight clustering tools and analyze their adaptability to our proposed architecture.

5.3 Cluster in OpenDayLight AD-SAL Architecture

The kernel of the ODL Controller design is based on the Service Adaptation Layer (SAL), that connects southbound protocol plugins to the northbound service and application plugins. Based on SAL, ODL provides two different types of service abstraction, AD-SAL and MD-SAL. AD-SAL, was the primary architectural implementation of ODL when we started this thesis, but now it is deprecated. In AD-SAL architecture, the SDN applications communicate with devices using separate application modules. Each southbound protocol interacts independently with its respective plugins. With the MD-SAL, SDN applications are defined from model objects where each API interacts directly with a model service instead of with plugins. Also, this model can communicate other application models. AD-SAL was designed for transparency between north-south bound communications, and MD-SAL disassociates the API from the southbound protocol plugins, which enables the modules to work together performing the operations on devices.

One of the SDN features in which we are particularly interested during my PhD is clustering. The first experiments we started were performed on ODL Hydrogen Release Base Edition, which allowed setting up a cluster communication between multiple ODL controllers. By adding one or more ODL instances, we can make the cluster highly available and increase resilience. On the southbound side, the enabled OF switches can connect to two or more instances of the ODL Controller via point-to-point TCP/IP connection. On the other hand, interaction between controllers and user's applications are performed via RESTful web services on the northbound side.

A connection manager handles the clustering state synchronization and transactions made by the nodes in a cluster. The connection manager of ODL provides two configurable schemes:

SINGLE_ CONTROLLER and *ANY_ CONTROLLER_ ONE_ MASTER*. By default, ODL Hydrogen release operates in the *ANY_ CONTROLLER_ ONE_ MASTER* scheme. The cluster connection schemes are explained below.

- *SINGLE_ CONTROLLER*: All the nodes are connected with a Single Controller. The Single Controller Scheme algorithm will determine the one controller to which all the nodes are connected. This is like Active-Standby model from a South-Bound perspective.
- *ANY_ CONTROLLER_ ONE_ MASTER*: Any node can be connected with any controller. But only one is the master controller in the cluster.

Once the connection manager has been set up, interactions controller-to-controller on the northbound side specify two modes of operations: equal and master/slave. This is essentially to have one OF switch multi-homed to multiple controllers. If a controller fails, another SDN controller can take the control to avoid system failures. The ODL Hydrogen provides clustering services using OpenFlow 1.2 with multiple controllers on the network as explained next.

- Equal interaction: in this case all controllers have read/write access to the switch, which means they have to synchronize in order not to step on each other's feet.
- Master/Slave interaction: in this case there will be one master and multiple slaves. Also there could be still multiple equals as well.

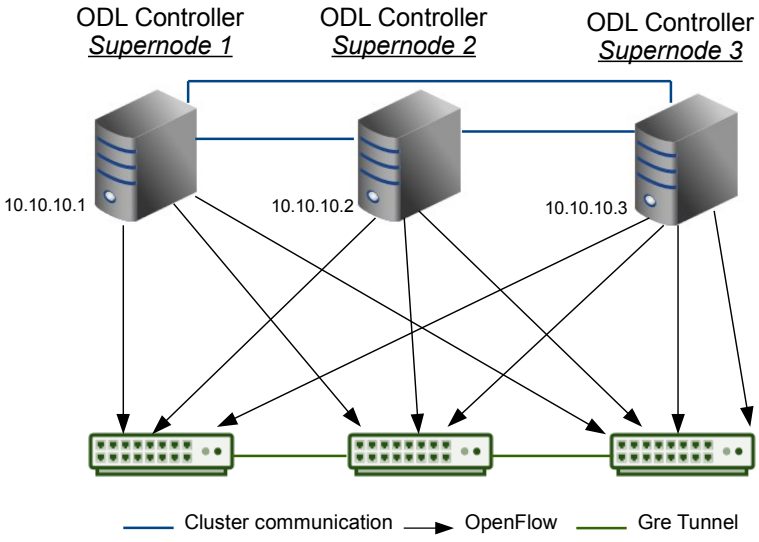

Figure 5.4: AD-SAL, cluster implementation architecture.

In setting up the cluster two or more nodes could be elected *supernode* to support high availability. It is recommended at least 3 nodes be set up in a cluster. A cluster can be formed with only 2 nodes, but, if one of the nodes fails the cluster will not work correctly, see Figure 5.4. Specific clustering configuration could be defined on the file server after the controller begins operating. When the *supernode* are chosen to be included in the cluster, it is important

to have IP connectivity among them, and to verify that the ports 7800, 12001 are available. The *supernodes* controllers that are member of the cluster create a full mesh peer-to-peer network. These *supernodes* exchange information about some control functions which allow to share the workload and handle East-West traffic. This traffic includes information such as the topology network, inventory, and control plane management. We will study the different options and applicability about this cluster implementation in practical scenarios via simulations which will be presented in Chapter 6.

The southbound plugin support by ODL Hydrogen in cluster deployment is via OpenFlow 1.0 as well as 1.3 version. Later, AD-SAL was upgraded to MD-SAL as the clustering solution provided by ODL Helium, Beryllium, Boron or Carbon release. In the next section, we provide details of the MD-SAL architecture for running the controllers in cluster mode.

5.4 Cluster in OpenDayLight MD-SAL Architecture

The Model-Driven Service Abstraction Layer (MD-SAL), is a new component provided by the last versions of ODL. For cluster deployment MD-SAL is based on the popular AKKA clustering technology [129]. AKKA modules provide a decentralized peer-to-peer network with high availability to avoid a Single Point of Failure (SPOF) or bottleneck. The members of clusters communicate by means of the gossip protocol and automatic failure detector. This feature enables every member of the cluster to join or leave the network without any configuration changes. AKKA clustering is based on the Amazon Dynamo [130] distributed system in particular the Riak distributed database. After implementation in Section 6.3.3, we provide an overview of the difference between AD-SAL and MD-SAL OpenDayLight cluster architectures.

The config files to modify are located in the ODL directory which will contain the ***akka.conf***, ***module-shards.conf*** and textbf*modules.conf* files. Every node of the cluster needs a unique *member-N* identity role defined in the ***akka.conf***. It should also indicate the seed nodes to know the new members joining the cluster. ***Module-shards.conf*** is used to segment some or all features of a ODL MD-SAL datastore. Data shard replicas could be defined on as many of the members as possible, because if in a cluster with 3 nodes and replicas is configured only into 2 nodes, the data shards will be not operational. There, a replica can be defined in which ODL instance will be stored and also, where the inventory or topology may be located. For example, inventory may be placed on member-1 and member-2, while the topology may be placed on member-1 and member-3 on the cluster (Fig. 5.5). To contain data, a module can be defined in separate shards on the ***modules.conf***. The default shard-strategy replicates all the data clustering information. Inventory shard stores and handles information about the OpenFlow switches ports (node connectors), MAC address, flow tables and statistics of network traffic. On the other hand, topology shard contains a graphical representation of network topology including the OpenFlow switches and end hosts. A sample configuration is provided in *akka.conf* and *module-shards.conf* files.

Once the configurations have been made and the cluster members are started, the first task begins by sharing gossip messages about the current state of other nodes in a cluster. Then, the gossip convergence process occurs, only if the node is reachable, to connect nodes to each other. Otherwise, if any cluster member is unreachable, it will be placed in down state and removed from the cluster. For the gossip propagation, a failure detection algorithm

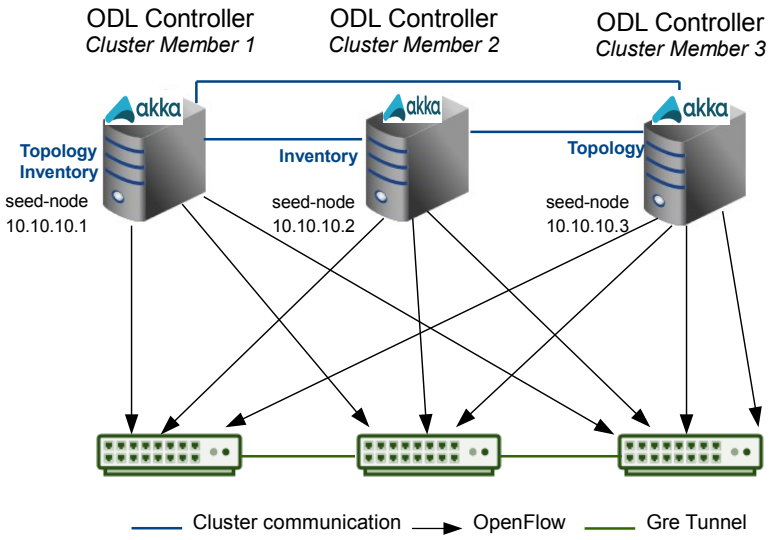

Figure 5.5: MD-SAL, cluster implementation architecture.

is used to detect the reachable and unreachable nodes on the cluster. As mentioned before, nodes communication depends on the gossip convergence where all members of the cluster share heartbeat messages. After convergence, an election process occurs to determine a leader based on the RAFT Consensus Algorithm. It serves only the role of cluster membership management. This role consists only in moving member nodes of the cluster into or out of active status within the cluster.

Currently the ODL Helium, Beryllium Boron or Carbon releases [123] identify two modes of operations for cluster deployment: active/active and active/passive interaction. These options are available in the ODL project, permitting support of a cluster based high availability model. Each controller has a partial view of the entire network, which can help to collaborate and exchange information. We analyze the available options for connecting the SDN cluster.

- In active/active mode, should the configured primary fail, should there be partial network partition of the primary controller's links or should there be full network partition (if monitoring is enabled), the passive secondary will be promoted to the active primary controller role as part of the failover. During a full network partition, both segments of the network may be independently managed.

- In active/passive mode, in the event of the configured primary's failure or partial network partition or failure of the primary controller's links, the passive secondary will be promoted to the active primary controller role as part of the failover. In the event of a full network partition (if monitoring is enabled) the passive secondary will not be promoted to the primary role but will instead suspend core controller functionality. The switches on the secondary's segment of the network will go unmanaged.

The akka.conf configuration file has instances of seed-nodes under the cluster. This file contains the IP address of the cluster node and the listening port to receive the requests from joining seed-nodes. Also, it is necessary to specify a role name for the instance. By default,

the role name is member-1. It is important to note that a role name must be unique for each node. A sample akka.conf content file on the cluster member-1 is as follows:

Sample akka.conf

```
odl-cluster-data {
  akka {
    remote {
      netty.tcp {
        hostname = "10.10.10.1"
        port = 2550
      }
    }
    cluster {
seed-nodes = [
"akka.tcp://opendaylight-cluster-data@10.10.10.1:2550",
"akka.tcp://opendaylight-cluster-data@10.10.10.2:2550",
"akka.tcp://opendaylight-cluster-data@10.10.10.3:2550"]
      roles = [
       "member-1"
      ]
    }
    persistence {
      journal {
        leveldb {

        }
      }
    }
  }
}
```

In the module-shards configuration file, it is possible to distribute different tasks between member nodes of the cluster. Each entity can store replica shards which include information about the topology, inventory and other features. A sample content from the file follows:

Sample module-shards

```
module-shards = [
    {
        name = "default"
        shards = [
            {
                name="default"
                replicas = [
                    "member-1"
                    "member-2"
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "topology"
        shards = [
            {
                name="topology"
                replicas = [
                    "member-1"
                    "member-2"
                ]
            }
        ]
    },
    {
        name = "inventory"
        shards = [
            {
                name="inventory"
                replicas = [
                    "member-1"
                    "member-3"
                ]
            }
        ]
    }
]
```

Chapter 6

Implementation and Simulation Development

Summary: Previously, in chapters 4 and 5 we presented an SDN-based architecture for ad-hoc and IoT networks, which has been one of the first approaches to use SDN technologies for management of these kinds of networks. Also in chapter 5 is presented the SDNCH, an SDN clustered architecture for WSN. In this chapter, we study different cases in which these types of proposed architectures are leveraged to improve the performance of implementing SDN in ad-hoc, IOT and cluster networks. The content of this chapter corresponds to our publications in [17, 18, 19].

6.1 Introduction

The experimental platform was developed based on SDN and enabled users easily to design network topology via web applications. The framework of the testbed would then allocate resources to the experiment, which is able to create routes between nodes, monitor requests sent to the controller and also decide policies delivered to each device. Our approach on the testbed provides an environment for mobile and ad-hoc network deployment. It allows nodes to be connected with others via one SDN-compatible OVS switch, and at the same time this switch is controlled by an SDN controller. This framework for Software-Defined Networking applications provides an external and an internal network deployment. The external network allows the management of virtual machines remotely, and the internal network enables the OpenFlow communication between hosted nodes, see figure 6.1.

The implemented testbed includes a virtualized testing environment combined with vmware vsphere, qemu system, Tinycore, software-based Open vSwitch (OVS) and OpendayLight controllers 6.1. The list of components used to build the testbed environment is described below:

- **Vmware Vsphere:** A platform that allows multiple virtual machines to run simultaneously on a physical host computer. It provides a web interface which allows users to connect remotely and also remote access by console. In this platform we can modify the processor, memory, storage, and resources on each virtual machine [131].

- **mRemoteNG:** An open source multi-tabbed connections manager that handles remotely the access to the VM's servers installed on the framework. mRemoteNG supports multiple remote connection protocols including Remote Desktop/Terminal Server (RDP),

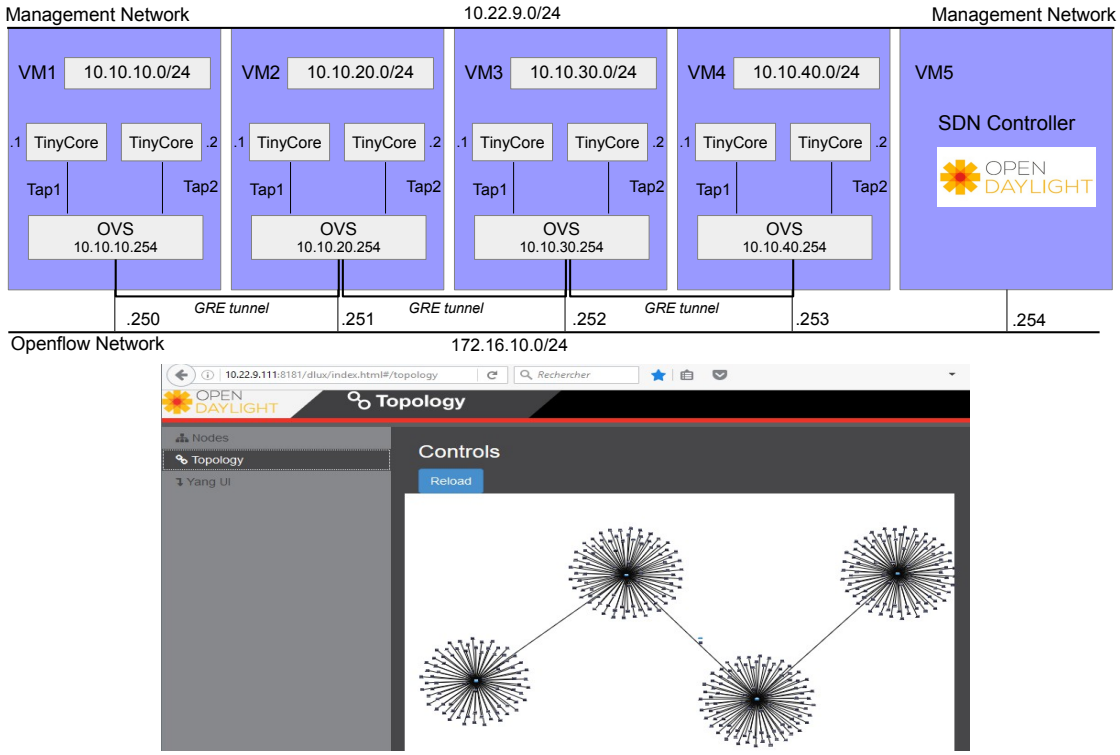

Figure 6.1: Network design of the based SDN testbed

Virtual Network Computing (VNC), SSH (Secure Shell), Telnet, rlogin and Raw Socket Connections [132].

- **Qemu System:** Is an open source machine emulator that can be used to virtualize embedded devices. Qemu emulates different peripherals such as the processor, network interfaces, hard drive, RAM memory, VGA, keyboard, serial, PCI, USB ports and so on. Also, this emulator enables a remote connection over SSH to securely log onto each remote device [133].

- **Tinycore:** A minimal Linux operating system with capabilities of embedded and IoT devices. Tiny Core has few available architectures for deployment such as ARM, x86 and x86_64 processors [134].

- **TightVNC:** Is a free software application that allows controlling and monitoring remote desktop. With TightVNC we can manage and configure remotely the Tinycore machines located on our local testbed network [135].

- **Open vSwitch (OVS):** Is a multilayer open source software licensed under Apache license that allows hypervisors to virtualize internetworking devices. The Tinycore guests are connected to virtual ports, allowing the communication between end devices. Open vSwitch supports the OpenFlow protocol so that it can include features, manage configuration, and modify flow table entries on the OF switches via a command line tool. With OVS, we create Generic Routing Encapsulation (GRE) tunnels between VM's server in order to encapsulate the OpenFlow traffic onto overlay networks [1].

- **OpenDayLight Controller (ODL):** An open SDN platform supported by the Linux Foundation. Its developement is via a collaborative project involving many contributors of the IT industry such as Cisco, Ericsson, Brocade, IBM, HP, Juniper Networks, Microsoft, Red Hat, Intel and many others. Via the southbound OF protocol, ODL allows us to program the network control functions of the networking devices connected on our testbed platform [123].

Table 6.1: Testbed Specifications

CPU	Intel Xeon Processor
Memory	16GB
OS	Ubuntu 14.04 Server & Debian 8.7
SDN Controllers	OpenDayLight Helium SR4 with OpenFlow 1.3 and AKKA cluster
Server Virtualization	VMware vSphere 5.1
Hosts Virtualization	Qemu 2.10.0, OVS 2.5.0
Embedded Systems	Tinycore 7.2 with 48MB RAM

The testbed implementation includes a virtualized environment combined with OpenDayLight SDN controller [1] and OF protocol version 1.3. Basically, according to the OpenFlow protocol, an OpenFlow switch forwards a packet to the controller if it does or does not have a matching flow rule for that packet. When host A wants to send an IP packet over Ethernet to host B by IP address, it needs an ARP reply from B. When a new flow from host A arrives at an OpenFlow switch, the switch forwards the packet to the SDN controller. The controller plays the role of ARP proxy. If there is not match with the flow entry installed, the packet will be rejected. Otherwise, an ARP request is allowed to pass toward host B. Upon receiving the ARP reply, the routing in the controller will send the flow to the switch connected to the source host and the switch connected to the destination host. Once, the link is established, a shortest route will be set for the switch. In this way, the switch is in charge of routing the packets between Host A and Host B (Fig. 6.2).

We have virtualized OpenFlow compatible switches (OpenVSwitch version 2.5.0) and small Linux (TinyCore with 48Mo of RAM). To evaluate the performance of Openflow, the experimental scenario deployed on the SDN testbed consists of OpenDayLight Helium-SR4 and five virtual machines (VM) connected to the same network. The PC running the controller uses an Ubuntu 14.04 operating system and was provided with 2 virtual CPUs and 16GB of RAM. Each VM had 8 virtual CPUs and 16GB of RAM, using the Debian 8.3 image with OVS preinstalled. We use KVM for machine virtualization over the VMs, to run TinyCore Linux 3.16.6 with 48MB of RAM. From this testbed, we will be able to run our protocol experiment with more than 500 devices in a first attempt. Each device used is a TinyCore Linux system.

