Security Analysis and Access Control Enforcement through Software Defined Networks

Salaheddine Zerkane

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UBO
Université de Bretagne Occidentale

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Salaheddine ZERKANE

Security Analysis and Access Control Enforcement through Software Defined Networks

Thèse présentée et soutenue à BCOM, site de Rennes, le 5 novembre 2018 à 10h
Unité de recherche : Lab-STICC – CNRS UMR 6285

Rapporteurs avant soutenance :
M. Zonghua ZHANG, Maître de Conférence–HDR, IMT Lille Doual
Mme Hakima CHAOUCHI, Professeur, Télécom Sud Paris

Composition du Jury :

Président : M. Mohamed MOSBAH, Professeur des Universités, Institut Polytechnique de Bordeaux

Examinateurs : M. Zonghua ZHANG, Maître de Conférence–HDR, IMT Lille Doual
Mme Hakima CHAOUCHI, Professeur, Télécom Sud Paris

Dir. de thèse : M. Philippe LE PARC, Professeur des Universités, Université de Bretagne Occidentale

Co-dir. de thèse : M. Frédéric CUPPENS, Professeur-HDR, IMT Atlantique
Co-enc de thèse : M. David ESPES, Maître de Conférence, Université de Bretagne Occidentale

Invité(s)
M. Jean VAREILLE, Maître de Conférence, Université de Bretagne Occidentale
M. Philippe BERTIN, Ingénieur de Recherche, Orange
M. Cao-Thanh PHAN, Ingénieur de Recherche, IRT B<>COM
Mme Nora CUPPENS, Professeur des Universités -HDR, IMT Atlantique

Abstract

The evolution of the internet, the explosion of information services and the emergence of new technologies sound the knell of the conventional network architecture. The reasons are related to its rigidity, complexity, and cost.

Software Defined Networking (SDN) is an emerging paradigm that promises to resolve the limitations of the conventional network architecture. Thanks to its programmability, centralization, federation and externalization, SDN strategy is to make it the backbone of the future internet and an enabler for new information services and technologies

The relation between SDN and security is a hot subject that contributes to reaching these goals. SDN and security have a reciprocal relationship. In this thesis, we study and explore two aspects of this relationship. On the one hand, we study security for SDN by performing a vulnerability analysis of SDN. Such security analysis is a crucial process in identifying SDN security flaws and in measuring their impacts. It is necessary for improving SDN security and for understanding its weaknesses. Thus, the thesis provides a generic classification of SDN vulnerabilities. It relies on the Common Vulnerability Scoring System (CVSS) to quantify their severity. Then, it enhances the severity scores by the Analytic Hierarchy Process (AHP). The thesis integrates SDN specific characteristics into the computation of the severity.

On the other hand, we explore SDN for security. Such an aspect of the relationship between SDN and security focusses on the advantages that SDN brings into security. The thesis explores this research direction by introducing SDN concepts to stateful firewalls. The thesis designs and implements an SDN stateful firewall that transforms the Finite State Machine of network protocols to an SDN Equivalent State Machine. The latter automatizes the firewall behavior in the infrastructure thanks to this dynamic transformation. Besides, the thesis evaluates SDN stateful firewall and NetFilter regarding their performance and their resistance to Syn Flooding attacks.

Furthermore, the thesis uses SDN orchestration for policy enforcement. It proposes a firewall policy framework to express, assess, negotiate and deploy firewall policies in the context of SDN as a Service in the cloud.

Through these research works, we contribute to the study of SDN security, to the improvement of firewalls and the enforcement of firewall policies.

Keywords

Software Defined Networks, Security, Vulnerability Analysis, Firewalls, Programmability, Orchestration

Résumé

L'évolution d'Internet, l'explosion des services d'information et l'émergence de nouvelles technologies sonnent le glas de l'architecture du réseau conventionnel. Les raisons sont liées à sa rigidité, sa complexité et son coût.

Les réseaux programmables (SDN) sont un paradigme émergent qui promet de résoudre les limitations de l'architecture du réseau conventionnel. Grâce à leurs propriétés qui allient programmabilité, centralisation, fédération et externalisation, les réseaux programmables ont vocation à devenir une colonne vertébrale pour le futur Internet, et un catalyseur pour les nouveaux services et technologies de l'information.

La relation entre les réseaux programmables et la sécurité est un sujet brûlant. Dans cette thèse, nous étudions et explorons deux aspects de cette relation. D'une part, nous étudions la sécurité pour les réseaux programmables en effectuant une analyse de leurs vulnérabilités. Une telle analyse de sécurité est un processus crucial pour identifier les failles de sécurité des réseaux programmables et pour mesurer leurs impacts. Elle est nécessaire pour améliorer la sécurité des réseaux programmables et pour comprendre leurs faiblesses. Ainsi, la thèse fournit une classification générique des vulnérabilités des réseaux programmables, en s'appuyant sur le système CVSS (Common Vulnerability Scoring System) pour quantifier leur sévérité. Ensuite, elle améliore les ordres de sévérité par le processus de la hiérarchie analytique (AHP). La thèse intègre les caractéristiques spécifiques des réseaux programmables dans le calcul de la sévérité en utilisant ce processus.

D'autre part, nous explorons l'apport des réseaux programmables à la sécurité. Cet aspect de la relation met l'accent sur les avantages que les réseaux programmables apportent à la sécurité informatique. La thèse explore cette direction de recherche en appliquant les concepts des réseaux programmables aux pare-feu à états. La thèse conçoit et implémente un pare-feu programmable qui transforme la machine à états finis des protocoles réseaux, en une machine à états équivalente pour les réseaux programmables. Ces derniers automatisent le comportement du pare-feu dans l'infrastructure grâce à cette transformation dynamique. En outre, la thèse évalue le pare-feu implémenté avec NetFilter dans les aspects de performances et de résistance aux attaques d'inondation par paquets de synchronisation. De plus, la thèse utilise l'orchestration apportée par les réseaux programmables pour renforcer la politique de sécurité dans le Cloud. Elle propose un Framework pour exprimer, évaluer, négocier et déployer les politiques de pare-feu dans le contexte des réseaux programmables sous forme de service dans le Cloud.

À travers ce travail de recherche, nous contribuons à l'étude de la sécurité des réseaux programmables, à l'amélioration des pare-feu et au renforcement de la politique de sécurité dans le Cloud.

Keywords

Réseaux programmables, SDN, cyber sécurité, pare-feu, orchestration, analyse,vulnérabilité, control d'accés

Acknowledgments

I thank my supervisors and mentors Prof. Philippe Le Parc, Prof. Fréderic Cuppens, and David Espes. During these years, you have taught me how to be a researcher and a teacher. Thank you for your trust and freedom in exploring different research directions. I have enjoyed working with you. I would like to express my gratitude for your contributions to this work including sleepless nights before deadlines (special mention to David), high quality mentoring in all the steps of my thesis, full support of our ideas and your tremendous help in my future projects.

I am also very grateful and thankful to Cao-Tanh Phan for his guidance, patience, and encouragement at all the stages of my thesis. Thank you for all that I have learned from you. Thank you for your trust, your support and your friendship.

I am also thankful to my friends and colleagues from the Network Architecture team in B<>COM, especially to Philippe Bertin, Olivier Choisy, Thomas Ferrandiz, Carol Bonan, Anne Coteaux, Michel Corriou, Tim Legrand, Cindy Martin, Marion Benetiére, Johan Pellay, Farouk Messaoudi and many others.

I would like to thank my adelphes deeply. The friends that are more than your sisters and brothers. All these people around the world that I have met and that have inspired me. your benevolence, your support. Thank you for sharing with me great and worse moments. Thank you for teaching me and learning from me.

I would like to thank also the friends and the people that have impacted my life during these years. Especially, Dr. Jean Vareille. A great intellectual and humanist that has open for me more perspectives in sustainablity, philosophy and so many other subjects. I want to thank him for his support and encouragements. I am thankful also to Abhijeet, you were a true incarnation of humility.

My immense gratitude goes to my parents and brothers. Thank you mother for your unconditional love, great sympathy, huge kindness, your courage and continuous struggles throughout my entire life to make us into humans. You are a true example of a great lovely mother. Thank you father and brothers for understanding my choices, for loving me and for your support.

List of Publications

International Conferences

- Zerkane, S., Espes, D., Le Parc, P., & Cuppens, F. (2016, May). Software defined networking reactive stateful firewall. In IFIP International Information Security and Privacy Conference (pp. 119-132). Springer, Cham.
- Zerkane, S., Espes, D., Le Parc, P., & Cuppens, F. (2016, September). A proactive stateful firewall for software defined networking. In International Conference on Risks and Security of Internet and Systems (pp. 123-138). Springer, Cham.
- Zerkane, S., Espes, D., Le Parc, P., & Cuppens, F. (2016, October). Vulnerability analysis of software defined networking. In International Symposium on Foundations and Practice of Security (pp. 97-116). Springer, Cham.
- Cuppens, N., Zerkane, S., Li, Y., Espes, D., Le Parc, P., & Cuppens, F. (2017, July). Firewall Policies Provisioning Through SDN in the Cloud. In IFIP Annual Conference on Data and Applications Security and Privacy (pp. 293-310). Springer, Cham.

International Journals

- Zerkane, S., Espes, D., Le Parc, P., & Cuppens, F. (Submitted). A Survey Of Relationship Between Software Defined Networks and Security. In the Journal Of Network and Computer Applications.
- Zerkane, S., Espes, D., Le Parc, P., & Cuppens, F. (Submitted). A Head to Head with Firewalls through Software Defined Networks. In Computer Networks: The International Journal of Computer and Telecommunications Networking.

Patent

- Salaheddine ZERKANE, Cao-Tanh PHAN, David ESPES, Philippe Le PARC, Frederic CUPPENS. (2016-10-05). Method for protecting a communications network, associated device, control system and computer program. EP3076615A1.

Contents

List of Figures

List of Tables

Part I

Thesis Background

Chapter 1

Introduction

“ Science is a way of life.
Science is a perspective.
Science is the process that
takes us from confusion to
understanding
in a manner that’s precise,
predictive and reliable - a
transformation,
for those lucky enough to
experience it,
that is empowering and emotional.
”

Brian Greene

Contents

1.1 General Introduction

The conventional network architecture is becoming inadequate for the requirements of new technologies such as Cloud Computing, Internet Of Things, Bring Your Own Device and for the expansion of internet services. These technologies and services need large-scale computing, high resource availability, dynamic infrastructure tailoring, automation, resilience, holistic knowledge and other needs. However, conventional architecture cannot satisfy them due to its rigidity, complexity, costs and lack of customization. Its rigidity is a barrier for network innovation because developers are limited to the capacities of proprietary devices and technologies. Its configuration and maintenance are complex and costly because in this architecture network devices are rigid, they use low-level abstractions and do not collaborate with each other. Besides, network devices in the conventional network architecture are vendor dependent, and their evolution relies on the vendor business model. The vendor may hamper network customization when it contradicts its goals.

Software Defined Networking (SDN) promises to resolve the limitations mentioned above. SDN is the softwarization of networks. This softwarization shapes the design and the development of systems and services from a monolithic architecture to a component-based architecture where multiple technologies and stakeholders can deploy their services and collaborate. SDN breaks the vertical integration of the conventional network architecture. As a result, SDN releases network innovation from the vendor dependence paradigm. SDN provides standardized programmable interfaces to network applications that enable them to reprogram the network infrastructure and customize it according to their needs. Moreover, SDN elevates user network programs from a Command Line Interface(CLI) model to a high abstracted model that federates different applications and users around network policies and holistic network knowledge. SDN reduces network provisioning and maintenance time because it automatizes their tasks. It reduces the manual intervention of network operators and their errors.

SDN and security have a particular relation. On the one hand, SDN inherits the vulnerabilities of the conventional network architecture, and it introduces new vulnerabilities that did not exist before. In this matter, security for SDN is the study of SDN security issues and their resolutions. On the other hand, SDN improves security applications thanks to its advantages. Its programmability automatizes security behavior in the network infrastructure. Its orchestration simplifies the expression and deployment of security policies. Its centralization provides a global knowledge that improves the consistency of security applications. Its programmable interfaces enable the pervasiveness of security policies in all the infrastructure.

In this thesis, we focus on this two aspects of the relationship between SDN and security. Concerning security for SDN, the thesis studies the vulnerabilities of SDN to identify them and to measure their severity. Regarding SDN for security, the thesis integrates SDN characteristics into the design of stateful firewalls to improve the firewall performance and operations.

The thesis performs an SDN vulnerability analysis to tackle the first aspect. SDN vulnerability analysis is a crucial process used to identify SDN security flaws and to measure their impacts. This process is necessary to improve SDN security and understand its weaknesses. It provides a generic classification of SDN vulnerabilities and evaluates their impacts on SDN security. These outcomes enable organizations to know the impacts of their conceptual and implementation choices. They help them to adopt suitable countermeasures against security attacks by making the best security decisions.

The thesis relies on the Common Vulnerability Scoring System (CVSS) to quantify the severity of SDN vulnerabilities. CVSS is based on qualitative and quantitative metrics that compute the severity of security vulnerabilities. Its computation procedures integrate three dimensions

related to the different characteristics of conventional networks: the essential generic features of computer systems, their temporal features, and their environment-related factors.

The thesis extends CVSS with the Analytic Hierarchy Process (AHP) which integrates SDN characteristics into the computation of the severity. AHP is a Decision-making process that enables the measurement of the weights related to the alternatives according to decision criteria. AHP is a multi-criteria decision making procedure. It is used to define and evaluate the importance of decision alternatives in the decision-making process. It decomposes complex problems into many levels of connected subproblems. Then, it evaluates the intensity of each subproblem in the overall set of problems.

Furthermore, the thesis explores the second aspect (SDN for security) to improve conventional firewalls and solve some of their limitations. They are complex and costly regarding provisioning and maintenance. They rely on perimeter security. As a result, they can not protect the network from insider attacks. Also, they do not collaborate because they are technology proprietary devices using different interfaces. This lack of collaboration reduces the consistency of firewall policies. In this respect, SDN can improve conventional firewalls by resolving these issues.

Thus, the thesis introduces SDN characteristics into stateful firewalls. It designs and implements an SDN stateful firewall based on the characteristics of SDN such as externalization, centralization, federation, and programmability. The thesis proposes a formalism that abstracts the concepts of the SDN firewall. It processes Network Finite State machines into their SDN Equivalent FSM (SEFSM). It uses high order logic and set theory to formalize the behaviors of the SDN firewall.

Besides, SDN characteristics bring many advantages to Cloud Computing regarding automation, global knowledge, and dynamicity. The thesis explores the enforcement of security policies in SDN as a service. The thesis proposes an orchestrator to express, assess, negotiate and deploy firewall policies.

1.2 Context

The context of this thesis is SDN vulnerability analysis, SDN firewall design and development, and SDN orchestration in Cloud Computing. We have performed this thesis in the Network Architecture laboratory of the IRT b<>COM according to a three years funding given by the Brittany region. The research of this thesis is also part of the academic research laboratory Lab-STICC. The thesis is a collaboration between Western Brittany University, IMT Atlantique Rennes and IRT b<>COM.

IRT b<>COM is a research and development institute that aims to bridge academic research with industry. IRT b<>COM operates in three principal axes which are Network and Security, E-Health and Hypermedia.

Lab-STIC (Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance) is organized around three areas of research. The first one (MOM Pole) focuses on Microwaves and Materials. The second one (CACS) focuses on communications, architecture, and circuits. The third area (CID) is related to knowledge, information and decision subjects.

Both SDN vulnerability analysis and SDN stateful firewalls research work have been integrated into the European sub-project TANDEM of the project CELTIC+. TANDEM addresses the challenge faced by a new network infrastructure regarding high volatile data traffic of mobile linked up objects. It aims to develop a secure Networking architecture for a Data Center Cloud in Europe. The critical infrastructure working group of the project has included our vulnerability analysis work. The SDN firewall application integrates the secure network architecture of SENDATE as one of its building blocks.

Furthermore, we have collaborated with another Ph.D. thesis to enhance the orchestrator of our firewall. This previous research work is related to the selection and negotiation of security policies. The research work provided a method for measuring the similarity between security policies. Similarly, it used a policy tree as configuration to store and manage security-aware preferences and requirements to negotiate security policies. Our orchestrator enhances this research by proposing a firewall policy expression, selection, negotiation and deployment in the context of SDN as a service.

1.3 Problem Statement

We address three problems in this thesis.

1. The first problem is the identification and quantification of SDN vulnerabilities. One of the most common assumptions taken in the previous SDN literature is that SDN introduces new vulnerabilities due to its architecture and features. However, there is not a clear piece of work that addresses these vulnerabilities. Also, all the existing vulnerability computation frameworks and models are inadequate for SDN because they do not integrate its specific properties.

2. The second problem is related to the enhancement of stateful firewalls with SDN characteristics. Another assumption of the literature is that SDN enhances security applications such as firewalls by its architecture and characteristics. Firewalls suffer from complexity. They are error-prone due to human interventions. They do not protect the network against insider attacks. They lack policy enforcement. The literature provides limited research work in the arena of SDN firewalls. Most of these works address the design of stateless firewalls. However, they do not address the implementation of SDN stateful firewall applications or their evaluation regarding performance and attack resistance. The literature also does not compare these evaluation metrics against conventional firewalls such as NetFilter.

3. The third problem is the lack of a complete orchestration mechanism for firewall policies management in the context of SDN as a service. The literature lacks works that take into account the firewall policy life cycle from the expression phase to the deployment phase as OpenFlow rules. The assumption taken in the literature is that SDN orchestration enables the pervasiveness of policies into SDN infrastructure, their automation, and their enforcement. A complete policy management framework based on SDN that supports policy expression, assessment, negotiation, and deployment is as an application for SDN. To the best of our knowledge, no method in the literature addresses all these steps using SDN.

1.4 Solution and Contribution Overview

In this thesis, we study the security of SDN and propose a solution that analyzes SDN vulnerabilities. Our Solution is the first piece of work to identify and to quantify SDN vulnerabilities. Besides, we design, implement and evaluate an SDN stateful firewall that integrates SDN features such as externalization, centralization programmability and federation to improve the issues of conventional firewalls. We also enhance the orchestration part of the proposed firewall by integrating into the orchestrator a policy management framework. We rely on our research work in this thesis to address the three problems mentioned above. The solutions brought by our thesis lead to the following contributions:

1. **SDN Vulnerability Analysis:** To address problem 1, we propose an approach to assess SDN security vulnerabilities by combining the reversion of security objectives with SDN assets. Then, we quantify the severity of the vulnerabilities using CVSS. CVSS computes the severity based on mathematical metrics that address the intrinsic, temporal and environment characteristics of vulnerabilities. However, we find that there are significant factors, specific to SDN, which are not covered by CVSS. These factors affect the security of SDN and enlarge its vulnerability surface. To cover these factors, we introduce into the severity computation AHP weights to determine the importance of each SDN assets regarding its importance for SDN specific features. AHP compounds four steps. In the first step, we determine the importance of each SDN specific feature in relation to the others. In the second step, we determine the importance of each SDN asset to the rest of the assets according to each SDN feature. In the third step, we combine all the results to determine the weight of each SDN asset. In the fourth step, we integrate the weights to CVSS computations to adapt CVSS to SDN.

2. **SDN stateful firewall:** To address problem 2, we propose a formalism using set theory and high order logic to formalize the behavior of our SDN stateful firewall. The proposed SDN firewall integrates the SDN architecture as an SDN application in the application layer and as an orchestrator in the management layer. The firewall application processes the FSM of network protocols according to two behaviors. The first behavior is reactive whereas the second one is proactive. Both behaviors generate a SEFSM that calls firewall and controller functions and installs firewall policies as OpenFlow rules. Besides, we implement our SDN firewall and deploy it in a data center platform. We evaluate its performance and its resistance to DDoS attacks. We also evaluate the performance and resistance of NetFilter in the same evaluation scenarios and configurations. Then, we compare both results to find the conditions in which our solution is better than a conventional solution.

3. **SDN Policy Orchestration:** To address problem 3, we integrate into the orchestrator a policy management framework that supports firewall policies life cycle. The framework relies on set theory and High Order Logic to unify the expression of firewall policies requirements and offers and to assess their relations. It selects the provider that best satisfies the requirements using a ranking algorithm. It provides a negotiation protocol to reach an agreement between the customer and the selected provider. Then, it deploys the agreement into the SDN infrastructure as OpenFlow rules.

1.5 Thesis Organization

This thesis is organized as follows. Chapter 2 covers key background knowledge on SDN. It describes the SDN paradigm, its specific features. It presents its architecture and OpenFlow. It discusses its challenges and benefits. Chapter 3 studies the relationship between SDN and security. It analyzes the aspects of security for SDN by describing SDN attacks and their security solutions. Moreover, it studies SDN for security concerning the improvement that SDN brings to security applications.

The main contributions of this thesis are in chapters 4, 5 and 6. Chapter 4 proposes a vulnerability analysis model for SDN. It presents SDN vulnerabilities. It calculates their severity using CVSS. Then, it adapts the results according to the impacts of SDN features on security using AHP. Chapter 5 describes the details of our SDN stateful firewall. It discusses traditional firewalls and our motivations for SDN firewalls. Then, it introduces the main ideas of our solution and its conceptual foundations. It also describes its implementation and evaluation.

Chapter 6 describes the policy management framework for SDN firewall policies provisioning. It presents the policy expression and the policy assessment formalism. It discusses the selection mechanism and the contract establishment processes. It introduces the policy deployment process. It presents the implementation of the model into the orchestrator and the evaluation of its performance.

Chapter 7 concludes the thesis and outlines future work.

Chapter 2

Software Defined networking

" We are all now connected by the Internet, like neurons in a giant brain. "

Stephen Hawking

Contents

2.1 Introduction

The internet became a conventional architecture for human communication because it leverages human interactions beyond time and space. This technology is largely deployed with a plethora of interconnected devices and services that process, manage and deliver mass information. However, conventional network architecture experiences many issues. It is based on the vertical integration of software and hardware into the same device. The data plane layer, the control layer, and the application layer are vertically integrated together in the same proprietary devices (see Figure 2.1).

As a result, services and middle-boxes such as firewalls are costly in terms of price, complexity and maintenance [1]. They cannot be adapted to the client's need without the intervention of their manufacturers; neither can they collaborate with the network devices of different manufacturers. Many middleware products have been developed to enable them to interact. Unfortunately, the adoption of these solutions has not been a success due to their complexity, their costs and the rapid evolution of services.

The rigidity of the network devices and their heterogeneity are a barrier for network innovation. It confines developers around the complexity of the hardware and its limitations. Furthermore, the administration of the network devices is costly because it is based on technology proprietary products. Administrators and engineers manually configure the plethora. As a consequence, they prompt errors and affect the security of the network.

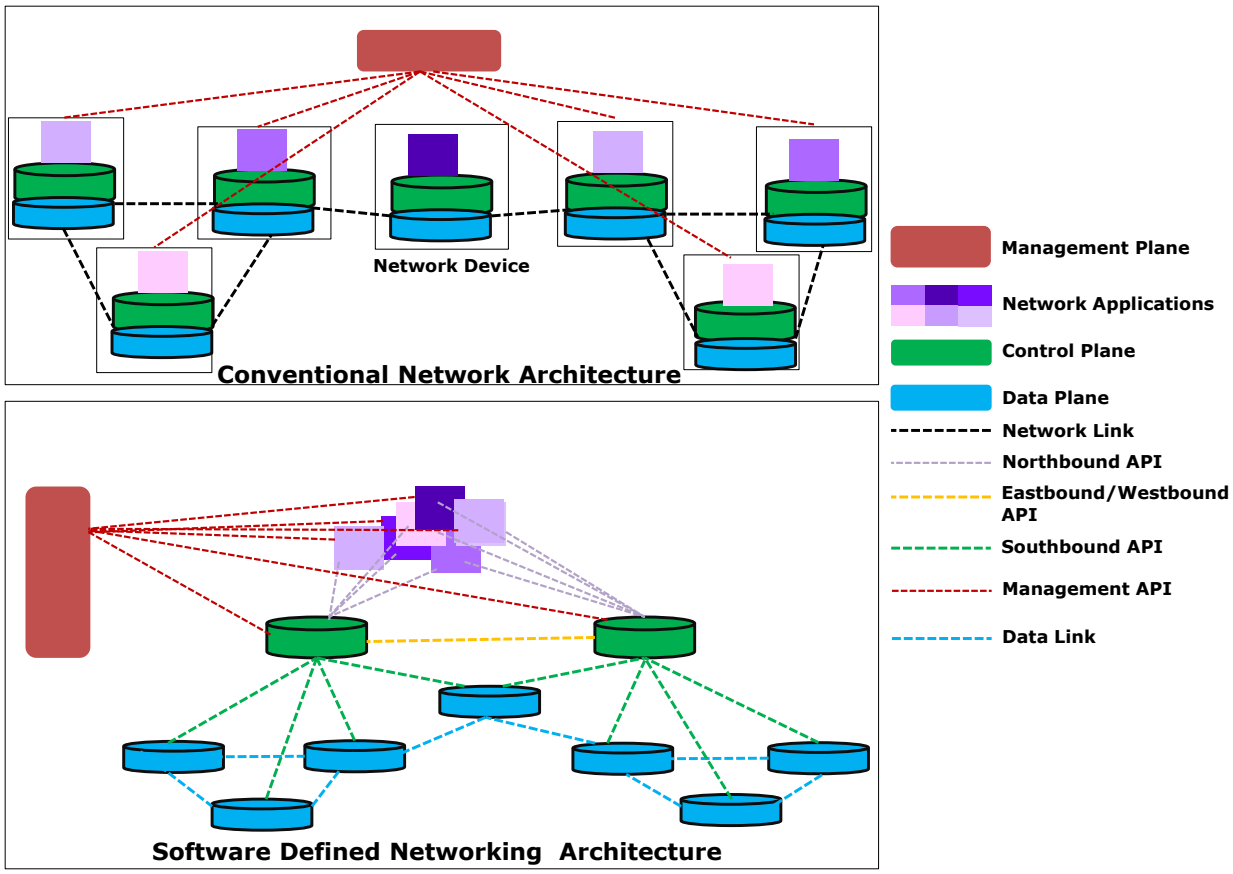


Figure 2.1 – Conventional Network Architecture Vs. SDN Architecture

Software Defined Networking (SDN) brings solutions to the issues mentioned above by breaking the vertical integration of the conventional architecture and standardizing the differ-

ent networking interfaces. As a result network devices become network elements that perform only data plane tasks such as forwarding. SDN separates the three layers by externalizing the control layer and the application layer from the data plane layer (see Figure 2.1). Therefore, an independent software entity performs control decisions. Applications become able to reprogram network elements without being limited by the constraints of hardware complexity.