We have created a custom script to simultaneously start multiple qemu-kvm virtual machines. It is important to note that this script enables the control of each device over an SDN environment and all communications are set up via the OpenFlow protocol. The shell script consists in launching the hosted virtual machines and the instructions to interconnect the switches. It facilitates a remote-control session and the ability to handle virtual machines

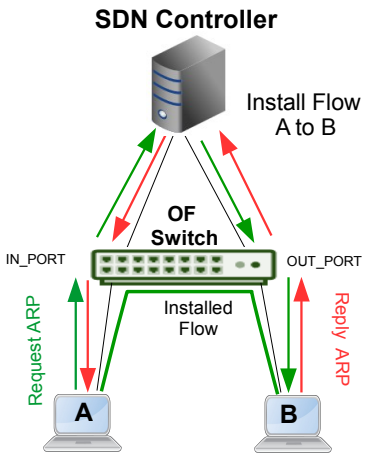

Figure 6.2: Openflow ARP Request

running on Hyper-V hosts. Once the initial set up and configuration are accomplished, into each VM a Tiny Core image is use to start the Hyper-V hosts. In order to establish connection between the nodes, a DHCP shell script assigns automatically the IP address previously configured on the startup TinyCore image. Every time that a node turns on, it sends a DHCP request to get an IP address based on the MAC address which was assigned by the initial script. At this point, every started node is listening for connections on an open port and the IP address via Virtual Network Computing (VNC). When the setup and configuration have been completed, it can be used for implementation and experimentation in an OpenFlow network emulation Testbed with a fully-virtualized environment.

Procedure 1 Power-on the TinyCore System

1: $img = TinyCore\_img$
2: $path\_img = /path/to/img$
3: $mem = \ 48\ MB$
4: $N = \ the\ number\ of\ VMs\ to\ turn\ on$
5: $OVS_i = OVS_{name}$
6: $y = 0$
7: **while** $1 + y \leq N$ **do**
8: tap =tap(1+y)
9: VM_{name} =Tiny_(1+y)
10: $new\_img = /path/to/save/img$
11: mac =00:00:00:00:00:(1+y)
12: $port$ =(5900+y)
13: qemu creates an image with qcow2 format in *newimg*
14: qemu starts the VMs with *mem, new_ img,* VM_{name}*, mac, port*
15: **if** $OVS_i = NULL$ **then**
16: add bridge *OVS*
17: set OF version to bridge *OVS,protocols=OpenFlow13*
18: add port *OVS, tap*
19: **else**
20: add port *OVS, tap*
21: **end if**
22: y=y+1
23: **end while**

6.2 Setting up OpenFlow Rules

To allow forwarding traffic between connected hosts on the network, a set of flow entries must be installed on the specified flow table of each OF switch. A flow table consists of a number of flow entries in which each datapath entry is associated to an action instruction. This action decides how the packets should be processed by the controller. The control path of the flow entries in the flow table is programmed on top of SDN controller. An OF switch can contains one or more flow tables and a group table processing forwarding decisions by adding new flow entries or modifying/removing the existing flow entries. To make any forwarding decisions, the flow entries can be added using POSTMAN Chrome API, YANG UI or *ovs-ofctl add-flow* tools.

For the L2/L3 forwarding decisions, the SDN controller knows the topology and also the devices attached to the topology as well as their identities IP/MAC addresses. Then, programming the switches via an open API, the OpenFlow controller may manage and constantly monitor the Openflow network update upon changes. In L2, the OVS acts as a regular layer-2 learning switch, automatically creating a learning table based on the source MAC address of the incoming frame and places it on an incoming port. Then it forwards the frame to the appropriate output port destination based on the *odl-l2switch-switch* feature provided by ODL. An OF switch is represented by an in_port, switch name and the out_port.

In L2, we have no flow to install because the ODL controller is able to discover the network

with LLDP and OFDP packets and the switch knows how to forward ARP packets. The OVS acts as a regular Layer-2 learning switch. It automatically creates a learning table based on the source MAC address of the incoming frame and places it on a MAC table.

To build a route L3, the forwarding operations at flow or packet level are therefore processed by the controller. It can be done by pushing rules into hardware devices with the *ovs-ofctl add-flow* command as well as ARP resolution between hosts and edge switches. If the switch receives a packet that does not match any entry in the flow table, the Openflow switch forwards the packet to the controller by an ARP request. When such request is received by the controller, it will be ignored because the controller does not know the NextHop IP address needed to handle all traffic forwarded to another subnet. To configure the OVS's to learn the NextHop, an ARP gateway IP is assigned to establish the connection between all nodes.

The flow entries added to each switch have an ARP gateway IP and a MAC address to send the frame to the out_port to which the host is connected. The ARP gateway carries traffic from one subnet to another. Each OpenFlow switch maintains a routing table to decide how to route the IP packets. Each routing entry consists of the destination address, subnet mask and the MAC address for the destination IP. Upon receiving the ARP reply, the routing in the controller will send the flow to the switch connected to the source host and the switch connected to the destination host. Once, the link is established, a shortest route will be set for both switches. In this way, the switches are in charge of routing the packets between Host A and Host B.

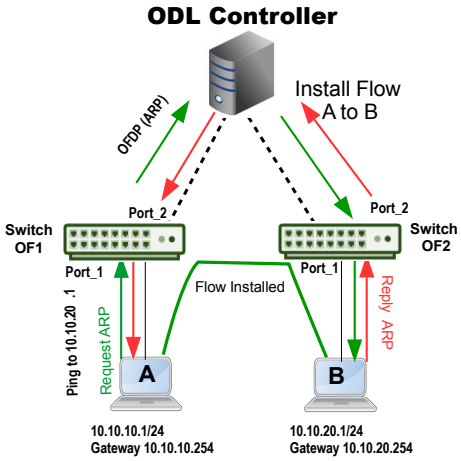

Figure 6.3: OpenFlow ARP Request.

The Figure 6.3 shows an example of the L3 routing process,in which each network node is configured with an IP, netmask and gateway. Then it forwards the frame to the appropriate output port to reach the destination. In this case, without a router between networks, we can forward packets between networks using flow installed by the controller into hardware devices. The command used is *ovs-ofctl add-flow* and ARP resolution process (see table 6.2 and table 6.3).

When implementing OpenFlow we have some scalability issues with IPv4, in fact it cannot be used easily to interconnect large numbers of devices. For optimum scalability all packets must be processed by an OF pipeline. The processing pipeline includes multiple flow tables

Table 6.2: OpenFlow Routing Process on Switch 1

	MAC_Source	MAC_Destination	IP_Source	IP_Destination	In_Port	Out_Port	MAC/ARP Gateway	IP/ARP Gateway
1	00:00:00:00:00:01	00:00:00:00:00:02	10.10.10.1	10.10.20.1	1	2		
2							00:00:5E:00:02:01	
3				10.10.20.0/24			00:00:5E:00:02:02	
4		00:00:00:00:00:02		10.10.20.1				
5		00:00:00:00:00:02				2		
6							00:00:5E:00:02:01	10.10.10.254

Table 6.3: OpenFlow Routing Process on Switch 2

	MAC_Source	MAC_Destination	IP_Source	IP_Destination	In_Port	Out_Port	MAC/ARP Gateway	IP/ARP Gateway
1	00:00:00:00:00:02	00:00:00:00:00:01	10.10.20.1	10.10.10.1	1	2		
2							00:00:5E:00:02:02	
3				10.10.10.0/24			00:00:5E:00:02:01	
4		00:00:00:00:00:01		10.10.10.1				
5		00:00:00:00:00:01				2		
6							00:00:5E:00:02:02	10.10.20.254

and each flow table contains multiple flow entries looking up a match rule in each table. By default, the packet matching process starts at table 0, then continues sequentially to look up in additional tables a match flow entry if necessary.

The custom pipeline 6.4 has been organized into multiple flow tables to reach an external subnet network. The pipeline starts at table 0, which provides a classification of network traffic that we are interested in filtering before pushing it along the OF pipeline. Also, it matches an Ethernet source address to route layer 3 traffic through an ARP gateway responder. Table 10 matches all IPv4 packets with sources address in CIDR prefix, then forwards the packet to the appropriate output port. In the table 20, forwarding table entries resolve the destination IP address to the correct MAC address for L2 forwarding. Table 30 forwards any layer 2 encapsulations. Layer 2 looks up to forward out the correct port in table 40. All incoming ARP Request are processed in table 50, then are turned into an ARP reply.

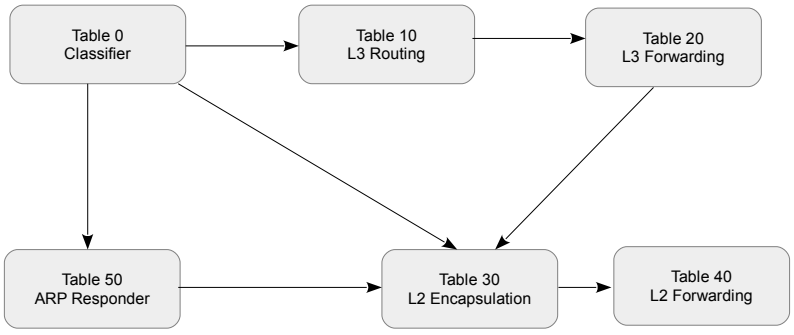

Figure 6.4: Subnet OpenFlow Pipeline

With the *ovs-ofctl dump-flows* command on both OF switches, it is possible to figure out that packets match the proper entries forwarding L3 traffic. All flow entries of the switches are printed in a console displaying the already installed OpenFlow messages. Each flow moves throughout the pipeline previously programmed, according to the OF match rules. Running the ping command on each end device, allows testing to determine if the network hosts are working properly and reaching the communication path.

This example shows only 2 subnet devices connected on our SDN network. To build a large scale of networks in a layer 3, IPv4 OpenFlow forwarding decisions requires hundreds of

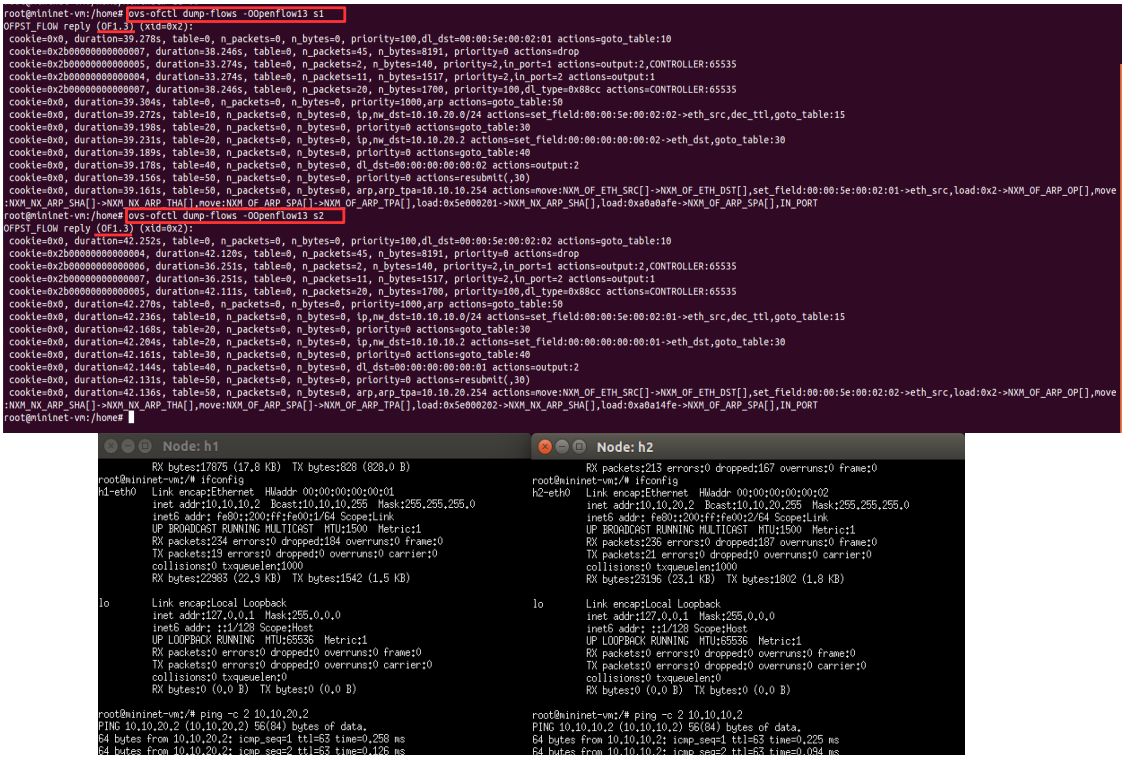

Figure 6.5: Subnet Openflow Flow entries

thousands of flows into a single switch to allow implementing scalable networks. It creates a new challenge to interconnect multiple devices in our developed framework. To set up a scalable OpenFlow testbed, we have developed a shell script to simplify the flow entries added to each flow table which thus enables the management of the whole network by an ODL controller (Fig. 6.2).

Procedure 2 Add OF Flow entries to Switch

1: OVS_{name}
2: $Controller_{port}$
3: $j = 0$
4: $N =$ *the number of VMs launched*
5: $Out_{port} = ((1 + N))$ *output port to interconnect switches*
6: **while** $j \leq v_{max}$ **do**
7: *list* =range (j , v_{max})
8: **if** $j = 0$ **then**
9: *addflowstatic*
10: **else**
11: *addflowdynamic*
12: **end if**
13: j=j+1
14: **end while**
15: **function** *range* (v_{min} , v_{max})
16: **for** $i = 1$ to v_{max} **do**
17: **if** $v_{min} = v_{max}$ **then**
18: **if** $i = (v_{max} - 1)$ **then**
19: *print* i break
20: **else**
21: *print* i
22: **end if**
23: **else if** $i = v_{max}$ **then**
24: *print* i
25: **else if** $i = v_{min}$ **then**
26: *print* i
27: **else**
28: *print* i
29: **end if**
30: **end for**
31: **end function**
32: **function** *addflowstatic*
33: add static flow entries into the OF pipeline only one time
ovs-ofctl add-flow OVS_{name} j *list* $Controller_{port}$
34: **end function**
35: **function** *addflowdynamic*
36: add dynamic flow entries into the OF pipeline more than once time
ovs-ofctl add-flow OVS_{name} IP_j MAC_j Out_{port}
37: **end function**

The procedure to add flow entries into the OF switches starts by reading the number of VMs started on the network, as well as the open port on the SDN Controller. Each connected VM is bound to one specific port and is used for the forwarding process described in the pipeline 6.4. The range function identifies the number of ports between v_{min} and v_{max}. Then

it creates an ordered list where an *input_port* is attached to the corresponding *output_port* of each connected VM. This function is very useful for the forwarding process in L2 when the *input_port* is used to flood packets on every port of the switch. If a port is selected *input_port* during the function range procedure, it cannot be considered as an *output_port*. Thence the OVS can automatically match the flows between *dl_src* (ethernet source address) and *dl_dst* (ethernet destination address) for forwarding them to the correct *output_port*. For operation of type L3 traffic, an ARP resolution was configured in the pipeline 6.4, in order to forward packets on different network handled purely by OpenFlow. To add flows into the pipeline, we introduce two functions *addflowstatic* and *addflowdynamic*. Function *addflowstatic* to install the flow entries that are set up only one time on the flow tables. In this case, the function sends the packet and its metadata to the SDN controller. The initial table 0 initializes the type of packet to be used such as, LLDP L2 and ARP L3. Another static flow entry is the ARP resolution gateway for routing decisions in L3 with their subnet address. *addflowdynamic* is used to add flows randomly on the flow tables. This function floods the packets received on the *input_port* through all listed ports by the *range* function. It also allows the routing process to include the IP and MAC address of each connected VM, attaching the *output_port* to send traffic out of their network as show in figure 6.7.

6.3 Cluster in SDN with OpenDayLight

In order to verify the AD-SAL Clustering and MD-SAL Clustering provided by ODL, we studied the OF performance by setting up both clusters into our prototype. The implementations are validated by deploying three ODL controllers which provide high availability, reliability and scalability (Fig. 6.6). If one of the ODL controllers is unavailable on the cluster, the resources are redirected and the workload is distributed to the other cluster nodes that can take the control over the network. In that way, the cluster operates normally without any downtime.

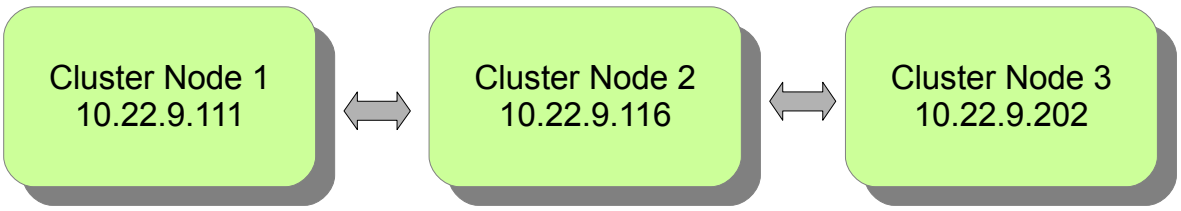

Figure 6.6: Cluster platform architecture

6.3.1 AD-SAL Clustering

First of all, to setup AD-SAL clusters it is necessary to install the pre-built zip file of ODL Hydrogen Release Base Edition available on the webpage of the OpenDayLight project. This ODL version is based on the Java platform and requires minimum Java 7 JDK installed on the Controller machine. To set the JAVA_HOME environment variable, it is necessary to add the following line into the bashrc file: export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64. Sometimes an "out of memory" error message is observed when the ODL controller

starts. The fix for this issue should be to increase the amount of memory by changing export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=1024m" and also in the bashrc file. The ODL Hydrogen is outdated now, but it was our first cluster tested platform.

6.3.1.1 ODL Hydrogen Structure

Before startup of the cluster, it is important to understand the main content of the OpenDayLight directory. A short definition describes the following ODL content files:

- configuration - startup-config file with initial set up
- lib - contains java libraries
- plugins - ODL OSGi plugins directory
- run.bat - script file to launch ODL on Windows systems
- run.sh - script file to launch ODL on Linux/Mac/Unix systems
- version.properties - specifies the version number and build time

6.3.1.2 Running ODL Hydrogen

The ODL Hydrogen can be initialized by running the *run.sh* script file from the command line with administrator privileges. After the ODL start, the GUI can be accessed in http://"ip-address-of-ODL-controller":8080 via a web browser interface. Then to log in, the user name and password are by default admin/admin. The ODL GUI provides an option to change or reset the password. By default ODL uses OpenFlow 1.0, to enable OpenFlow 1.3 version, and it is necessary to add the *-of13* option with the launch script file.

6.3.1.3 Clustering Test

The first thing to do is to verify the connectivity between controllers. Once the initial configuration is done in all the nodes, its can start the clustering services. To set up clusters in ODL Hydrogen, we perform the command -Dsupernodes on a terminal session of each controller.

After running the command, the controllers discover the network topology with the switches and all the nodes connected to them. By connecting a web browser to remote controllers IP address and port 8080, the network devices can be show on a graphical interface, as shown below:

After running the command, the supernodes create a full mesh network. At any time, controllers can come or leave the cluster. The 3 ODL clustered nodes display on the devices tab the learned switches OF|00:00:d6:89:0d:9d:21:4e and OF|00:00:da:98:b3:18:af:4d. In order to experiment with cluster performance in ODL Hydrogen, we use 2 OF switches with 10 interconnected TinyCore clients. The clustered nodes identify the switches connected with the end nodes, as well as their MAC and IP address. It is important to note that our framework is able to add dynamically more VM clients if necessary. For making forwarding decisions on the flow tables, we only need to launch script file add flows 6.2. At this point all end nodes are able to connect to each other on the OF network.

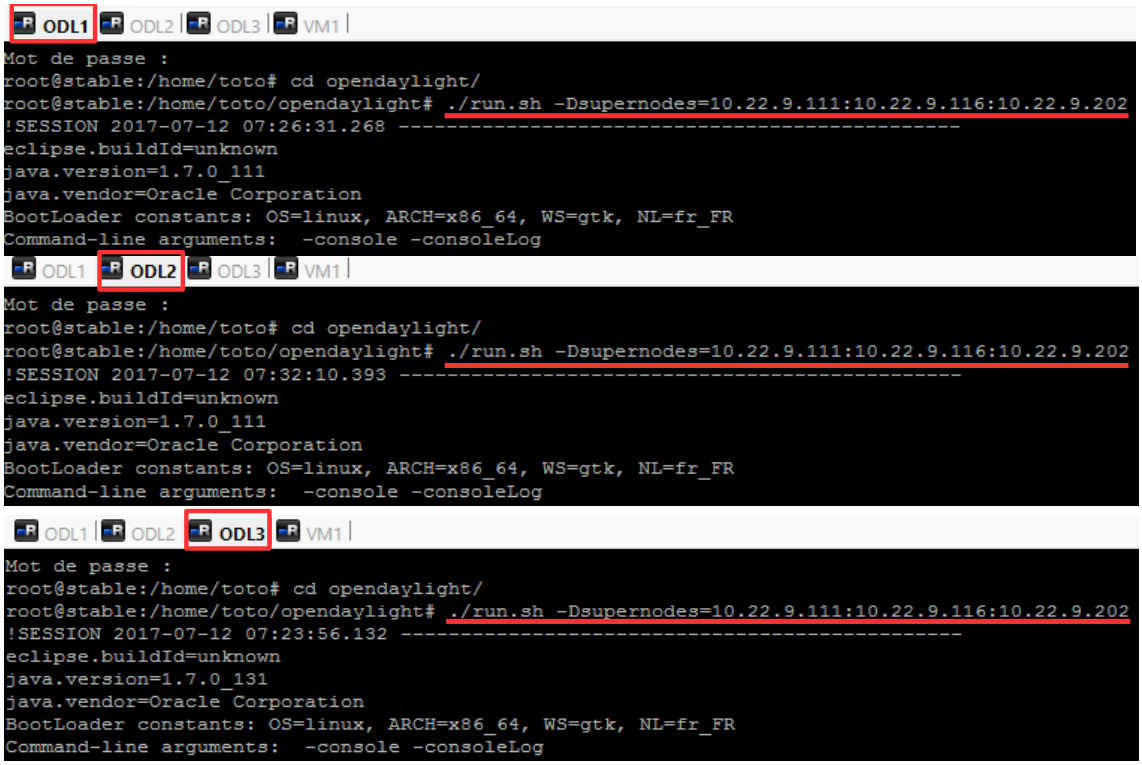

Figure 6.7: Start up the Cluster

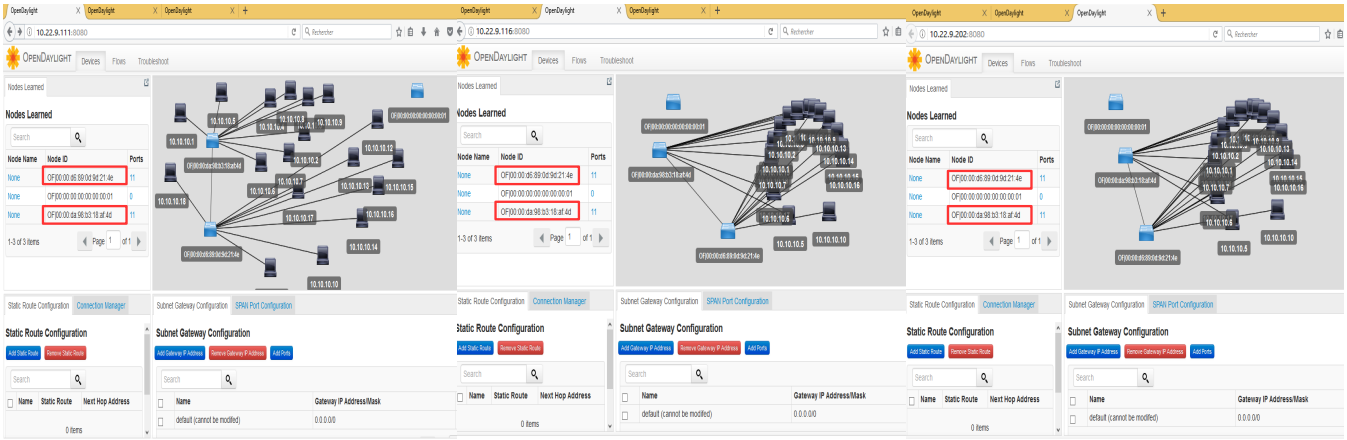

Figure 6.8: Cluster on ODL Hydrogen

6.3.2 MD-SAL Clustering

We also explore how to set up the configuration deployment using a MD-SAL Clustering architecture with the latest version of OpenDayLight. Before starting the cluster, we verify the connections between the ODL cluster nodes. Once the connections are verified, the cluster nodes are able to obtain set up configuration and can work together as one entity. Depending on the distribution to deploy the cluster, it can be downloaded on the ODL web server repository [123].

6.3.2.1 Setting up the cluster nodes

The latest versions of ODL use a Raft algorithm to provide high availability. This means that it needs at least three cluster nodes to have the majority of cluster members available. One of the most important features of clustering in MD-SAL is module shards. This feature allows allocation of the shards inventory, topology, toaster and default in different node replicas within a cluster. In the set-up configuration, we can decide where shard replicas will be placed.

With the ODL distribution file on each cluster member, the controller can be start up using *./bin/karaf* script file located on the Karaf distribution directory (Fig. 6.9). Once the controllers are brought up and running, it is possible to install karaf features which allow start up of some applications. The *feature:list -i* command permits exploration of the available features to install. Some of the features are installed by default for internal process on the controller. At the Karaf command, line the clustering is enabled by running *feature:install odl-mdsal-clustering* command.

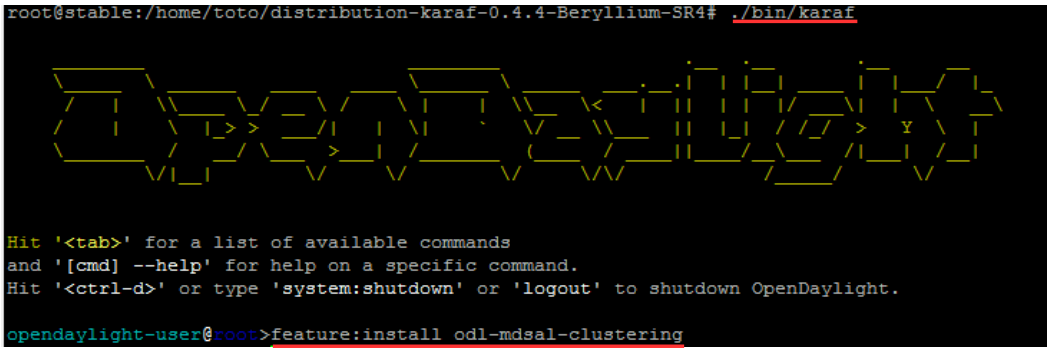

Figure 6.9: Cluster on ODL Beryllium

On each cluster node the initial configuration files akka.conf (Fig. 6.9) and module-shards.conf need to be configured with the clustering information. In akka.conf file, the configuration to set is the IP address of host and the port in which the node receives the joining nodes requests. Also, configure all seed nodes members of the cluster and define the role of each one.

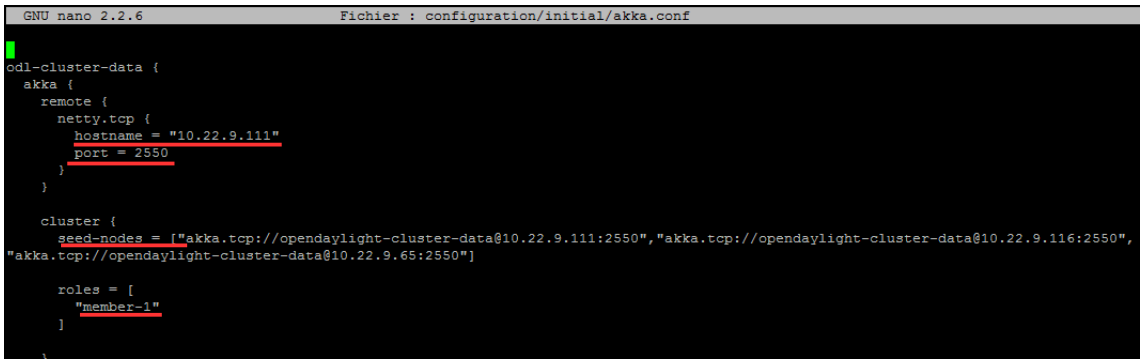

Figure 6.10: AKKA configuration

In the module-shards.conf file it is necessary to specify where the shard will be replicated

on the other clustered nodes. Therefore, the cluster may be initialized without any change on this file. The cluster sharding allows avoiding excessive loading of a node by distributing replicas of the shards across different nodes if needed. Once the configuration is done in all cluster members, the clustering services can start up with the three-cluster nodes running together as only one entity. Another way to configure the cluster is executing the bash shell script *./bin/configure_ cluster.sh index 1..N seed_ nodes_ list* provided only by the most recent distributions of ODL.

6.3.2.2 Clustering Test

Validation that the cluster configuration is correct is accomplished by using the GUI provided in ODL. It is possible to verify that all nodes are running up on the clustered network. This validation tests if the cluster is running properly, and also checks that the shard replications are satisfactorily distributed among nodes. Installing DLUX (DayLight User eXperience) user interface features on the text console of the started up ODL controller, provides a web service application for development and testing the cluster deployment. Access to DLUX is via the web browser at http://Remote-IP-Controller:8181/index.html.

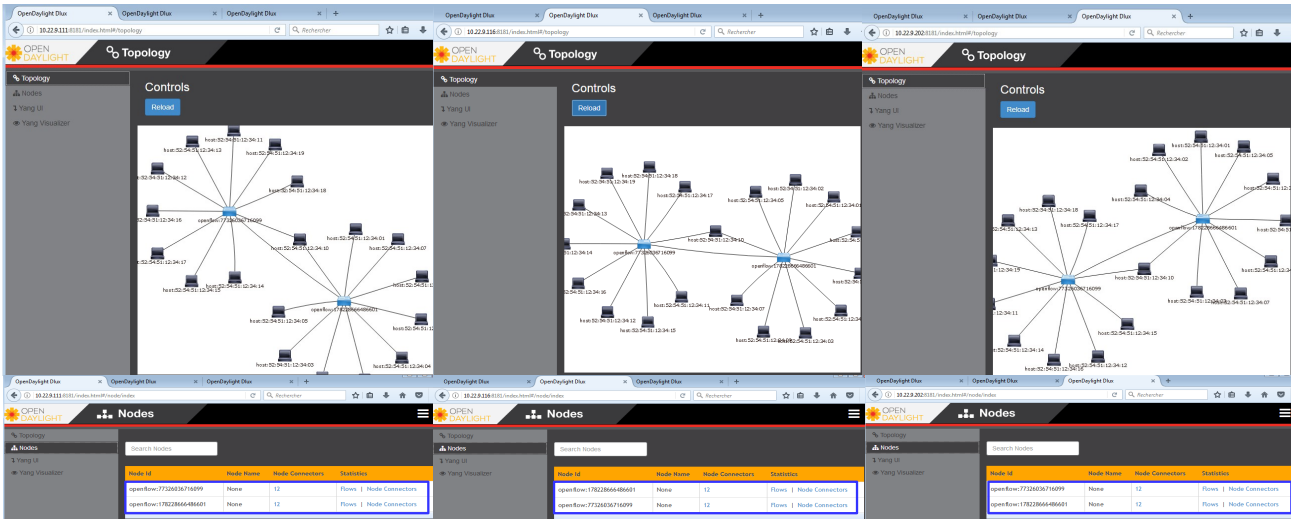

Figure 6.11: Cluster on ODL Berrylium

Using Postman HTTP requests allows identifying shard replicas information for every cluster member. By installing the Jolokia feature with the *bundle:install -s mvn:org.jolokia/jolokia-osgi/1.1.4* commands on the ODL console, we are able to explore the shards replication content. To read specific information about a shard, use the query GET `http://Remote-IP-Controller:8181/jolokia/read/org.opendaylight.controller:Category=Shards,name=member-1-shard-inventory-config,type=DistributedConfigDatastore`. This example returns shard inventory information. An example of information given is shown in the figure 6.12. The request made on cluster member 2 returns the shard leader member, the peer addresses and the state of the current follower node in the cluster. This detailed output helps to understand the shard configuration that was set up on the module-shards.conf, and also identifies statistics, counters and many other sharding operations.

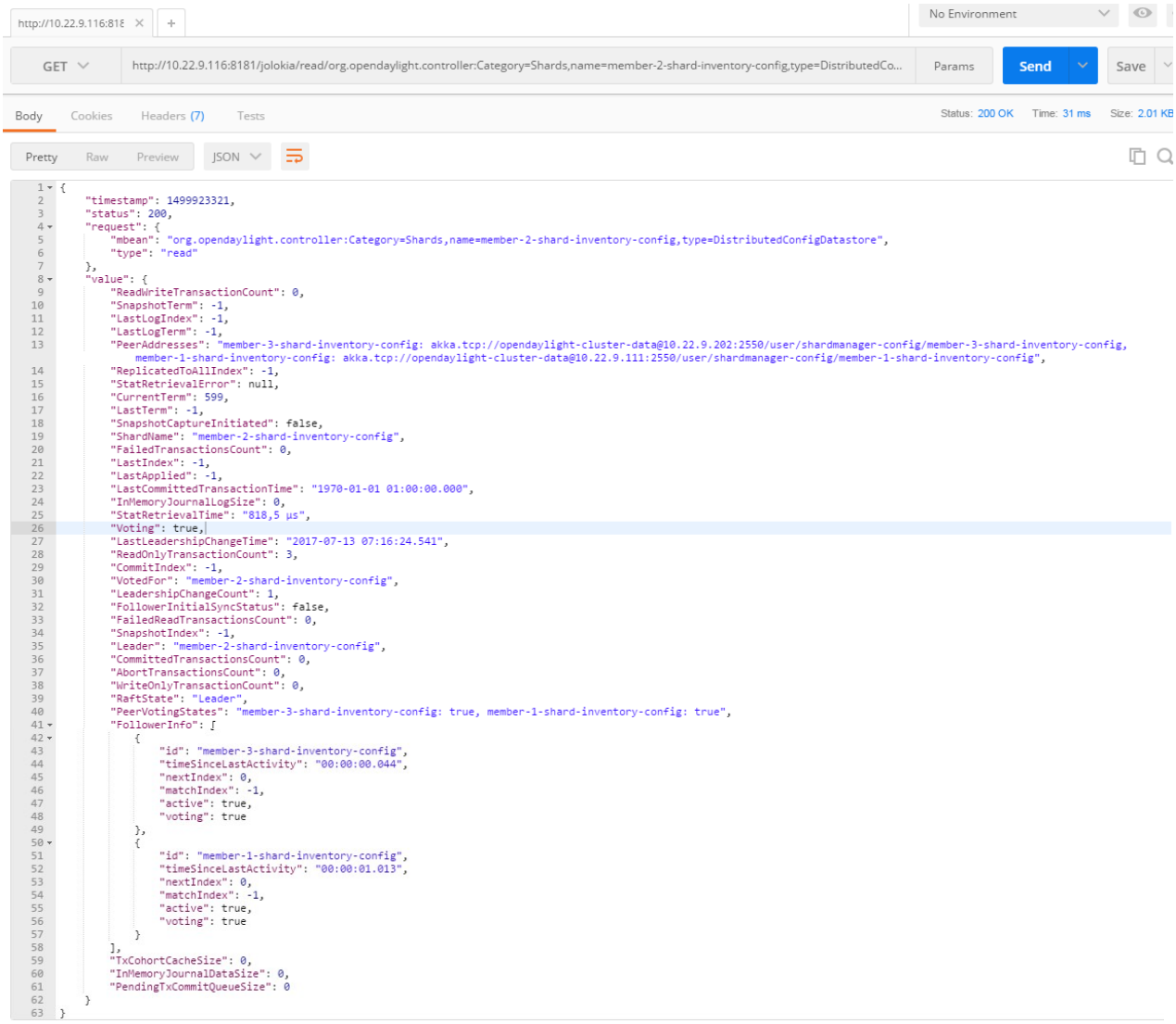

Figure 6.12: Shard inventory config

6.3.3 Difference to AD-SAL and MD-SAL clustering

In the clustering architecture in AD-SAL with ODL Hydrogen each clustered node is called *supernodes*. There are no configuration files to be configured. By only including *supernodes* at the bring up controller command line and the IP address of the reachable nodes, the cluster can be deployed. Clustering with AD-SAL solution is not complicated, but it has some scalability limitations. In addition, the *supernodes* store all the data clustering information in the same place, and we cannot distribute network information as can be done with MD-SAL cluster on ODL.

In the MD-SAL clustering solution provided in the latest version of ODL, it is possible to distribute the data store architecture. For example, the inventory data can go on one cluster node and topology on another node. Then, the data may be replicated to a defined number of nodes on the cluster and also eventually may be replicated to all clustered nodes using the gossip protocol.

During the experimentation, we observed that the OpenDayLight clustering solution does not have the appropriate mechanisms to deploy our proposed architecture. The cluster architecture of OpenDayLight replicates all the network data information on multiple controllers. With this procedure we cannot distribute routing functions and security policies on each border controller. This issue can be solved by deploying a router node in order to establish the communication link between the border controllers. This intermediate node is in charge of implementing all the routing decisions and the distribution of the data.

6.4 SDN Security

The Internet has grown rapidly in the recent decades and continues to be developed in terms of dimension and complexity. Many security challenges have been raised such as user authentication and accountability. Security is one of the most important network functions meant to protect the network and prevent attacks such as man-in-the-middle, unauthorized access, spoofing and denial of service, to only name a few methods. Network security can be enhanced using SDN features, for example programmability, that contain awareness and some global network state information. The SDN architecture provides many potential opportunities to achieve the exponential growth of the Internet. However, many SDN technologies are quite new, and they present many security challenges to optimizing the SDN deployment.

Motivated by the SDN's potential to bring solutions to many unsolved security problems and an enhancement of existing solutions, several research works have discussed the use of OpenFlow protocol to provide network security enforcement. In the following paragraph the main OpenFlow-based security application are discussed. The main objective of this section is to provide an overview of SDN security issues and present our approach for distribution of SDN applications.

6.4.1 OpenFlow-based security applications

OpenFlow protocol is the most used and widely deployed on the southbound interface of the SDN. It allows communications between network devices and SDN controllers via a control channel. Nevertheless, OF does not enforce security in this communication channel by itself at the startup implementation. This is because encryption with TLS is a suggestion and not a requirement, and it runs directly over plaintext TCP connections. Additionally, some vendors have not yet implemented any solution for securing the transportation channel of the message exchanges between an OF controller and an OF switch 6.4. An attacker can exploit this lack of authentication on a unsecured OF channel to infiltrate the network and eavesdrop or spoof network traffic on the southbound interface. It is particularly dangerous because if the attacker gain full control access, it can to take down the OF switches, alter OF rules into the switch, capture sensitive traffic and monitor how the controller manages the OF packets.

In the context of our framework, the OpenFlow channel communication of the OVS and the ODL controller is encrypted by using SSL/TLS connection based on the Public Key Infrastructure (PKI) model. The PKI must be initialized with *ovs-pki init* command which creates the Certification Authority (CA). With the command *ovs-pki req+sign sc ctl controller* we can generate the controller private key and certificate files. Next, the OpenSSL toolkit allows

Controllers/Switches	TLS Support
OpenDayLight	✓
POX	✗
NOX	✗
Beacon	✗
Floodlight	✓
Ryu	✓
ONOS	✓
FlowVisor	✗
Pica8	✓
OpenVSwitch	✓
Brocade	✓
HP	✓
Big Switch Networks	✗
Arista	✗
Indigo	✗
Cisco	✓

Table 6.4: Brief of Controllers and Switches that support TLS

creation of keystore in PKCS12 format which contains the controller private and public keys. Then, the keystore is imported into a JKS format keystore adapted to be configured on the OF configuration file of the ODL controller (Fig. 6.13). To create the public key, private key, and certificate for the OVS, the command *ovs-pki req+sign sc switch* makes the CA files. Using the *ovs-vsctl set-ssl* command to point to the directory of the previously-generated public and private keys allows configuration of the OVS with SSL encryption. Also, it is necessary to add the controller certificate *cacert.pem* which authorizes the connection to the ODL controller by verifying the signature of this CA certificate. At this stage, the TLS/SSL connection between OVS and ODL controller is working together to communicate securely in both directions. Nevertheless, SSL/TLS communications are vulnerable to interception and modification through a man-in-the-middle attack if the private key is leaked. All private keys must be keep secret in a secure space, since anyone in possession of the matching private key can decrypt a message encrypted with the private key.