This chapter introduces SDN through its principles and its specific features. It presents its architecture. It discusses OpenFlow. It also highlights its advantages and challenges.

2.2 SDN Paradigm

Software Defined Networking [2, 3] applies software engineering concepts such as modularity, abstraction, and separation of concerns to make flexible, scalable and efficient networks. Software evolved from a monolithic bloc of functions, data, and interfaces running on dedicated sealed hardware into decoupled and flexible modules that are hardware agnostic. This evolution was possible because of the introduction of operating systems, the standardization of the different interfaces, the separation of data from functions and the abstraction of hardware complexity.

SDN operates with the same idea. It breaks the vertical integration between the control layer and the data layer in networks. It separates the data plane layer from the control layer. It centralizes the control layer logically in a software entity called the controller. Furthermore, it introduces programmable interfaces between the different new layers to offer applications network abstractions while making them able to reprogram the network devices.

SDN is the softwarization of the network. SDN enables network software to change and control the behavior of network infrastructure independently of the underlying hardware. The software can be business applications, management applications, security applications or other network services. The software express their logic in high-level languages and policies without the constraints of hardware details. Then, these high-level rationales are interpreted through open programmable SDN interfaces to hardware specifications. The latter can be forwarding behavior, infrastructure configuration, resource requests, and other low-level specifications. Finally, network elements change their behavior and states according to these specifications.

SDN is founded on four characteristics [3–8]. **Externalization** of control decouples the data plane layer from the control layer. **Centralization** centralizes the control layer logically in an external software entity. **Federation** opens and standardizes SDN interfaces. It guarantees the interoperability between applications, controllers, and network elements. **Programmability** automatizes the decisions of network application and the controller. It enables them to change the states and behaviors of network elements dynamically.

2.2.1 Externalization

SDN separates the control functions from the forwarding behavior by decoupling the data plane layer from the control layer. It breaks the vertical integration between the control software and the network element. Upon stripping control functions from network elements, the latter becomes simple forwarding elements, and all the intelligence and decisions are externalized in the control entity.

Decoupling the control from the data plane makes their deployments and fates independent. Both can evolve and change technologically. They can be provisioned and managed without being limited to the technical constraints of the other. Furthermore, control separation accelerates network innovation because it liberates the control and applications from the limitations of network infrastructure and its technology proprietary restrictions.

2.2.2 Centralization

SDN centralization means that all the control functions are concentrated in a single entity called the controller. The latter is the gravity center of all the network in SDN. It can be physically distributed on many commodity servers, but its control functions behave seamlessly as a single logical entity. The logical centralization simplifies network management and infrastructure abstraction. It allows the convergence of network applications by consistently spreading their logic on network elements.

Besides, centralization builds a global network knowledge. This abstraction represents the state of the network and its information. It is created according to a set of criteria that generalizes the characteristics of the network. It increases the usability of network resources because it simplifies their representation and hides their implementation. Besides, centralization increases network performance by grouping small resources into a logical entity while ensuring their synchronization and cooperation. It also enables the reduction of configuration errors because the centralized controller processes the decisions of all the applications together according to its global knowledge. Centralization offers applications only the information that they need. This specific representation limits the sphere of influence of application in the network [9].

2.2.3 Federation

SDN introduces the concept of open interfaces to enable controllers and network applications to manage and configure the network infrastructure. Their purpose is the convergence of all SDN actors around the same set of protocols, interaction models, and architecture. Besides, they enable different controllers and network applications to collaborate, to share the network state and to unify the methods of expressing their requirements.

Open interfaces offer data models, standardized operations, communication protocols and programmability functions that can be translated into low-level commands. These mechanisms deal with the vertical interactions between applications, controllers, and network elements but also with horizontal interactions in inter-domain networks.

On another note, SDN promotes open source technology with the standardization of open interfaces. It aims to open the control layer technology to set an open universal controller that can be used, enhanced and customized by any user. The main controllers used are open source. This global network operating system will federate all the stakeholders on a unified technology while breaking the lock-in from the control layer technology. Along the same line, SDN also strives to unify and open the infrastructure layer into common technology. Open-Virtual Switch (OVS) [10] is an effort in this way.

2.2.4 Programmability

SDN programmability enables network applications to express their intents concerning policies and configurations without the constraints of low-level infrastructure details. The control layer verifies the consistency of these requirements. Then, it translates them into low-level specifications and sends them to the network elements. Upon installing them into the infrastructure, network elements update their states and change their behaviors according to these intents.

Thanks to programmability, and network applications, the controllers can perform fine-grained control of the network infrastructure. It enables programs in the control layer and in network applications to reprogram the behaviors of network elements on aggregate levels such as MPLS tunnels or protocol layers such as an HTTP connection. For example, an application can specify how network elements will behave with traffic flow. For this purpose, it expresses

a set of policies that specify the criteria to identify the flow. When the controller receives the requirements, it interprets them into low-level specifications. The latter can contain the desired states, traffic information, and appropriate actions. Then the rules are installed into the suitable network elements.

2.3 SDN Architecture

SDN architecture [7, 11–18] compounds four layers (see Figure 2.1). The bottom layer is the Infrastructure Plane (Data Plane Layer). It processes network traffic and executes network forwarding/routing behaviors. The application layer is localized on the top of the architecture. It runs the business logic of network stakeholders. It defines different network services such as load balancing, firewalls, VoIP, and other services. The control layer (controller) is situated between the previous two layers. It provides the application layer with network state and network data. It interacts through Northbound APIs with the application layer and Southbound APIs with the infrastructure layer. Besides, the control layer uses East/West APIs to exchange with other controllers. The management layer is connected to the three layers mentioned above. It manages the configuration, provisioning, and administration of their resources.

2.3.1 Infrastructure layer

The infrastructure layer (Data Plane Layer) consists of a set of interconnected network elements. Data links ensure this interconnection in the infrastructure. Each link connects a port of a network Element to the port of its neighbor. A network element performs computation, storage and forwarding operations on network traffic. Network elements can be virtual or hardware switches, routers, middle-boxes, and other network components. They integrate open programmable interfaces. The control layer uses these interfaces to reprogram their behaviors and to extract their local states. They operate directly on network traffic by performing forwarding behavior, routing tasks, caching, packet inspection and other networking operations.

Open Virtual Switch is a concrete example of an SDN network element. This open source software switch is included in every Linux distribution [19]. It supports many SDN protocols such as OpenFlow. Its modular design eases its integration with vendors switches. This integration guarantees the acceleration of switching and the support of OpenFlow without implementing it in vendors' hardware [20]. Vendors use the provided interface to run OVS and OpenFlow in their Hardware. OVS implements its modules in two spaces [21–23]. In the user space, it integrates OpenFlow in the switch daemon Ovs-vswitchd. In the kernel space, it proposes a kernel module (Datapath Kernel) that abstracts the forwarding behavior based on flows.

2.3.2 Southbound API

The Southbound API (Data Control Planes Interface (D-CPI)) is an open programmable interface between the infrastructure layer and the control layer. It supports the interactions between both layers by offering them all the necessary abstractions and protocols. It has mainly two functions. It provides the control layer with abstractions to communicate with network elements. Besides, it executes the commands of the control layer in network elements.

There are many available D-CPI such as Protocol Oblivious Forwarding (POF) [24, 25], OpenVSwitch Data Base Management Protocol (OVSDB) [26], OpenState [27], Programmable Abstraction of Datapath (PAD) [28], Hardware Abstraction Layer (HAL) [29], LISP [30] and Open Policy Flexible Protocol (OPFlex) [31]. However, OpenFlow [32] is the most widely used D-CPI. This standardized open interface enables the control layer to reprogram the forwarding behavior of the infrastructure layer and to pull its state. Mainly, it uses Flow rules to configure the

Data Plane Layer. Flow rules are a set of matching fields associated with actions and an instruction that describes how the network element should execute these actions. Flow rules are kept in OpenFlow tables inside network elements.

2.3.3 Control layer

The control layer (also called the controller) is considered the brain of the network. It is a logically centralized entity that embeds interfaces, network operations, network state and a development environment to control the behavior of network elements. It abstracts the network infrastructure, processes network states and creates a domain knowledge of the network. Network applications use these abstractions to reprogram the network elements. Moreover, SDN controllers interpret applications policies into low-level rules. They guarantee the consistency and validity of rules installation in network elements. Besides, they monitor the network elements and network traffic by extracting network state, network events, metrics and other parameters.

The control layer can be considered as a Network Operating System that simplifies the development of network applications. It drives open programmable interfaces by interpreting applications requirements into hardware specifications and by deploying them in the network elements. It collects network states also using these interfaces. It guarantees network consistency by dynamically managing rule updates throughout the entire network. It offers a development environment to implement control functions. It constructs a holistic view of the network using the collected infrastructure states. Besides, it manages the virtualization of the infrastructure layer.

Physically, the control layer can be a single entity running on one server or a set of distributed controllers running on many servers. In the latter case, the distributed controllers cooperate to harmonize the control logic and to guarantee network consistency. The distributed control architecture is advantageous regarding scalability and performance because it enables the distribution of the load on many controllers. As a result, it prevents the control layer from becoming a bottleneck. Also, it limits overheads due to frequent network events handled by a single controller. Distributed controllers are more resilient than centralized controllers because they can handle failures while keeping control of the network.

Furthermore, the controller can include Network Programming Languages. Network applications use them to program controller functions. These languages can offer formal validation mechanisms to check the consistency of network functions. FML [33], Nettle [34], Frenetic [35], Procera [36]and Pyretic [37] are examples of SDN languages.

The SDN community has developed many open source controllers. NOX [38] is the first SDN controller that has been developed as an open source control layer. This single-threaded controller is implemented in the C language. Opendaylight [39] is a control layer project that has been developed to satisfy network operators' needs. It is implemented in Java, and it provides many SDN interfaces. It provides development environments that create control functionalities and their interaction patterns. ONOS [40] is an open source NOS. It is a distributed architecture based on clustering the control layer to ensure high performance and scalability. RYU [41] is a software component SDN controller. Its software architecture is multi-threaded. It supports all the versions of OpenFlow.

2.3.4 Eastbound and Westbound APIs

Eastbound and Westbound APIs (Control Control Planes Interface (C-CPI)) enable controllers to communicate within the control layer. Depending on the communication peer, there are two types of Eastbound and Westbound APIs. An Eastbound interface manages communications between an SDN controller and a legacy network. It integrates the mechanisms that interpret

SDN interactions into their none SDN counterparts and vice-versa. A Westbound interface manages communications between only SDN controllers. It ensures the interoperability between different types of SDN controllers. It improves network consistency and inter-domain policy deployment. Depending on the position of controllers, intermediary controllers integrate both types of interfaces, whereas edge controllers integrate one type. The locality of controllers is essential for running the proper Interface with one of the other peers.

Eastbound and Westbound APIs are used to control traffic between multiple domains that are governed by different controllers. This coordination has different aspects. Controllers can be from different vendors. In this case, the interface ensures the interoperability between them. They can have different functions and network applications. Another controller can solicit the functions through the interface. Different administrative authorities can govern the domains. In this case, the interface is used to route the packet to its destination or to calculate the costs of the paths. Finally, the interface can be used to divide a load of a controller on different controllers to ensure suitable scalability.

C-CPI is not standardized yet. For example, SDNi [42] is a Westbound interface for SDN multi-domain communications. It defines multi-control orchestration and coordination interactions across multiple SDN domains. It enables controllers to collaborate to reprogram the behaviors of their network elements which is crucial for inter-domain network operations such as path computing, routing, load balancing and other global network services. SDNi also handles information exchanges like QoS, SLAs, flow setup requests, reachability updates between controllers. It also enables the sharing of domain Knowledge and resources such as QoS, domain topology and other information.

2.3.5 Northbound API

Northbound API (Application Control Planes Interface (A-CPI)) is a programmable interface between controllers and network applications. It enables the latter to interact with the control plane and to deploy its requirements across the network infrastructure [43]. This abstraction enables network applications to profit from the full potential of SDN features such as global knowledge, programmability, automation, and policy deployment. It offers network applications universal data models, network states, management information and other functions. The latter use the interface to reprogram the behavior of network elements and to update the network state.

The interface ensures the interoperability of network applications. Thus, the latter can interact with any controller without relying on its implementation. In the same way, network applications become entirely independent from the infrastructure implementation. It hides all the control functions' complexity while integrating the proper modules that plug network applications with the control layer functions. Besides, it abstracts the details of the infrastructure by hiding the implementation of the forwarding behavior of network elements.

The interface features abstract the functions of the controller to simplify the infrastructure configuration. For instance, network applications express through the northbound interface network policies. The controller uses the policies to perform network computation (for example, routing). The latter includes network constraints such as traffic priorities, QoS, racing conditions, consistency checking and other control operations. Then, through the southbound interface, it interprets the received policies and the results into southbound API rules such as OpenFlow rules.

In contrast to other interfaces, the northbound interface expresses the business logic of network applications intuitively. It offers programming models and data structures to describe business requirements and goals [44]. For instance, these requirements can be services that are needed to run network applications, the model to chain them together, their access to network

resources, the virtualization of the infrastructure, QoS and other high-level policies. The goals deal with QoE (Quality of Experience), application performance and scalability, optimization and monetization of available resources and other objectives.

So far, there is not a standardized northbound API. Discussions and works about the definition and standardization of northbound interfaces are still in progress within the SDN community. In these terms, many SDN controllers such as Opendaylight implement their northbound interface. REST API [45] is used as a northbound interface by many controllers. It enables network applications to express the forwarding behavior of network elements. However, because it is specific, it does not guarantee the interoperability, portability, and re-usability of network applications with other controllers. NetIDE [46, 47] is another Northbound interface. It offers an integrated environment that supports the development and all the life cycle of Network Application in SDN. NetIDE enables developers to perform network application's requirement collection, design, implementation, deployment, testing and debugging. Furthermore, it offers diagnosis tools to analyze the performance of the resulting network. It also handles network state such as topology information and offers a mean to express high-level policies.

2.3.6 Application layer

The application layer resides at the top of the SDN architecture. It compounds network applications. It abstracts the business logic, the requirements, and goals of network stakeholders. This layer generally interacts with third-party applications, administrators, developers and other actors. It transforms these interactions into policies and network strategies. Then, it sends them through the northbound interface to the control layer to configure the network infrastructure or collect its state.

Network applications are computer programs that process network information and produces network services. They offer QoS, Access Control, security, network optimization and other services. They reprogram the behavior of the infrastructure layer through high-level policies. They request the controller's functions and resources such as topology, network events, infrastructure resource capabilities and other state information. They can act as any client requesting services and data from controllers, but also they can provide the control layer with high-level information such as network policies.

The application layer ensures innovation across SDN. For instance, traffic engineering applications such as path computing and load balancing process network paths more efficiently and timely thanks to global network knowledge and forwarding programmability provided by the controller. In the same way, Integrated Network Management System Applications in SDN benefits from such advantages. They provide fault tolerance, network discovery, configuration changes, inter-domain routing and real-time end-to-end flow deployment. Regarding SDN security applications, many solutions were also developed and integrated into the SDN application layer to handle DDoS attacks, authentication, access control, firewalls and Intrusion Detection Systems (IDS).

2.3.7 Management layer

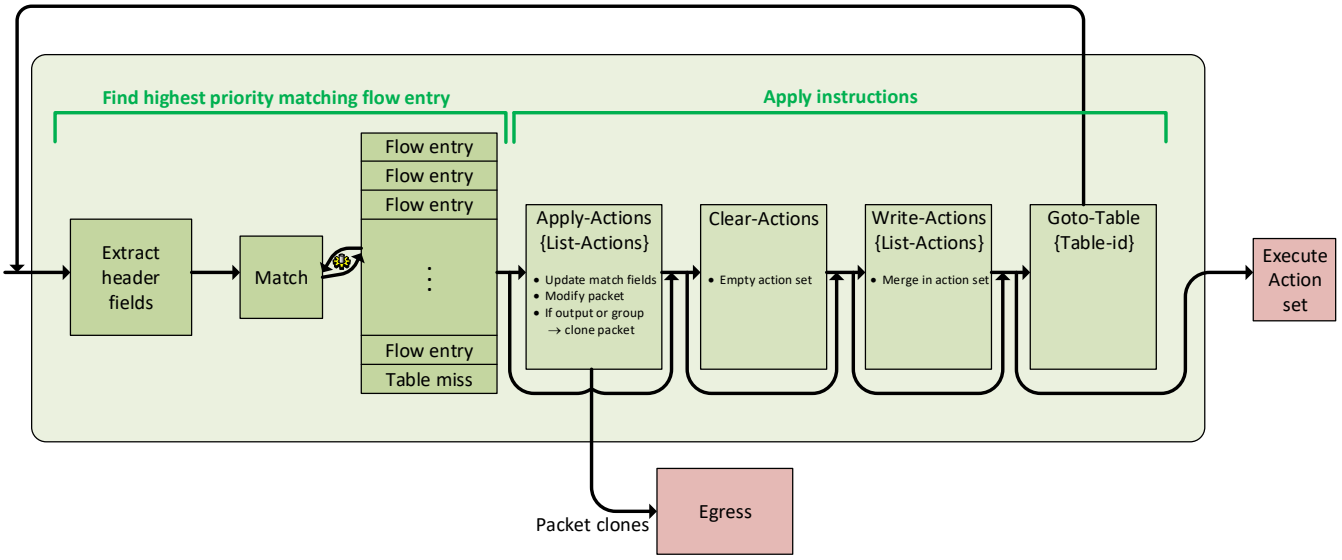
The management layer (Manager) steers the configuration and orchestration of software and hardware resources in the other SDN layers. In addition, it interacts with the administrators to enable them to manage and monitor the SDN layers. The Manager maintains the software and hardware resources of SDN. It deploys them in the proper hardware with the adequate configurations. It adapts them according to the network traffic and in the context of business logic. It controls all their life cycle. It monitors the utilization, the operations, and health of SDN resources. It collects the domain knowledge of each controller. It combines the latter with the monitoring data to construct a global knowledge of the network.

2.4 OpenFlow

OpenFlow defines how network elements execute forwarding functions on network traffic. It also standardizes the interactions between the network elements and the controllers. It consists of two parts [48–50]. The first part is the Datapath. It is localized inside the network elements. It enables them to process, store and forward network traffic. The datapath is a chain of flow tables which connects flow tables to each other to enable flow lookups. A flow table is a collection of flow entries [51]. The second part is the OpenFlow Protocol. It defines the syntax and the semantics of the open flow logic. It includes data structures, communication mechanisms and the operations supported by OpenFlow. It is used to exchange messages, reprogram the infrastructure and pull its state. It compounds two classes; one is localized in the controller, and the other is localized in the network elements.

OpenFlow integrates a processing pipeline in Datapath to handle network traffic (see Figure 2.2). When the network element receives a packet, it sends it to the processing pipeline. The latter parses the packet to extract its header. It finds the highest priority flow entry because OpenFlow processes packets according to a descending order of priority. Then, it compares the extracted header against the matching fields of the primary entry. If it founds a correspondence between both then, the appropriate instructions with the corresponding set of actions are executed. However, if it does not find a match, the processing pipeline moves the matching towards to the next flow entry in the priority order. In case there is not matching at all with all the rules, the processing pipeline executes the table-miss entry. Depending on its configuration, this entry forwards the packet to the controller or drops it.

Figure 2.2 – OpenFlow Processing Pipeline



Flow entries contain mainly a matching set, priority, counters, actions, and timeouts. Match set contains many matching fields. Each field has a type and a value. Types can be L2, L3, MPLS, VLAN characteristics such as Destination_IP, Protocol_Type, VLAN_ID, ... Each value is linked to a wildcard that indicates the masked bits of the value. Values can be partially or wholly masked using wildcards.

Counters perform statistics on packets, flow entries, flow tables, ports, queues, group of flow tables and other network elements components. They enable the controller to monitor the forwarding behavior of network elements. Statistics are part of the network element state. The controller uses them to construct a global knowledge of the infrastructure. For example, counters track the duration of the installation of a flow rule, the number of packets passing through a port, the amount of data matching a flow entry and other counters.

Each flow entry has two ending terms that express their lifetime. Network elements run these timeouts according to the configuration received from the controller. Idle-Timeout causes the eviction of the flow entry when it has not matched a packet for the Idle-Timeout value in seconds. Besides, Hard-Timeout expresses the lifetime of the flow entry.

Instructions define and modify the desired actions on the packet. They encapsulate an Action Set. The latter defines the forwarding behavior and operations that the network element should perform on the matched packet. Forward-To-Port, Forward-To-controller, Drop, Go-To-Table, and Rewrite-Field are examples of supported actions. Instructions can modify the Action Set that will be executed after leaving the flow table. There are six instructions:

- Apply-Actions: actions are immediately applied to the packet. If this instruction contains an output action, a copy of the packet is forwarded to its current state to the destination port.
- Clear-Actions: it deletes all actions of the Action Set.
- Write-Actions: it adds actions to the Action Set. If these actions overlap current actions, they take precedence.
- Write-Metadata: it writes the metadata value into the metadata field of the packet-in.
- Stat-Trigger: it generates an event for the controller if a threshold value for a statistic is reached.
- Goto-Table: it indicates the next table in the processing pipeline.

Table-miss feature reflects the forwarding mode of the network element. There are two modes supported by SDN. In the reactive mode, all the packets that do not match the installed OF rules are forwarded systematically to the control layer. Network element forwards the entire packet to the controller; however, if its buffer is activated, it stores the packet and forwards only the first bytes of the packet. In both cases, the network element encapsulates the traffic by adding an SDN part before sending it to the controller. This encapsulated packet is called a packet-in message. Packet-in adds an OpenFlow header to the first traffic to indicate to the controller that the network element does not know how to handle the packet. Then, when the controller receives the packet-in, it de-encapsulates it. It processes it to decide whether to drop it or to install the corresponding rules in the network element. The second mode is the proactive mode. In this mode, the controller installs in advance all the OpenFlow rules ion the network elements. Network elements drop the unknown traffic without being forwarded to the control layer. The latter receives only the traffic when an OpenFlow rule executes a Forward-To-controller action.

The communication mechanisms between the control layer and network elements are crucial for OpenFlow operations. OpenFlow supports three types of interactions [52]:

- **controller-to-Switch messages** are initiated by the control layer. They enable the controller to add, remove or update OpenFlow rules, to configure flow tables and the switch, to forward network packets (packets-out) from the controller and to initiate the handshaking with network elements.
- **Asynchronous messages** are instigated by network elements. They inform the control layer with messages about network state information, flow notifications and new incoming packets (Packets-in).
- **Symmetric messages** are started by either the control layer or network elements. They are used to maintain the communication channel (Echo messages, Hello messages and

Error messages) or to upgrade switch capabilities with customized OpenFlow's structures and operations (Experimenter feature).

2.5 Expected SDN Benefits

Thanks to its features externalization, centralization, programmability and federation, SDN offers many benefits [53–55]. For instance, SDN leverages the network with simplification, convergence, agility, operation automation, global knowledge, orchestration, efficiency, and openness. These advantages make SDN a key solution to integrate into operational environments. Cloud Computing, data centers, Wide Area Networks (WAN), mobile networks, Network Function Virtualization (NFV) and many others can benefit from SDN as an innovation enabler. Moreover, on the business level, SDN contributes to cost reduction, simplifies network administration and enhances network quality.