Parallel into traditional network architectures (Fig. 6.14), the control and data planes of network interconnecting devices are bundled together. As we described previously, the control plane provides functions to build forwarding/routing decisions to send data link layer frames (L2) and network Layer packets (L3). The data plane functions by carrying out the commands of the control plane through forwarding tables, routing tables and ARP tables, among others. With this principle, the network nodes must compute the path to the destination after exchanging routing information. As a result, any other process requires many successive operations compared to the SDN architecture in which the routing decisions are made only by the controller. Network devices will only handle packets based on the flow table entries received from the controller via the OpenFlow protocol. With this in view, each flow entry has a match/action set up against fourteen required header match fields which provide a higher

```
*42-openflowplugin-He.xml ×
      <module>
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">prefix:openflow-switch-connection-provider-impl</type>
        <name>openflow-switch-connection-provider-default-impl</name>
        <port>6633</port>
        <transport-protocol>TLS</transport-protocol>
        <switch-idle-timeout>15000</switch-idle-timeout>
        <tls>
         <keystore>configuration/ssl/ctl.jks</keystore>
         <keystore-type>JKS</keystore-type>
         <keystore-path-type>PATH</keystore-path-type>
         <keystore-password>opendaylight</keystore-password>
         <truststore>configuration/ssl/truststore.jks</truststore>
         <truststore-type>JKS</truststore-type>
         <truststore-path-type>PATH</truststore-path-type>
         <truststore-password>opendaylight</truststore-password>
         <certificate-password>opendaylight</certificate-password>
         <cipher-suites>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_DHE_DSS_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
             </tls>

      </module>

      <module>
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:openflow:switch:connection:provider:impl">prefix:openflow-switch-connection-provider-impl</type>
        <name>openflow-switch-connection-provider-legacy-impl</name>
        <port>6653</port>
        <transport-protocol>TLS</transport-protocol>
        <switch-idle-timeout>15000</switch-idle-timeout>
         <tls>
         <keystore>configuration/ssl/ctl.jks</keystore>
         <keystore-type>JKS</keystore-type>
         <keystore-path-type>PATH</keystore-path-type>
         <keystore-password>opendaylight</keystore-password>
         <truststore>configuration/ssl/truststore.jks</truststore>
         <truststore-type>JKS</truststore-type>
         <truststore-path-type>PATH</truststore-path-type>
         <truststore-password>opendaylight</truststore-password>
         <certificate-password>opendaylight</certificate-password>
          <cipher-suites>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_DHE_DSS_WITH_AES_256_GCM_SHA384</cipher-suites>
         <cipher-suites>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</cipher-suites>
</tls>
    </module>
```

Figure 6.13: OpenFlow configuration file in ODL to support TLS.

level of granularity than the traditional forwarding technique.

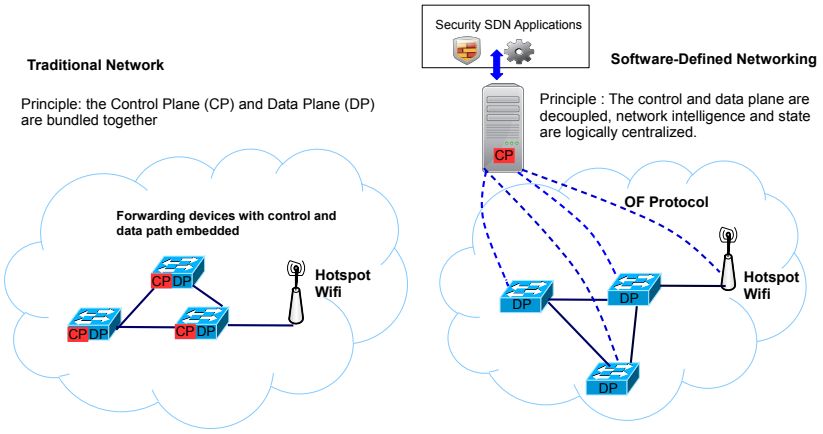

Figure 6.14: Traditional Network and Software-Defined Networking.

Since its standardization by the Open Networking Foundation, many OpenFlow versions are now available and supported by a variety of vendors' devices. At the time of the writing of this dissertation and to best of our knowledge, OpenFlow 1.5 is the latest version of the protocol. As noted in the previous paragraph, in this version, packets can be matched against fourteen required header match fields which provide a higher level of granularity than the traditional forwarding technique that uses only the layer 2 destination address. Note that not all header fields have to be used in all flow tables. The SDN controller can ask the switch about which match fields are supported in every flow table. Each flow table may contain many flow entries. By Customizing an OF pipeline we can determine the manner in which the packets should

interact with those flow tables.

Field	Description
OXM_OF_IN_PORT	Physical or logical port
OXM_OF_ACTSET_OUTPUT	Egress port from action set
OXM_OF_ETH_DST	Ethernet destination address
OXM_OF_ETH_SRC	Ethernet source address
OXM_OF_ETH_TYPE	Ethernet type of the OpenFlow packet payload
OXM_OF_IP_PROTO	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	IPv4 source address
OXM_OF_IPV4_DST	IPv4 destination address
OXM_OF_IPV6_SRC	IPv6 source address
OXM_OF_IPV6_DST	IPv6 destination address
OXM_OF_TCP_SRC	TCP source port
OXM_OF_TCP_DST	TCP destination port
OXM_OF_UDP_SRC	UDP source port
OXM_OF_UDP_DST	UDP destination port

Table 6.5: OpenFlow matching fields

However, the Openflow protocol sends to the controller only information it contains, called flows (see Table 6.5). In the case of monitoring network devices and the exchanged information, we would also be able to control not only IP, TCP or UDP packets but also application-level packets like HTTP or FTP. Openflow does not provide the appropriate mechanism to manage the security of a network at the application level. Also, once a flow entry is installed on the OVS, OF requests do not reach the controller. All packets will be processed only at the network forwarding device level, layer 2-3.

In a most recent work like [33], an alternative protocol to OpenFlow, OpFlex has been proposed by Cisco. This SDN paradigm offers a new potential solution for overcoming the challenge of managing packets at the application level. In conjunction with industry partners and open source communities, Cisco offers a compatible plugin of Opflex with the OpenDayLight Helium release and OpenVSwitch. This Protocol is based on two key elements, the Policy Authority and the Policy Agent. The policies are defined in the Policy Authority. Then they are resolved asynchronously by the OpFlex Policy Agent. The device or host is connected via the Policy Agent, empowering the nodes to make more decisions about how to execute the policies.

Another approach to managing the application level is to use the Group Based Policy (GBP)

RESTCONF northbound API provided by ODL. The GBP groups the EndPoints (EPs) into EndPoints Groups (EPGs), then it applies policy contracts to the traffic between groups. EPs may include the virtual interface, physical interface or another network device. Each EP should be assigned to an EPGs. The EPGs use contracts to define what kinds of traffic are allowed on the network. These contracts decide how EPs can communicate between themselves. A contract provides a subject which defines some rules to match the traffic network and clauses determines how the contract may be applied to EPs and EPGs.

For development of our framework, Openflow protocol defines control messages that enable the SDN controller to establish a secure connection with the network devices, read their current state and install forwarding instructions. But, with Openflow the network devices are only able to handle the flow/forwarding action by predefined rules. For instance, if one device is corrupted by a virus, this threat can propagate itself in the network without any control. Moreover, anyone can connect his device on the network. Our work could be extended the use of the OF protocol to include the management of the network protocol headers such as an http header, icmp, ssh and many others. Using Opflex or GBP will allow distribution of the policy security rules to the endpoint devices optimizing network management operations at layer 4-7. At each exchange, the interconnecting devices would verify policy rules previously defined on the northbound API controller. With OF the network intelligence resides on the controller, but with a GBP the network intelligence can be distributed to the network devices allowing them to implement forwarding decisions and apply services without requiring additional information from the controller. This feature is very useful because in case an OF controller without cluster architecture fails, the network will no longer be functional.

Figure 6.15 illustrates security architecture for enterprise provided by Cisco and the main idea of the proposed approach combining SDN and GBP with OpFlex.

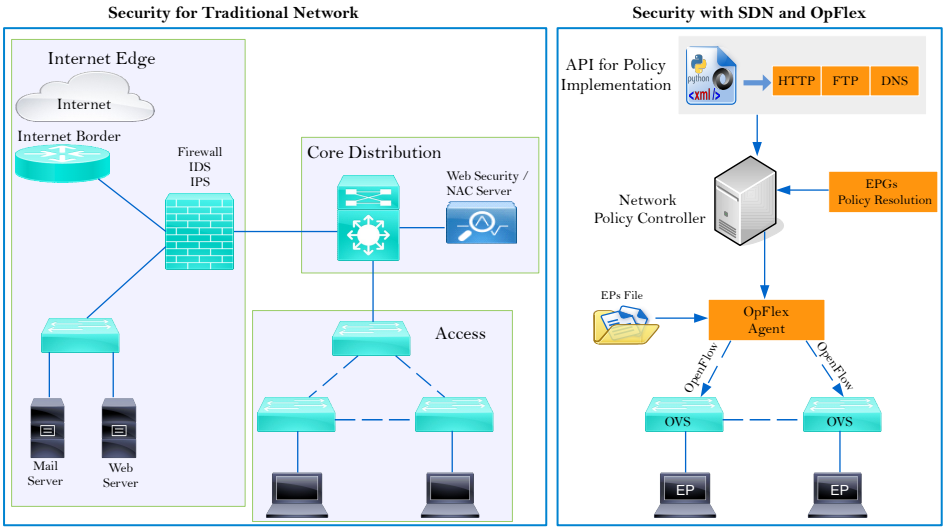

Figure 6.15: Traditional security network architecture and security based on SDN policies.

6.4.2 Some SDN Drawbacks and Solutions

The centralized architecture of SDN entails dependency on the control plane, it exposes the controller to a potential Denial of Service (DoS) attack. This can be mitigated by the High-Availability (HA) architecture with multiple controllers. Redundant SDN controllers ensure network availability, avoiding network failover as would be the case if a single controller should become unavailable. However, without a careful security policy design, all cluster members can be exposed to DoS attacks.

Another weakness of many SDN controllers can be found on the northbound HTTP API, because they have no any encryption and authentication method of communication. ODL have SSL encryption on the northbound HTTP API, by default it is disabled. In addition, the access to GUI in ODL is a basic HTTP authentication with a weak default password. This means that an attacker can gain full control of a network controller directly in the absence of stringent security measures. Despite having multiple controllers in place, the weaknesses can be exploited by bypassing one of the controllers and listening to all services on the other controllers. However, it is possible to set up stronger password and enforce the access policies by using the Authentication, Authorization and Accounting (AAA) feature provided in ODL. In this way, the possibility of attacks on the northbound interface controller may be reduced.

If an attacker successfully gains unauthorized access to the controller, it will be able to add/remove access, change flow table contents, modify network topology, upstream/downstream controllers and intercept sensitive data. Like any emerging technology, the SDN introduces new risk and it will be potentially vulnerable to new types of currently unknown attacks. That is especially the case with many relatively new SDN systems which come with new security challenges. But some possible attacks on SDNs are not new, and there are a considerable numbers of network security solutions for dealing with this kind of attack.

6.4.3 Distributed Security Solution on the SDN Cluster

Many works have studied network security using the SDN architecture, either by implementing firewalls, IPS, and IDS modules on top of the SDN controller [72, 71, 73], or by installing security policies into OpenFlow switches [136]. The emergence of the next generation Internet architecture, requires even higher security levels, such as authenticating network devices and user objects connecting to users using both wired and wireless technologies. Furthermore, we need to monitor the behavior of both the users and the objects, establish trust boundaries, and use accounting methods along with software verification. However, existing security mechanisms [136, 72, 71, 73] do not provide levels for securing the next generation Internet architecture.

Inspired by existing Network Access Control and security techniques [22], we suggest an extended secure SDN based architecture for the IoT. To explain our architecture, we first present a simple solution in which a controller manages the security of one SDN domain. Second, we can extend this first solution in order to include multiple controllers with regard to the available resources on each control platform. In addition, we extend the distributed control architecture by interconnecting all SDN domains via border controllers, which takes us close to a secure model for the IoT.

Traditional Ad-Hoc architecture does not provide network access control or global traffic monitoring, due to the absence of the network infrastructure. The proposed architecture

overcomes those security limitations and enables dynamic network configuration and security policies deployment. Our purpose is to achieve maximum synchronization of SDN Controllers within a security perimeter, enabling granular control over network access and continuous monitoring of network endpoints.

In our testbed framework, we have studied the OpenDayLight MD-SAL clustering solution. We explored the effectiveness and performance of SDN controllers in the OpenDayLight cluster deployment context. The experimental results are shown in section 6.3. To build a cluster, we need to define the member nodes of the cluster in the configuration files of the controllers. When the clustering service comes up, the cluster member-1 replicates all control functions among the cluster members. Our goal is not to replicate all control functions on multiple controllers but to distribute routing functions and security rules on each controller of each cluster. In this case, the cluster definition used is not the same as that we have defined in the literature of ad-hoc networks. We cannot use it to design our SDN clustered architecture.

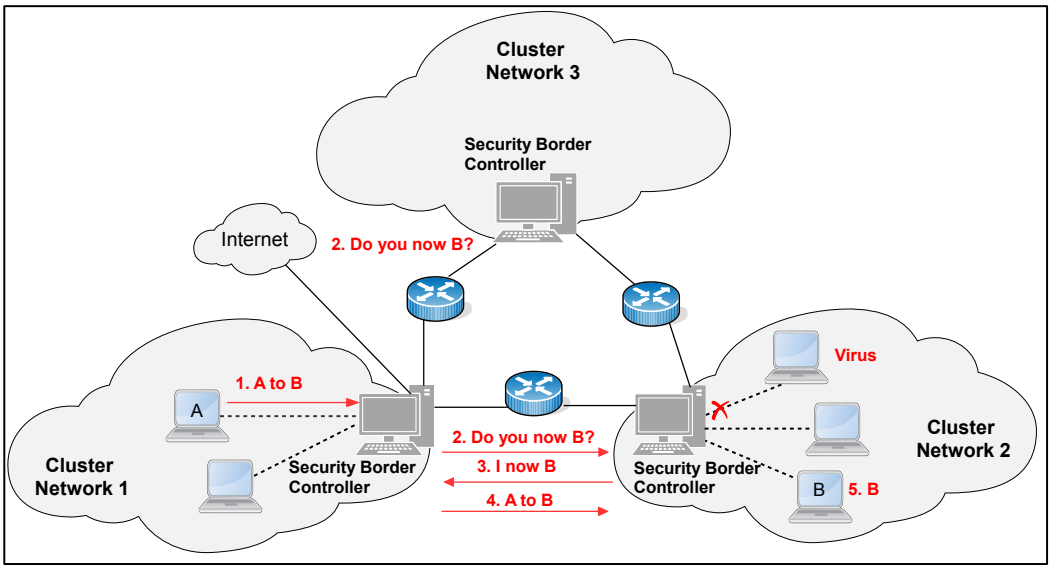

Figure 6.16: Grid of Security in SDN Domain.

To manage the clustered network traffic in our proposed architecture, we propose the solution of Figure 6.16. Each communication link between the SDN controllers is managed by a router node. This intermediate device is responsible for transmitting and sharing the OpenFlow messages. When a device transmits the data packet at the switch ingress port, the switch will automatically deliver the contents to its SDN controller. The packet will get a current connectivity state to match against the flow rules, then, it can be forwarded. If the destination of the packet is in another cluster, this packet is sent up to the other controller via the router node, and this controller will process the data on the data path to the next hop. The router nodes use OSPF protocol to forward packets towards the next hop controller. If the controller node is unavailable, the devices cannot forward the received data packets to another device.

In order to secure network access and network resources, the SDN controllers:

1. Begin by authenticating the network devices,
2. Once the OpenFlow secure connection between the switch and the controller is established,

3. *T*he controller blocks switch ports directly connected to the users,
4. *A*fter that, the controller authorizes only user's authentication traffic,
5. Once the user is authenticated, and based on the authorization level of the user,
   (a) *T*he controller will push the appropriate flow entries to the software or the hardware access switch.
   (b) In IoT, we extend this authentication process to each device.
   (c) Each device has to associate itself with an OpenFlow enabled node, and all such nodes are connected to one controller in their domain.

The whole concept of grid security network is to extend the SDN domain concept to multiple domains (Fig. 6.16), and each controller of each domain exchanges their security rules with other controllers. There are SDN controllers which behave as a security guards on the edges of extended SDN Domains to ensure network safety. The safety connection between domains could be provisioned and added to SDN Controllers, so that only recognized traffic could be accepted. The controllers know policies within their domains, but they don't know policies of the other domains. So when a node wants to communicate with another node of another domain, the flow has to be forwarded to the Security Controller, also called the Border Controller. The security controller asks each neighbor controller if it knows the destination of the information (Fig. 6.16).

There is further potential to exploit this architecture; with a security controller in every domain we avoid users opening new services without authorization. Only services authorized by a security controller can be used for endpoint devices. In order to avoid any person or device initiating communication with another person or device, when any entity wants to open a communication port, it must make a request to the SDN controller. For instance, assuming that one user or device wants to open a web service, the first task is to request to its controller if there exists a web service opened on the network. Then each border controller asks each controller of each domain. If a such service exists on a device, then all messages will be forwarded to this device.

Furthermore, the Extended SDN controllers periodically monitor and check the flow table entries of the software switches, because they are deployed at the user side. In the proposed architecture, we deployed software switches at the user side to enhance ad-hoc devices' forwarding capabilities. Moreover, the deployed software switches allow the SDN controller to apply and enforce the security policies inside the ad-hoc area.

Conclusion of Part II

In this part, we present a new SDN-based architecture which can be used in the context of ad-hoc network and IoT. In ad-hoc networks, nodes in equal interaction roles have the capability to interconnect with other nodes/devices. In one domain, we can have networks with or without infrastructure. This approach is a promising solution which simplifies network management using the SDN. In the same way, nodes without OF capabilities may be associated to nodes with enough resources to support OF. The communications between domains is accomplished with a special node called a border controller. This border controller is responsible to control all traffic only for its domain. The Border Controller is able to deal with multi-distributed controllers in order to guarantee the independence of each domain in case of failure.

Also, we present a new SDN-based architecture for cluster network, called the Software Defined Clustered Sensor Network architecture (SDCSN). This approach is a first attempt to analyze the opportunities and challenges of applying the SDN clustered in WSN. On the top of the SDCSN architecture, we propose an SDNCH which is the coordinator on the domain. The main benefit is that the SDNCH can not only manage the network, also it can manage the security of one SDN domain. Therefore, the proposed cluster can not solve problems with routing processes in distributed SDN-architecture. But by combining routing protocols and SDN, a new effective controller-routing method can be used to distribute routing functions on each SDNCH. This promising solution was extended to include multiple controllers by interconnecting SDN domains via border controllers using AKKA-based clustering technology. Our simulation results in chapter 6 showing the efficiency and the adaptability of SDNCH performing many configurations in realistic scenarios. In the SDN principle the network intelligence is centralized but we distributed multiple SDN controllers to increase the overall scalability and availability.

We have implemented the OpenFlow protocol performing different test environments and also analyzed how to deploy a cluster architecture with the OpenDayLight controller. To improve the system performance, we built an SDN framework which allows a large number of implementations, including scalability, high availability and persistence of OpenFlow networks. With the developed model it was possible to observe the behavior of the OF protocol in large-scale experiments, including the test performance over the cluster service. One of the biggest parts of this research was to further understanding of the process of OpenFlow messages exchanged between the OF switch and the controller. Also, we analyzed how to add, modify or delete rules in the forwarding tables. We have developed a shell script which can process multiple OF packets based on pipeline processing. It is very useful for generating and installing flow entries onto the OF switches when there are multiple devices connected on the network. Additionally, we analyzed the clustering services behavior on ODL and showed how it works.