2.5.1 Simplicity and convergence

SDN hides the complexity of hardware thanks to federation, programmability, and externalization. Network applications can reprogram the network elements and collect data without worrying about the details of the underlying layers. This simplicity enhances network cost savings, reduces its complexity and improves innovation. Developers need to integrate into their implementation only the proper SDN interface module instead of developing a specific application that runs only on a specific network device.

The controller abstracts the complexity of the infrastructure layer by offering high-level languages and models. This simplicity leads to the convergence of different vendors technologies around the same concepts and models. SDN convergence enhances data transfer, application collaboration and operations upgrades. It enables the exchange of information between the vendor's solutions, the control layer, and network elements. It transforms them into intelligible and unified data models that can be reused in different environments.

2.5.2 Agility

SDN agility adapts the network state dynamically without the intervention of humans. It also improves the reconfiguration cycle of the network according to the knowledge of the controller. The controller and applications react to the new state by updating on-the-fly network elements behavior. Administrators specify and remotely their policies using the high-level abstractions. Then, the controller adapts their policies dynamically with its knowledge to ensure the consistency of the network with its operations.

Also, SDN agility enables the automatic customization of the infrastructure according to the business logic of stakeholders and their requirements. The controller configures the network elements dynamically according to the requirements of the application layer. It delivers to the application layer only the data that corresponds to their business logic while ensuring that the controller deploys their policies on the infrastructure layer appropriately.

2.5.3 Automation

Automation [56] is an outcome of SDN programmability. It enables the manipulation, the deployment and change of network control logic without the manual intervention of humans. Besides, the control plane enables network applications to manipulate the infrastructure layer by programming its behavior. It deploys network policies without needing the intervention of humans. The controller adapts dynamically to network changes by changing network state

and by reprogramming the infrastructure layer when necessary, thanks to the implementation of the proper programming functions that query the different programming interfaces.

The controller handles the installation of the application layer policies in the infrastructure layer. The control layer interacts seamlessly with network elements to deploy these policies effectively. By minimizing human intervention and enhancing applications with the ability to automatize network operations, SDN reduces network instability introduced by operator's errors.

2.5.4 Global knowledge and orchestration

The global knowledge is an outcome of centralization and programmability. The former enables SDN to observe and collect the local states and events happening in the network. Programmability offers the abstractions to consolidates these states and events into a global knowledge. It provides adapted holistic views to each application depending on its requirements.

The global knowledge improves network orchestration. The latter optimizes the behavior of the infrastructure layer and improves its consistency according to the network state. It detects network inconsistencies and corrects them. SDN orchestration manages multi-tenancy requirements. SDN expands its knowledge gathering across multiple sites to collect network information. Then, it uses such knowledge to orchestrate tenant's cross-site performance while ensuring the proper isolation between tenants' resources [57]. Also, SDN orchestration makes SDN a leading light of end-to-end network operations that need holistic information to operate such as routing, optimization, and security.

2.5.5 Network Efficiency

SDN improves network efficiency compared to traditional networks. In the latter, each device runs its control software to process the traffic. The same packet that passes through many networking devices is processed through each one. If we take into consideration the number of networking devices in the legacy architecture and the cost of processing a packet, we find that the legacy architecture wastes a huge amount of resources. The reason is related to the control functions which are repeated in many network devices of the legacy architecture.

SDN resolves this issue by externalizing the control and centralizing it. One controller can manage a set of network elements. Its function can process flow only once. Then, it installs the proper rules to deal with all the packets of the flow without reprocessing them. As a result, it reduces the burden significantly on network resources.

2.5.6 Openness

The adoption of OpenFlow as an open standard and the openness of other SDN technologies are boosting network innovation [58] thanks to SDN federation. Thus, many open source communities emerged and are developing SDN technologies such as controllers, switch software, programmable interfaces and SDN network applications. The leading SDN solutions are open source. Furthermore, SDN openness is decisive in unlocking the technological bolts. For instance, it enables the scientific community to access easily to open source solutions, implement many ideas, experiment them and collaborate with the industry.

2.6 SDN Challenges

Many challenges are impacting the broad spread adoption of SDN [59, 60]. SDN challenges are related to scalability, interoperability, consistency, security and flow limitations. Scalability and security are the primary issues in SDN. The former is related to the ability of SDN to extend its

resources, especially network elements and traffic, without damaging network performance. Security is the intrinsic characteristic of SDN that protects its integrity, availability, and confidentiality while also, not endangering those of other entities with its operations. Both challenges are significant concerns because they can even be counteracting forces. For instance, security affects the performance of SDN and scalability can expose SDN to new vulnerability vectors.

2.6.1 Scalability

SDN centralization affects its scalability and performance. The controller can become a bottleneck [61] because a centralized controller deals with a huge amount of data. OpenFlow particularly introduces this issue. Its reactive mode puts the entire burden on the controller's resources. The latter receives network traffic and reacts by installing new rules. By directing a huge volume of network packets into the controller, network elements jeopardize its performance. This reactivity leads to overload the controller but also, it overflows network elements. Moreover, an attacker can exploit this issue as a vulnerability to make successful DDoS attacks on SDN.

Flow setup in OpenFlow can lead to scalability issues in both network elements and controllers. When a new flow arrives in a network element and does not match any flow rule; the network element forwards it to the control layer. The latter processes the new flow. Then, it generates new flow rules, and it sends them with the flow to the infrastructure layer. Network elements update their tables. Then, they apply the new rules on the received flows. The update depends on network element's and controller's capabilities, and on the performance of their software. If one of these factors cannot scale; then, the process will lead to performance issues.

Controller's placement is also another issue in SDN [62]. Increasing the number of controllers improves reliability because a controller can handover its state to other controllers. In case it fails, the other controllers will take in charge the operations. However, the number of controllers needs to be determined without affecting SDN scalability. An inadequate number of controllers can increase network latency. The controller placement also raises other issues. How their placement affects their interactions and their security?

2.6.2 Interoperability

Controller-controller interoperability is still an open challenge. As we have aforementioned, there is not a standard C-CPI. This interoperability needs to specify the data structures that controllers can share, the functions that they can handover, resources reachability and other features. Furthermore, it has to put in place some collaboration mechanisms that enable controllers to synchronize their states, to provide support and resources to each other and to build common universal knowledge. C-CPI standardization must also take into consideration the security concerns that it introduces. How to protect the confidentiality of one resource from other controllers? How to prevent the attacker from reaching other controllers or third-party entities via the Westbound and Eastbound APIs, if he attacks the controller? How to identify each controller?

We find the same issue in the northbound API because A-CPI has not been yet standardized. The northbound API must grant the interoperability vertically and horizontally. The vertical interoperability is between the application layer and the control Layer. The horizontal Interoperability is between SDN applications. The interoperability must formalize all the collaboration mechanisms and data structures for both cases.

2.6.3 Consistency

Network consistency is also a challenging objective in SDN. The controllers reprogram the behavior of network elements by changing their configuration or by modifying their rules. Then, the network elements confirm the execution of the new updates to their controllers by notifying them. In this process, each controller relies completely on the confirmation of its network elements. The latter can lead the control layer to inconsistent states if they notify them without really executing their updates. For instance, OpenFlow neither specify any mechanism to validate the controller rules nor specify the rest of SDN interfaces. All the controllers must share the same global view to guarantee network consistency [63].

Flow racing and conflicts are also among SDN consistency issues. When many applications query the same controller to install their rules, how to determine which rules the controller must install first? How can the controller be sure that the network elements have installed them? The issue of racing is within the control layer network elements. The latter can delay some rules while running first the simplest one; especially, when they receive messages that enclose many chunks of rules. Conflicts between rules can occur when many applications have contradictory goals. For instance, one application installs its rules, and another one uninstalls these rules. Contradictions can also be in some parts of the rules. For example, a rule that forwards traffic and another one that drops the same traffic.

2.6.4 Security

Concerning security, the controller is the most attractive SDN entity for attackers [64]. The reason is related to the role of the control layer in SDN. The controller is the brain of the network [65]. If an attacker takes hold of the control layer; he will seize all the rest of the network. He can corrupt the behavior of network elements. He can modify network traffic. He will be able to expose network applications and even other controllers. An attack can also put down the control layer by targeting its availability. In this case, the entire SDN network availability will be affected.

Due to the sturdy dependability and reachability between SDN layers, an attack on a layer can affect the other layers. Depending on its sophistication and complexity, it can open new breaches towards other layers. Also, it can reduce their performance. An attacker can exploit API's vulnerabilities to access SDN resources in different layers. He can modify network traffic and even SDN functionalities [66]. Also, he can insert itself in the middle of two layers to snoop the communications between SDN entities. Therefore, interfaces need to integrate into their heart security mechanisms such as access control, encryption and integrity checks to protect the network.

Network elements rely entirely on the control layer. The latter specifies their forwarding behavior. In the case of the connectivity between both layers failed, network elements will not be able to handle network traffic because they do not have any intelligence. In this case, the forwarding and routing capabilities of network elements become inoperative, and they induce the network elements to drop the traffic. As a result, the network becomes unreliable [67]. An attacker can exploit this issue to take control of network elements. For example, it initiates a first threat that disconnects a network element from its controller. Then, it connects an impersonated controller to this network element to command it. As a consequence, the attacker reprograms all the connected network elements.

2.6.5 Flow Limitations

Flow operations in SDN present many lacks due to the design of OpenFlow [68]. SDN allows only a unidirectional flow transmission using one open flow rule between a source and a des-

tination. The return transmission needs an additional new flow rule provisioned by the controller. Moreover, SDN does not enable the infrastructure layer to manage the traffic according to the session or the state of the connection, especially, because the infrastructure layer is stateless in SDN. As a result, the controller performs the tasks of creating return flow rules and managing stateful transmissions. In the reactive mode, this situation can lead to scalability and performance issues.

The size of flow tables can affect network quality. The increase in the number of flow rules inside flow tables decreases the performance of the network due to the rise of the lookup time to find a match inside the flow tables. Moreover, installing flow rules with timeouts set at null values accelerates the filling of flow tables. When the latter is full, the network elements reject the installation of new flow rules and their performance reduces.

2.7 Discussion

SDN is a tremendous evolution in networking. It unifies and connects current network ideas. Unlike its ancestors, SDN succeeds its industrial adoption by many key players. Its implementation and deployment in production environments witness about the so-called SDN fever. The other evidence is the rapid rise of the Open Networking Foundation (ONF) [69]. This community compounds the major worldwide networking operators, vendors and system integrator. They work all together to push innovation around SDN.

It describes SDN architecture, its main components, and their characteristics. It discusses the advantages of SDN such as efficiency, automation, simplicity, network optimization, global knowledge, and openness. Finally, it presents SDN open challenges such as interoperability, consistency and flows limitations while focusing more on scalability and security.

Besides, this chapter presented OpenFlow. The latter has some limitations related to its scope, performance, and security. All its abstractions are only focused on network forwarding behavior while the network is rich of much other information needed by controller and network applications [39]. For example, QoS metrics, topology information, link status, hardware information, and other information. Besides OpenFlow functions are limited to substitution of some packet values. It does not propose arithmetic functions on packet header, matching on the application layer, actions on installed flow rules.

SDN is vulnerable to many attacks. The load on the controller and network element links (especially in OpenFlow) can lead to bottlenecks. Moreover, SDN reactive mode makes the controller vulnerable to DDoS attacks because an attacker can flood the control layer with unknown traffic using network elements. Therefore, we need to understand the relation between SDN and security. In this regards, the next chapter expands the comprehension of SDN and its security.

Chapter 3

Security in Software Defined Networking

" If security were all that mattered,
computers would never be turned on,
let alone hooked into a network with
literally millions of potential intruders
"

Dan Farmer

Contents

3.1 Introduction

The relation between SDN and security is bipolar. On the one hand, SDN enables the improvement of network security thanks to its concepts. In fact, SDN centralization provides means for the construction of a holistic knowledge of the network. This knowledge can be used for monitoring, detecting and preventing network attacks. Programmability enables security applications to automatically configure network devices and adapt their behavior to the network state. This reactive behavior can be deployed the nearest possible next to the attack source. Separating the network layers limit the attack surface because the implementation of the three layers is different. By opposition, the dependence and biding between the implementations of the three layers are strong in the conventional architecture.

However, on the other hand, SDN introduces new attack vectors that were not present in the conventional architecture. SDN intensifies the severity of some attacks. For example, an attacker who takes the commands of a controller will be able to access the other SDN components and corrupt them. An adversary can even use an SDN asset as an attack vector or a zombie to assault another one. Moreover, The breakdown of the controller will result in the unavailability of services running in the application layer. An attacker can even take advantage of this situation to replace the legitimate controller with its corrupted controller. This usurpation will give him the commands of the controller.

In this chapter, we first describe computer security concepts. Then, we present security for SDN and SDN for security. In the former, we analyze its aspects. We describe SDN attacks and their solutions. We classify them according to security properties. In the latter, we study the enhancements that SDN provides for security.

3.2 From Network Security to SDN Security

According to the NIST [70], Computer security is:

> "The protection afforded to an automated information system to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications)."

From this definition, we can understand that computer security goals are to protect the integrity, availability, and confidentiality of computer assets.

In the case of computer networks, these assets are all the valuable resources that constitute the structure and operations of the network. Network devices, data links, network packets, computing and storage resources and network services are examples of network assets. When we speak about protection, we need to highlight also the goals and processes that preserve network assets. In this context, network security protects assets from network threats and network attacks. Using the definitions provided by [71] we can define network threats and attacks as the following :

- **Network threats** are potential for violation of network security, which exists when there is a circumstance, capability, action, or event that could breach and cause harm to the security dimensions of network assets.
- **Network attacks** are assaults on network security that derives from a deep threat. They are creative acts and deliberate attempts, methods or techniques to evade the security properties of network assets.

Network security aims to protect and defend network assets to preserve their security dimensions. Security protection includes detection, prevention and mitigation mechanisms against network threats, before the occurrence of network attacks. Security defense includes detection, remediation, and mitigation against the actions of network attackers.

Security dimensions are classified into the following categories based on [72, 73]. These properties (or dimensions) are as follows:

1. **Access control** aims to protect network assets against unauthorized access.

2. **Authentication** ensures the validity of the claimed identities from the ones participating in authorized and legitimate communications.

3. **Non-repudiation** ensures that an entity can not deny its behaviors. It provides a proof of all the actions and events related to an entity to prevent the latter to repudiate them.

4. **Confidentiality** ensures that unauthorized entities can not understand the information assets.

5. **Integrity** ensures that illicit entities have not modified the assets.

6. **Availability** ensures that the access, use, and operations on network assets is not refused to authorized entities.

7. **Privacy** ensures the protection of the information that might be derived from the operations on network assets.

An attacker misappropriates the security dimensions of the network to reach its goals. He exploits the vulnerabilities of the network to perform its attacks [74]. A network vulnerability is a weakness in the network; when it is exploited, it triggers the transition of the network from a normal state to a flawed state [75]. The network vulnerabilities can be weaknesses in the software technology deployed in the network. For example, a portion of a code that can generate buffer overflow under certain conditions. They can also be related to weaknesses in network protocols. For example, in TCP an adversary can exhaust the resources of a destination by sending some synchronization messages (SYN) to this destination. The weakness of TCP is in the reservation of resources capacities for each new synchronization message in the server.

Besides, network policies can also produce new vulnerabilities such as opening access to unauthorized intruders or blocking an authorized source. For example, a network policy that blocks a client in a LAN when the ingress traffic going to the firewall reaches a specific threshold. In this case, an internal adversary can spoof the identity of the victim. Then, he sends a volume of traffic using this fake identity to trigger the policy against the victim. As a result, the latter is blocked because of the masquerade of the attacker. Administrators and users in the network can also introduce network vulnerabilities by misconfiguring their network and its assets. Using old versions of software, plain-text communications, weak authentication mechanisms and other misconfiguration behaviors can be exploited by attackers to perform network attacks.

Network attacks are classified into the following four main categories [76]:

1. **Reconnaissance attacks** use illicit techniques to collect information on the network to discover its state, its vulnerabilities or spy on its traffic and assets. These attacks generally precede the other categories of attacks and are combined with them. They help the adversary to prepare himself and establish a plan for his next move. An attacker who performs reconnaissance abuses the confidentiality and privacy security dimensions.

2. **Access attacks** enable an unauthorized entity to penetrate into the network. These attacks abuse the access control and authentication security dimensions. They exploit vulnerabilities in network policies and network misconfigurations to gain access.

3. **Disruption attacks** aim to exhaust the resources of the network, downgrade its performance and block legitimate access to network assets. They exploit the vulnerabilities in software technologies such as using the weaknesses found in the micro-codes of network applications to generate errors. Besides, they use protocol vulnerabilities to block the access or downgrade network performance. For example, an attacker can provoke amplification attacks on a target by forging DNS requests that require large DNS responses. The amplification factor (the relation between DNS requests size and DNS answers size) is the vulnerability that the attack exploits in DNS. More the amplification factor increases more the attack exhausts the resource of the target [77]. In fact, the DNS response size can be 66 times the size of DNS requests [78, 79]. Thus, the attacker sends this type of DNS requests to many DNS legitimate resolvers that answer with amplified DNS responses towards the target. At a certain threshold, the network traffic entropy of the target drops drastically leading to the exhaustion of its network resources. Disruption attacks are threatening and dangerous [80] because they use sophisticated techniques such as automation, large capacities, hiding behind legal entities (such as zombies) and they are fast to perform. These attacks abuse the availability and integrity of network assets.

4. **Malicious Modifications** corrupt the network assets with viruses, worms, Trojan, malicious traffic and fake data. They enable the attacker to implement backdoors in the network to resist security mechanisms. They also enable him to theft confidential information to gain access or spy on the communications between legitimate users. Besides, they enable him to damage the behavior of network assets and turning them even to attackers or bots that perform DDoS and other attacks. These types of attacks abuse the integrity and the non-repudiation security mechanisms.

The relation between security and SDN has two facets. They are determined according to the contribution of SDN and security to each other. On the one hand, security for SDN integrates network security principals to protect and defend SDN assets. On the other hand, SDN for security integrates SDN benefits (such as simplification, agility, global knowledge, automation, and interoperability) to improve traditional protection and defense mechanisms.

Security for SDN does not stop with the threats that SDN inherits from the legacy architecture. SDN also introduces new threats and empowers the attackers in some situations. The main reasons are the vulnerabilities that its features introduce unlike in the legacy architecture. For example, attacking the controller will put all the network resources under the commands of an attacker. This most threatening attack [81] was not possible in the legacy architecture because the control was distributed and sealed.

SDN for Security rethinks many security mechanisms because SDN improves them by design, unlike the legacy architecture [82]. SDN simplification enhances the expression of security policies while hiding the complexity of the network. SDN interoperability enables many security mechanisms to collaborate and share their knowledge. SDN global knowledge empowers security solutions with the ability to secure the network from end to end while keeping the consistency of their operations. Besides, SDN automation programs security behaviors inside network elements and security agility adapt security solution to the evolution of the network state and attack dynamically.

In the following sections, we show how SDN impacts security for the better and the worse. We highlight the impacts of SDN principles on network security regarding the introduction of vulnerabilities and the attraction of attackers. Then, we introduce the benefits of SDN for

improving network security. We assess existing SDN security solutions, and we discuss their advantages and drawbacks.

3.3 Security for SDN

Security for SDN aims to protect the assets of SDN from attackers. A secured SDN preserves the security dimensions of its assets. It forbids unauthorized access to its components, resources, and services. It ensures the identity of its components in the different layers. These identities must be unique to prevent any impersonation. For example, an adversary who can pretend to be a legitimate controller will be able to destroy the network. Besides, SDN must track and save the actions and events of all its assets. This information is valuable to monitor the network and to construct a global view of the network. For example, a service provider that offers QoS and QoE to a client via SDN must prove that he ensures the SLA to its customer.

Besides, information is valuable for security. SDN must also protect its information assets by keeping them secret to unauthorized entities. For example, OpenFlow may be based on plain-text information exchanges between the controller and the network elements. An adversary can exploit this vulnerability to forge knowledge on the contents of network elements tables and their properties, on the traffic that is forwarded to the controller, on its behavior but also on the network state such as resource information, network events, network policies and other information.

These reconnaissance attacks can push the adversary to prepare for the next move with a DDoS attack. Moreover, on the privacy level, SDN needs to protect information that can be derived from observing its assets. For example, the control port, the type of controller, its behavior, the installed OpenFlow rules, the SLAs and other knowledge are derived information that has to be protected in SDN.

Inserting illegitimate entities or updating illegally the SDN components is a violation of SDN integrity. SDN must prevent an attacker from abusing the integrity of its components, its information, and their resources. An attacker can inject malicious code into the controller to corrupt its behavior. He can corrupt legal OpenFlow rules by placing himself as a man in the middle on the control-data link. With this type of attacks, an attacker can even reach end-users by deviating their communications, changing their QoS properties or blocking them in the network elements.

The communications in SDN must be secured. All its interfaces and their protocols must protect their communications. A vulnerable interface is an open door to attack an SDN component and spread in domino effects the attacks to other components in different layers. For example, an attacker intercepts the messages on the East/West interface between two domain controllers. Then, he corrupts them to gain access to the domains of the two controllers. He will be able to collect information on their states, abuse the integrity of their components and even reaches their end users.

The availability of SDN components is a significant requirement for the services that rely on SDN but also for its security. SDN must protect its resources from exhaustion, its assets from being interrupted. Besides, it must also prevent attackers from transforming SDN components as DDoS vectors behaving like bots or zombies that attack other SDN components or even end users. For example, the reactive mode of OpenFlow enables an attacker to exploit some vulnerabilities of SDN such as flow table oversize, controller overload and southbound API overflow. When an attacker exploits these vulnerabilities, they enable him to turn network elements to high vectors for DDoS attacks against the controller.

It is essential to study the security risks of the SDN architecture and how to resolve them with prevention, correction, mitigation and remediation processes. This aspect of SDN is still immature and open for research. Although some research works propose security mechanisms

to secure SDN, there is not a standardized reference architecture or mechanisms to secure SDN. The lack of a standard is probably due to the scalability issues that these security mechanisms can introduce. Besides, few works tackle in depth the risks assessment, vulnerabilities analysis and attack evaluation in SDN. A consistent comprehension of SDN security risks is needed to secure the architecture.

In this regards, we have assessed the main works on SDN security analysis. The majority of them focuses on OpenFlow. The papers [83–87] provide security assessments of OpenFlow. These SDN studies highlight attacks on OpenFlow especially those affecting availability such as DoS. The study in [88] provides an OpenFlow risk analysis based on attack trees and STRIDE approach [89]. It derives from the latter a list of OpenFlow vulnerabilities. The authors in [90] propose a security assessment methodology for Software Defined Networking Based Mobile Networks (SDN-MN). The mechanism uses attack graphs to define and to generate attack paths while taking into account SDN-MN's specific features. Besides it uses the Analytic Hierarchy Process (AHP) [91] to quantify the influence of SDN-MN dynamic factors on its security regarding attacks costs.

According to [92] programmability and centralization features of SDN are attractive for attackers. They empower attackers with abilities that were not possible in the legacy networks. Besides, they expose the vulnerabilities of SDN. Security threats are magnified within the control plane, because it becomes a single point of failure in an SDN environment and because it relies heavily on automation [93]. An attacker can use the components of the data plane layer or the application layer to exhaust the controller resources. Moreover, a contaminated controller propagates the attack to all the other SDN layers including other controllers. The attacker also can reprogram the network devices and corrupt network traffic. Thus, the combination of both SDN features gives the attacker superpowers to reach its goals. We discuss the security impacts of each SDN feature in the next subsections.

3.3.1 SDN externalization threats

The separation of the control plane from the data plane introduces new threats on the access to the controller, SDN communications, and controller distribution [94]. The externalization of the controller opens the access to the control layer by contrast to the legacy network where the control was sealed and coupled in a network device. An attacker identifies and penetrates to the components of the control layer without passing by the complexity of all the layers such as in the legacy network architecture. In SDN, the attacker can study and attack the control layer without going through the complexity of the data plane while in the legacy architecture his attacks need to take into account the data plane at least. Externalization introduces vulnerabilities that can lead to significant attacks such as corrupting the controller, impersonating it, or even breaking the entire network.

Communications between the application, control and data layers in the legacy architecture are internal between its sealed components using low level (and proprietary) languages. Attacking these links needs physical access to the network device. However, in SDN, thanks to externalization, these links become exposed. They are no longer internal and hidden. They become network channels that are vulnerable to many security attacks. For example, An attacker can steal data on these links by identifying them and remotely accessing them.

The distribution of the controllers enlarges the threat landscape of SDN. While the modularity brought by externalization retains the attacks exploiting externalization vulnerabilities locally for the application and data plane layers, those in the controller can spread to other controllers. For example, let's suppose an attacker who identifies a controller. He studies his behavior and code. Then, he discovers a severe vulnerability in a controller function. Let's suppose that the vulnerability is about an unhandled exception in the code of the controller; when

the controller receives a bogus packet, the controller restarts. The attacker will identify all the controllers that integrate this function, then attack them all.