With the aim of understanding the roles and the sharding strategy to any member in the cluster, we provide a detail description of how to set up the configuration needed to deploy a cluster architecture. The simulation results prove that the proposed framework can achieve the performance, scalability and flexibility needed for our approach to ad-hoc, IoT and cluster computing systems.

An overview of security weaknesses and solutions of SDN are provided in this part. Also, we have proposed a promising security solution addressed to SDN clustered architecture. We have highlighted the security vulnerabilities in the northbound and southbound interfaces of SDN. Then, we have presented existing solutions to secure the communication between these interfaces and the controller. SDN has many benefits for enhancing and securing the network infrastructures, but at the same time it brings new security challenges.

In order to enhance the developed SDN cluster platform, we provide some security approaches to security measures which can effectively defend against and react to potential attacks. In this way, the clustered controllers can not only manage the network, but also monitor and efficiently secure the network, against outside and inside attacks. Existing security threats in traditional networks can be extended to the SDN architecture, because of the centralized nature of the control plane. However, the research community is already working to develop innovative security solutions and applications for SDN. Consequently, we have made a review of current research focused on OF and SDN vulnerabilities.

Résumé Partie II : Nouvelles Architectures pour le réseaux ad-hoc et l'IoT *Version Française*

Architecture des réseaux Ad-hoc basée sur le SDN

Au cours de ces dernières années, les réseaux ad-hoc ont fait l'objet de nombreuses recherches et développement grâce au large éventail d'applications fournit par type de réseau. Les réseaux ad-hoc sont adaptés à de nombreuses applications dont il n'est pas possible de construire un réseau avec infrastructure qui puisse prendre en charge une forte mobilité et auto-organisation. Certaines applications de réseaux ad-hoc tels que : le travail collaboratif, les opérations militaires, la santé, l'environnement, domotique, l'industrie et la surveillance sont quelques exemples d'utilisation de MANET. Par rapport aux réseaux informatiques traditionnels, le MANET a de nombreuses contraintes qui doivent être traitées au fur et à mesure. Certaines de ces contraintes du système limitent sa puissance, sa mémoire, sa bande passante de communication sans fil, son espace de stockage, sa puissance de calcul et par-dessus tout sa sécurité. Dans cette section, nous présentons un bref résumé des réseaux ad-hoc et décrivons l'architecture SDN proposée pour aborder la gestion d'un routage flexible pour les périphériques embarqués. Certaines parties de ce chapitre ont été présentées et publiées sous la forme d'articles scientifiques de conférences avec comités de lecture [12, 13, 15].

Dans un réseau ad-hoc, le processus de routage entre noeuds dépend de nombreux facteurs. En effet, il comprend des tâches complexes comme les modifications de topologie, l'initiation de requêtes et le calcul de la meilleure route pour déterminer rapidement et efficacement l'itinéraire de paquets dans le réseau. En outre, le calcul de la route vers la destination avec un grand nombre de noeuds connectés au réseau requièrent une grande quantité de trafic. De plus, les noeuds peuvent se déplacer de manière aléatoire. Par ailleurs, la sélection d'une route incertaine peut entraîner l'interférence du signal sur le canal de communication sans fil qui sera perturbée sur le trajet.

Le routage dans un MANET est différent du routage traditionnel trouvé dans les réseaux maillés, également nommés réseaux de capteurs sans fil *Wireless Sensor Network* (WSN). Normalement, le maillage du WSN consiste en un noeud-puits ou une station de base capable de gérer et d'optimiser les communications entre les noeuds. MANET se compose d'un groupe de noeuds qui se déplacent dynamiquement dans toutes les directions sans un contrôleur central fixe. Par conséquent, leurs itinéraires sont reconfigurés à plusieurs reprises. Les protocoles de routage réseau ad-hoc peuvent être classés en trois catégories : proactifs, réactifs et hybrides. Il existe un large nombre de protocoles de routage pour les réseaux ad-hoc qui ont été proposés par la communauté scientifiques. Par la suite, nous ne mentionnerons que les protocoles

de routage ad-hoc les plus importants tels que : *Destination-Sequenced Distance-Vector Routing* (DSDV), *ad-hoc On-Demand Distance Vector Routing* (AODV), *Dynamic Source Routing* (DSR), *Temporally ordered routing algorithm* (TORA), et *Zone Routing Protocol* (ZRP).

Dans le but d'activer une plateforme qui permettra de surmonter les limitations précédemment décrites sur le routage des réseaux ad-hoc, nous présentons une approche de politique basée sur des règles définies par la notion du SDN. La portée de la gestion du réseau ad-hoc est très large, mais avec son extension au SDN, elle peut augmenter les possibilités de prendre le contrôle, de gérer les comportements et de maintenir une connectivité transparente sur les noeuds ad-hoc de façon à simplifier son management. Dans l'architecture basée sur le SDN pour les réseaux ad-hoc, un noeud peut être consulté (Fig. 6.17) comme une combinaison de:

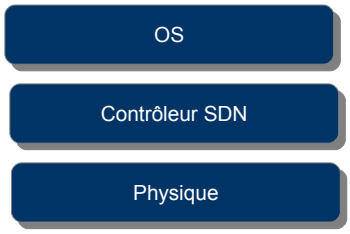

Figure 6.17: A noeud dans un reseau ad-hoc.

- Interfaces : La couche physique qui comprend les interfaces physiques ou les périphériques connectés dans un réseau ad-hoc. Il s'agit de la couche la plus basse avec des ressources matérielles telles que des téléphones, des tablettes et d'autres éléments numériques physiques. Chaque périphérique doit être une entité fournissant des abstractions virtuelles pour permettre le calcul.

- Couche programmable : Cela consiste en un commutateur virtuel compatible avec le SDN et un contrôleur du SDN. Ces commutateurs virtuels permettent aux utilisateurs ad-hoc de communiquer entre eux sur le même réseau. Le contrôleur transmet également les paramètres de connexion de commutateur virtuel aux dispositifs physiques permettant la communication à l'intérieur et à l'extérieur du réseau. Chaque commutateur virtuel dispose d'un ensemble de règles connues sous le nom de flux qui sont utilisés par le contrôleur SDN pour manipuler le trafic entrant et sortant du réseau.

- Système d'exploitation et ses applications : Ce sont les applications, en plus du système d'exploitation (OS), qui exécutent des tâches de gestion en utilisant l'interface OS. À un niveau plus élevé, l'OS gère les applications d'interface de gestion de réseau requises par les utilisateurs du réseau ad-hoc.

Un des avantages de cette nouvelle architecture de réseau ad-hoc basée sur le SDN est sa compatibilité avec le réseau du SDN. Puisque chaque noeud dans le réseau ad-hoc à un commutateur du SDN compatible intégré et un contrôleur du SDN, nous pouvons interconnecter le réseau ad-hoc sur le réseau existant pour construire un domaine du SDN étendu (Fig. 6.18). En outre, comme tous les contrôleurs du domaine du SDN étendu sont en interaction égale, toutes les règles seront synchronisées. Dans un travail plus récent comme [121], le domaine du SDN est

limité au réseau avec infrastructure. Dans cette configuration, les utilisateurs ad-hoc doivent se connecter par d'autres noeuds (la passerelle de réseau) directement liés au domaine du SDN. Dans l'architecture proposée, le domaine du SDN est étendu afin d'inclure tous les dispositifs ad-hoc. Notre approche consiste à déployer un commutateur logiciel OpenFlow, comme Open vSwitch [122] dans chaque noeud ad-hoc. Cette configuration permet aux noeuds ad-hoc de se connecter au réseau dans le cadre du domaine du SDN, ce qui permet d'appliquer les mêmes règles aux utilisateurs de domaine du SDN. Comme le montre la figure 6.18, l'architecture proposée prend en charge les réseaux avec ou sans infrastructure.

Dans ce type de réseau hétérogène, nous avons deux types de noeuds dans un domaine. Si un noeud a suffisamment de ressources pour supporter le SDN, il est nommé OpenFlow noeud (OF-N). Les noeuds sans ressources pour SDN sont appelés noeuds intelligents ou capteurs (S-N). Pour les périphériques IoT ou les réseaux de capteurs, chaque périphérique ne peut pas disposer d'un commutateur compatible SDN intégré et d'un contrôleur SDN comme nous l'avons proposé précédemment. Mais nous supposons que chaque périphérique avec faible ressource peut être associé à un noeud voisin qui possède la capacité SDN. Chaque domaine possède un contrôleur SDN de périphérie qui puisse contrôler tout le trafic uniquement dans son domaine.

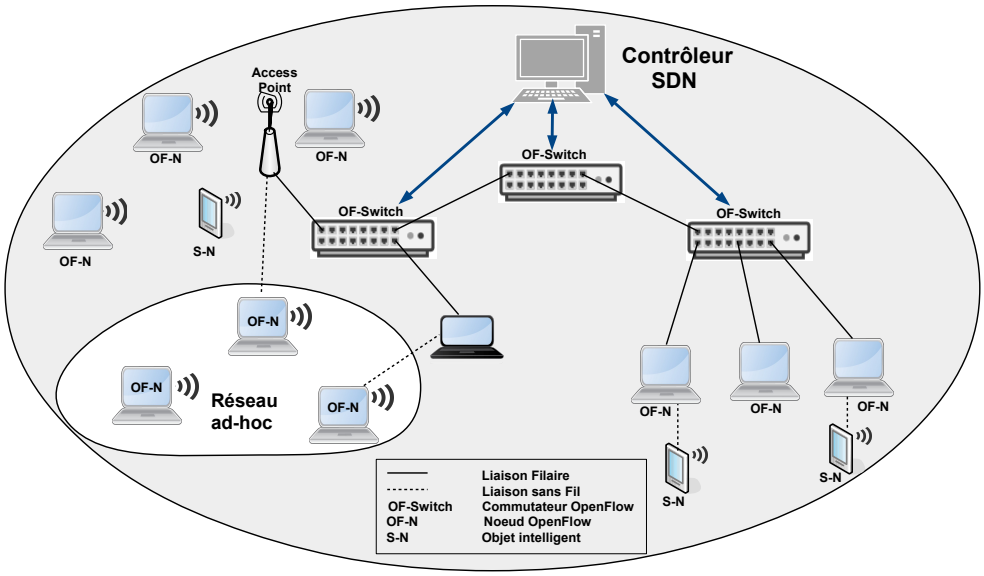

Figure 6.18: Domain Etendu du SDN .

Comme chaque noeud ad-hoc a son propre contrôleur du SDN, le plan de contrôle du SDN doit gérer l'évolution de chaque commutateur virtuel du SDN sur chaque appareil ad-hoc. Quand un nouveau dispositif ad-hoc se connecte ou quitte le réseau, cela provoque de nombreux échanges de messages, qui pourrait saturer le réseau ou le contrôleur central. Afin d'assurer l'évolutivité et la tolérance aux pannes, une architecture distribuée du SDN est préférée avec de multiples contrôleurs comme dans [22]. Pour veiller à cela, nous ajoutons de manière dynamique de nouveaux contrôleurs à la zone de réseau ad-hoc et autorisons les noeuds spéciaux à exécuter l'opération de contrôle. Les nouveaux contrôleurs partagent la même vision globale du réseau. Cependant, leurs fonctions et le domaine de gestion du SDN seront limités à une petite zone ad-hoc. En outre, ces contrôleurs seront chargés de surveiller le comportement des commutateurs

logiciels, déployés du côté de l'utilisateur.

Solution baséé sur le cluster

Dans le section précédent, nous avons présenté une architecture SDN pour les réseaux ad-hoc et l'IoT. Nous identifions également le nombre de paquets OpenFlow traités par les contrôleurs SDN pendant le processus de découverte de topologie du réseau dans un seul domaine. Ce chapitre présente une approche qui consiste à distribuer plusieurs contrôleurs du SDN en clusters. En outre, ce modèle permet de distribuer des fonctions de routage et des règles de sécurité parmi les contrôleurs. Nous étudions un modèle de cluster de haute disponibilité basé sur la couche d'abstraction de service, *Application-Driven* (AD-SAL) et le modèle *Model-Driven* (MD-SAL), fournie par le contrôleur OpenDayLight (ODL) [123].

Une des fonctionnalités du SDN qui nous a particulièrement intéressées au cours de ce projet de recherche est le *clustering* (Grappe de serveur). Les premières expériences que nous avons réalisées ont été effectuées sur ODL *Hydrogen Release Base Edition*, ce qui nous a permis de configurer plusieurs contrôleurs ODL dans un premier temps. En ajoutant une ou plusieurs instances ODL, nous pouvons rendre le cluster hautement disponible et augmenter la résilience. Du côté de l'interface Sud (*Southbound*), les commutateurs OF actifs peuvent se connecter à deux ou plusieurs instances du contrôleur ODL via une connexion TCP/IP point à point. D'autre part, l'interaction entre les contrôleurs et les applications de l'utilisateur est effectuée via les services Web RESTful du côté l'interface nord (*Northbound*).

Notre dernière expérience porte sur le *Model-Driven Service Abstraction Layer* (MD-SAL), qui est un nouveau composant fourni par les dernières versions d'ODL. Pour le déploiement de cluster, MD-SAL est basé sur la technologie populaire de clustering AKKA [129]. Les modules AKKA fournissent un réseau décentralisé *peer-to-peer* avec une grande disponibilité pour éviter un point unique de défaillance (SPOF) ou les goulots d'étranglement. Les membres du cluster communiquent par le moyen du gossip protocole et d'un détecteur de panne automatique. Cette fonctionnalité permet à chaque membre du cluster de rejoindre ou de quitter le réseau sans aucune modification sur le fichier de configuration. Le clustering AKKA est basé sur le système distribué *Amazon Dynamo* [130] en particulier la base de données distribuée *Riak*.

Basé sur ce modèle du cluster, nous présentons le concept cluster de réseaux de capteurs définis par logiciel (SDCSN) en supposant que le réseau de capteurs sans fil peut contenir des centaines ou des milliers de noeuds capteurs. Normalement, un réseau à grande échelle ne peut fonctionner efficacement sans une structure fixe organisée. Pour cette raison, nous proposons de regrouper le réseau en supposant que chaque cluster head est un contrôleur SDN. Dans chaque domaine SDN, le cluster head de réseau définie par le logiciel (SDNCH) [137] est en charge de gérer le fonctionnement des noeuds capteurs (Figure 6.19). Avec cette approche de *clustering*, les informations collectées sur l'environnement sont traitées dans le domaine par des noeuds et sont ensuite acheminées vers le SDNCH. En outre, le contrôleur est le noeud le plus puissant du cluster. En utilisant l'interaction active/active entre SDNCH, le contrôleur aura un accès complet au commutateur et aux mêmes règles des flux. Sur la base de cette approche, nous avons pu définir les paramètres de configuration, stocker les données et agréger les informations collectées dans le domaine ou envoyer les informations au puits ou à un autre SDNCH.

Chaque noeud capteur échange des informations avec ses voisins plus proches ou ses voisins

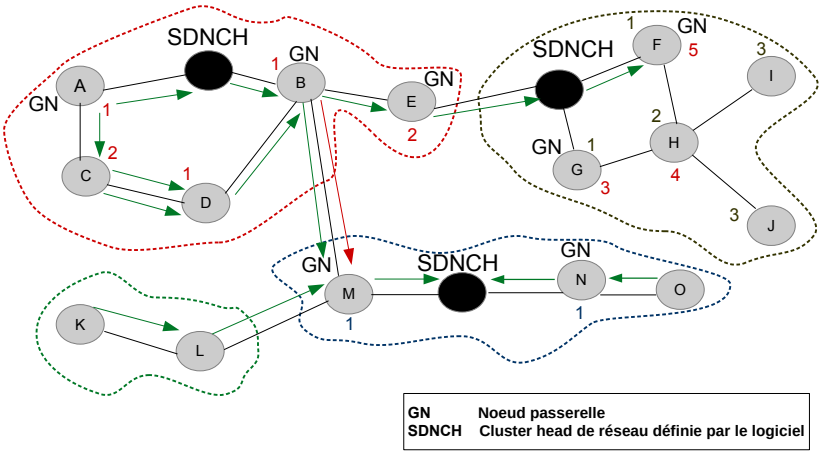

Figure 6.19: L'acheminement sur le cluster head de réseau définie par le logiciel.

au 2-saut. D'après l'hypothèse d'une connectivité en graphes, les informations générées sur un capteur peuvent atteindre le SDNCH en acheminant l'information via le réseau. La passerelle est engagée pour l'agrégation et la transmission des données de l'ensemble du noeud capteur dans le domaine vers les autres domaines. Ainsi, lorsqu'une passerelle n'est plus disponible, la communication entre les domaines est déconnectée et les capteurs associés au SDNCH sont isolés. Le SDNCH joue le rôle de coordinateur dans tous les domaines. Chaque SDNCH agit comme une station de base temporaire dans son domaine, et il est capable de communiquer avec d'autres SDNCH. Un domaine est donc composé d'un SDNCH, des passerelles et des noeuds capteurs.

Pour éviter la gestion de plusieurs clusters avec quelques contrôleurs, nous construisons un nouveau groupe de clusters appelé domaine. Un domaine est un ensemble de clusters gérés par un contrôleur SDN appelé contrôleur de périphérie. Avec cette approche, nous étendons le concept vers un SDSN domaine. Dans chaque domaine, tout le trafic est contrôlé par le contrôleur de périphérie SDNCH, qui a ses propres politiques de sécurité et stratégie de gestion pour distribuer les fonctions de routage et les règles de sécurité à d'autres contrôleurs de périphérie. Nous adoptons cette architecture pour garantir la sécurité avec le concept d'une grille de sécurité [126] intégrée dans chaque contrôleur et ainsi éviter les risques potentiels d'attaque. Lorsqu'un noeud capteur doit transmettre les données collectées dans un autre domaine, le flux doit être transféré vers le contrôleur sécurisé de périphérie du domaine le SDNCH.

Dans l'architecture SDN que nous avons proposée pour l'IoT [12], l'idée est d'étendre l'architecture du domaine SDN à plusieurs domaines, chacun ayant un ou plusieurs contrôleurs SDN qui contrôlent tout le trafic et gèrent seulement les périphériques dans son domaine. Un domaine représente un réseau d'entreprise ou un datacenter. Avec ce modèle, nous pouvons étendre le domaine SDN pour inclure des objets intelligents tels que des tablettes, des téléphones intelligents et des véhicules mobiles, pour n'en nommer que quelques-uns. Cela peut être réalisé avec le développement d'une plateforme qui puisse intégrer les commutateurs logiciels OpenFlow dans ces périphériques. Notre approche ne consiste pas à distribuer les fonctions de contrôle sur plusieurs contrôleurs, mais plutôt de distribuer les fonctions de routage et les règles de sécurité sur chaque contrôleur. Ces contrôleurs seront alors responsables d'établir les

connexions et d'échanger des informations avec d'autres contrôleurs voisins. Pour sélectionner un chemin approprié entre les noeuds connectés au cluster, le processus de routage des flux a été indiqué dans (Fig. 6.19).

Implémentation et évaluation

Auparavant, dans le section 6.4.3 and 6.4.3, nous avons présenté une architecture SDN pour les réseaux ad-hoc et IoT qui a été une des premières approches à utiliser les technologies SDN pour la gestion de ces types de réseaux. En outre, le chapitre 6.4.3 développe le concept de SDNCH, une architecture SDN clustered pour WSN. Dans ce chapitre, nous étudions les expérimentations que nous avons réalisés, dans les architectures proposées sont exploitées pour améliorer les performances de la mise en oeuvre du SDN dans les réseaux ad-hoc, IOT et cluster. Certaines parties de ce chapitre ont été présentés et publiées sous la forme d'articles scientifiques aux conférences avec comité de lecture [16, 18].