Externalizing the controller enables the attacker to study the controller to impersonate it. The attacker deploys its malicious controller in the distributed network of controllers to spy on the communications, corrupt SDN operations or execute disruption attacks. The impersonification of the controller is also related to the lack of security mechanisms to ensure trust between controllers [92].

3.3.2 SDN centralization Threats

SDN centralization reduces the resilience of the network and introduces many disruption vulnerabilities in the controller [95]. In fact, centralization makes the controller as a single point of failure [96] because of the strong dependency of SDN components on the controller. As a consequence of the centralization, the attacker sees the controller as an easy target [97] that empowers him with super privileges over the network. A compromised centralized controller will let the genie out of the bottle because an attacker who takes the commands of the controller will be able to take control of all the network.

Although centralization empowers network applications with holistic knowledge, an attacker can harvest such information to forge a solid knowledge of its potential targets [98]. The control information assets are sensitive because they maximize the probability of the attack success and minimize their complexity. On the other hand, an attacker who tampers such information will corrupt the view of the controller and the operations of the SDN components in the other layers. If an attacker destroys such information, he will deplete the SDN services.

Besides, the single controller is a concentration point for communications with the other layers. Its limited resources are continuously used for network decisions because the other layers depend on its decision engine. An attacker who interrupts the centralized controller will cause the breakdown of all the network. An attacker can use the dependency and the concentration to transform the other assets of SDN as attack vectors against the centralized controller.

For example, let assume the following Distributed Denial of Service (DDoS) scenario where the attacker floods the controller with unknown packets:

1. We suppose an SDN with 1 controller and n network elements in reactive mode.
2. The attacker sends unknown traffic to each network element with a flow rate of x B/s.
3. Each network element encapsulates the packets and sends them to the controller with a flow rate of $\mathrm{T}_i(x)$ B/s (where T_i is the encapsulation function in the network element i).
4. The controller receives packets-in with an input flow rate SF which is computed according to the following formula:

$$\mathrm{SF} = \sum_{i=1}^{n} \mathrm{T}_i(x) \textbf{Byte per Second (B/s)} \tag{3.1}$$

5. It processes reactively the packets-in with a load of $\oint(\mathrm{SF})$. (where $\oint$ is the controller processing function that generates OpenFlow rules and packets-out).
6. $\oint(\mathrm{SF})$ generates an output flow SF'_i towards each network element i.
7. Υ_i is the bandwidth utilization between the controller and the network element i. Υ_i is determined as a function of $\mathrm{T}_i(x)$ and SF'_i.
8. We assume $\hbar_i$ is the maximum bandwidth of the communication link between the controller and the network element i.

9. We assume ℓ is the maximum load on the controller.

The attacker will success its attack when one or both of the following conditions are fulfilled:

$$\begin{cases} \text{If } \oint(\text{SF}) > \ell & \text{Controller ressources are overloaded} \\ \text{If } \Upsilon_i > \hbar_i & \text{Controller and network element i link is overfloaded} \end{cases}$$

As a result, the controller performances decrease resulting in a domino effect that reduces performances of the network elements and the SDN applications in the application layer.

3.3.3 SDN federation threats

Unifying the communications between the different layers of SDN introduces new vulnerabilities in SDN. The homogenization of the SDN interfaces reduces the complexity of attacks that target or misuse them; while in the legacy network, their heterogeneity improves their resilience. Thanks to the federation, the attacker targets an open, standardized interface that is shared by many SDN components. It becomes easier for the attacker to study the interface, detect its vulnerabilities and exploit them to perform its attacks successfully. Besides, the federation creates a large attack vector that connects each vulnerability points of the interfaces. This link between all the SDN layers can be used by an attacker as an underground passage to attack SDN components while hiding behind their legitimate peers. For example, let's assume an SDN with an internal attacker that aims to disclose the traffic of a victim to third parties. The following steps take place:

1. The attacker corrupts an SDN application.
2. The malicious application generates a policy that forwards the egress traffic of a victim to all SDN applications in the application layer.
3. It sends the policy to the controller through its northbound API.
4. The controller receives the policy and interpret it into OpenFlow rules.
5. The network elements install the attacker OpenFlow rules.
6. The egress traffic of the victim is disclosed to all SDN applications whatever their northbound API implementations thanks to the controller federation.

Thanks to the interoperability brought by the federation in SDN, the attacker is now able to talk to all the SDN components from any layer. He relies on the interfaces to translate its malicious policies at any SDN layer. Besides, the attacker in SDN can misuse the collaboration between SDN applications and controllers to interrupt, corrupt and steal SDN components. He can even misuse an interface to poison another SDN component. For example, he can inject in the northbound API false topology information for an SDN application.

3.3.4 SDN programmability threats

The softwarization in SDN provides means to program the network with software applications. As a consequence, an attacker can corrupt an SDN application (or inject a malicious application) to reprogram the behavior of network elements. Programmability can add fuel to the flames of the attacker because the adversary can subvert the operations of other legitimate applications in SDN. For example, he programmes the network with conflicting policies that contradict the legal rules. The attacks can become persistent and dynamic because SDN programmability replaces the network management entirely with automation [99]. An attacker

can inject malicious code in a legitimate application to react automatically to network events. The malicious application dynamically reprogram the network elements when it observes the network events. Even if its policies disappears, its behavior will exhibit again when observing the network events. Furthermore, the bugs in applications can affect the entire network because applications can program network elements and manipulate the traffic automatically. This bugs can express a subtle and complex behavior that is not connected to their cause [100].

Moreover, both programmability modes (see Chapter 2) introduce security vulnerabilities in SDN. On the one hand, all the programmability tasks rely on the controller decision engine in the imperative mode. As a result, the controller becomes a single point of failure. An attacker can use this vulnerability to exhaust the resources of the controller with malicious policies through the northbound API. He can also amplify its malicious behavior to all the network elements that are connected to the controller by programming them on-the-fly.

On the other hand, the decision engine is distributed between the controller and the network devices in the declarative mode. An attacker, in this case, can divert the network elements without the controller being aware of it. He can misuse the network elements as zombies to attack the SDN components or to poison the holistic knowledge of the controller. The latter includes all the information of the existing OpenFlow rules in the network elements. According to this knowledge, the controller takes its decisions. For example, let us assume an SDN with a declarative mode that compounds a controller and a network element. The attacker goal is to control all the traffic going through the network device without the control plane knowing about it. As a result, the network view of the controller is poisoned by the malicious behavior of the attacker. The steps of the attack are as follow:

1. An attacker injects a malicious code in the network element.
2. The malicious code creates a new OpenFlow table OFAT*.
3. The malicious code learns the network state by observing the network events. According to this information, it installs in OFAT* new OpenFlow rules that drop any new traffic.
4. It configures the output ports as ingress ports that forward all the traffic to OFAT* when the processing pipeline of the controller finishes dealing with the traffic
5. When new traffic comes, it is processed by the controller pipeline.
6. It is sent to the output port.
7. The network element redirect the traffic to OFAT*.
8. The controller is not informed because the decision is not part of its scope.
9. The malicious code generates an OpenFlow rule to drop the new traffic.
10. The network element drops any packets that belong to the flow.

3.3.5 SDN Attacks and their countermeasures

SDN attacks exploit the new attack vectors that SDN introduces to reach their goals. [92] derives its attack vectors from the vulnerabilities of SDN. [101] constructs their vectors by combining some network attacks with SDN layers and their components. [102] combines both previous works to derive many SDN security attacks. They focus, especially on OpenFlow. They propose an OpenFlow security assessment checklist. For example, [92] proposes seven new threat vectors of SDN that can exploit its vulnerabilities. The proposed vectors are as follows:

1. Forged or faked traffic flows.

2. Attacks on vulnerabilities in switches.
3. Attacks on control plane communications.
4. Attacks using vulnerabilities in controllers.
5. Lack of mechanisms to ensure trust between the controller and management applications.
6. Attacks using vulnerabilities in central stations.
7. Lack of trusted resources for forensics and remediation.

According to the authors only the vectors 3, 4 and 5 are specific to SDN. The other vectors are found in the legacy architecture and are inherited in SDN. Control plane communications (vector 3) refers to the attacks on the communication between the controller and SDN layers (including other controllers). This type of attacks targets the southbound, the northbound and East/West APIs. Attacks on the vulnerabilities of the controller (vector 4) target the controller components. For example, DDoS attacks on the controller exhaust its resources. Finally, the fifth vector enables the attacker to integrate malicious applications to corrupt the network state. For example, an application may update the security policies made by other applications or create conflicts and breaches with its new policies.

SDN reconnaissance attacks & countermeasures

Attackers use SDN reconnaissance attacks to collect information on SDN components. The attacker exploits this information to detect the vulnerabilities of SDN and to assault the system with active attacks. The attackers collect legacy network data and SDN specific information. The former can be port status, QoS information, topology and other information. The attacker gathers SDN specific information from vectors 3, 4 and 5. This information can be the size of the flow table, the flow table contents, the type of the controller, the list of the different SDN interfaces and other SDN data. Based on all these information, the attacker builds an SDN attack knowledge. It uses this knowledge to plan his next move. Reconnaissance attacks can monitor traffic using the vector 3 to eavesdrop the network or to extract its patterns to analyze it. For example, Know Your Enemy (KYE) attack [103] is an SDN specific reconnaissance attack that an attacker uses to collect information on SDN such as network elements flow rules, SDN configuration and deployed defense mechanisms. The attacker sends scanning probes (for example forged traffic) to network elements. He monitors the return traffic of the probes. Then, he infers the flow tables contents in network elements.

The solution in [104] proposes a fingerprinting attack implementation, evaluation, and countermeasure. The attack targets the communication channel between the controller and network elements (vector 3) by a remote attacker. It monitors their interactions and the RTT of packets to know which packets trigger the installation of new OpenFlow rules. Their attack evaluation shows that the accuracy of this fingerprinting attack is around 98%. They propose as a countermeasure a packet delay mechanism. The first packets of installed OpenFlow rules are delayed inside the network elements. Their RTT increases and approximates the RTT of the packets without OpenFlow rules. As a result, the accuracy of the attack is decreased; however, the performance of the network is also impaired.

The solution in [105] monitors the response times of packets to determine whether a network is SDN or not. It uses a scanning method called Header Field Change Scanning (HFCS) to infer the network element behavior 'Forward_to_controller' by observing the response times of its probing packets. They propose as a countermeasure to activate the wildcards of OpenFlow

rules to compress the number of flows managed by each rule. In this case, the response times of the probing packets become steady. As a result, the accuracy of the HFCS reduces.

[106] combines many fingerprinting techniques to detect which type of controller is in an SDN (vectors 3 and 4). It relies on two techniques. The first one is Timing Analysis fingerprinting. This technique infers timeout values and controller processing time. Then it compares the values to the times of different controllers using a reference database to find the controller. The second technique is Packet-Analysis inference. This technique infers from the packet the knowledge to identify the type of the controller. It analyzes LLDP packets contents and intervals because they differ from one controller to another. This information is compared with the controller reference DB to infer the type of the controller. Besides, it analyzes the ARP responses to determine if the distribution version of the OpenDaylight controller is Hydrogen.

SDN access attacks & countermeasures

SDN access attacks abuse the authentication and access control dimensions of SDN to steal or corrupt SDN components. They use techniques such as masquerading, Man-in-the-middle, elevation of privileges, trust and password attacks. The details of these SDN attacks are as follows.

1. **SDN masquerading attacks**: In masquerading attacks, the attacker pretends to be a legal SDN component. He can exploit the vector 6 to gain access. Then, he impersonates the identity of an SDN controller (vector 4), a network element (vector 2) or an SDN application (vector 5) to corrupt the SDN. For example, an attacker targets an SDN controller to impersonate it. The attacker performs the following steps:

   (a) He installs a malicious application inside the server that is running an SDN controller by exploiting the vector 5.

   (b) The malicious application collects identity information of the controller such as control ports, controller IP_Address, network elements IP_Addresses and other control configuration data.

   (c) The malicious application provokes the restart of the controller application.

   (d) When the control application restarts, the malicious application declares itself on the different interfaces as a controller using the legitimate identity.

   (e) The malicious application blocks the accesses to the legitimate controller.

2. **SDN Man-in-the-middle attacks**: They target the SDN communications through the SDN interfaces. The attacker places himself inside the SDN interface between two SDN components while allowing these assets to communicate. He exploits the vector 3 or 5. Then, he listens to or modifies the communications inside the interface. For example, an attacker that targets the northbound interface as a man-in-the-middle will perform the following steps:

   (a) The attacker places his malicious application in the northbound API.

   (b) The malicious application provokes the restarting of a legitimate SDN application.

   (c) The malicious application spoofs the identity of the legitimate application

   (d) The malicious application connects through the northbound API to the legitimate controller using the spoofed identity.

   (e) The malicious application modifies the control information in the legitimate application configuration with its identity to impersonate a legitimate controller.

(f) When the legitimate application starts, it connects to the malicious application as a controller.

(g) The malicious application plays the role of a transparent proxy between the legitimate controller and application to spy their communications

3. **SDN Elevation of privileges attacks**: They enable the attacker to modify its access privileges and its trust boundaries within SDN components. They exploit the lack of trust mechanisms in SDN (vector 5) and the vulnerabilities in the central station (vector 6). These attacks empower the attacker with full control of the SDN. They can open many backdoors for the preparation of the next attacks. For example, an attacker targets OpenFlow to elevate the priorities of its SDN malicious application. He will perform the following steps:

   (a) The attacker gain access to the controller by the administrator station.

   (b) The attacker installs his malicious application in the application layer.

   (c) The attacker modifies the configuration of network policies in the controller.

   (d) He sets the highest priorities for the OpenFlow rules of his malicious application.

   (e) He decreases the priorities of all the OpenFlow rules that will be generated from the policies of SDN security applications.

   (f) He gives access to his application to communicate with all the network elements and other controllers.

   (g) Once the malicious application sends its network policies, the controller translates them into OpenFlow rules with the highest priority over the other OpenFlow rules.

4. **Trust and password attacks**: In trust attacks, an attacker compromises a trusted SDN component. He exploits the vector 5. Then, he misuses its trust boundaries to penetrate the SDN domain. Password attacks break users passwords. Then, they use these passwords to access SDN components. They exploit the vulnerabilities in the administrative realm (vector 6).

Implementing SDN access control mechanisms reduces the SDN attack surface [107] (the SDN vulnerabilities, their severities, and likelihoods); however, the literature lacks works that propose access control solutions for SDN components. Some works [107, 108] discuss in general the introduction of access control and authentication into the design of SDN architecture without formalizing and implementing their ideas. Other works focus on controlling the access of SDN application to protect the controller. [109] proposes PermOF, a fine-grained permission system based on a set of 18 OpenFlow specific authorizations. The solution controls the access of OpenFlow applications in the controller entry of the northbound API. OperationCheckpoint [110] is another SDN access control solution that enables only authorized SDN applications to access controller resources. For each SDN application, the solution defines a permission list to the controller operations. OperationCheckpoint controls the access in the northbound API according to this list. If an application tries to access unauthorized resources, the solution prevents the access and logs the event to profile the application.

SDN disruption attacks & countermeasures

SDN distribution attacks decrease the capacities of SDN components and interrupt them. Their primary goal is to deny the access [111] of legal entities to an SDN component or to interrupt the entire SDN domain. Disruption attacks constitute a pandora's box in the control layer because the controller is a single point of failure [112]. These disruption attacks can have three

aspects in SDN [113] such as dead data, access denial, and flooding. In the first attack, an attacker exploits the vulnerabilities in the source code of SDN components to find which malformed data cause the corruption of an SDN component behavior or its termination. Access denial attacks deny the legitimate access of an SDN component to the resources of other SDN components or the network. The attacker performs these attacks by neutralizing the data of the victim. Besides, these attacks can also manipulate network policies. For example, an attacker that denies the flow processing in a network element will perform the following actions:

1. The attacker places himself as a man-in-the-middle in the OpenFlow API.
2. He listens to the interactions between the controller and network elements.
3. Based on his knowledge, he forges an OpenFlow rule that deletes all the flows in a network element without notifying the controller.
4. He sends the modified OpenFlow rules to the network element.
5. The network elements install the corrupted OpenFlow rule.
6. The network elements empty all its tables.

The last type of SDN disruption attacks is flooding techniques. In these attacks, the attacker flood an SDN component with random network traffic or SDN data from different sources [114]. The goal of this attacks is to exhaust the SDN resources (bandwidth, storage, and processor) until they are overloaded. For example, an attacker that wants to flood the controller can perform the following actions:

1. The attacker place himself in the OpenFlow interface.
2. He captures any OpenFlow rule.
3. He modifies the action of the rule to Forward_To_controller systematically and activates the notifications coming from the network elements.
4. He releases the modified OpenFlow rules.
5. He repeats the modified OpenFlow rules to all the network elements.
6. The network elements install the modified rules.
7. All the network elements forward systematically all their traffic to the controller.

This attack transforms the network elements to high flooding vectors that exhaust the control data link and the controller resources. Many DoS attacks and their countermeasures have been reported in the literature. [115] shows a DoS that denies the access of an SDN network element in floodlight controller. The attacker spoofs the Data_Path_ID and the Mac_Adress of an existing network element. Then, he connects the fake network element to the same controller. As a result, the controller disconnects the legitimate network element and connects to the fake entity. The DDoS attack on SDN reported by [105, 116] floods an SDN controller with random traffic from the network elements. The controller reacts to this traffic by installing the corresponding OpenFlow rules. As a result, the attack overloads the controller resources and overfloods the bandwidth with the network elements.

There are in the literature many solutions to mitigate this attack. [117] proposes a solution that reduces the Hard_Time_out and Idle_Time_out of OpenFlow rules. [118] proposes another solution based on Moving Target Defense (MTD) [119]. Once the controller DDoS threshold is triggered, the solution activates a backup controller from the pool of controllers. It installs

into the network elements that have triggered the threshold two OpenFlow rules. The first rule diverts the new flows that have not any matches to the bloom filter [120] module of the solution. The second OpenFlow rule changes the role of the network elements to connect them with the backup controller.

Other disruption attacks target the network elements resources. [121] discusses and evaluates a DoS attack that exhausts the flow table size. The attacker sends continuously bogus flows that are slightly different from the previous flows. The network elements forward them to the controller. The latter reacts to the bogus flows by installing OpenFlow rules. The attacker continues until the OpenFlow table is full. [122] proposes a defense strategy to mitigate this attack. The solution relies on the collaboration between all the network elements. When the flow table of a network element reaches a severe size, the other network elements help the loaded peer by handling its new traffic. The solution diverts all the new traffic from the loaded network element to its peers. Moreover, [123] reports a DoS attack that targets the network element buffer to overflow it. The attack sends to the network element new packets with high payloads. The network element stores these large packets in its buffer and encapsulates their headers as packet_in to ask the controller for the OpenFlow rule. When the buffer becomes full, the network element will drop the legitimate traffic because it can not store it. One countermeasure is to control the buffer size. The controller monitors the buffer. Then, when its size reaches a predefined threshold, it asks the network element to forward all the new packets without buffering them.

SDN malicious modification attacks and countermeasures

Malicious modifications in SDN introduce changes into SDN components to hijack their behavior, values, and resources. These attacks breach the integrity and non-repudiation security dimensions of SDN. They can use three attack techniques to modify SDN components. The first technique is data modification. The attacker adds, deletes or alters SDN information assets (such as OpenFlow rules, packets, topology and configuration files). This type of attacks provides means for the attacker to bypass security mechanisms and poison the global knowledge of the controller. In the former, for example, the attacker injects in network elements OpenFlow rules with high priority to prevent network elements from forwarding specific traffic to a firewall [124]. In Global knowledge poisoning, the attacker inserts, deletes and alters network events, notifications, and other monitoring data. As a result, the controller constructs an incomplete and wrong view of the network. Besides, the attacker can also starve the SDN components with this attack. For example, a man-in-the-middle in the northbound API neutralizes all the data systematically from/to an application. The data modification can also serve for repudiation attacks. In this case, the attacker tampers the actions of SDN components or falsifies their logs.

The second technique is the malicious injection of code into the SDN components. The attacker detects vulnerabilities in the SDN components and their system. Then, he modifies their software by injecting its code to corrupt their behaviors. The malicious injection of code needs remote or local access to the SDN components. Besides, it requires also privileges and trust boundaries. For these reasons, most of the time these attacks occur from the administrative domain. The attacker hacks the administrator session to gain unauthorized access to the SDN components. Besides, these attacks can also be instigated by the application layer. For example, in the application layer, it is possible to have SDN applications that communicate with third-party applications. In this case, an attacker can inject a worm in an SDN application. Then, the malicious SDN application contaminates the other SDN applications by exploiting the vulnerabilities of the system running the applications or those of the northbound API. A malicious injection into the controller has a critical severity because a rogue controller can

corrupt all the SDN components.

The last technique is based on replaying information. An attacker captures the interactions between SDN components. Then, he repeats the capture when certain conditions are in place. This type of attacks enables the attacker to gain access to an SDN component, to modify it or masquerade its identity. For example, an attacker will perform the following actions to control the network elements by a replay attack:

1. He places himself as a man-in-the-middle on the northbound API.
2. He listens to all the policies sent by an SDN application controller and saves them in its database.
3. The attacker replays the policies according to a time interval.

[125] discusses the malicious modification of traffic with malicious network elements. When the attacker instigates a malicious modification on the network elements, the latter can have the following misbehavior:

1. **Traffic loss**: In this case, the attacker inserts OpenFlow rules that drop randomly or selectively the traffic. As a result, the malicious network element drops the traffic. The proposed countermeasure is based on the flow conservation principle. It detects the attack if the outgoing flows are less than the incoming flows in each network element.
2. **OpenFlow data fabrication**: The malicious network element produces bogus packets (or packets-in) and sends them to the controller. Sphinx [126] is an SDN security solution that detects this attack. It verifies the metadata of the packets sent to the controller with the policies of the administrator. If they abuse the conformance, it detects the attack.
3. **Mis-routing**: The attacker installs or modifies network elements rules to forward the traffic to wrong destinations. The authors propose to forward the traffic without matching to the controller to detect the attack.
4. **Traffic modification**: In this case, the attacker inserts OpenFlow rules in the network elements that alter the contents of packets. One countermeasure is to implement in the network element hash function as integrity code to ensure the integrity of OpenFlow modification messages.
5. **Traffic delay**: An attacker modifies the OpenFlow rules to put certain flow in queues with low QoS. He can also modify the lifespan of OpenFlow rules by minimizing it. The network element will forward the traffic to the controller because of the deletion of the expired OpenFlow rules. As a result, the jitter of the packets increases due to the interaction delay between the network elements and the controller. Monitoring the arrival times of packets in the network elements to detect the attack can be a countermeasure to this misbehavior.
6. **Traffic reordering**: The attacker injects a malicious code in the network element that alters the sequence number of packets. For example, he can perform the malicious modification by extending the experimenter of OpenFlow to handle a modify action on sequence numbers. The authors propose a detection mechanism that keeps an order list for packets digests in each network element. It compares the packet order with these lists to detect the network elements that reordered the traffic.

The solution in [127] detects malicious behaviors in network elements by active probing. The mechanism selects a set of OpenFlow rules from random network elements periodically.

Then, it constructs artificial packets. It sends the packets to the selected network elements. It traces the packet forwarding path. It compares the path with the programmed behavior. If they are not similar, it detects the attack and signals it to a Security Information and Event Management (SIEM) application [128]. In addition to an active probing module, the solution of [129] also integrates a statistic checking module and a packet obfuscation module to detect and mitigate malicious modifications on network elements. The statistic checking module verifies the flow statistics to detect network elements that falsify their notifications. The obfuscation module encrypts flows contents and headers to protect their integrity.

Global knowledge corruption is another malicious modification attack on SDN. It enables an attacker to instigate a DoS attack or even to take control of the SDN. [130] discusses and evaluates two types of topology poisoning attacks that subvert the SDN global knowledge. The host location attack deceives the controller with false data that inform it about the move of a target host to a new location (which is, in reality, the adversary location). As a result, the controller updates its knowledge with the new false location. It programs the flows of the target towards the attacker location. Besides, the link fabrication attack is the second topology poisoning attack. The attacker fakes LLDP packets to fabricates non-existing internal links between network elements. The authors propose some countermeasures to mitigate these attacks. For example, a security solution that verifies the legitimacy of host migration according to migration precondition and post-condition. In the former condition, the solution checks that the controller has received a port down notification from the network element before the end of host migration. In the post-condition, the solution verifies that the host is unreachable from its previous location after the end of the migration. The other countermeasures include the authentication of hosts and the signature of LLDP packets with integrity codes.

3.4 SDN for security

The concept of SDN for security uses the advantages of SDN to enhance network security solutions. For instance, centralization empowers security applications with holistic knowledge. This global knowledge can be used to optimize security operations, to reinforce security policies and to react efficiently to network attacks. Programmability enables network applications to deploy their behavior automatically on network devices. Federation and externalization provide means for security applications to collaborate with SDN components, stakeholders and third party entities using standardized means. As a result, security policies configuration and management become simpler because administrators use high level common standardized interfaces to express their policies. The controller interprets them to the suitable low-level rules according to the proper southbound interface. Besides, the separation enables the controller to verify the consistency and accuracy of security policies to validate them.