Nous avons utilisé le contrôleur du SDN OpenDayLight [1] pour les implémentations en utilisant le protocole OpenFlow 1.3. Selon le protocole OpenFlow, un commutateur OpenFlow transmet un paquet au contrôleur qu'il possède ou non la fonction de correspondance pour ce paquet. Lorsqu'un hôte veut envoyer un paquet IP sur Ethernet vers un autre hôte, l'acheminement du paquet sera disponible à travers une requête ARP. Lorsqu'un nouveau flux d'un hôte arrive à un commutateur OpenFlow, le commutateur achemine le paquet vers le contrôleur SDN. Le contrôleur joue le rôle de proxy ARP. S'il n'y a pas de correspondance avec une règle de flux installée, le paquet sera rejeté. Sinon, une demande ARP sera autorisée à passer vers l'hôte destination. Lors de la réception de la réponse ARP, le routage dans le contrôleur envoie le flux au commutateur connecté à l'hôte source et au commutateur connecté à l'hôte de destination. Une fois le lien établi, le commutateur décide du chemin le plus court pour acheminer le paquet.

Pour réaliser les simulations, nous avons virtualisé des commutateurs compatibles OpenFlow (OpenVSwitch version 2.5.0) avec les distributions Linux (TinyCore avec 48Mo de RAM). Pour évaluer les performances du protocole OpenFlow, le scénario expérimental déployé sur notre SDN testbed consiste en OpenDayLight Helium-SR4 et cinq machines virtuelles (VM) connectées au même réseau. La machine qui exécute le contrôleur utilise un système d'exploitation Ubuntu 14.04 et dispose de 2 CPU virtuelles et de 16 Go de RAM. Chaque machine virtuelle a 8 CPU virtuelles et 16 Go de RAM, avec le système d'exploitation Debian 8.3 et OVS préinstallé. Nous utilisons KVM pour la virtualisation des machines client qui exécute les images TinyCore Linux 3.16.6 avec 48 Mo de RAM. À partir de cette plateforme de type testbed, nous pourrons exécuter nos simulations de protocole et de l'architecture du SDN avec plus de 500 dispositifs lors d'une première tentative. Chaque périphérique utilisé est un système TinyCore Linux.

Pour effectuer les décisions de transfert du trafic entre les hôtes connectés au réseau, avec OF il faut installer un ensemble de flux de données sur des tables de flux (flow table) spécifiées sur chaque commutateur OF. Une table de flux est composée d'un certain nombre d'entrées de flux dans lesquelles chaque entrée d'acheminement est associée à une instruction d'action. Cette action décide comment les paquets doivent être traités par le contrôleur. Le contrôle d'acheminement de chaque entrée dans la table de flux est programmé sur le contrôleur SDN. Un commutateur OF peut contenir une ou plusieurs tables de flux et peut regrouper les dé-

cisions de transfert dans une table de flux ce qui permet d'ajouter de nouvelles entrées ou modifier/supprimer les entrées de flux existantes. Pour prendre ces décisions d'acheminement, les entrées de flux peuvent être ajoutées à l'aide d'outils tels que POSTMAN Chrome API, YANG UI ou *ovs-ofctl add-flow*.

Le contrôleur SDN peut prendre des décisions de transfert au niveau couche 2-3, car il a une vue globale de l'ensemble du réseau. Par conséquent, il a connaissance de la topologie, ainsi que des périphériques attachés à la topologie avec adresses IP/MAC identités. Ensuite, la programmation des commutateurs peut être faite via une API ouverte, ce qui permet au contrôleur de gérer et surveiller constamment la mise à jour du réseau Openflow même pendant des modifications. Au niveau couche 2 avec le contrôleur ODL, l'OVS agit comme un commutateur d'apprentissage de couche 2 régulier, il crée automatiquement une table d'apprentissage en fonction de l'adresse MAC source de la trame entrante et la place sur un port d'entrée. Par la suite, le commutateur transmet la trame vers la destination du port de sortie appropriée en fonction de l'application *odl-l2switch-switch* fournie par ODL. Un commutateur OF est composé d?un nom_port, d'un commutateur et d'un out_port.

Pour créer le routage à la couche 3, les opérations de transfert au niveau du flux ou du paquet sont donc traitées par les règles définies par contrôleur dans les périphériques matériels avec la commande *ovs-ofctl add-flow* ainsi que la résolution ARP entre les hôtes et les commutateurs voisins. Si le commutateur reçoit un paquet qui ne correspond à aucune entrée dans la table de flux, le commutateur Openflow transmet le paquet au contrôleur par une requête ARP. Lorsque la requête est reçue par le contrôleur, et s'il ne connaît pas l'adresse IP NextHop nécessaire pour gérer tout le trafic transféré vers un autre sous-réseau cette requête sera ignorée. Pour configurer le routage sur l'OVS il faut ajouter le NextHop, incluant une IP adresse passerelle affectée pour établir la connexion entre tous les noeuds.

Dans notre prototype, nous avons créé un script personnalisé pour démarrer simultanément plusieurs machines virtuelles qemu-kvm (Procédure 6.1). Ce script consiste à lancer les machines virtuelles hébergées sur une machine hôte et les instructions pour interconnecter les commutateurs. Cela facilite un contrôle de session à distance et la capacité de gérer des machines virtuelles fonctionnant sur des hôtes hyperviseur. Chaque noeud démarré est en écoute des connexions entrantes sur un port ouvert et son adresse IP via Virtual Network Computing (VNC). Lorsque les configurations ont été complétées, la machine virtuelle peut être utilisée pour la mise en oeuvre et l'expérimentation dans une émulation de réseau OpenFlow testbed avec un environnement complètement virtualisé.

Pour construire un réseau à grande échelle à la couche niveau 3, les décisions de transfert IPv4 OpenFlow nécessitent des centaines de milliers de flux en un seul commutateur pour permettre la mise en oeuvre des réseaux avec scalabilité. Cela a fait naitre un nouveau défi pour interconnecter plusieurs périphériques dans notre prototype. Pour configurer un tesbed d'OpenFlow scalable, nous avons développé un script shell pour simplifier les entrées de flux ajoutées à chaque table de flux qui permet ainsi la gestion de l'ensemble du réseau par le contrôleur ODL (Procédure 6.2).

Nous avons expérimenté les clusters de haute disponibilité AD-SAL et MD-SAL, qui sont fourni par ODL. Ces simulations nous ont permis d'étudier la performance d'OF avec les configurations des deux clusters dans notre prototype. Les implémentations sont validées avec le déploiement de trois contrôleurs ODL, qui fournissent simultanément une haute disponibilité, une fiabilité et une évolutivité (Fig. 6.20). Si l'un des contrôleurs ODL n'est pas disponible

sur le cluster, les ressources sont redirigées vers un autre contrôleur et la charge de travail est distribuée aux autres noeuds du cluster qui peuvent prendre le contrôle sur le réseau. De cette façon, le cluster peut fonctionner normalement sans interruption.

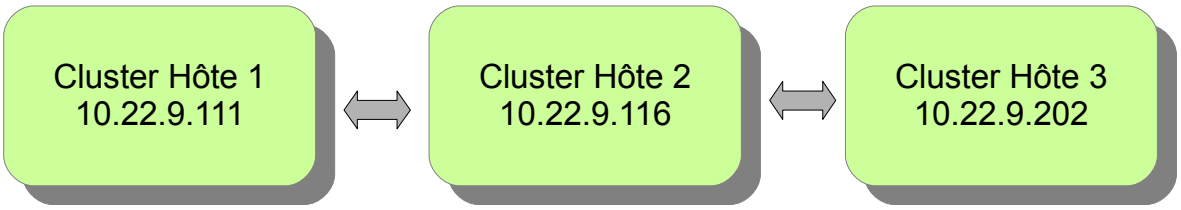

Figure 6.20: Cluster prototype architecture

Sécurité SDN

L'Internet s'est considérablement étendu au cours des dernières décennies et continue d'être développé en termes de dimension et de complexité. De nombreux problèmes de sécurité ont été soulevés tels que l'authentification et la traçabilité des utilisateurs. Au cours du temps, la sécurité est une des fonctions de réseau les plus importantes destinée à protéger le réseau et à prévenir les attaques telles que l'attaque de l'homme du milieu , l'accès non autorisé, l'usurpation et le déni de service entre autres. La sécurité du réseau peut être améliorée à l'aide des fonctionnalités du type SDN. Par exemple, la programmation qui contient la connaissance et les informations globales sur l'état du réseau. L'architecture SDN offre de nombreuses opportunités pour suivre la croissance exponentielle d'Internet. Cependant, la plupart des technologies SDN sont tout à fait nouvelles et présentent de nombreux problèmes de sécurité pour optimiser le déploiement de SDN. Ce chapitre fourni un aperçu de certains problèmes de sécurité sur le SDN et présente notre approche pour la distribution des applications SDN.

Le protocole OpenFlow est le plus utilisé et largement déployé sur l'interface sud du SDN. Il permet la communication entre les périphériques réseau et les contrôleurs SDN via un canal de communication. Néanmoins, OF n'applique pas de mesure de sécurité dans ce canal de communication par lui-même lors du démarrage. Les techniques de cryptage avec TLS sont possibles mais non obligatoire, car il s'exécute directement sur des connexions TCP en clair. En outre, certains fournisseurs n'ont pas encore mis en place de solution pour sécuriser le canal de communication des échanges de messages entre un contrôleur OF et un commutateur OF. Un pirate peut exploiter ce manque d'authentification sur un canal OF non sécurisé pour infiltrer le réseau et écouter ou falsifier le trafic réseau sur l'interface sud du SDN. Ce manque de sécurisation est particulièrement dangereux si l'attaquant obtient un accès de contrôle complet. Il peut alors arrêter les commutateurs OF, modifier les règles dans le commutateur, capturer de trafic sensible et surveiller comment le contrôleur gère les paquets OF. Dans le cadre de notre prototype, le canal de communication OpenFlow de l'OVS et du contrôleur ODL est crypté en utilisant la connexion SSL/TLS en fonction du modèle PKI (*Public Key Infrastructure*).

Avec le protocole OpenFlow 6.5), chaque entrée de flux a une correspondance/action qui peut être configurée contre quatorze champs d'en-tête de correspondance qui fournissent un niveau de granularité plus élevé que celles des techniques d'acheminement traditionnelles. Cependant,

le protocole Openflow envoie uniquement les informations (flux) qu'il contient au contrôleur. Dans le cas de la surveillance des périphériques réseau et des informations échangées, nous voudrions également contrôler non seulement les paquets IP, TCP ou UDP mais aussi les paquets au niveau des applications comme HTTP ou FTP par exemple. Openflow ne fournit pas le mécanisme approprié pour gérer la sécurité d'un réseau au niveau de l'application. En outre, une fois qu'une entrée de flux est installée sur le OVS, les requêtes OF n'atteignent pas le contrôleur. Tous les paquets seront traités uniquement au niveau de l'équipement de réseau, couche 2-3. Dans un récent travail un protocole alternatif à OpenFlow, OpFlex [33] a été proposé par Cisco. Ce paradigme SDN offre de nouvelles perspectives pour surmonter le défi de la gestion des paquets au niveau de l'application. Une autre approche pour gérer le réseau au niveau d'application est d'utiliser l'API *Group Based Policy* (GBP) RESTCONF *Northbound* fournie par ODL. Ces techniques seront incluses dans les travaux futurs.

L'architecture centralisée de SDN implique une dépendance sur le plan de contrôle. Cette centralisation expose le contrôleur à des attaques potentielles comme le déni de service (DoS). Cela peut être atténué en utilisant une architecture de haute disponibilité *High Availability* (HA) avec plusieurs contrôleurs. Les contrôleurs SDN redondants garantissent la disponibilité du réseau, évitant le basculement du réseau comme ce serait le cas avec un seul contrôleur qui deviendrait indisponible. Cependant, sans une conception prudente de la politique de sécurité, tous les membres du cluster peuvent être exposés aux attaques DoS. Il existe une autre faiblesse sur de nombreux contrôleurs SDN qui se trouve sur l'API HTTP à l'interface nord, ces derniers n'ayant aucune méthode de cryptage sur les communications et l'authentification. Nous avons expérimenté un cluster de haute disponibilité pour éviter la présence d'un noeud critique et une défaillance du système. Pour sécuriser les API dans l'interface nord il est nécessaire d'installer et configurer la fonctionnalité *Authentication, Authorization and Accounting* (AAA) fourni par ODL.

Dans le cas de notre prototype, nous nous sommes inspiré du concept de Contrôleur d'accès au réseau et des techniques de sécurité existante [22], pour concevoir une architecture sécurisée exploitant le SDN et étendue à l'IoT. Pour expliquer notre architecture, nous présentons d'abord une solution simple dans laquelle un contrôleur gère la sécurité d'un domaine du SDN. Deuxièmement, nous appliquons cette solution en incluant plusieurs contrôleurs liés aux ressources disponibles sur chaque plate-forme de contrôle. Ainsi, nous élargissons l'architecture de contrôle distribuée par l'interconnexion à tous les domaines du SDN par les contrôleurs de périphérie, ce qui nous rapproche d'un modèle sécurisé pour l'IoT.

En raison de l'absence d'infrastructure du réseau, l'architecture traditionnelle ad-hoc ne prévoit pas de contrôle d'accès au réseau ou à la surveillance du trafic global. L'architecture proposée dans cet dissertation surmonte ces limites de sécurité et permet la configuration du réseau et le déploiement de politiques de sécurité.

Notre but est de réaliser la synchronisation maximum de contrôleurs du SDN dans un périmètre de sécurité permettant un contrôle à grain fin sur l'accès au réseau et la surveillance continue des points finaux.

Afin de garantir l'accès au réseau et aux ressources de réseau, les contrôleurs du SDN:

1. Commencent par authentifier les périphériques réseau;

2. Une fois la connexion sécurisée par OpenFlow, la communication entre le commutateur et le contrôleur sera établie;

3. *Le contrôleur* bloque les ports de commutation connectés directement aux utilisateurs;
4. *Le contrôleur* autorise seulement la circulation d'utilisateurs authentifiés;
5. Suivant le niveau d'autorisation de l'utilisateur;
   (a) *Le contrôleur* va transmettre les entrées appropriées de flux d'accès au commutateur pour le logiciel ou le matériel informatique;
   (b) Dans IoT, nous étendons ce processus d'authentification aux périphériques;
   (c) Chaque dispositif doit s'associer à un noeud OpenFlow actif, lui-même relié à un contrôleur dans son domaine.

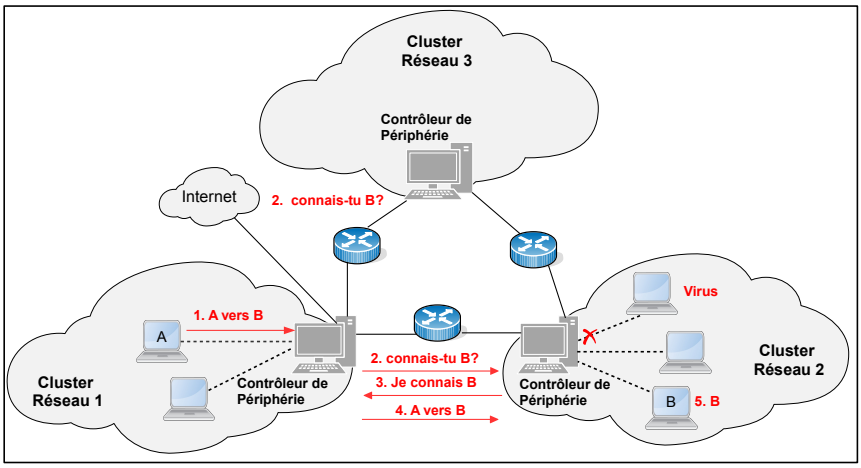

Figure 6.21: Grille de Sécurité SDN Domaine.

Le concept de grille de sécurité du réseau [126] est d'étendre la notion de domaine du SDN à plusieurs domaines, et que les contrôleurs de différents domaine échangent leurs règles de sécurité. Certains contrôleurs du SDN se comportent comme des agents de sécurité au bord du domaine du SDN étendu pour assurer la sécurité du réseau. Les connexions entre les domaines de sécurité seront gérées uniquement par les contrôleurs du SDN. Seul le trafic certifié sera accepté. Chaque contrôleur ne connait que les politiques de son propre domaine. Donc, quand un noeud veut communiquer avec un noeud d'un autre domaine, le flux doit être transmit vers le contrôleur de sécurité, également appelé contrôleur de périphérie. Le contrôleur de sécurité demande à chaque contrôleur voisin s'il connaît la destination de l'information et le flux n'est transmis que si la destination est connue (Fig. 6.21).

Chapter 7

Conclusions and Future work

Over the last few years, network virtualization has become an interesting topic for the industry and research community. Traditionally, network interconnecting devices use proprietary closed systems which constrain the innovation and flexible configuration management. Network communications in the modern day are challenging issues due to increasing number of embedded devices with seamless demand of connectivity. A promising technology such as SDN can be addressed to simplify the programmability and scalability of future heterogeneous network challenges, although, some technical aspects of SDN are still under development and need to be further investigated. We conclude this dissertation by summarizing our main contributions and identifying new directions for future research in the area of SDN, including new protocols and security approaches.

7.1 Outlook of the Work

In this thesis, we were interested in the evaluation and implementation of the new networking paradigm, the SDN, addressed to support the emerging heterogeneous networks architectures. This heterogeneity consists of different network components manufactured by various vendors. The diversity of configuration across a vary protocols vendors and proprietary hardware/software increase the complexity of network management. The proposed design is an efficient alternative to manage the exponential growth of interconnected devices. Separating the control functions from the data forwarding of devices, allows a flexible programmability centralized via the SDN controller. The centralized control mechanism simplifies the network configuration without having to configure each network device individually. In an SDN architecture, the southbound APIs facilitate the communication between the network devices and the SDN controller. OpenFlow is the most commonly known southbound protocol, standardized by the Open Networking Foundation (ONF). On the northbound side, currently no standardized API exists. This interface allows communication between the controller and applications or higher-level control programs. Traditional networks use firewalls or IPS to control the network behavior. By installing these applications on the SDN controller, it can monitor all requests received via northbound interfaces.

We believe that SDN is a promising technology that will be addressed to the next generation of network architecture. The network is becoming more complex to manage with the increasing number of devices with heterogeneous connectivity. The ad-hoc network is one of

most popular topics studied by the research community in the last decade. The devices on an ad-hoc network use peer-to-peer communication to route packets from the source to destination among themselves without a fixed infrastructure. The network intelligence is placed into each device. Due to the decentralized nature of ad-hoc devices, the emerging SDN technology offers new opportunities to configure, control and manage this kind of network in a more efficient manner. We have proposed an SDN-based architecture for ad-hoc networks in which each ad-hoc node is connected to a virtual switch in the legacy interface, which at the same time may be manipulated by an SDN controller. Establishing the OF role in *OFPCR_ROLE_EQUAL* operation in each ad-hoc node, these devices can synchronize the policy rules with the others through their embedded SDN compatible switch. Taking advantage of the flexible management of SDN, we extend our approach to handle networks with or without infrastructure. This is accomplished by organizing the nodes into different types of nodes in a domain. The nodes with enough resources to support SDN capability are called an OpenFlow Nodes (OF-N) and the nodes without SDN capability such as IoT or embedded systems are called smart nodes (S-N). The S-N are associated with a neighbor node which supports SDN technology. A border controller controls and monitors all communications by setting up security rules, software verification and control access authorization on the domain. Also, this border controller shares the traffic load management with border controllers of the other domains.

The communication between domains is established via a multi-distributed border controller in a clustered SDN architecture. This distributed system achieves high availability, scalability and reliability of the information. By setting up multiple controllers, we avoid the single point of failure or bottleneck weakness in the SDN architecture. We have analyzed the clustering solutions provided by the OpenDayLight project, with the aim of better understanding the behavior of multiple controllers under a cluster deployment. As a starting point for this research, the first model studied is the AD-SAL service abstraction to build an SDN cluster architecture. The AD-SAL model was explained in detail, the implementation as well as system performance. Currently, this model is deprecated in the most recent ODL versions. The second model is the MD-SAL service abstraction provided in the all latest versions of ODL. MD-SAL uses the AKKA clustering tool to bring up cluster functionality. This toolkit provides allocation of the network information in different node replicas within a cluster. Based on this cluster mechanism we have proposed a Software Defined Clustered Sensor Network (SDCSN) to manage WSN. The SDN controller is placed on the cluster head called the (SDNCH) which is in charge of managing all kinds of operations of the sensor nodes. All collected information is processed in the domain by the sensor nodes. Then it is sent to the SDNCH which determines the best routing decision, configuration and maintenance for each node.