There are many other advantages that SDN brings for security. SDN is neither focused on a specific protocol nor is related to a particular architecture [131]. This independence enables many security solutions to benefits from SDN aspects by opposition to the legacy architecture. Furthermore, security applications adapt to the changes happening in the network thanks to SDN. For example, SDN enables security applications to operate on different granular levels in the network. A security application can behave differently with different traffic according to many levels of security instead of applying the same behavior to all traffic every time. This agility improves the efficiency of network security applications.

For example, a traditional Intrusion Detection System (IDS) in the legacy network will be deployed as hardware in a network node. The administrator configures the IDS manually to protect the network. He configures in every network element the proper policies to send the traffic to the IDS. Each time the policies change in the IDS or the network, the administrator needs to reconfigure everything manually. These tasks are costly, tedious and error-prone. Be-

sides, all the traffic is sent to the IDS before it reaches its destination. This type of transmission increases the network delay. Another way to minimize the negative impact of the network delay is to deploy many IDS in many nodes. However, in addition to the costs and the nightmare of policy management complexity, all the deployed IDS need to collaborate, to share knowledge and to coordinate their operations. In the opposite case, they will process many times the same traffic. They can have conflicting policies that lead to security breaches. The administrator can introduce new vulnerabilities due to its manual interventions.

Introducing and designing the IDS within SDN resolves the problems mentioned above because SDN has many advantages for security applications. SDN enhances the mitigation of network disruption attacks. It decreases the associated costs. It eases the management and the enforcement of security policies. Thus, in the case of SDN IDS, the administrator configures in a high-level language the policies on the IDS application. The controller verifies and validates that all the policies are consistent and accurate. It resolves the problems automatically by inferring new rules. When there is a change, the controller detects it and modifies all the concerned policies automatically. If the IDS is distributed on many nodes, the controller manages the interactions between them. It provides them with the holistic knowledge to optimize their operations and enable them to cooperate.

3.4.1 SDN simplification and independence for security

The design and deployment of new security solutions are more straightforward and useful in the application layer thanks to the standardized API and the independence of the control from network elements. A developer implements its security application in any programming language and independently from the source code of the controller. The northbound API hides the network complexity. The application only needs to integrate the proper interface to communicate with the controller. The role of the northbound API is to ease the integration of the security application into the SDN ecosystem while allowing the application to keep its independence and individuality. By contrast, this flexibility was not possible in the legacy architecture. A developer must know the implementation details of the network device. Besides, the implementation of the application is confined in the programming language and features that the network system supports. The developer also faces other constraints such as the inaccessibility to all the network system artifacts and the hardware limitations of the network device. The former limits the innovation on security applications while the latter increases the complexity of their implementations.

Masters of complexity administer security applications in legacy networks [132]. The administrators face a lack of abstraction and security policy in low-level languages. The security configuration in the legacy architecture is error-prone due to this complexity. It uses proprietary consoles that are based on low-level languages. It is also fastidious and risky because the administrator configures the devices one by one.

On the other side, SDN simplifies the security configurations of network elements [133]. Through SDN abstraction, administrators express their security policies in high level and simplified languages that abstract the complexity of the network. The controller interprets these security policies using the proper SDN interfaces into low-level specifications. All the complexity related to the underlying software and hardware is hidden. Furthermore, security applications can easily configure network elements in SDN because of two reasons [134]. It moves the complex modules to the control plane. It enables security applications to extend the data plane layer with new features. Thus, SDN simplification improves network management, reduces security services costs and ensures the portability of security policies [99, 135].

Thanks to the independence that SDN offers, the fates of SDN components are separated. Each asset is developed separately from the others while in the legacy architecture, the devel-

opment was based on the vertical integration between the three network layers. Besides, the SDN component runs in a different system and hardware. This diversity elevates the resilience of the SDN architecture. For example, an error in the source code of an SDN component will not generate another error in the source code of another SDN component. Besides, an attacker who detects a software technology vulnerability in an SDN asset will not be able to exploit the same vulnerability in other SDN assets. He needs to study and analyze the source codes of all the SDN assets. Thus, the SDN independence makes his tasks costly and burdensome.

3.4.2 SDN agility for security

SDN agility is the ability to adapt dynamically to network changes. It leverages security application with many features [136]. SDN enables the deployment of security applications by creating virtual networks dynamically. These virtual networks support VM mobility and flow adjustment. SDN controls the flow dynamically to drive it to the proper security application. Moreover, SDN provides means to adapt and deploy security policies in network elements dynamically. Thanks to SDN, it is now possible to deploy security applications dynamically next to the source of the attack and then direct only the suspicious traffic to this application [137].

The advantages of SDN adaptability are the customization of security solutions according to the network state and their effective deployment of security. The customization of security in SDN is on two levels. On the one hand, SDN adapts the functions of a security application according to the changes happening in the network. This adaptation includes the activation or deactivation of security modules, the upgrade of the security application, the construction of security service chains and other adjustment features. It also provides an effective security mechanism that changes the defense behavior dynamically to respond to the mutations of attacks [138]. The goal of the SDN adaptation is to ease the evolution of security applications with network states. On the other hand, SDN tailors the security policies according to the requirements of customers [139]. It satisfies their needs by installing fine-grained rules that satisfy their security policies.

The effective deployment of security applications in SDN enables their functionalities to scale up or down according to network state [140]. For example, suppose an SDN IDS that analyzes all the application layer traffic. The network elements send the traffic to the IDS via their channel with the controller. When the burden on the data link becomes essential and starts to affect the performance of the network, the SDN IDS reacts to this situation by programming a rule on the data plane devices to receive only specific part of the traffic. The IDS can even install such rule proactively. The rule will be triggered when the conditions are in place.

Another strategy is the deployment of many SDN IDS on the other neighboring controllers. The first controller asks them to analyze the traffic via the East/West API's. Then its IDS installs OpenFlow rules to redirect a part of the traffic to each IDS to divide the load among the neighbors. The strategy scale-down the controllers' resources to return to the initial state when those of central controller becomes available.

3.4.3 SDN global knowledge for security

SDN for security provides means for security applications to program network elements. Security applications rely on SDN interfaces to collect network state and events. These data are necessary to analyze traffic and detect security attacks. SDN holistic knowledge improves the operations of security applications that share the global knowledge without waiting for the convergence time to adapt their behavior [54]. For example, if an attack is detected, the alert is shared over all the security applications. This knowledge sharing can be used to adapt the security behavior globally according to the state of the network.

Unlike in SDN, security applications in the conventional architecture have only partial views of the network. They do not collaborate with each other. As a consequence, their functions are limited by their narrow scopes. Moreover, their partial knowledge affects the consistency of network security because of two reasons. The behavior of a security application can overlap with the one of another security application such as a security application that processes traffic that has already been processed by another one. The other reason is about behavior conflicts. The policies of security applications can contradict the security policies of another. The global knowledge of SDN resolves these issues. The controller uses its holistic knowledge to verify, validate and reinforce security policies. The utilization of the global knowledge for policies is very valuable for firewalls and security monitoring applications [95].

3.4.4 SDN convergence for security

SDN convergence sustains the collaboration between security applications on two levels. On the one hand, it enables users to express their security policies using the proper SDN interface to configure other security applications. Then, it interprets these artifacts by bridging them to the interfaces that are used by the other applications. For example, a user with legitimate access can configure seamlessly the policies of different security applications using the northbound interface unlike in the legacy network. On the other hand, the SDN convergence enables security application to exchange information in standardized formats. This information is the global knowledge, network events, statistics, and other resource information.

Thanks to this collaboration, the principle of the mix and match of different security applications from different vendors become simpler and operational within SDN [141]. It enables different proprietary solutions such as NAT, authentication solutions, IPS, firewall, IDS and others to collaborate and communicate. It offers them the proper bridges to exchange information and share their results. It chains them on different paths, virtual networks, and network domains to redirect to them the proper traffic, the needed information. It synchronizes their actions to optimize security and prevents conflicts and racing between their behaviors. Besides, SDN convergence enables applications to update their security and systems rapidly, and adjust their operations. These frequent updates are critical requirements for cloud computing [142].

3.4.5 SDN automation for security

SDN automatizes security. It enables security applications to reprogram the network elements proactively according to their security behaviors. In this case, a security application installs dynamically its behavior on network elements before the events that trigger it happen. Besides, security applications react effectively to network events because the controller interprets their actions to security rules and installs them on the network elements next to the source of the attack. The outcome of this automation is a fast resolution of network attacks [143].

Moreover, SDN offers to security application highly reprogrammable network elements and the capacity to redirect dynamically network traffic [144]. As a result, the configuration and deployment of security policies within the data plane become less error-prone and effective. In contrast to the legacy network, in SDN the user expresses its security policies, then the controller propagates them automatically on the proper network entities. A security application can monitor the traffic then updates the security policies to improve the network security. When the administrator changes the policies, the controller performs the updates automatically without needing manual interventions on each network device such as in the legacy network. This policy flexibility reduces downtime and improves the diagnosis of security issues [145].

Thanks to SDN automation, security policies elevates the behavior of network elements. As a result, these elements become security devices that perform security behavior. They execute security rules dynamically. This transformation is cost-effective because SDN reduces the need to buy new security devices [134]. Besides, SDN automation reduces network instability and network vulnerabilities [146]. SDN automatizes the deployment of security policies updates following the network state. It also alleviates the network of manual interventions that introduce security vulnerabilities.

3.4.6 SDN orchestration for security

Using programmability, abstraction, centralization, and federation, SDN orchestrates security operations efficiently. This orchestration can steer the proper traffic in the right security application [147]. It enables security applications to process the network from end-to-end. It can adjust the behavior of security applications according to the evolution of the network state. Also, the orchestration provides a means to hide the complexity of security policies. It automatizes their deployment and management. It ensures the interoperability between different security applications. It unifies them around the same policy management model.

As a result, SDN enables security applications to overcome the complexity of detecting distributed attacks coming from different sources, whereas in a legacy network device it is impossible to detect them [148]. Thus, the SDN security orchestration becomes like an immune system that collects network states from all its components, detects the network anomalies and then spreads the right countermeasures into the contaminated elements [145]. Unfortunately, the literature lacks propositions that integrate SDN orchestration for security. There is an open research field on the subject, especially, regarding SDN for security as a service and security policies based on SDN orchestration in the cloud.

CloudWatcher [149], FRESCO [150, 151], Flowguard [152] and OpenSec [153] are SDN solutions for security. They rely on their specific languages to specify security policies inside the network. They enable users to configure security policies. They spread these policies through the network elements. Besides, they can collect network events from network elements to offer them to security applications. These propositions need an SDN orchestrator because they lack interoperability between them. Besides, They do not integrate a policy management process to interact with different users. They do not reinforce security policies and share knowledge. They can introduce security vulnerabilities by contradicting and racing the policies of other security applications. An SDN orchestrator enables them to collaborate together and with different users and third-party applications. It abstracts their specific languages to a unified model. The latter will enable a user to compose services that are offered by these solutions. The orchestrator enables the users to express their security requirements while hiding the complexity of the underlying layers. Besides, the orchestrator gathers events observed by the different solutions to construct a holistic knowledge. The information of this knowledge can be used by the orchestrator to update the security solutions and reinforce security policies.

Some security solutions integrate aspects of SDN orchestration such as security policy expression and security policy enforcement. Tang et al. [154] develop a service-oriented high-level policy language to specify security service provisioning between end nodes. Batista et al. [155] propose the PonderFlow as an extension of Ponder [156] language. PonderFlow can be used to specify and reinforce security policies based on OpenFlow. Finally, EnforSDN [157] proposes a management process that exploits SDN orchestration to separate the policy resolution layer from the policy enforcement layer in security service appliances. The concept improves policy enforcement management, network utilization, and communication latency without compromising security policies. However, it cannot handle stateful security applications.

3.4.7 SDN solutions for security

Many security solutions have been developed using the SDN benefits that were mentioned earlier. SDN offers three features to enhance security solutions:

1. SDN offers the possibility to plug the northbound API inside the software architecture of a security application. In this case, the developers include in the code of their security applications the northbound API parts that enable their applications to interact with SDN controllers.

2. SDN offers Domain Specific Languages (DSL) to applications and users. These DSLs are used to express security policies and re-program network elements according to these policies.

3. Some SDN controllers offer programming environments such as development frameworks, libraries, plugins, and resources to develop security applications inside the controller. These security applications behave like controller functions. They use controller services. They access the global controller knowledge. They program the network elements directly. For example, FRESCO [150, 151] is an SDN framework that can be used to develop SDN security monitoring and detection applications. It offers a scripting language that is used for implementing security functions as modular libraries in the controller. Besides, the scripting language offers also the data structures and resources to compose complex security applications by linking and sharing the modular libraries.

These SDN features gave birth to many SDN security solutions such as Authentication & authorization, Moving Target Defense, Firewalls, and IPS & IDS solutions. We present some examples of these solutions in the following subsections.

Authentication & Authorization Solutions based on SDN

Authentication & Authorization solutions based on SDN have been developed as OpenFlow applications. For example, [158] proposes an authentication and access control mechanism called AuthFlow. This OpenFlow application authenticates hosts on the control data link layer. A host starts EAP authentication according to IEEE 802.1X standard [159] with AuthFlow's Authenticator. The latter de-encapsulates the authentication messages. It validates the authentication against the Radius server. Then, it sends the results to the AuthFlow application running in POX controller. If the host is successfully authenticated, the AuthFlow application determines its access to SDN components using its credentials and identity.

Flowidentity [160] is another authentication solution based on SDN. It uses the same authentication mechanisms of AuthFlow. Besides, it introduces an access authorization enforcement module based on a role-based firewall. This module can dynamically update the security policies of authenticated sessions. Thanks to SDN global knowledge, automation, and agility, FlowIdentity performs dynamically stateful policy updates to existing sessions unlike in the legacy network.

By contrast, the Authentication and Authorization application in [161] does not use IEE 802.1X software. This SDN application uses the northbound API to perform authentication and authorization of Machine To Machine (M2M) communications. It authenticates hosts based on their credentials. It allocates a bandwidth to each host. It authorizes the host to access other hosts based on an access list. It sends the high-level specification to the controller. The latter interprets them into OpenFlow rules and installs them on the network elements. When the host logs out, it uninstalls all its OpenFlow rules.

Moving target defenses based on SDN

Moving target defense technics defend the network against reconnaissance attacks. They change continuously in time the properties of the network to prevent or make difficult for an attacker to learn the network resources and their vulnerabilities [162]. As a result, the attacker will spend more efforts and time to look for the network vulnerabilities [163]. SDN enables active MTD thanks to the combination of SDN agility and global knowledge. The former enables SDN to adapt its MTD strategy seamlessly with the evolution of the network state and the attack. The SDN global knowledge leverage MTD with a holistic view that improves the reliability of its operations. As a result, MTD strategy is enforced according to an end to end security strategy. For example, OF-RHM [164] is an MTD solution based on SDN. It performs a host transparent IP address replacement by hiding real IP addresses with virtual IP addresses. It associates to the real IP addresses short living virtual IP addresses that are updated according to a time interval. The solution selects randomly from the unused IP address space these virtual IP addresses. Then, it installs the OpenFlow rules that perform the IP address mutation in all the network elements without affecting the end host. By the use of SDN features such as agility, automation, and global knowledge, OF-RHM mutates IP addresses with high unpredictability and speed to distort the attacker knowledge.

Firewall solutions based on SDN

A firewall is a security mechanism that protects a network from unauthorized access. It filters the traffic by matching the packets' headers with a set of security policies to allow only the current traffic to access the network [165]. SDN elevates firewalls with all its features. Simplification enables administrators to manage firewall policies without worrying about the complexity of the underlying infrastructure. Agility adapts firewall rules dynamically according to the network state. Global knowledge enables firewall applications to spread their rules all around the network. Interoperability enables firewall applications to install their rules and collaborate through a common northbound API. Finally, automation enables a firewall application to install its rules autonomously and manage them without the intervention of administrators.

There are two categories of firewalls based on SDN. Full-SDN-Firewalls rely on the control layer to express their behaviors and to collect network state (see Figure 3.1). Hybrid-SDN-firewalls are independent of the controller (see Figure 3.2). The details of all these firewalls are as follows:

1. **Full-SDN-Firewalls** Full SDN firewalls take advantage of SDN to secure the network. Mainly, their behavior is expressed using the southbound API features. This API can reprogram the network elements and collect network events. They do not depend on a specific data plane technology. Besides, their implementation can be independent of the control layer thanks to the northbound interface. However, the controller manages their behaviors and interactions. There are four types of Full-SDN-firewalls. Control function firewalls are firewall functions that are implemented inside the control layer. SDN application firewalls are implemented in the application layer. Extended OpenFlow firewalls extend OpenFlow features to support firewall behavior in OpenFlow. Finally, SDN firewall frameworks can be used to construct firewall functions in the control layer.

    - **Control Function Firewalls** [166–186]: They integrate a firewall module as a controller function. The stateless firewall denies or permits the traffic without managing the dynamic properties of connections such as its state or port allocations. In these Firewalls, the administrator configures the controller with security policies. Then, the firewall function interprets them into OpenFlow rules. It calls the OpenFlow primitives directly. The OpenFlow Agent constructs the appropriate OpenFlow

rules and installs them on the network elements. SDN controllers such as RYU, Floodlight and POX [167] integrate an OpenFlow stateless firewall as a control function.

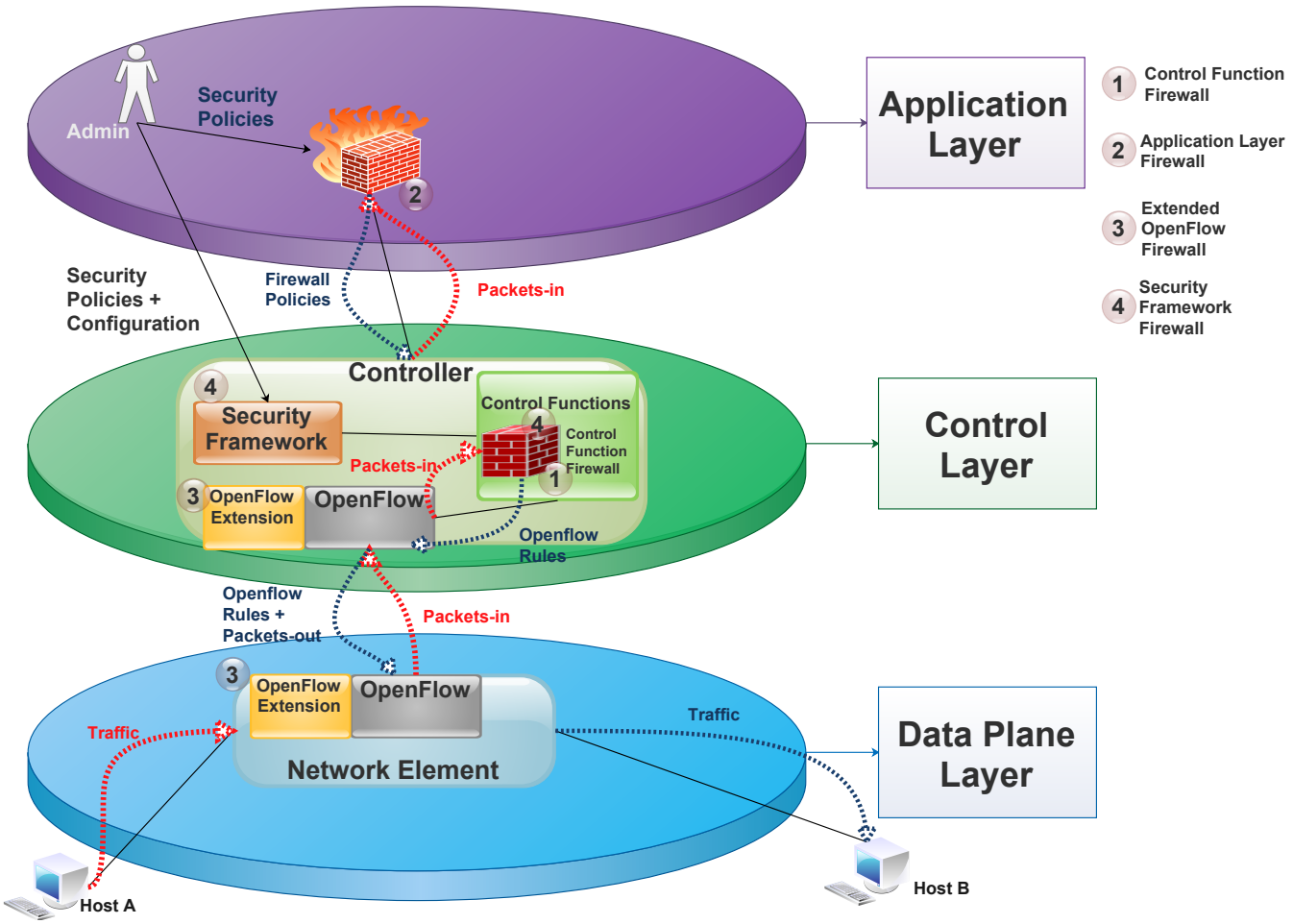


Figure 3.1 – Full SDN Firewalls

The advantage of these firewalls is their performance because they can use the primitives of the control layer directly to call OpenFlow capabilities instead of relying on a high-level interface. However, they are dependent on the type of the controller. They are not portable to other controllers. It is difficult to maintain them due to their dependencies with other controller functions. Developers need to modify the code of the controller to upgrade these firewalls which imply knowledge of the controller code. This complexity is a burden for innovation on both the firewall and the control layer. Moreover, the intervention of the firewall administrator can lead to failures in the controller. If he misconfigures the firewall function in the controller, the failure can propagate to other controller functions. The administrator can even introduce new vulnerabilities to the controller through the firewall function.

- **Applications layer firewalls** [187–194]: SDN firewall applications are developed independently from the control Layer. Principally, they are integrated into the application layer. They interact with the controller through the northbound interface. They send the security policies (expressed in a high level of abstraction) to controllers that interpret them into OpenFlow rules and install them on the network elements. Their implementation, maintenance, and upgrade neither depend on the source code of the control layer nor affect it.

To the extent of our knowledge, all the SDN application layer firewalls that have been implemented are stateless. Furthermore, they do not use the SDN global knowledge to perform end to end security. They are not able to reinforce security policies in the network by avoiding traffic reprocessing. They do not share knowledge with other firewall Applications in other SDN domains. Besides, they do not consolidate their domain knowledge to construct a holistic view of the network.

- **Extended OpenFlow firewalls**[195–199]: They extend OpenFlow protocol to handle a part of the firewall behavior in the data plane layer. This behavior is mainly related to connection state processing. For example, FleXam [195–197] extends the OpenFlow with a new artifact that can build firewall functions. It adds to OpenFlow a channel that specifies flows filters, sample schemes (parts of packets that will be sampled) and new actions to filter and sample flows. FleXam forwards all samples to controller systematically to inform it about the results of the sampling. Then, the controller verifies them with firewall policies and installs the associated OpenFlow rules.

  Although these firewalls introduce stateful processing into OpenFlow; they have some issues. Regarding performance, they consume more memory and processing resources because they need more memory space in the data plane devices and more CPU in both controller and network elements. They do not process the global knowledge to reinforce firewall policies. Besides, they are neither compatible with the existing SDN data plane devices nor compatible with the OpenFlow specification.

- **SDN firewall frameworks** [150–152, 200]: Some researchers propose SDN security frameworks that can implement SDN Firewalls. However, these frameworks are not appropriated for designing stateful Firewalls. They only allow the construction of stateless firewalls. Besides, they neither use nor share the global knowledge. They do not enable multiple firewall applications to collaborate in different domains to reinforce the security policies. Besides, Their maintenance is fastidious because it affects the code of the framework and the controller that supports them.

  FRESCO [150, 151] can develop SDN stateless firewalls. It instantiates predefined security modules and assembles them into an OpenFlow firewall application. However, FRESCO is not appropriate for building SDN stateful firewalls because it cannot track the states of a packet and handle dynamic traffic information such as port allocation. Another example of an SDN framework is Flowguard [152, 200]. It allows building OpenFlow stateless firewalls. It proposes a mechanism to detect and resolve firewall policies violations caused by traffic modifications and OpenFlow rules dependencies.

2. **Hybrid-SDN-Firewalls**: they introduce control function into the network elements or collaborate with the legacy architecture. The controller in these firewalls only installs the OpenFlow rules to direct the traffic towards them and to resubmit the ingress traffic coming from them into the OpenFlow pipeline. As a result, the control layer has neither visibility nor any controllability on the traffic processing inside these firewalls. The different types of these firewalls are as follows.