We have created a framework that was used to prove the functionality and applicability of our approach in different practical, realistic scenarios. Using this prototype, we demonstrated with simulation results the effectiveness of the proposed SDN-based architecture incorporating the following components:

- **Evaluation of Openflow** : As part of this research study, an important goal is to analyze in-depth the Openflow protocol operations and forwarding actions. The implementation was done using the OF version 1.3. Through this protocol the SDN controller can dictate to the network forwarding devices the process for routing packets. The SDN controller can add, modify or remove flow entries dynamically organized from flow tables using OF

based rules. Via POSTMAN Chrome API, YANG UI or ovs-ofctl add-flow tools, these matching and forwarding rules may be programmed. According to OF specification, all packets should be forwarded to the controller if there is or is not a matching flow entry. We noticed that if a flow rule is installed on the OF Switch, it does not ask the controller for the routing process. The OF switch decides how to process the packets which make management difficult at the forwarding/routing level when a packet inspection is needed. In layer 2 with the *odl-l2switch-switch* feature, the ODL controller learns the MAC addresses and the ports of the connected devices. Then it creates automatically a flow table including all forwarding decisions. To build a layer 3 at packet level operations for scalable networks, the process is more complex and requires many successive operations. To help solve this issue, we have developed an algorithm that effectively programs into the OF devices the routing decisions organized by pipelines. In simulation, we observe also that the OF protocol does not have appropriate traffic classification mechanism at higher layer applications. To achieve this, a promising alternative is to combine OF with OpFLex or GBP tools for best filtering and monitoring of network traffic information.

- **Cluster deployment** : The SDN architecture presents a potential single point of failure risk and a target to potential attack, due to the centralized control functions on a single node. With the centralized control plane, the controller has a global view of the entire network, but in case of controller failure, the entire network becomes unavailable. To overcome this weakness, a distributed architecture is needed, implementing multiple SDN controllers. Inspired by ODL AD-SAL and MD-SAL clustering solutions, we have experimented with a distributed SDN architecture. Our implementation with several clustered nodes enables reducing the latency, increasing high availability, scalability and fault tolerance. We provided in detail through experimental results the deployment and behavior of multiple ODL controllers running over a cluster architecture. In this cluster arragement, all end-point devices are organized in domains. Each domain is managed by an ODL controller that at the same time shares the communication and traffic exchange with other controllers via AKKA clustering mechanism.

- **System Performance** : The developed framework aims to minimize the complex management of heterogeneous networks. This experimental platform provides a virtualized environment which allows creating and deciding the policies delivered to each device. Also, it monitors the network state of controller and devices in order to filter, process and distribute routing decisions among different controller instances. This experiment carries out operation of large-scale and distributed SDN networks performing the OF protocol and ODL controllers with more than 500 devices. The results show a system with high performance, flexibility and scalability that demostrates a better network performance compared to traditional networks. Nevertheless, some of the SDN technologies are still under development to deal with a lack of information about deployment and security threats. We highlighted SDN vulnerabilities and provide some tools that can be integrated to the SDN architecture to deal with these threats.

7.2 Future work

Based on the results obtained in this dissertation, we consider that there are new research directions to be explored. We have designed, implemented and evaluated an SDN-based distributed architecture for heterogeneous networks, but some issues and challenges still remain unsolved in this area of research. The developed system is able to handle the communications between clustered nodes, but future work needs to set up a dynamic routing cluster protocol over SDN clusters. Such an approach will provide a better routing paths for objects/devices deployed on the cluster. In addition, we plan to include in our simulations the functionalities of the Opflex protocol. More simulations are needed to evaluate the performance of the proposed architecture for IoT networks. Another future work could be large scale deployment including a combination of IPv4 and IPv6.

7.3 Conclusion et travaux futurs *Version Française*

Durant ces dernières années, la virtualisation des réseaux est devenue un sujet qui attire l'attention de l'industrie et de la communauté de la recherche. Traditionnellement, les dispositifs d'interconnexion réseau utilisent des systèmes propriétaires fermés ce qui limite l'innovation et la gestion de configuration d'une façon flexible. Au cours du temps, la communication réseau a évolué rendant difficile sa gestion, en raison de l'augmentation du nombre de périphériques intégrés avec une haute demande de connectivité sans interruption de service. Une technologie prometteuse telle que le SDN peut être utilisée pour simplifier le défi de la programmabilité et l'évolutivité des réseaux hétérogènes à l'avenir. Il est cependant important de noter que certains aspects techniques du SDN sont encore en cours de développement et doivent faire l'objet d'une étude plus approfondie. Nous concluons cette thèse en résumant nos principales contributions, ce qui ouvrira par la suite de nouvelles orientations des futures recherches dans le domaine du SDN, en incluant de nouveaux protocoles et approches de sécurité.

7.4 Travaux de thèse

Au cours de ce projet de thèse, nous nous sommes intéressés à l'évaluation et à la mise en oeuvre du nouveau paradigme de réseau, le SDN. Celui-ci est également destiné à soutenir les nouvelles architectures émergentes des réseaux hétérogènes. Cette hétérogénéité consiste en différents composants de réseau fabriqués par divers fournisseurs. La diversité de configuration à travers des protocoles variables et le matériel/logiciel propriétaire augmente la complexité de la gestion du réseau. La conception proposée est une alternative efficace pour gérer la croissance exponentielle des dispositifs interconnectés. En séparant les fonctions de contrôle du transfert de données des équipements de réseaux, une programmabilité flexible est envisageable et centralisée via le contrôleur SDN. Le mécanisme de commande centralisé simplifie la configuration du réseau sans devoir configurer chaque équipement de réseau individuellement. Dans une architecture SDN, les API vers l'interface sud facilitent la communication entre les périphériques réseau avec le contrôleur SDN. OpenFlow est le protocole le plus connu sur l'interface sud, et normalisé par la *Open Networking Fondation* (ONF). Du côté de l'interface nord, il n'existe actuellement aucune API standardisée. Cette interface permet la communication entre le contrôleur et les applications ou des programmes de contrôle de niveau supérieur. Le réseau traditionnel utilise un pare-feu ou IPS pour contrôler le comportement du réseau, en installant ces applications sur le contrôleur SDN, il peut surveiller toutes les demandes reçues via les interfaces nord.

Comme nous l'avons déjà évoqué, le SDN est une technologie prometteuse qui peut être adressée aux nouvelles générations d'architecture réseau. Le réseau devient de plus en plus complexe à gérer avec le nombre croissant de périphériques avec une connectivité hétérogène. Le réseau ad-hoc est l'un des sujets les plus populaires étudiés par la communauté scientifique durant la dernière décennie. Les périphériques sur un réseau ad-hoc utilisent une communication peer-to-peer pour l'acheminement des paquets de la source vers la destination entre eux sans aucune infrastructure fixe. L'intelligence du réseau est placée dans chaque périphérique. En raison de la nature décentralisée des périphériques ad-hoc, la nouvelle technologie SDN offre de nouvelles opportunités pour configurer, contrôler et gérer ce type de réseau de manière

plus efficace. Nous avons proposé une architecture SDN pour les réseaux ad-hoc où chaque noeud ad-hoc est connecté à un commutateur virtuel dans l'interface existante qui, en même temps, peut être manipulé par un contrôleur SDN. Avec l'établissement de l'opération OF rôle *OFPCR_ROLE_EQUAL* dans chaque noeud ad-hoc, ces périphériques peuvent synchroniser les règles définies avec les autres noeuds via leur commutateur compatible SDN intégré. Cela permet de prendre l'avantage de la gestion flexible de SDN, pour étendre notre approche de gestion du réseau avec ou sans infrastructure. Ceci est accompli en organisant les noeuds en différents types dans un domaine. Les noeuds qui ont des ressources suffisantes pour prendre en charge la capacité SDN s'appellent noeuds OpenFlow (OF-N) et les noeuds sans capacité SDN tels que l'IoT ou systèmes embarqués s'appellent des noeuds intelligents (S-N). Les S-N sont associés à un noeud voisin qui peut prendre en charge la technologie SDN. Un contrôleur de périphérie contrôle et surveille toutes les communications en configurant les règles de sécurité, la vérification du logiciel et l'autorisation au contrôle d'accès dans le domaine. De plus, ce contrôleur de périphérie partage la gestion du trafic de chargement avec d'autres domaines de contrôleur de périphérie.

La communication entre domaine est établie via un contrôleur de périphérie multi-distribué dans une architecture SDN en cluster. Ce système distribué offre une grande disponibilité, évolutivité et fiabilité de l'information. En configurant plusieurs contrôleurs, éviter l'échec d'un point unique de défaillance ou la faiblesse des goulots d'étranglement de l'architecture SDN. Nous avons analysé les solutions cluster fournies par le projet OpenDayLight, dans le but de mieux comprendre le comportement de plusieurs contrôleurs sous un déploiement cluster SDN. En tant que point de départ de cette recherche, le premier modèle étudié et configuré est le service de abstraction AD-SAL pour construire une architecture de cluster SDN. Le modèle AD-SAL a été expliqué en détail, la mise en oeuvre ainsi que la performance du système. À l'heure actuelle, ce modèle est obsolète dans les dernières versions de l'ODL. Le deuxième modèle est le service de extraction MD-SAL fournie dans toutes les dernières versions d'ODL. MD-SAL utilise l'outil de clustering AKKA pour mettre en place des fonctionnalités du cluster. Cette boîte à outils fournit l'allocation des informations de réseau dans différentes réplicas de noeud dans un cluster. Basés sur ce mécanisme de grappe de serveurs, nous avons proposé un réseau de capteurs à cluster défini par logiciel (SDCSN) pour gérer le WSN. Le contrôleur SDN est placé sur cluster head appelée (SDNCH), qui est en chargée de gérer toutes sortes d'opérations des noeuds capteurs. Toutes les informations collectées sont traitées dans le domaine par les noeuds capteurs, puis elles seront envoyées au SDNCH qui prend la meilleure décision de routage, la configuration et la maintenance sur chaque noeud.

Nous avons créé notre propre prototype d'expérimentation qui a servi à prouver la fonctionnalité et l'applicabilité de notre approche dans différentes pratiques et scénarios réalistes. En utilisant ce prototype, nous démontrons par simulation, l'efficacité de l'architecture SDN proposée en incorporant les résultats suivants:

- **Évaluation du protocole OpenFlow** : Dans le cadre de cette étude de recherche, un des objectifs principal est d'analyser en profondeur les opérations du protocole Openflow et ses actions de transfert de paquets. La mise en oeuvre a été effectuée en utilisant la version OF 1.3. Grâce à ce protocole, le contrôleur SDN peut dicter aux équipements de réseaux le processus pour acheminer les paquets. Le contrôleur SDN peut ajouter, modifier ou supprimer des entrées de flux organisées dynamiquement à partir de différents

tableaux de flux en utilisant des règles basées sur le protocole OF. En utilisant POSTMAN Chrome API, YANG UI ou des outils de flux add-flow, ces règles de correspondance et de renvoi peuvent être programmées. Selon la spécification OF, tous les paquets doivent être transmis au contrôleur s'il existe ou non une entrée de flux correspondante. Nous avons remarqué que si une règle de flux est installée sur le commutateur OF, elle ne demande pas au contrôleur pour le processus de routage. Le commutateur OF décide comment traiter les paquets, ce qui rend difficile la gestion au niveau de l'acheminement/routage lorsqu'une inspection par paquets est nécessaire. Dans la couche 2 avec la fonction *odl-l2switch-switch*, le contrôleur ODL apprend l'adresse MAC et les ports des périphériques connectés, puis crée automatiquement une table de flux qui comprend toutes les décisions d'acheminement. Pour construire un réseau à la couche niveau 3, les opérations d'acheminement de paquets pour des réseaux évolutifs, le processus est plus complexe et nécessite de nombreuses opérations successives. Pour résoudre ce problème, nous avons développé un algorithme qui programme de façon dynamique et efficace les décisions de routage organisées par pipelines dans les commutateurs OF. Au cours des simulations, nous observons également que le protocole OF n'a pas de mécanisme approprié de classification du trafic dans la couche au niveau applicatif. Pour dépasser ce challenge, une alternative prometteuse est de combiner OF avec les outils OpFLex ou GBP pour obtenir un meilleur filtrage et suivi des informations de trafic du réseau.

- **Cluster déploiement** : L'architecture SDN représente un risque potentiel d'attaque en raison des fonctions de contrôle centralisées sur un seul noeud qui représente un point critique du réseau. Avec le plan de contrôle centralisé, le contrôleur a une vue globale de l'ensemble du réseau, mais en cas d'échec du contrôleur, tout le réseau devient inaccessible. Pour surmonter cette faiblesse, une architecture distribuée est nécessaire pour implémenter plusieurs contrôleurs SDN. Inspiré par la solution de cluster ODL AD-SAL et MD-SAL, nous avons expérimenté une architecture SDN distribuée. Notre implémentation avec plusieurs noeuds en cluster permet de réduire la latence, en augmentant la haute disponibilité, l'évolutivité et la tolérance aux pannes. Nous avons fourni en détail des résultats expérimentaux au déploiement et au comportement de plusieurs contrôleurs ODL exécutés sur une architecture de cluster. Dans ce cluster, tous les périphériques de point final sont organisés dans des domaines. Chaque domaine est géré par un contrôleur ODL qui, en même temps, partage l'échange de communication et de trafic avec d'autres contrôleurs via le mécanisme de cluster AKKA.

- **Performance du Système**: Avec le prototype développé nous avons souhaité minimiser la gestion complexe des réseaux hétérogènes. Cette plateforme d'expérimentation fournit un environnement virtualisé qui permet de créer et de décider les règles d'acheminement sur chaque périphérique. Le but étant également de surveiller l'état du réseau du contrôleur et des périphériques, afin de filtrer, traiter et distribuer les décisions de routage entre différentes instances de contrôleur. Cette expérience nous a permis de créer des réseaux SDN à grande échelle et au même temps distribués. Cela nous permet d'évaluer en pratique les performances du protocole OF et des contrôleurs ODL avec plus de 500 périphériques connectés. Les résultats montrent un système à haute disponibilité, flexibilité et évolutivité qui engendre une meilleure performance du réseau par rapport aux réseaux traditionnels. Néanmoins, certaines des technologies SDN sont encore en cours

de développement ce qui implique un manque d'informations sur le déploiement et les menaces de sécurité. Nous avons décrit certaines de ces vulnérabilités et nous fournissons des outils qui peuvent être intégrés à l'architecture SDN pour faire face à ce type de menaces.

7.5 Travaux futurs

Basés sur des résultats obtenus dans cette dissertation, nous considérons l'existence de nouvelles orientations de recherche à explorer. Nous avons conçu, mis en oeuvre et évalué une architecture distribuée basée sur le SDN pour des réseaux hétérogènes, mais certains problèmes n'ont pu être résolus à ce jour et il reste des défis à relever dans ce domaine de recherche. Néanmoins le système développé est capable de gérer les communications entre les noeuds en cluster, et le travail futur doit configurer un protocole pour distribuer des fonctions de routage dynamique et des règles de sécurité sur le cluster SDN. Cette approche fournira de meilleurs acheminements de routage des objets/dispositifs déployés sur le cluster. En outre, nous prévoyons d'inclure sur notre prototype les fonctionnalités du protocole Opflex. D'autres simulations sont nécessaires pour évaluer les performances de l'architecture proposée pour les réseaux IoT. Un autre travail futur pourrait être un déploiement à grande échelle, qui comprendra une combinaison d'IPv4 et d'IPv6.

List of Abbreviations

AAA	Authentication, Authorization, Accounting
AD-SAL	Application-Driven
AODV	Ad-hoc On-Demand Distance Vector Routing
API	Application Programming Interface
BATMAN	Better Approach to Mobile Adhoc Networking
DSR	Dynamic Source Routing
DSDV	Destination-Sequenced Distance-Vector Routing
EC	Extension Controller
ETSI	European Telecommunications Standards Institute
GBP	Group Based Policy
GUI	Graphical User Interface
LISP	Locator/ID Separation
M2M	Machine to Machine
MD-SAL	Model-Driven Service Abstraction Layer
MANET	Mobile Ad-hoc Network
ODL	OpenDayLight Controller
OF	OpenFlow Protocol
OSGi	Open Services Gateway initiative
OGMs	Originator Messages
OS	Operating System
OVS	Open vSwitch Database
QoS	Quality of Service
RDP	Remote Desktop/Terminal Server
RREQ	Route Requests Messages
RREP	Route Replies Messages
SAL	Service Adaptation Layer
SDNCH	Software-Defined Network Cluster Head
SDSN	Software Defined Clustered Sensor Networks
SDN	Software-Defined Network
SSL	Secure Sockets Layer
TORA	Temporally ordered routing algorithm
TLS	Transport Layer Security
VM	Virtual Machine
VNC	Virtual Network Computing
ZRP	Zone Routing Protocol

Bibliography

[1] Open Networking Fundation "OpenFlow Switch Specification, Version 1.5.0", White Paper, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, Dec. 2014. xv, xvii, 19, 20, 24, 67, 86, 87, 116

[2] M. Dye, R. McDonald, and A. Rufi, *Network Fundamentals, CCNA Exploration Companion Guide*. ACM, 2007. xvii, 12, 13

[3] Open Networking Foundation. "Software-Defined Networking: The New Norm for Networks", White Paper, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, Apr 2012. xvii, 14, 15, 53, 54

[4] Cisco Systems Inc. "OpFlex: An Open Policy Protocol White Paper", White Paper, http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html, Apr. 2012. xvii, 15, 18

[5] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 7–12, Aug. 2013. xvii, 16

[6] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 19–24, ACM, 2012. xvii, 16, 17

[7] F. N. N. Farias, J. J. Salvatti, E. Cerqueira, and A. J. G. AbelÃ©m, "A proposal management of the legacy network environment using openflow control plane.," in *NOMS*, pp. 1143–1150, IEEE, 2012. xvii, 29, 30

[8] S. Kuklinski, "Programmable management framework for evolved sdn," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–8, May 2014. xvii, 30

[9] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar,

"Carving research slices out of your production networks with openflow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 129–130, Jan. 2010. xvii, 31, 32, 39

[10] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: Perspectives and challenges," *Wirel. Netw.*, vol. 20, pp. 2481–2501, Nov. 2014. xvii, 42, 43, 56, 57

[11] D. Evans, "The internet of things: How the next evolution of the internet is changing everything." White Paper, http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, Apr 2011. 2, 6

[12] O. Flauzac, C. Gonzalez, A. Hachani, and F. Nolot, "Sdn based architecture for iot and improvement of the security," in *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, pp. 688–693, March 2015. 4, 8, 18, 63, 74, 76, 111, 115

[13] F. Olivier, G. Carlos, and N. Florent, "New security architecture for iot network," *Procedia Computer Science*, vol. 52, pp. 1028 – 1033, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015). 4, 8, 63, 111

[14] F. Olivier, G. Carlos, and N. Florent, "Sdn based architecture for clustered wsn," in *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 342–347, July 2015. 4, 8, 73

[15] O. Flauzac, C. Gonzalez, and F. Nolot, "Original secure architecture for iot based on sdn," in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, pp. 1–6, July 2015. 4, 8, 63, 111

[16] O. Flauzac, C. Gonzalez, and F. Nolot, "Developing a distributed software defined networking testbed for iot," *Procedia Computer Science*, vol. 83, pp. 680 – 684, 2016. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops. 4, 8, 73, 116

[17] C. Gonzalez, O. Flauzac, F. Nolot, and A. Jara, "A novel distributed sdn-secured architecture for the iot," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 244–249, May 2016. 4, 8, 48, 58, 85

[18] C. Gonzalez, S. M. Charfadine, O. Flauzac, and F. Nolot, "Sdn-based security framework for the iot in distributed grid," in *2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, pp. 1–5, July 2016. 4, 8, 85, 116

[19] O. Flauzac, C. Gonzalez, and F. Nolot, "Software-defined clustered sensor networks approach to secure iot networks," in *In Proceedings of International Conference on Internet of Things, Data and Cloud Computing (ICC'2017,)*, pp. 1–6, June 2016. 4, 8, 85