   - **Network element firewalls**[27, 201–210]: They integrate firewall functions in network elements by modifying the data plane architecture. They are powerful because they process the traffic directly in the network elements with dedicated devices. Their hardware structure is adapted to process network traffic and to perform

firewall functions. However, the control layer depends on them to know which traffic has been processed and how to manage it.

Unfortunately, there is not an SDN controller able to control these firewalls because they require extensions in the southbound API. Besides, they contradict SDN philosophy because they incorporate full control functions into network elements likewise in the legacy architecture. In these firewalls, the control is vertically integrated with other network element functions. As a result, the control layer loses its controllability.

The authors in [201] design an SDN stateful firewall inside the network element. The solution modifies the data plane architecture by adding new tables in the network element and new messages to the OpenFlow API. A State Table handles the actual state of a connection, and a Shifted Flow Table processes the next states. A third table is added at the level of the control layer to store the information about the states. The solution introduces state-in messages as a new feature to synchronize with the State Table. In this solution, the states of the connections are managed by the network elements. The controller is only informed about the updates because all the firewall control function is inside the data plane. Furthermore, State-in messages generate more traffic with the Controller. Moreover, the data plane sends systematically any unknown traffic to the Controller. As a result, an attacker can use this breach to flood the control layer and perform DDoS attacks.

In [202] the authors integrate a distributed firewall into OVS. The idea aims to improve the performance of SDN firewalls and to elevate data plane features by handling layers 4 to layer 7 data. They incorporate into OVS, the Linux connection tracking system Conntrack [211]. The latter provides a stateful firewall with advanced features to perform deep packet inspection. The solution adds a new action to OpenFlow to send the flows to the Conntrack modules. These modules process the traffic according to its pre-configured security policies. The processing steps in this firewall are as follows:

(a) The flow that corresponds to an OpenFlow entry with a Conntrack action is sent to the Conntrack modules.

(b) The Conntrack module inspects them to determine their state.

(c) It updates the state connection tracking tables with the new state.

(d) Conntrack module forwards the flow with its state to the OF table.

(e) The flow is recirculated inside the flow tables to match with another OF Rule.

This solution uses deep packet inspection and network address translation in the data plane. However, the controller cannot intervene in this process and has not any visibility on the processing done by Conntrack because the behavior of Conntrack is independent of the control layer.

Moreover, this firewall cannot use the global view of the network to reinforce security policies. Its behavior can lead to issues of coordination and discrepancy with the control layer because of the integration of the control inside the network elements. Besides, each data plane device needs to be configured individually and manually with security policies by the administrator. As a result, problems of manual errors, policy incoherence can occur due to this lack of policy automation.

OpenState [27] modifies also the data plane architecture and extends the OpenFlow specification. It integrates into the OpenFlow network element a state table that represents the Extended Finite State Machine (XFSM) of the traffic protocol. The state table enables the data plane layer to control the connections by tracking their states. Moreover, OpenState adds new OpenFlow structures and messages to enable

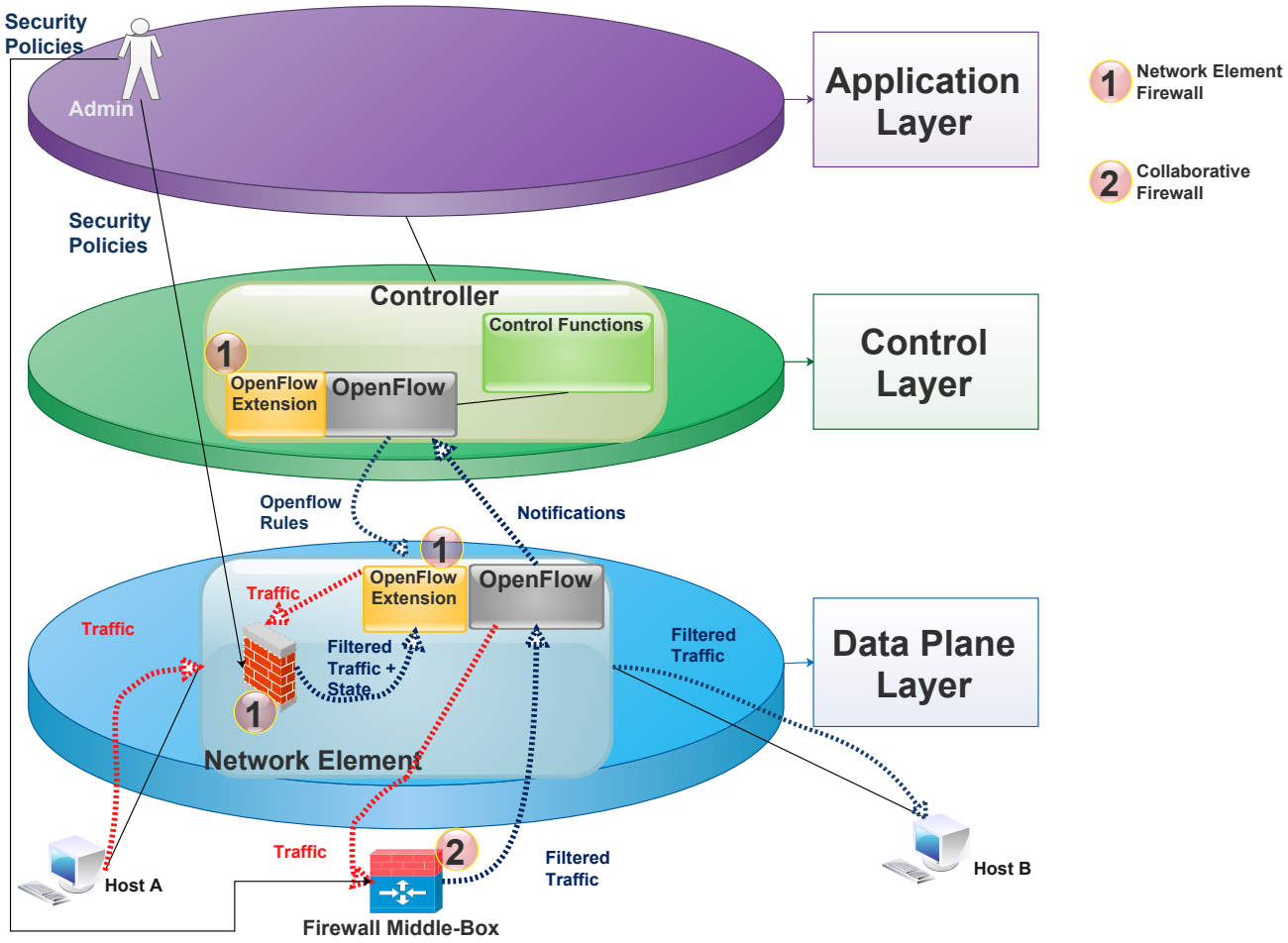


Figure 3.2 – Hybrid SDN Firewalls

the controller to initialize the connection tracking in the network elements. The developers of Openstate assume that their solution is more efficient than application SDN Firewalls. However, it has some drawbacks. It is not able to adapt its behavior to the global knowledge provided by SDN. Besides, it is not compatible with the existing SDN network element architecture. It puts a part of the control inside the data plane layer. The Openstate network element becomes able to take decisions independently from the control layer.

Although network element firewalls are considered to have better performance than other SDN firewalls; they enlarge the vulnerability surface of SDN. They introduce new vulnerabilities that can be exploited by attackers such as unbounded flow state memory allocation, triggerable CPU intensive operations and lack of a central state management [212]. In unbounded flow state memory allocation, the attacker exhausts the memory of the network element by pushing the latter to extend and update its state table continuously. In triggerable CPU intensive operations, an attacker forces the network element to execute intensive CPU operations that exhaust or slow the performance of the network element. An example of an intensive CPU operation is the notification of the controller each time there is an event on the state table. Lack of central state management gives means to an attacker to poison the view of the controller and to compromise the consistency of traffic states. This vulnerability is due to the absence of a centralized entity that involves the processing of states and their synchronization between network elements.

- **Collaborative firewalls** [157, 213–224]: They connect hardware or virtual firewalls

(ex. middlebox) to network elements to filter network traffic. In this case, the controller installs the appropriate OpenFlow rules to forward the traffic to the middlebox and to resubmit the ingress flows coming from the middlebox inside the flow tables. Collaborative firewalls reuse existing legacy firewalls with SDN; however, their behaviors can neither be controlled nor be automatized by SDN. Their architecture induces a hop for each link between a middlebox and a network element. These additional routes increase traffic delays because the network elements send traffic to the middle-box and wait for its processing before forwarding it. Besides, The controller has no visibility on their behavior and their security policies. Thus, they cannot use the global knowledge. Moreover, because they do not collaborate in a distributed environment or a multi-domain network, problems of security policies inconsistencies can arise. For example, a middlebox in a domain A can block traffic which is authorized in a domain B.

IPS & IDS based on SDN

Both IPS and IDS based on SDN rely on automation and SDN global knowledge. The SDN automation provides means to prevent and react to security attacks by installing the proper countermeasures and directing the relevant traffic. The SDN global knowledge improves the monitoring of network events to detect security attacks. Monitoring is an important function in both solutions. SDN offers two types of network monitoring [225]. Passive monitoring consists of collecting information about SDN assets such as flow statistics and resource usability. It supports two modes of information collection. In the push mode, the SDN assets send the information to the controller when an event happens. In the pull mode, the controller request the needed information from the SDN components. The second type of monitoring that SDN provides is the active monitoring. SDN controller collects the traffic or a part of it in the form of packets-out from network elements. By combining these types of monitoring, SDN leverages the detection of attacks with network deep and wide knowledge.

The combination of global knowledge, agility, and automation enable SDN to harvest intelligence from IDS and IPS [226]. This knowledge can be shared among other SDN security applications and controllers to reinforce security in the network. For example, when the IDS detects an attack, it informs the controller. The latter notifies an access control application about the detection to update the permissions in the network.

The IDS of [227] combines both types of monitoring to detect faulty traffic and DDoS attacks. It installs OpenFlow rules based on the IP addresses of known sources of attack and other hosts. The OpenFlow rules abstract blacklists, whitelists, geo-localization data and event severity related to these IP addresses. Furthermore, they direct the flow statistics and packets-out using these rules to the controller to monitor the behavior of all the sources. The controller IDS function performs traffic inspection to detect and respond to network anomalies. It uses the SDN programmability to install OpenFlow rules that drop bogus flows and direct DDoS traffic to a DDoS washing machine.

Another example is the IPS of [228]. This collaborative OpenFlow IPS performs active threat detection and prevention using SDN automation and global knowledge. The solution integrates a security control application in the controller to reprogram the network elements and collect network events. It extends the Floodlight controller with built-in modules in its northbound API. The new modules ensure the interactions between the controller and the other subsystems of the IPS such as Snort application (a lightweight network and open source intrusion detection tool) [229, 230] and the honeypot. They also interpret the directives of Snort into OpenFlow rules. This SDN IPS combines the following mechanisms:

1. Botnet/Malware blocking mechanism: it is based on a Snort machine with 5000 bot-

net/malware Snort rules. The controller directs the traffic to this machine. When snort detects a botnet or a malware, it informs the IDS control application. Then, the latter installs the appropriate OpenFlow rules to drop the malicious traffic in the network elements.

2. Scan filtering mechanism: it is based on an anti-scanning algorithm that runs in all the hosts. When the algorithm detects scanning attacks on the host, it informs the IPS control application. Then, the latter reprogram the network elements to block the traffic of the attack source.

3. Honeypot mechanism: it uses a Virtual Machine (VM) that pretends to be a potential victim to attract attackers. The VM relies on configuring the DoS and DDoS rules in its snort to detect the attacks. When it detects the disruption attack, it alerts the IPS control application that reprograms the network elements to block the traffic of the attack source.

3.5 Discussion

In this chapter, we have discussed the security for SDN and the improvements that SDN brings to security. On the one hand, we have shown that SDN lacks security solutions that protect its security dimensions. Its features amplify attacks in SDN. As a result, they enhance the attacker with superpowers. Besides, SDN assets are vulnerable to many attacks. They can be even used as attack vectors against other SDN and non-SDN assets. The controller is a single point of failure and a potential backdoor to contaminate other assets. For this reason, it is considered by attackers as attractive and a potential Pandora box.

On the other hand, we have discussed the benefits of SDN for security. SDN is an enabler for many security applications. It leverages monitoring, detection, prevention and defense mechanisms and mitigation. It has accelerated the research work and innovation about enhancing security solutions with automation, global knowledge, interoperability, agility and network simplification. Many security applications have been improved thanks to its features.

Nonetheless, its security is still a challenging hot subject that reduces the usability of security solutions based on SDN [231]. Many security aspects of SDN are not addressed. The research in SDN security analysis is still not appropriately tackled [232] because it does neither quantify the impacts of SDN security vulnerabilities nor study their security. Therefore, we propose an analysis of SDN security vulnerabilities in the next chapter.

Part II

Security for SDN

Chapter 4

Software Defined Networking Vulnerability Analysis

" Vulnerability is the birth place of innovation, creativity and change. "

Brené Brown

Contents

4.1 Introduction

Network security analysis faces many challenges such as uncertainty, lack of adequate data and technological mutations [233]. The last challenge becomes the most impressive driving force for the overhaul of security analysis in computer networks. Especially with the advent of SDN, security analysis needs to cover the specific characteristics of this paradigm to enable stakeholders to take effective security decisions that protect SDN assets.

Vulnerability analysis is an essential process in security because it enables discovering the weaknesses of a system and their impacts. Nonetheless, analyzing a system to detect its unidentified weaknesses is still a complex and a subjective process [234] because there is neither a universal classification nor a standardized methodology in vulnerability analysis. This fact is particularly amplified in a dynamic and emerging environment such as SDN where there are neither documented SDN vulnerabilities nor quantified regarding their severity.

We propose in this chapter the first vulnerability analysis of SDN in the literature. Our work is part of the European project TANDEM SENDATE [235, 236]. It builds an SDN vulnerability model and quantifies their severity. It calculates the vulnerabilities severity using the Common Vulnerability Scoring System (CVSS). Then, it adapts the results according to the impacts of SDN features on security (see Chapter Section 2.2) using the Analytic Hierarchy Process (AHP).

4.2 Problem Statement

SDN is facing substantial challenges in security because it inherits security flaws from classical network architecture and it introduces new vulnerabilities. Besides, its features empower the attackers with a superpower that increase the severity and likelihood of attacks. Therefore, a general vulnerability analysis for SDN is necessary to improve SDN security and understand its weaknesses. This analysis needs to provide a generic classification of SDN vulnerabilities and evaluates their impacts on network security. Vulnerability analysis of SDN enables organizations to know the impacts of their SDN conceptual and implementation choices. It supports them to adopt suitable countermeasures against security attacks by reducing the attack surface.

We rely on the Common Vulnerability Scoring System (CVSS) [237–240] to quantify the impacts of SDN security vulnerabilities. CVSS is based on qualitative and quantitative metrics that define the severity of security vulnerabilities. Its computation procedures integrate three dimensions related to the different characteristics of legacy networks: their essential generic features, their temporal features, and their environment-related factors.

However, SDN features are not covered by CVSS but they affect the security of SDN and enlarge its attack surface. For example, the centralization of the controller transforms its components to precarious shared resources among other SDN assets. In this case, all the SDN assets are exposed by the vulnerabilities of the controller. In this case, CVSS on SDN will not include the controller influence on other SDN assets. This influence is due to the impacts of SDN features on each asset. Thus, a proper vulnerability analysis using CVSS needs to integrate SDN features into the evaluation of vulnerability because they determine the importance of each asset in SDN. Some assets are more influenced by SDN features and can expose other assets to security attacks. In this context, we rely on decision-making procedures to measure the weights of SDN assets using SDN features.

Decision making is the process to choose among alternatives based on multiple factors [241]. Although decisions are subjective judgments and they depend on the knowledge and experience of domain experts, some methods can be used to rationalize decisions and quantify

them mathematically. The Analytical Hierarchy Process (AHP) [91, 242–245] is one of these approaches. It is a multi-criteria decision making procedure. It is used to define and evaluate the importance of decision alternatives in the decision-making process. It decomposes complex problems into many levels of connected subproblems. Then, it evaluates the intensity of each subproblem in the overall set of problems.

AHP enables us to evaluate the impacts of SDN features on its assets to determine their weights. These weights are integrated to the severities of each asset vulnerability to adapt CVSS for SDN. The steps of our generic vulnerability analysis are as follows :

1. We classify SDN assets.
2. We propose a set of security attack objectives based on security dimensions.
3. We construct a matrix of SDN vulnerabilities by combining SDN assets with security attack objectives.
4. We quantify the severity of SDN vulnerabilities using CVSS.
5. We adapt the quantification by including into the process SDN features intensities, using AHP.

4.3 Vulnerability Analysis Concepts

We use CVSS to evaluate the severity of SDN vulnerabilities. We adapt CVSS logic to SDN context. This adaptation enables us to choose the proper criteria for SDN vulnerabilities using CVSS (Section 4.3.1). Besides, we introduce AHP into the quantification of the vulnerabilities severity (Section 4.3.2). AHP transforms the CVSS results according to SDN features. The details of CVSS and AHP are as follows.

4.3.1 Common Vulnerability Scoring System

CVSS [237–240, 246, 247] offers an open framework to assess and evaluate the impacts of computer security vulnerabilities. It uses a set of standardized metrics to score vulnerabilities and compute their severity scores. It enables organizations to improve their security policies because its neutrality improves the identification and remediation of vulnerabilities. It supports the evolution of systems because it provides contextual scoring that represents the severity of the vulnerability according to the actual context. Also, its framework is open-source. The parameters and the quantification methods can be accessed and modified by users.

CVSS provides three groups of metrics [248] that evaluate the severity of vulnerabilities. It offers a set of equations to calculate the severity score in each group. The equations have been elaborated based on intensive research work on vulnerabilities and their analysis. They rely on a set of mathematical rules and rationals that varies depending on the metric type [249] (the details of the equations is in [250]) The metrics of the base group evaluate the unchangeable characteristics of vulnerabilities that are not influenced by time nor by user environments. On the other hand, the metrics of the temporal group process the characteristics that are subject to change over time. Besides, those of the environment group focus on the proprieties that are influenced by user environments. CVSS associates with each several metric options that determine the difficulties for an attacker to exploit the vulnerability. Each option has a different value that expresses the degree of making the attacker task easier or harder. These values are the parameters of the formulas that calculate the severity of vulnerability in each group. Table 4.1 describes all the CVSS metrics according to SDN requirements. Table 4.2 [248] lists the options of each metric and their numerical values.

The base group represents the features of a vulnerability that neither change in time nor the implementation. It specifies many characteristics. Attack Vector is the location where the attacker accesses the vulnerability. Attack Complexity is the complexity of the vulnerability exploitation. Privilege Required are the privileges the attacker needs for exploiting the vulnerability. Scope refers to the spread of the vulnerability to other assets. User Interaction determines if the attacker needs to interact with the user. The impacts of the vulnerability on security dimensions define the influence of the vulnerability on confidentiality, integrity, and availability. The base group severity is the maximum impact value of severity and the worst case. The Temporal metrics adapt the value of the base by reducing it according to their characteristics. The base group score SV_{base} is computed according to the following steps:

1. CVSS computes the Exploitability of the vulnerability E(V). This metric represents the capacity of the attacker to exploit the vulnerability. CVSS uses the following formula:

$$\begin{aligned} E(V) = {} & 8.22 \times \text{Attack Vector} \times \text{Attack Complexity} \\ & \times \text{Privilege Required} \times \text{User Interaction} \end{aligned} \tag{4.1}$$

2. CVSS computes the base Impact Sub Score (ISC_{base}) using the following formula:

$$ISC_{base} = 1-((1-Impact_{Confidentiality}) \times (1-Impact_{Integrity}) \times (1-Impact_{Availability})) \tag{4.2}$$

3. CVSS calculates the Impact Sub Score (ISC) by taking into consideration the scope of the vulnerability. It uses the following formulas:

$$ISC = \begin{cases} 6.42 \times ISC_{base} & \text{If scope = Unchanged} \\ 7.52 \times (ISC_{base} - 0.029) - 3.25 \times (ISC_{base} - 0.02)^{15} & \text{If scope = changed} \end{cases} \tag{4.3}$$

4. CVSS calculates the vulnerability severity of the base group SV_{base} using the following formula:

$$SV_{base} = \begin{cases} Round\_up\ (Minimum((ISC + E(V)), 10)) & \text{If scope = Unchanged} \\ Round\_up\ (Minimum(1,08 \times (ISC + E(V)), 10)) & \text{If scope = changed} \\ 0 & \text{If ISC <= 0} \end{cases} \tag{4.4}$$

Table 4.1 – CVSS metrics for SDN

Group	Metric	Definition
Base	**Attack Vector**	The path by which an attacker exploits the vulnerability, it can be remote from an external network (internet), Adjacent from a neighboring network (ex. another domain), Local from within the SDN network or physical by physically accessing the assets
	Complexity	The efforts costs that the attacker needs to exploit the vulnerability. Low efforts refer to direct actions that lead to repeatable success. High efforts use reconnaissance, preparation and pentesting
	Privilege Required	The special rights that the attacker needs to exploit the SDN vulnerability. The options are: None, Low: user privileges and high: management privileges
	User Interaction	Determines if an SDN user is needed to participate in the exploitation of the vulnerability
	Scope	Determines if the vulnerability exploitation impacts other SDN assets
	Confidentiality	The impact of the vulnerability exploitation on SDN confidentiality. The options are: None, Low (Asset leaks to authorized entities), High (Asset leaks to unauthorized entities)

Table 4.1 continued from previous page

Group	Metric	Definition
	Integrity	The impact of the vulnerability exploitation on the veracity and trustworthiness of SDN assets. The options are: None, Low (assets are altered but the attacker neither controls the process nor measures the consequences), High (the attacker controls the modification process and the consequences propagates in SDN)
	Availability	The impact of the vulnerability exploitation on SDN accessibility. The options are: None, Low (the asset availability is partially impacted, it is unavailable for a certain time only or available all the time with some interruptions), High (the asset is completely inaccessible)
Temporal	**Exploit Code Maturity**	The maturity of the attacker techniques that enable the attacker to exploit the SDN vulnerability. The options are: Unproven (No exploit is available or it is theoretical), Proof of Concept (the exploit code is not functional in all SDN environments and it requires first a modification by skilled attackers), Functional (the code works in SDN but it requires skilled attackers), High (exploit code works in SDN and is widely-available, reliable and easy to use)
	Remediation Level	The level of fixes and patches to correct the SDN vulnerability. The options are: Official Fix (a complete SDN vendor solution is available), Temporal Fix (a temporal tool or hot-fix is available), Workaround (unofficial fix and a non-vendor patch is available), Unavailable (there is no solution to correct the vulnerability)
	Report Confidence	The level of technical knowledge on the vulnerability that is available to potential attackers. Unknown (there are documents that are reporting the vulnerability but its causes are not identified), Reasonable (significant details are available but the vulnerability is not proven in practice), Confirmed (detailed vulnerability reports are available and functional reproduction is possible)
Environmental	**Confidentiality Requirement**	It enables the customization of confidentiality for the SDN asset relatively to other metrics
	Integrity Requirement	It enables the customization of integrity for the SDN asset relatively to other metrics
	Availability Requirement	It enables the customization of availability for the SDN asset relatively to other metrics
	Modified Base Metrics	The 8 metrics customize base metrics according to the influences of the environment

The temporal group specifies the values of the vulnerability characteristics that change over time. This group includes three vulnerability metrics. Exploit Code Maturity refers to the availability and the usefulness of techniques and malicious code to exploit the vulnerability. Its options can vary in a scale that starts with theoretical exploitation. Then, it evolves to proof of concept code and finally to functional exploit code. Then, it becomes fully mature to be used by unskilled attackers. The Remediation Level is the second metric. It refers to the availability and level of maturity of corrections and fixes for the vulnerability. Finally, Report Confidence evaluates the credibility degree of the existence of the vulnerability. CVSS calculates the vulnerability severity of the base group SV_{temp} using the following formula:

$$SV_{temp} = Round\_up\ (SV_{base} \times \text{Exploit Code Maturity} \times \text{Remediation Level} \times \text{Report Confidence}) \tag{4.5}$$

The environment group measures the scores of the vulnerability features that are impacted by the implementation or deployment in a specific IT or user environment. Each environment has a different impact on vulnerability and by extension on the security of a system. The environment influences the characteristic of the vulnerability of the base group. For example, suppose we have an SDN Implementation. We deploy the SDN in a data center platform that includes an AAA solution. This environment changes the access metrics because it makes the tasks of the attacker harder. The environment group modifies the metrics of the base group.

Table 4.2 – CVSS Metrics, options and their numerical values

Number	Metric	Option	Numerical Value
1	Attack Vector (Base group) & Modified Attack Vector (Environment group)	Network	0.85
		Adjacent Network	0.62
		Local	0.55
		Physical	0.2
2	Attack Complexity (Base group) & Modified Attack Complexity (Environment group)	Low	0.77
		High	0.44
3	Privilege Required (Base group) & Modified Privilege Required (Environment group)	None	0.85
		Low	0.62 (0.68 if Scope or Modified Scope are Changed)
		High	0.27 (0.50 if Scope or Modified Scope are Changed)
4	Scope (it modifies the value of Privilege Required & Modified Privilege Required)	Unchanged	Does not impact
		Unchanged	See previous metric values
5	User Interaction (Base group) & Modified User Interaction (Environment group)	None	0.85
		Required	0.62
6	Confidentiality (C), Integrity (I), Availability (A) Impact (Base group) & Modified C,I,A Impact (Environment group)	High	0.56
		Low	0.22
		None	0
7	Exploit Code Maturity (Temporal group)	Not Defined	1
		High	1
		Functional	0.97
		Proof of Concept	0.94
		Unproven	0.91
8	Remediation Level (Temporal group)	Not Defined	1
		Unavailable	1
		Workaround	0.97
		Temporary Fix	0.96
		Official Fix	0.95
9	Report Confidence (Temporal group)	Not Defined	1
		Confirmed	1
		Reasonable	0.96
		Unknown	0.92
10	Security Requirements: C,I,A Requirements (Environment group)	Not Defined	1
		High	1.5
		Medium	1
		Low	0.5

Besides, it enables the prioritization of the security dimensions depending on the business model of the environment. These priorities are Confidentiality Requirement (CR), Integrity Requirement (IR) and Availability Requirement (AR). For example, in a banking environment, confidentiality and integrity will be the priority while in a video streaming use case, availability will be the focus. The environment group severity SV_{env} is computed according to the following steps:

1. CVSS computes the Modified Exploitability sub score $E(V)_{Mod}$ which represents the impacts of the environment properties on the exploitability of the vulnerability. It uses the following equation:

$$\begin{aligned} E(V)_{Modified} = 8.22 \times AttackVector_{Mod} \times AttackComplexity_{Mod} \\ \times PrivilegeRequired_{Mod} \times UserInteraction_{Mod} \end{aligned} \tag{4.6}$$

2. CVSS computes the environment Impact Sub Score (ISC_{env}) using the following formula:

$$\begin{aligned} ISC_{env} = Minimum(1 - (((1 - Impact_{mod\_Confidentiality}) \times CR) \times \\ (1 - (Impact_{mod\_Integrity} \times IR)) \times (1 - (Impact_{mod\_Availability} \times AR)))) , 0,915) \end{aligned} \tag{4.7}$$

3. CVSS calculates the Modified Impact Sub Score (ISC_{mod}) by taking into consideration the scope of the vulnerability. It uses the following formulas:

$$ISC_{mod} = \begin{cases} 6.42 \times ISC_{env} & \text{If scope = Unchanged} \\ 7.52 \times (ISC_{environment} - 0.029) - 3.25 \times (ISC_{env} - 0.02)^{15} & \text{If scope = changed} \end{cases} \tag{4.8}$$

4. CVSS calculates the impact score of the temporal group IST on the environment group. It uses the following equation:

$$IST = Exploit\_Code\_Maturity \times Remediation\_Level \times Report\_Confidence \tag{4.9}$$

5. CVSS calculates the vulnerability severity of the base group SV_{env} using the following formula:

$$SV_{env} = \begin{cases} Round\_up\ (Min((ISC_{mod} + E(V)_{mod}), 10)) \times IST & \text{If scope = Unchanged} \\ Round\_up\ (Min(1,08 \times (ISC_{mod} + E(V)_{mod}), 10)) \times IST & \text{If scope = changed} \\ 0 & ISC_{mod} =< 0 \end{cases} \tag{4.10}$$

The final scores of CVSS values SV_x ($x \in$ [Base, Temporal, Envirenment]) are standardized within the interval [0 10]. They can be mapped to the following qualitative judgments:

$$\begin{cases} \text{If } SV \in [0,1 - 3,9] & \text{Then the severity is Low} \\ \text{If } SV \in [4 - 6,9] & \text{Then the severity is Medium} \\ \text{If } SV \in [7 - 8,9] & \text{Then the severity is High} \\ \text{If } SV \in [9 - 10] & \text{Then the severity is Critical} \\ \text{Otherwise} & \text{None} \end{cases}$$

Table 4.3 – AHP Scale of importance for pairewise comparaison

Qualitative judjement	Equal Importance	Equally to Moderately	Moderate Importance	Moderately to Strong	Strong Importance	Strongly to to very strong	Very strong Importance	Very strong to extremely	Extreme Importance
Direct numeric rate (importance of a to b)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Reciprocal numeric rate (importance of b to a)	**1/9**	**1/8**	**1/7**	**1/6**	**1/5**	**1/4**	**1/3**	**1/2**	**1**

4.3.2 Analytic Hierarchy Process

AHP [91, 242–245] is a decision-making model that optimizes decisions when dealing with qualitative, quantitative and conflicting choices. It decomposes a qualitative problem (a decision goal) into many levels of hierarchies. The decider chooses many criteria and evaluates them according to many decision alternatives to reach a decisive goal. The criteria are the features that characterize the goal. The alternatives are the possible goals that can be reached. AHP evaluates the weights of each alternative according to all the criteria. It uses this decision method to help decide on the most weighted alternatives. In the first level of the hierarchy, AHP measures the relation between the different criteria based on their intensities for the decisive goal. The values of the criteria are evaluated according to their pairwise comparisons. Then, AHP calculates the weight of each criterion. The higher the weight of a criterion, the more dominant it is compared to other criteria. Then, in the other levels, the criteria are rationalized with the different alternatives according to matrix algebra. The alternatives are evaluated in pairwise comparisons according to each criteria dimension. The higher is the score of the alternative; the more important is its performance for the criteria comparing to other decision alternatives. Finally, AHP combines the weights of all the criteria with the scores of the alternatives. It determines the global value of each alternative regarding all the criteria. AHP evaluation has four main steps. Their details are as follows.

Weighting the criteria

In this step, AHP uses a pairwise matrix to compare the criteria between them. It determines the intensity level of criteria in respect to other criteria. The values of the comparison are based on transforming a qualitative relation scale to a numeric scale. These values fulfill a standardized AHP scale (see table 4.3) [251] from 1 (two criteria with the same intensity) to 9 (a criteria is extremely intense than the other one). The following calculations take place in this step:

1. AHP calculates the pairwise matrix A($n \times n$) (where n is the number of the criteria). The matrix A ensures the following conditions:

$$a_i^j \times a_j^i = 1 \tag{4.11}$$

   (where a is the intensity value, i and j are the criteria indexes from 1 to n).

2. The matrix A is normalized to the matrix B according to the following equation:

$$b_i^j = \frac{a_i^j}{\sum_{k=1}^{n} a_k^i} \tag{4.12}$$

(where b_i^j are the elements of the matrix B).

3. AHP calculates the weight vector W based on the Eigen vector method [252]. The weights w_i of the vector are computed according to the following equation:

$$w_i = \frac{\sum_{j=1}^{n} b_i^j}{n} \tag{4.13}$$

Scoring the alternatives

AHP uses a pairwise comparison to estimate the importance of each alternative. For each criterion dimension, it evaluates the intensities of an alternative compared to the other alternatives. These values represent the performance of the alternative in the criterion dimension. Therefore, it builds n pairwise matrices where each matrix represents a criterion. The following calculations take place in this step:

1. For each criterion c_i, AHP calculates the pairwise matrix $\mathrm{A}_{c_i}(m \times m)$ (where m is the number of alternatives). The matrix A_{c_i} ensures the following conditions:

$$ac_k^l \times ac_l^k = 1 \tag{4.14}$$

(where ac is the intensity value, k and l are the alternative indexes from 1 to m).

2. The matrix A_{c_i} is normalized to the matrix B_{c_i} according to the following equation:

$$bc_l^k = \frac{ac_l^k}{\sum_{p=1}^{m} a_p^l} \tag{4.15}$$

(where b_l^k are the elements of the matrix B_{c_i}).

3. AHP calculates the weight vectors $\mathrm{W}^{(c_i)}$ based on the Eigen vector method. The weights $w_l^{(c_i)}$ of the vector are computed according to the following equation:

$$w_l^{(c_i)} = \frac{\sum_{k=1}^{m} bc_l^k}{m} \tag{4.16}$$

4. AHP constructs the Matrix of scores S where each column corresponds to a criteria weight vector.

$$\mathrm{S} = (\mathrm{W}^{(c_1)}, \mathrm{W}^{(c_2)}, ..., \mathrm{W}^{(c_n)}) \tag{4.17}$$

Consolidation of the alternative with the criteria

The weights of the alternatives are consolidated with weights of the criteria in order to evaluate the overall intensities of each alternative. This evaluation enables the decision maker to rank its alternative by its global importance in its decision. AHP calculates the global ranking vector V which is the product of the matrix S with the criteria vector W. Each element $v^{(c_i)}$ (with $i = 1..m$) of V is calculated according to the following equation:

$$v^{(c_i)} = \sum_{k=1}^{n} (s_i^k \times w_k) \tag{4.18}$$

Consistency computation

AHP provides consistency evaluations to check if the construction of the pairwise matrices and the previous calculation contains inconsistencies. Inconsistencies can be introduced by the decision maker in any pairwise matrix. For example, let's suppose we have 3 criteria. An inconsistency arises if the decision maker evaluates a first criterion to have moderate importance to the second criterion. Then, he evaluates the second criterion to have strong importance to the third criterion. However, he evaluates the third criterion to have equal importance to the first one. This pairwise comparison reduces the consistency of the AHP evaluation because by transitivity the first criterion should be evaluated to have very strong importance to the third one. Therefore, AHP proposes to estimate the consistency of its process by computing the Consistency Ratio (CR). The value of (CR) is compared with the consistency rule to evaluate the consistency of the AHP process. The details of the computations are as follows:

1. AHP calculates the ratio matrix R using the following equation:

$$r_i = \frac{\sum_{j=1}^{n} a_i^j \times w_j}{w_i} \qquad (r_i \in \mathrm{R},\ a_i^j \in \mathrm{A}\ and\ w_j \in \mathrm{W}) \tag{4.19}$$

2. AHP calculates the consistency index CI using the following equation:

$$\mathrm{CI} = \frac{(\mathrm{A}verage(\mathrm{R}) - n)}{(n-1)} \tag{4.20}$$

3. AHP calculates the consistency ratio CR using the following equation:

$$\mathrm{CR} = \frac{\mathrm{CI}}{\mathrm{RI}} \tag{4.21}$$

   The random consistency index has been normalized in many studies. It is defined according to the value of n. Many studies has been performed to estimate its values by running many simulations with a number of matrices. We choose the RI values of Alfonso & Lamata [253] (see table 4.4) because their computations are the most precise. They calculate their random consistency indexes using 5.10^6 matrices.

Table 4.4 – Alonso and Lamata RI values

n	1	2	3	4	5	6	7	8	9	10
RI	-	-	0,5247	0,8816	1,1086	1,2479	1,3417	1,4057	1,4499	1,4854
n	11	12	13	14	15	16	17	18	19	20
RI	1,5140	1,5365	1,5551	1,5713	1,5838	1,5978	1,6086	1,6181	1,6265	1,6341

4. The decision maker compares CR value with the consistency rule to judge the AHP consistency. If the AHP is inconsistent then the decision maker needs to correct the pairwise comparison and repeat all the AHP. The conditions of the rule are as following:

$$\text{Consistency Rule} = \begin{cases} \text{completely Consistent}, & \text{if CR} = 0 \\ \text{Acceptable Consistency}, & \text{if CR} <= 10\% \\ \text{Inconsistent}, & \text{otherwise} \end{cases}$$

4.4 SDN Asset Classification

We classifies SDN assets in Table 4.5 according to section 2.3 of chapter 2. Table 4.5 describes all the components of our system under study. The asset class indicates the family of the SDN asset and its location. It can be SDN applications (located in the application layer), controllers (located in control layer), network elements (located in data plane layer), SDN interfaces (located in the 3 previous SDN layers), managers and coordinators (located in the management layer). The asset components represent the SDN assets. They are the logical objects of SDN that need to be protected. Table 4.5 provides a description for each SDN asset. Besides, it instantiates each SDN assets with existing examples in SDN environments.

4.5 SDN Vulnerability Procedure

We propose a set of SDN generic vulnerabilities. We have built them by combining security threats with SDN assets. The identified vulnerabilities are generic because they are neither linked to the implementation of SDN assets nor any specific SDN environment. They are part of security by design because we introduce the vulnerability assessment into the design of SDN architecture rather in its implementation or deployment. Our objective is to identify SDN vulnerabilities in the earliest phases of SDN life cycle.

We have identified 120 (20×6) generic SDN vulnerabilities by applying a set of inverted security principals (see section 3.2 of chapter 3) to the assets of our system under study. We have performed the following steps:

1. Reverse the security principals (see Table 4.6).
2. Combine each reversed security object to all the identified assets.

Table 4.5 – SDN Assets

Asset Class	Asset Component	Description	Instanciation
Application	Application Function	It defines the operations and processes of network Service	SDN Firewall
	Application Content	It defines the information used and generated by applications	Firewall Policies
controller	Controller Content	It represents the control information needed by its operations	Libraries
	Controller Function	It represents the control operations	Traffic Monitor
	C-Agent	It offers control services to other entities	Ryu manager
	Controller RDB	It represents the control knowledge	Topology
Network element	Data Processing Engine	It is a set of functionalities that process data traffic	OVS Kernel Module
	Data Source	It delivers and transmits data traffic	Ports, Bridges
	Data Sink	It stores the network data	Buffers, Flow Tables
	Network Element RDB	It stores the data plane local knowledge	OVS DB
SDN Interface	A-CPI Agent	It manages the interactions of an application with the controller	REST Client
	A-CPI	It manages the interaction of the controller with applications	REST Server
	C-CPI	It manages the interactions between controllers	ForCES API
	D-CPI	It manages the interactions of the controller with network elements	Openflow-controller
	D-CPI Agent	It manages the interactions of the network element with the controller	Ovs-vswitchd
Manager	Management Function	It orchestrates management operations on SDN	Neutron
	Management Content	It contains administration information on SDN	Neutron DB
Coordinator	Application Coordinator	It manages administration operation in applications	Openstack plugins agents
	Controller Coordinator	It manages administration operation in controllers	Neutron plugins agents
	Network Element Coordinator	It manages administration operation in network elements	CLI, SNMP Agent

Table 4.6 – Reversion of security principals

Security object	Reversion	Description of Reversion
Access Control	Open Access	The asset can be accessed by any element without restriction
Authentication	Non identification	The asset lacks of identification in a distinctive way
Confidentiality	Non secrecy	The asset reveals its features and discloses its communications
Non-Repudiation	Repudiation	The asset can deny the actions that it has done and its behavior can not be traced to it
Integrity	Alterability	The asset can be tampered in part or entirely
Availability	Disruption	The asset can be temporally, partly or totally interrupted

As a result, we obtain Table 4.7. The first row of the table contains the reversed security principals. The first columns of the table contain the SDN assets. By combining both the rows and the assets, we obtained 120 generic vulnerabilities. Each cell starting from the second row and second column contain the ID of the vulnerability, the troubled asset, and potential security issue. For example, Vulnerability V36 is located in controller function, and it allows the attacker to perform disruption attacks on the asset.

Table 4.7 – SDN vulnerabilities

Threat / Asset	Open Access	Non identification	Non secrecy	Repudiation	Alterability	Disruption
Application Function	V11	V12	V13	V14	V15	V16
Application Content	V21	V22	V23	V24	V25	V26
Controller Function	V31	V32	V33	V34	V35	V36
Controller Content	V41	V42	V43	V44	V45	V46
C-Agent	V51	V52	V53	V54	V55	V56
Controller RDB	V61	V62	V63	V64	V65	V66
Data Processing Engine	V71	V72	V73	V74	V75	V76
Data Source	V81	V82	V83	V84	V85	V86
Data Sink	V91	V92	V93	V94	V95	V96
Network Element RDB	V101	V102	V103	V104	V105	V106
A-CPI Agent	V111	V112	V113	V114	V115	V116
A-CPI	V121	V122	V123	V124	V125	V126
C-CPI	V131	V132	V133	V134	V135	V136
D-CPI	V141	V142	V143	V144	V145	V146
D-CPI Agent	V151	V152	V153	V154	V155	V156
Management Function	V161	V162	V163	V164	V165	V166
Management Content	V171	V172	V173	V174	V175	V176
Application Coordinator	V181	V182	V183	V184	V185	V186
Controller Coordinator	V191	V192	V193	V194	V195	V196
Network Element Coordinator	V201	V202	V203	V204	V205	V206

Besides, the generated vulnerabilities are generalizations of all the inherent vulnerabilities that may occur on SDN assets. They can also be instantiated according to the environment and temporal variables of SDN. For example, vulnerability V36 generalizes the disruption vulnerabilities on controller functions. If we instantiate the latter according to the SDN environment of RYU controller; then, Traffic Monitor of RYU controller is the controller function. Therefore,

V36 can be instantiated to "unsupported exception event" in Ryu Traffic Monitor. For instance, an attacker can interrupt Ryu Traffic Monitor by generating an unsupported exception event during its execution.

4.6 Vulnerability severity results

First, we compute the severity of the proposed vulnerabilities (see Section 4.5) using the CVSS calculator 3.0 [254] and the descriptions provided by Table 4.1. Finally, we introduce into the results AHP to adapt them to SDN specific design features. We have chosen the options of the CVSS metrics according to the following assumptions:

1. In the base group, the vulnerabilities of the application and network element assets can come from the network and may require user interactions (metrics n#1 and n#5 of Table 4.2).

2. In the base group, the vulnerabilities of the controller can come from a limited vector (adjacent neighbors), do not require user interactions and need privileges (metrics n#1, n#3 and n#5 of Table 4.2).

3. In the base group, the vulnerabilities of the control layer are complex (metric n#2 of Table 4.2).

4. In the base group, the vulnerabilities of SDN interfaces and agents amplify the scope of any attack and do not require user interactions (metrics n#4 and n#5 of Table 4.2).

5. In the temporal group, Exploit Code Maturity is between Unproven and Proof-Of-Concept because there is not a threat code that works in any SDN implementation (metric n#7 of Table 4.2).

6. In the temporal group, the Remediation Level value is set to Official Fix when the deployment of TLS (Transport Layer Security protocol) [255] mitigates or prevents the exploit (metric n#8 of Table 4.2).

7. In the temporal group, the Report Confidence of D-CPI and D-CPI Agent is reasonable because the majority of security reports in the literature are focused on the interface between the control plane and the data plane (metric n#9 of Table 4.2).

8. In the environment group, we assume that TLS is deployed in the interfaces between the Control layer, the Data Plane layer and the Application layer. This security measure modifies the required privileges and complexity to high (metrics n#2 and n#3 of Table 4.2).

9. We assign different values to the metrics Confidentiality, Integrity and Availability according to the security objective of the vulnerability. For example, V35 (controller Function Alterability) impacts Integrity but does not affect Availability and Confidentiality (metrics n#6 and n#10 of Table 4.2).

4.6.1 Preliminary Vulnerability Scores

We compute the scores of the SDN vulnerabilities in the base, temporal and environment groups according to the equations in Section 4.3.1. We display the scores respectively in Figures 4.1a, 4.1b and 4.1c. The calculations are based on the assumptions as mentioned earlier. For example, we proceed according to the following logic to estimate the severity of V36 (Controller Function Disruption). V36 is exploited by attackers to perform denial of service on Controller

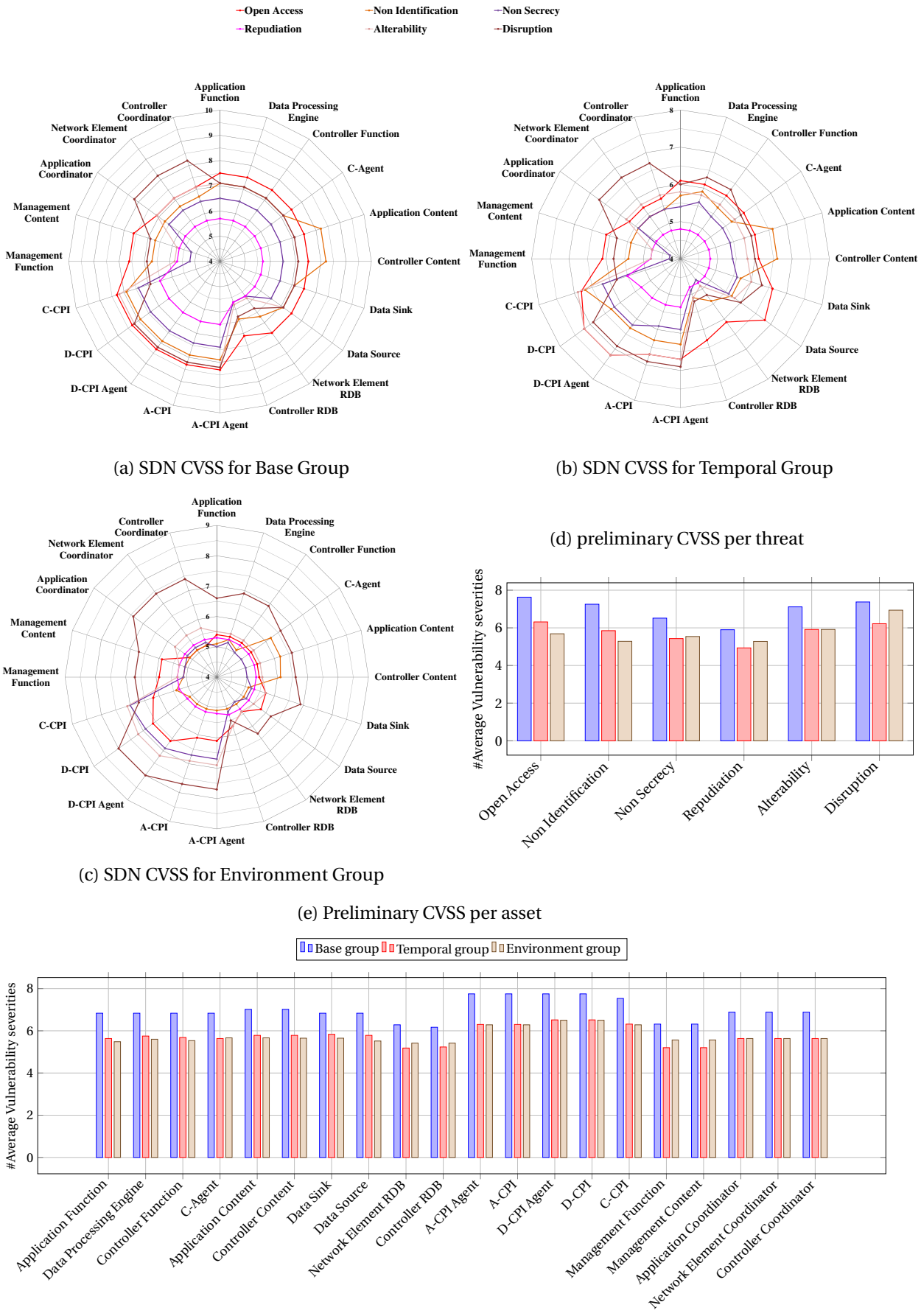
Function. We have seen in Section 3.3.5 of Chapter 3 that attackers flood controller functions with unknown traffic from network elements to exhaust the controller functions. Therefore, in the base group, the Attack Vector of the vulnerability is adjacent because the flooding source is Data Processing Engine. The complexity of the attack is low because the attacker misuses the reactive behavior of network elements and he can repeat the attack. He neither needs special privileges nor user interactions. Furthermore, this vulnerability impacts the availability of Controller Function highly, and in some cases, it can lead the target to alter the contents it receives and to interrupt the communications with other entities. As a result, the base group score of V36 is 7.1. There are proof-of-Concepts related to the exploit technique with a non-official solution and reasonable reports discussing this DoS Attack. Thus, the vulnerability score in the temporal group is 6.3. In the environment group, the TLS deployment modifies the privileges to Low. Thus, network elements need to open a secure channel before talking with the controller. As a consequence, the score in the environment group reduces to 6.9.

The preliminary average scores per asset are displayed in Figure 4.1e. The preliminary average scores per threat are displayed in Figure 4.1d. We observe that the vulnerability surface is between 8.3 (High) and 5.2 (Medium) in the base group spider diagram (Figure 4.1a). The different interfaces and their agents (A-CPI, A-CPI Agent, D-CPI, D-CPI Agent and C-CPI) have the highest scores because they expose other assets in different layers and enlarge the attack scope. Furthermore, the vulnerabilities related to Open Access and Disruption are the most severe. Their averages are respectively 7.9 and 7.2 in the base group.

In the Temporal group (Figure 4.1b) the vulnerability surface becomes between 7.2 and 4.3. This decrease is due to the unavailability of mature malicious code and confirmed attacker techniques (for almost all the vulnerabilities). As a result, the exploitation of the vulnerabilities is more difficult and expensive, especially in the Control layer. Besides, the majority of the vulnerability exploits (excluding Disruption and alterability in Application Logic and network element) are not reported.