[20] "Information processing systems – open systems interconnection – basic reference model, iso 7498:1984." Information Processing Systems (ISO). https://www.iso.org/standard/14252.html, sept 1984. 11

[21] Cisco Express Forwarding (CEF). http://www.cisco.com/c/en/us/support/docs/routers/12000-series-routers/47321-ciscoef.html, 2014. 13

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008. 15, 54, 67, 71, 105, 113, 119

[23] G. Jayavardhana, B. Rajkumar, M. Slaven, and P. Marimuthu, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, pp. 1645–1660, Sept. 2013. 15, 54

[24] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 7–12, ACM, 2012. 15, 16, 54, 55

[25] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, ITC '11, pp. 1–7, International Teletraffic Congress, 2011. 16, 55

[26] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2010. 17

[27] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, (New York, NY, USA), pp. 351–362, ACM, 2010. 17

[28] S. Schmid and J. Suomela, "Exploiting locality in distributed sdn control," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 121–126, ACM, 2013. 17

[29] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 1–6, ACM, 2012. 17

[30] G. Shuai, Z. Yujing, L. Hongbin, and Z. Hongke, "Scalable area-based hierarchical control plane for software defined information centric networking," in *Computer Communication and Networks (ICCCN)*, pp. 1–7, Aug 2014. 17, 77

[31] L. Xu, P. Djukic, and Z. Hang, "Zoning for hierarchical network optimization in software defined networks," in *Network Operations and Management Symposium (NOMS)*, pp. 1–8, May 2014. 17, 77

[32] A. Aissioui, A. Ksentini, and A. Gueroui, "An efficient elastic distributed sdn controller for follow-me cloud," in *Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 876–881, Oct 2015. 18

[33] OpFlex Control Protocol, https://tools.ietf.org/html/draft-smith-opflex-00, 2014. 18, 103, 119

[34] The Locator/ID Separation Protocol (LISP), https://tools.ietf.org/html/rfc6830, 2013. 18

[35] Network Configuration Protocol (NETCONF), https://tools.ietf.org/html/rfc6241, 2011. 18

[36] J. Biswas, A. A. Lazar, J. F. Huard, K. Lim, S. Mahjoub, L. F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein, "The ieee p1520 standards initiative for programmable network interfaces," *IEEE Communications Magazine*, vol. 36, pp. 64–70, Oct 1998. 18

[37] Forwarding and Control Element Separation (ForCES) Forwarding Element Model, https://tools.ietf.org/html/rfc5812, 2011. 18

[38] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The softrouter architecture," in *In ACM HOTNETS*, 2004. 18

[39] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008. 26

[40] D. Erickson, "The beacon openflow controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 13–18, ACM, 2013. 26

[41] Z. Cai, *Maestro: Achieving Scalability and Coordination in Centralizaed Network Control Plane*. PhD thesis, Rice University, Houston, TX, USA, 2012. AAI3521292. 26

[42] H. Li, X. Que, Y. Hu, G. Xiangyang, and W. Wendong, "An autonomic management architecture for sdn-based multi-service network," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, pp. 830–835, Dec 2013. 30, 33

[43] O. M. C. Rendon, F. Estrada-Solano, and L. Z. Granville, "A mashup-based approach for virtual sdn management," in *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pp. 143–152, July 2013. 31

[44] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. H. M. K. Costa, and O. Duarte, "Omni: Openflow management infrastructure," in *Network of the Future (NOF), 2011 International Conference on the*, pp. 52–56, Nov 2011. 31

[45] Y. Zhang, X. Gong, Y. Hu, W. Wang, and X. Que, "Sdnmp: Enabling sdn management using traditional nms," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*, pp. 357–362, Jun 2015. 31

[46] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 79–84, ACM, 2012. 31

[47] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, T. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." RFC 7348 (Informational), august 2014. 32

[48] S. Bemby, H. Lu, K. Zadeh, H. Bannazadeh, and A. Leon-Garcia, "Vino: Sdn overlay to allow seamless migration across heterogeneous infrastructure," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 782–785, May 2015. 32

[49] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of ip multicast in overlay networks using openflow," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 91–96, ACM, 2012. 32

[50] A. Basta, A. Blenk, Y.-T. Lai, and W. Kellerer, "Hyperflex: Demonstrating control-plane isolation for virtual software-defined networks," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 1163–1164, May 2015. 32

[51] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, (New York, NY, USA), pp. 7:1–7:6, ACM, 2011. 32

[52] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, (New York, NY, USA), pp. 323–334, ACM, 2012. 32

[53] R. McGeer, "A safe, efficient update protocol for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 61–66, ACM, 2012. 33

[54] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 67–72, ACM, 2012. 33

[55] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford, "Hotswap: Correct and efficient controller upgrades for software-defined networks," in *Proceedings of the Second*

ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, (New York, NY, USA), pp. 133–138, ACM, 2013. 33

[56] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates.," in *HotSDN* (N. Foster and R. Sherwood, eds.), pp. 49–54, ACM, 2013. 33

[57] S. Wang, D. Li, and S. Xia, "The problems and solutions of network update in sdn: A survey," in *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pp. 474–479, May 2015. 33

[58] W. Liu, R. Bobba, S. Mohan, and R. Campbell, "Inter-flow consistency: A novel sdn update abstraction for supporting inter-flow constraints," in *Communications and Network Security (CNS), 2015 IEEE Conference on*, pp. 469–478, Sept 2015. 33

[59] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," in *ACM SIGCOMM*, August 2009. 34

[60] H. Nikhil, S. Seetharaman, M. Flajslik, A. Gember, N. Mckeown, G. Parulkar, A. Akella, N. Feamster, R. Clark, A. Krishnamurthy, V. Brajkovic, and T. Anderson, "Aster*x: Load-balancing web traffic over wide-area networks," in *ACM SIGCOMM*, June 2010. 34

[61] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, (Berkeley, CA, USA), pp. 12–12, USENIX Association, 2011. 34

[62] S. Kaur, J. Singh, K. Kumar, and N. Ghumman, "Round-robin based load balancing in software defined networking," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, pp. 2136–2139, March 2015. 34

[63] C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, and C. YanRen, "A novel design for future on-demand service and security," in *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, pp. 385–388, Nov 2010. 34

[64] J. Suh, H.-g. Choi, W. Yoon, T. You, T. . Kwon, and Y. Choi, "Implementation of Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure," in *1st European NetFPGA Developers Workshop*, Sept. 2010. 34

[65] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10, (Washington, DC, USA), pp. 408–415, IEEE Computer Society, 2010. 35

[66] X. Liu, H. Xue, X. Feng, and Y. Dai, "Design of the multi-level security network switch system which restricts covert channel," in *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 233–237, May 2011. 35

[67] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with openflow/nox architecture," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pp. 7–12, Oct 2011. 35

[68] J. Wu, J. Bi, M. Bagnulo, F. Baker, and C. Vogt, "Source Address Validation Improvement (SAVI) Framework." RFC 7039 (Informational), Oct. 2013. 35

[69] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 121–126, ACM, 2012. 35

[70] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 127–132, ACM, 2012. 35

[71] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building robust firewalls for software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), pp. 97–102, ACM, 2014. 36, 105

[72] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in openflow," in *Communications (ICC), 2013 IEEE International Conference on*, pp. 1974–1979, June 2013. 36, 105

[73] R. Jin and B. Wang, "Malware detection for mobile devices using software-defined networking," in *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pp. 81–88, March 2013. 36, 105

[74] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-defined ids for securing embedded mobile devices," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pp. 1–7, Sept 2013. 36

[75] J.-H. Lam, S.-G. Lee, H.-J. Lee, and Y. Oktian, "Securing distributed sdn with ibc," in *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*, pp. 921–925, July 2015. 36

[76] A. Lara and B. Ramamurthy, "Opensec: Policy-based security using software defined networking," in *Network and Service Management, IEEE Transactions on*, Jan 2016. 36

[77] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 125–126, Jan. 2010. 36

[78] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for introducing innovation into wireless mobile networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '10, (New York, NY, USA), pp. 25–32, ACM, 2010. 37

[79] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, "The stanford openroads deployment," in *Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization*, WINTECH '09, (New York, NY, USA), pp. 59–66, ACM, 2009. 37, 38

[80] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pp. 1–6, July 2011. 37, 66

[81] A. Mahmud and R. Rahmani, "Exploitation of openflow in wireless sensor networks," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, vol. 1, pp. 594–600, Dec 2011. 37

[82] J. Chung, G. Gonzalez, I. Armuelles, T. Robles, R. Alcarria, and A. Morales, "Experiences and challenges in deploying openflow over a real wireless mesh network," in *Communications (LATINCOM), 2012 IEEE Latin-America Conference on*, pp. 1–5, Nov 2012. 37

[83] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmsdn)," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, pp. 89–95, Oct 2013. 37

[84] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise wlans with odin," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 115–120, ACM, 2012. 37

[85] M. Gerola, R. D. Corin, R. R. F. D. Pellegrini, E. Salvadori, H. Woesner, T. Rothe, M. Sune, and L. Bergesio, "Demonstrating inter-testbed network virtualization in ofelia sdn experimental facility," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 39–40, April 2013. 38, 39

[86] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler, and C. Peylo, "Cloudmac x2014; an openflow based architecture for 802.11 mac layer processing in the cloud," in *Globecom Workshops (GC Wkshps), 2012 IEEE*, pp. 186–191, Dec 2012. 38

[87] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 1–6, Oct 2012. 38, 48, 49, 58, 59

[88] C. Chaudet and Y. Haddad, "Wireless software defined networks: Challenges and opportunities," in *Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013 IEEE International Conference on*, pp. 1–5, Oct 2013. 38

[89] F. Pakzad, M. Portmann, and J. Hayward, "Link capacity estimation in wireless software defined networks," in *Telecommunication Networks and Applications Conference (ITNAC), 2015 International*, pp. 208–213,, Nov 2015. 38

[90] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 255–270, Dec. 2002. 38

[91] G. Gibb, D. Underhill, A. Covington, T. Yabe, and N. Mckeown, "Openpipes: Prototyping high-speed networking systems," in *In Proc. ACM SIGCOMM Conference (Demo)*, 2009. 39

[92] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010. 39

[93] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 24–29, Oct 2012. 39

[94] J. Schultz, R. Szczepanski, K. Haensge, M. Maruschke, N. Bayer, and H. Einsiedler, "Opengufi: An extensible graphical user flow interface for an sdn-enabled wireless testbed," *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pp. 770–776, Oct 2015. 39

[95] Carnegie Mellon University, CMU SCS Coke Machine, http://www.cs.cmu.edu/~coke/history_long.txt, 1982. 41

[96] S. Li, L. D. Xu, and S. Zhao, "The internet of things: A survey," *Information Systems Frontiers*, vol. 17, pp. 243–259, Apr. 2015. 41, 42, 56

[97] P. Sethi and S. R. Sarangi, "Internet of things: Architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 243–259, Jan. 2017. 42, 56

[98] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, pp. 2787–2805, Oct. 2010. 42, 44, 56

[99] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 2347–2376, Fourthquarter 2015. 44

[100] H. Huang, J. Zhu, and L. Zhang, "An sdn_based management framework for iot devices," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pp. 175–179, June 2014. 45, 48, 49, 58, 59

[101] Y. Jararweh, M. A. Ayyoub, A. Darabseh, E. Benkhelifa, M. A. Vouk, and A. Rindos, "Sdiot: a software defined based internet of things framework," *J. Ambient Intelligence and Humanized Computing*, vol. 6, no. 4, pp. 453–461, 2015. 45, 48, 49, 58, 59

[102] Q. Zhijing, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *Network Operations and Management Symposium (NOMS)*, pp. 208–216, May 2014. 46, 48, 58, 73

[103] D. Wu, D. Arkhipov, E. Asmare, Z. Qin, and J. McCann, "Ubiflow: Mobility management in urban-scale software defined iot," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, Apr 2015. 46, 48, 49, 58, 59, 73

[104] P. Thubert, M. R. Palattella, and T. Engel, "6tisch centralized scheduling: When sdn meet iot," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 42–47, Oct 2015. 46, 48, 49, 58, 59

[105] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *2012 European Workshop on Software Defined Networking*, pp. 7–12, Oct 2012. 46, 49, 59

[106] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, (New York, NY, USA), pp. 163–174, ACM, 2013. 46, 48, 49, 58, 59

[107] M. H. Kabir, "Novel architecture for sdn-based cellular network," *International Journal of Wireless & Mobile Networks (IJWMN)*, vol. 6, p. 71, December 2014. 47, 48, 49, 58, 59

[108] I. F. Akyildiz, P. Wang, and S.-C. Lin, "Softair: A software defined networking architecture for 5g wireless systems," *Computer Networks*, vol. 85, pp. 1–8, July 2015. 47, 48, 49, 58, 59

[109] H. H. Cho, C. F. Lai, T. K. Shih, and H. C. Chao, "Integration of sdr and sdn for 5g," *IEEE Access*, vol. 2, pp. 1196–1204, 2014. 47, 48, 49, 58, 59

[110] M. Usman, A. A. Gebremariam, U. Raza, and F. Granelli, "A software-defined device-to-device communication architecture for public safety applications in 5g networks," *IEEE Access*, vol. 3, pp. 1649–1654, 2015. 47, 48, 49, 58, 59

[111] T. Luo, H. P. Tan, and T. Q. S. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, pp. 1896–1899, November 2012. 47, 48, 49, 58, 59

[112] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of software-defined sensor networks," in *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pp. 410–413, Dec 2013. 47, 48, 58

[113] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Mobile Ad-hoc and Sensor Networks (MSN)*, June 2014. 48, 49, 58, 59, 74

[114] B. T. de Oliveira, L. B. Gabriel, and C. B. Margi, "Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks," *IEEE Latin America Transactions*, vol. 13, pp. 3690–3696, Nov 2015. 48, 49, 58, 59

[115] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, (New York, NY, USA), pp. 234–244, ACM, 1994. 65

[116] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing." RFC 3561 (Experimental), July 2003. 65, 75

[117] D. B. Johnson, D. A. Maltz, and J. Broch, "Dsr: The dynamic source routing protocol for multihop wireless ad hoc networks." RFC 4728 (Experimental), February 2007. 65

[118] V. Park and S. Corson, "Temporally-ordered routing algorithm (tora)," july 2011. 65

[119] Z. J. Haas, M. R. Pearlman, and P. Samar, "The zone routing protocol (zrp) for ad hoc networks," July 2002. 65

[120] S. Koh, J. Kim, and S. Lee, "A proposal of openflow controller to improve transfer rate in mesh network," in *2017 International Conference on Information Networking (ICOIN)*, pp. 509–511, Jan 2017. 66

[121] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-defined ids for securing embedded mobile devices," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, 2013. 70, 112

[122] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7, Nov. 2013. 70, 113

[123] OpenDaylight, *Network Functions Virtualization (NFV)*. http://www.opendaylight.org/, 2014. 73, 81, 87, 96, 114

[124] P. Diogo, L. Reis, and N. Vasco Lopes, "Internet of things: A system's architecture proposal," in *Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on*, June 2014. 73, 77

[125] P. Martinez and A. Skarmeta, "Empowering the internet of things with software defined networking," in *FP7 European research projec*, June 2014. 73

[126] O. Flauzac, F. Nolot, C. Rabat, and L. A. Steffenel, "Grid of security: A new approach of the network security," in *2009 Third International Conference on Network and System Security*, pp. 67–72, Oct 2009. 76, 115, 120

[127] P. Lin, J. Bi, Z. Chen, Y. Wang, H. Hu, and A. Xu, "We-bridge: West-east bridge for sdn inter-domain network peering," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 111–112, April 2014. 77

[128] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–4, May 2014. 77

[129] Akka, *Akka Documentation*. http://doc.akka.io/docs/akka/current/scala/index.html, 2014. 80, 114

[130] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, Oct. 2007. 80, 114

[131] VMware. vSphere Virtual Machine Administration, https://docs.vmware.com/en/VMware-vSphere/6.0/vsphere-esxi-vcenter-server-601-virtual-machine-admin-guide.pdf. 85

[132] mRemoteNG Documentation, https://github.com/mRemoteNG/mRemoteNG/wiki. 86

[133] QEMU version 2.10.0 User Documentation, https://qemu.weilnetz.de/doc/qemu-doc.html. 86

[134] L. Kasanen, "Into the core a look at tiny core linux." http://distro.ibiblio.org/tinycorelinux/corebook.pdf, 2013. 86

[135] TightVNC for Windows: Installation and Getting Started, https://www.tightvnc.com/doc/win/TightVNC_for_Windows-Installation_and_Getting_Started.pdf. 86

[136] B. A. A. Nunes, M. A. S. Santos, B. T. de Oliveira, C. B. Margi, K. Obraczka, and T. Turletti, "Software-defined-networking-enabled capacity sharing in user-centric networks," *IEEE Communications Magazine*, vol. 52, pp. 28–36, September 2014. 105

[137] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of software-defined sensor networks," in *Mobile Ad-hoc and Sensor Networks (MSN)*, pp. 410–413, Dec 2013. 114

Titre en français : Gestion d'une architecture hétérogène distribuée à l'aide du SDN

Les acteurs majeurs d'Internet ont mis en place dans leurs datacenters de plus en plus de virtualisation pour permettre de faire fonctionner plusieurs systèmes d'exploitation simultanément sur un même serveur physique. Cette technologie a permis de faire des économies énergétiques et financières importantes. Elle utilise également au niveau de la recherche depuis peu de temps, en particulier dans le domaine des réseaux. Traditionnellement, ce sont des équipements physiques tels que des commutateurs ou des routeurs qui se chargent du transfert de l'information, à la suite d'une programmation effectuée par les administrateurs. Dorénavant, ces équipements sont également virtualisés et la décision prise pour l'acheminement de l'information se fait de manière logiciel. Des expérimentations de cette technologie de virtualisation du réseau, aussi appelé Software-Defined Network (SDN), ont été mise en place par Google pour relier ses principaux datacenters [1], au travers le monde.

Dans le cadre de ce projet, nous présentons une nouvelle architecture basée sur les concepts du SDN, pour les réseaux avec ou sans infrastructure. Cette architecture est composée de réseaux filaires, sans fil et ad-hoc. Elle est ensuite proposée pour intégrer des objets communicants dans un domaine du réseau SDN. Différents domaines sont alors interconnectés pour que la gestion du réseau soit distribuée, sans toutefois réduire le niveau de sécurité. Cette étude propose une nouvelle architecture sécurisée et distribuée pour l'IoT (Internet des Objets).

Mots-clés en français : Réseaux ad-hoc, SDN, Openflow, Sécurité informatique, Cluster, Routage

Titre en anglais: Management of a heterogeneous distributed architecture with the SDN

Recently, the giants of the Internet are adopting every day more and more the benefits of virtualization within the data center. This technology reduces significantly power consumption, energy consumption, as well as operational cost. Furthermore, not long ago, this promising solution is studied by the research communities to be extended for network virtualization deployment. At this time, the physical networking devices can be virtualized, providing an intelligent abstraction via virtual network software that makes easy to deploy and manage network resources. The search giant Google has deployed SDN to experiment with the inter-connection between their data center around the world [1].

With the exponential growth of devices connected to the Internet, security network is one of the hardest challenge for network managers. In this context, the new networking paradigm, the Software Defined Networking (SDN), introduces many opportunities and provides the potential to overcome those challenges. In our approach, we first propose a new SDN based architecture for networking with or without infrastructure, that we call an SDN domain. This domain includes wired network, wireless network and Ad-Hoc networks. Next, a second architecture includes sensor networks in an SDN-based network and in a domain. Third, interconnecting multiple domains and we describe how we can enhance the security of each domain and how to distribute the security rules in order not to compromise the security of one domain. Finally, we present a new secure and distributed architecture for ad-hoc networks and IoT (Internet of Things).

Mots-clés en anglais: Ad-hoc networks, SDN, Openflow, Grid of Security, Cluster, Routing

Discipline : INFORMATIQUE

Spécialité : Réseaux et Télécommunication