The vulnerability surface becomes between 8.0 and 5.0 in the environment score spider (Figure 4.1c). However, it is not Open Access vulnerabilities (such as in the base group) that have the most severe impact. The Disruption vulnerabilities of the interfaces become the most severe; the SDN specification recommends the deployment of TLS in the southbound and the northbound interfaces. This deployment reduces the severity of Open Access vulnerabilities and those of Non-Secrecy, alterability, and Nonidentification (see Figure 4.1d). Besides, we note also that the vulnerabilities of C-CPI diverge from the other interfaces vulnerabilities scores in the temporal and environmental groups. The main reason is that C-CPI's security is unexplored and an untapped subject.

The results indicate a significant relation between SDN assets. We can see these relations in Figure 4.1e. In the Base group, the average vulnerabilities scores of controller Function, C-Agent, and controller Content are equal to the average vulnerabilities scores of Application Function, Application Content, Data Processing Engine, Data Sink and Data Source (there are minor disparities in the temporal and environment groups). The same equality relation is observed between the different interfaces in the control layer and their respective agents in the application layer and the data plane layer. For example, another target of the DoS attack is the OpenFlow tables (Data Sink). Controller Function answers with Flow rules and packet-out. Data Processing Engine processes them and saves them in OpenFlow tables. However, because flow table size is limited, the tables can overflow. As a result, Data Processing Engine's performance reduces, and it rejects the new OpenFlow rules (including those for legitimate traffic). In this scenario, the DoS attack abuses the limited size of OpenFlow tables which corresponds to V96 (Data Sink Disruption). The scores of this vulnerability in the three groups (respectively 7.1, 6.3 and 6.9) equal the scores of V36; however, there is an issue with this equality. The DoS attack on Controller Function disturbs the entire network and enlarges the impacts to other as-



(a) SDN CVSS for Base Group

(b) SDN CVSS for Temporal Group

(c) SDN CVSS for Environment Group

(d) preliminary CVSS per threat

(e) Preliminary CVSS per asset

Figure 4.1 – Preliminary CVSS Results

sets; network elements, SDN applications, and controllers will experience considerable answer delays (and even communication interruptions) from the affected controller. At the same time, the second attack on the network element has a restricted scope.

The severity of the controller vulnerabilities should be higher than the other assets. The reasons for this equality issue are related to the function of the CVSS method. It focuses only on the characteristics of conventional network systems. It neither takes into consideration the specific features of SDN nor the importance of each SDN asset towards the others. Therefore we need to enhance the CVSS to take into account the design characteristics of SDN. By improving the CVSS scores, we will adapt CVSS to SDN architecture and obtain more accurate scores reflecting SDN features.

4.6.2 Enhanced Vulnerability Scores

We have discussed in Section 3.3.1 of Chapter 3 the impacts of SDN specific features on its security. In fact, we think that SDN architecture enlarges the attack surface because its features increase the severity of its assets vulnerabilities. As a result, these vulnerabilities should be different depending on their weights in SDN and how its features impact them. Therefore, we need to enhance the CVSS computations by integrating into preliminary scores the weights of SDN assets. We use AHP to evaluate these weights. The goal of our decision making is to find the most important asset that is influenced by SDN features. The first dimension of our AHP is the SDN features. They are our criteria because they define SDN and influence its assets. In this dimension, we use AHP to compute the impacts of SDN features on its architecture. The second dimension is the SDN assets. They are our alternatives because each asset can be one choice among others to reach our goal. This dimension measures the importance of each asset inside each SDN feature. After these computations, we combine the final weights into the preliminary vulnerability scores to quantify the new CVSS. We undertake the following steps:

1. We describe the criteria and alternatives of our analysis.
2. We calculate the weights of each SDN asset using the steps of AHP (see Section 4.3.2).
3. We enhance the CVSS scores by integrating the AHP results into the calculations of CVSS.

SDN criteria and alternatives

The first level of our hierarchy (the criteria) represents the SDN features that affect SDN vulnerabilities. The second level (the alternatives) represents the SDN assets (see Table 4.5). The criteria of our analysis are programmability, centralization, federation, and externalization. We have described them in Section 2.2 of Chapter 2.

Programmability (see Section 3.3.4 of Chapter 3) increases the vulnerabilities of SDN assets because it enables attackers to automatize their threats, to adapt them to the evolution of the network and to spread them dynamically and widely to other assets. Centralization (see Section 3.3.2 of Chapter 3) defines the density of assets links and their reliance. A centralized asset is critical, and its vulnerabilities spread to all other assets because it increases logical dependence between assets. Federation (see Section 3.3.3 of Chapter 3) enables an attacker to extend their exploits to all SDN assets that rely on the same interface to interact. It reduces the complexity of vulnerabilities and eases their generalization and portability to other SDN environments. Externalization (see Section 3.3.1 of Chapter 3) opens and exposes SDN assets by breaking the legacy lock-ins. It enables the design of modular SDN assets that can be directly accessed, identified and studied. Thanks to externalization, an attacker can access and study SDN assets to identify their vulnerabilities without passing by the complexity that was prominent in the legacy architecture.

SDN AHP results

We apply in this step the AHP method to calculate the weights of SDN assets according to SDN features. We have determined the weights of the criteria according to the following assumptions:

1. **Programmability**: its vulnerabilities are local, but they have major impacts when they are spread through federation and centralization. An attack on the programmability will affect the functions of the layer and all the assets that depend on them. If SDN programmability is down, the operations of the control layer, application layer, and management layer stop.

2. **Centralization**: its vulnerabilities are the most global. An attack on the centralized controller will affect and will spread to all other assets. If centralization is down in the controller, all the SDN will be down.

3. **Federation**: its vulnerabilities impacts can spread to all the assets that have the same interface. An attack on the interfaces will affect all the communication of the assets that share common interfaces. If an SDN interface, all the assets that use the interface will not be able to interact.

4. **Externalization**: its vulnerabilities are local to each asset. If externalization is attacked in an asset, the attack affects only the asset; However, in the case of distributed controllers, the attacker will be able to exploit the same vulnerability in other controllers. Other SDN assets can work if externalization of an asset is attacked.

We have determined the weights of the alternatives in each criteria dimension according to the following assumptions:

1. **Programmability**: Controller Function and Application Function have equal importance because both of them aim to program the network elements. Besides, the controller offers application functions as Control Function and programmable frameworks and languages as C-Agent. Management alternatives are also important because they can configure other alternatives using simple programmable services.

2. **Centralization**: The control layer alternatives are the most important compared to other alternatives. Also, Controller Function, Controller Content, Controller RDB and C-Agent are more important than the rest of the control layer alternatives because the latter can be more distributed and duplicated (depending on the type and number of interfaces). Network element alternatives are the less important. Management layer alternatives are also important (including Controller Coordinator).

3. **Federation**: All the interfaces alternatives are the most important. We gave to Controller Function and Controller Content moderate importance because we think that they need to be standardized and all the controllers have almost the same Controller Function and Controller Content. They differ only in their implementations. The rest of the assets are less important for federation because they are technology proprietary and heterogeneous.

4. **Externalization**: all the alternatives have equal importance except for the control layer alternatives which are equally to moderately important because SDN focuses more on separating the control layer from the network element. Besides, externalization vulnerabilities affect the control layer when it is distributed.

The details of the computations are as follows :

1. **Weighting the criteria**: We construct the pairwise matrix A(4X4) between the ordered set of criteria C as follows:

   C = [P$rogrammability$, C$entralization$, F$ederation$, E$xternalization$].
   The matrix A is ordered according to C, its values are as follows:

$$A = \begin{pmatrix} 1 & 1/3 & 3 & 4 \\ 3 & 1 & 5 & 7 \\ 1/3 & 1/5 & 1 & 2 \\ 1/4 & 1/7 & 1/2 & 1 \end{pmatrix}$$

   The pairwise comparison has been performed according to the following comparisons (see table 4.3):

   - Programmability is moderately more important than federation and moderately to strong important than externalization.
   - Centralization is moderately more important than programmability. It is strong important than federation and it is very strong important than externalization.
   - Federation is equally to moderately important than externalization.
   - Externalization is the least important.

   We calculate the normalized matrix B and the weight vector W:

$$B = \begin{pmatrix} 0.2182 & 0.1989 & 0.3158 & 0.2857 \\ 0.6545 & 0.5966 & 0.5263 & 0.5000 \\ 0.0727 & 0.1193 & 0.1053 & 0.1429 \\ 0.0545 & 0.0852 & 0.0526 & 0.0714 \end{pmatrix}$$

$$W = \begin{pmatrix} 0.2546 \\ 0.5694 \\ 0.1100 \\ 0.0660 \end{pmatrix}$$

   We calculates the ratio matrix R, the consistency index CI and the consistency ratio CR (n=4 and RI=0.8816):

$$R = \begin{pmatrix} 4.077887108 \\ 4.118970711 \\ 4.004942058 \\ 4.032489718 \end{pmatrix}, \quad CI = \frac{Average(R) - 4}{3} = 0.019524133 \text{ and } CR = \frac{0.019524133}{0.8816} = 0.02$$

   The consistency ratio is 2% which means that our computations have acceptable consistency.

2. **Scoring the alternatives**: We construct the pairwise matrices $A_P(20 \times 20)$ for programmability, $A_C(20 \times 20)$ for centralization, $A_F(20 \times 20)$ for federation and $A_E(20 \times 20)$ for externalization. The matrices compare the ordered alternatives (assets) in each criterion dimension. The alternatives are extracted using the same order as the first column of table 4.3.1. The order starts from Application Function and ends with Network Element Coordinator. The details of all the computations are in Appendix .1. The weights of the alternatives are respectively W_P, W_C, W_F *and* W_E.

3. **Consolidation of the alternatives with the criteria**: We compute the final weights V:

$$S\begin{pmatrix} W_P & W_C & W_F & W_E \end{pmatrix} X\ W = V.$$

$$\begin{pmatrix}
0.094651944 & 0.019938648 & 0.015873573 & 0.037037037 \\
0.02574031 & 0.019938648 & 0.015873573 & 0.037037037 \\
0.094651944 & 0.110115604 & 0.050805513 & 0.074074074 \\
0.026711184 & 0.103426694 & 0.03166262 & 0.074074074 \\
0.105693977 & 0.123871379 & 0.093410539 & 0.074074074 \\
0.028311184 & 0.100756297 & 0.029226164 & 0.074074074 \\
0.018140808 & 0.017748509 & 0.017280425 & 0.037037037 \\
0.018140808 & 0.017748509 & 0.017280425 & 0.037037037 \\
0.018140808 & 0.017748509 & 0.017280425 & 0.037037037 \\
0.018140808 & 0.018989245 & 0.017280425 & 0.037037037 \\
0.069244285 & 0.020074889 & 0.093365575 & 0.037037037 \\
0.069244285 & 0.061994522 & 0.093365575 & 0.074074074 \\
0.082520136 & 0.061994522 & 0.093365575 & 0.074074074 \\
0.060385272 & 0.061994522 & 0.093365575 & 0.074074074 \\
0.060385272 & 0.020074889 & 0.093365575 & 0.037037037 \\
0.048357688 & 0.065478606 & 0.011979719 & 0.037037037 \\
0.024178844 & 0.063350946 & 0.011770514 & 0.037037037 \\
0.045786814 & 0.018951013 & 0.067816068 & 0.037037037 \\
0.045786814 & 0.056853039 & 0.067816068 & 0.037037037 \\
0.045786814 & 0.018951013 & 0.067816068 & 0.037037037
\end{pmatrix} \times \begin{pmatrix} 0.2546 \\ 0.5694 \\ 0.1100 \\ 0.0660 \end{pmatrix} = \begin{pmatrix}
0.0396 \\ 0.0221 \\ 0.0973 \\ 0.0741 \\ 0.1126 \\ 0.0727 \\ 0.0191 \\ 0.0191 \\ 0.0191 \\ 0.0198 \\ 0.0418 \\ 0.0681 \\ 0.0715 \\ 0.0658 \\ 0.0395 \\ 0.0534 \\ 0.0460 \\ 0.0324 \\ 0.0539 \\ 0.0324
\end{pmatrix}$$

Each line of the matrices S and W corresponds to an element of the collection A*l* with respect of the same order:
A*l* = (A*pplication* F*unction*, A*pplication* C*ontent*, *controler* F*unction*, *controler* C*ontent*, C−A*gent*, *controler* RDB, D*ata*P*rocessing* E*ngine*, D*ata* S*ource*, D*ata* S*ink*, N*etwork* E*lement* RDB, A − CPI A*gent*, A − CPI, C − CPI, D − CPI, D − CPI A*gent*, M*anagement* F*unction*, M*anagement* C*ontent*, A*pplication* C*oordinator controler* C*oordinator*, N*etwork* E*lement* C*oordinator*)

We calculate the ratio matrices for each dimension, the consistency index CI and the consistency ratio CR (n=20 and RI=1.6341):

$$CR_P = \frac{0,052865064}{1.6341} = 0.03 \quad CR_C = \frac{0,024413925}{1.6341} = 0.01$$
$$CR_F = \frac{0.018688237}{1.6341} = 0.01 \quad CR_E = \frac{0}{1.6341} = 0$$

The consistency ratios are less or equal to 3% which means that all our computations have acceptable consistencies.
We observe in vector V that control layer assets have the highest weights (0.1126 for C-Agent, 0.0973 for controller Function, 0.0741 for controller content, 0.0727 for controller RDB, 0.0715 for C-CPI, 0.0681 for A-CPI and 0.0658 for D-CPI) because SDN features impact more the control layer assets. The controller agent and function have the highest weights because they are the most affected by all the criteria. C-CPI has a greater weight than the other controller interfaces because in programmability , it has more important weights. This interface enables an attacker to reach all the SDN layers. In contrast, Network Element assets have lower intensities (0.0191 for each one of them) because SDN criteria have lower impact on them (except for D-CPI Agent). Application Function has a higher weight (0.0396) than network element assets because it is more impacted

by programmability. Management function and controller coordinator have also higher weights than the application assets because they are impacted by centralization and programmability.

Integration of the AHP weights

We calculate the new vulnerability severities by integrating the weights of V into CVSS scores. For each alternative, its weight is multiplied by upper bound of the CVSS to normalize the weights according to CVSS. Then, these values are added to the previous vulnerability scores of the alternative. We use the following equation for each alternative i:

$$\text{CVSS}'_i = \text{CVSS}_i + (10 \times v_i) \tag{4.22}$$

We limit the new values in the original CVSS interval [0 , 10] to avoid the score to exceed the maximum authorized value (10). We use the following equation:

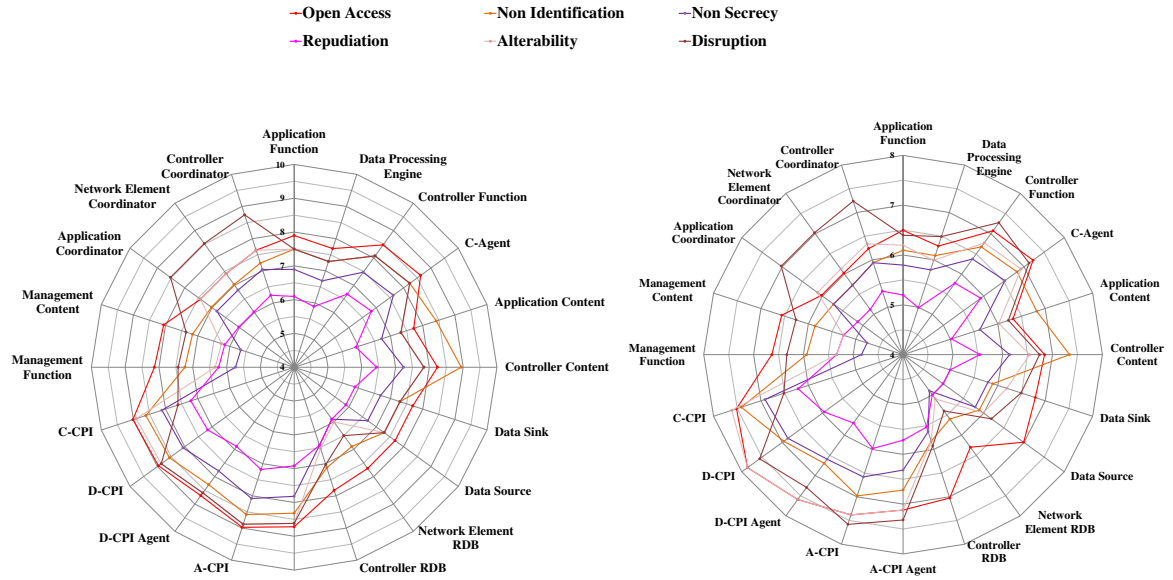
$$\text{CVSS}''_i = \text{M}in(\text{CVSS}'_i, 10) \tag{4.23}$$

The enhanced results are displayed in Figures 4.2a, 4.2b, and 4.2c. The average enhanced scores per asset are displayed in Figure 4.2e. The average enhanced scores per threat are displayed in Figure 4.2d. In contrast to the previous CVSS results, we observe that the vulnerability surface increases in the 3 groups. It now ranges in the base group between 9 and 5.7 (before between 8.3 and 5.2), in the temporal group between 7.9 and 4.8 (before between 7.2 and 4.3) and in the environment group between 8.7 and 5.2 (before between 8 and 5). This increase is due to the impact of SDN features which enlarge the vulnerability surface. Besides, Open Access stands with the highest average score (8.1) in the base group and in the temporal group (6.8) while Disruption has the highest average severity (7.4) in the environment group (see Figure 4.2d).

Furthermore, we observe important changes in the severity values compared to the previous results; the control layer vulnerabilities scores raise due to the introduction of AHP. We see these increases in Figure 4.2e. They exceed their counterparts in the other layers while in the previous results they had similar or smaller values. For example, in the 3 groups, the average scores of C-Agent(8, 6.8, 6.8), Controller Function (7.8, 6.6, 6.5) and Controller Content (7.8, 6.5, 6.4) exceed those of Application Function (7.3, 6, 5.9), Application Content (7.2, 6, 5.9), Data Processing Engine (7, 5.9, 5.8) and Data Sink (7, 6, 5.8). Another example of the changes is in Open Access between Controller Function (moved from 7.5 to 8.5) and Data Processing Engine (moved from 7.5 to 7.6).

In the preliminary CVSS results, the interface assets were the most impacted, but they had almost equal scores. The interfaces enable an attacker to spread its attack to other assets. In the enhanced CVSS results, the leading average CVSS scores belong to the interface assets of the controller which are: D-CPI (8.4, 7.2, and 7.2), A-CPI (8.4, 7, and 7), and C-CPI (8.3, 7, and 7). They exceed the scores of A-CPI Agent (8.2, 6.7, 6.7) and D-CPI Agent (8.1, 6.9, 6.9). The leading scores are followed by the enhanced scores of the rest of the controller assets.

The CVSS scoring enhancement breaks the equality relation between the control layer assets and the other assets in the network element and the application layer. For example, let's suppose an attacker that performs a DoS Attack on SDN. He can exploit four vulnerabilities which are C-Agent Disruption, Controller Function Disruption, Data Processing Engine Disruption and Data Sink Disruption. In the previous CVSS results, they have equal values in the base groups (7.1) and slight differences in the other groups, although they belong to two different layers. Thanks to CVSS enhancement, we resolve this incoherence by incorporating their weights to the preliminary results. Hence, the new scores shifted for C-Agent Disruption to (8.2, 7.1, 7.7), Controller Function Disruption to (8.1, 7.3 and 7.9), for Data Processing Engine Disruption to (7.3, 6.5, 7.1) and Data Sink Disruption to (7.3, 6.5 and 7.1).

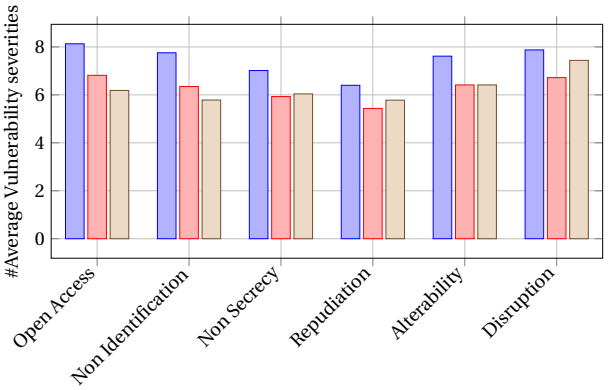


(a) Enhanced SDN CVSS for Base Group

(b) Enhanced SDN CVSS for Temporal Group

(c) Enhnaced SDN CVSS for Environment Group

(d) Enhanced CVSS per threat



(e) Enhanced CVSS per asset

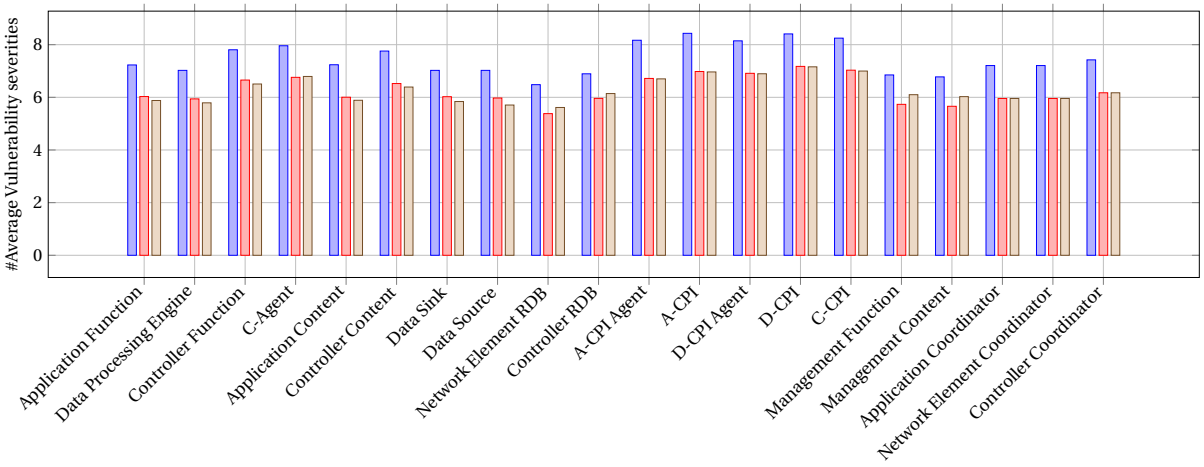


Figure 4.2 – Enganced SDN CVSS Scores

4.7 Discussion

In this chapter, we have analyzed the vulnerabilities of SDN. We have constructed a list of generic SDN vulnerabilities by inverting security objectives and combining them with SDN assets. We have used the CVSS model to compute the severities of these generic vulnerabilities. The preliminary CVSS results have shown inconsistencies because their computations do not take into account SDN specific features. Therefore, we integrate to the preliminary results the intensities of specific SDN features based on AHP to adapt CVSS according to SDN.

Our findings indicate that SDN has many vulnerabilities with high and medium severities because of the weaknesses inherited from classical network architecture and due to its specific characteristics. We have shown that CVSS is agnostic to SDN specific features. It assigns equal scores to the different assets. We resolve this issue by integrating AHP to CVSS and adapting the latter to SDN specific features. As a result, the SDN vulnerability surface has increased, and the control layer vulnerabilities become the most severe. Besides, In both preliminary and enhanced results, vulnerabilities related to Open Access are the most severe in the base group, while the severities of disruption are the highest in the environment group.

SDN needs to be protected from access and disruption vulnerabilities. In the next chapter, we propose an SDN stateful firewall. We use the advantages of SDN for security to improve firewalls (see section 3.4 of chapter 3). Besides, we take into account the vulnerability analysis results in the design of our SDN firewall to protect SDN assets from attacks that exploit disruption vulnerabilities.

Part III

SDN for Security

Chapter 5

Centralized SDN Firewall

" Use a personal firewall. Configure it to prevent other computers, networks and sites from connecting to you, and specify which programs are allowed to connect to the internet automatically. "

Kevin Mitnick

Contents

5.1 Introduction

We develop an SDN stateful Firewall. We integrate SDN concepts into legacy firewalls to improve their behavior. We have two objectives by making SDN firewalls. On the one hand, we want to prove that the softwarization of firewalls by SDN is advantageous for network security; it brings the de-perimeterization of security. Thus, we incorporate to the design of legacy stateful firewalls SDN concepts such as programmability, federation, externalization, and centralization. Then, we measure the effects of SDN automation on firewalls; we compare the performance of our SDN firewall against the performance of a well-known legacy firewall in many scenarios. On the other hand, we introduce into our SDN firewall a mechanism to protect SDN from disruption attacks that use vulnerabilities described in Chapter 4.

We automatize the firewall application by programming its behavior using OpenFlow. The SDN firewall elevates the behavior of network elements by transforming them into firewall devices. As an effect, the firewall logic pervades along the network. The network elements execute the firewall behaviors according to OpenFlow rules. These behaviors are programmed through the controller using its holistic view and OpenFlow. Furthermore, at the level of the application plane, the firewall application makes sure that the firewall logic fulfills the security policies. Our solution is part of the European project TANDEM SENDATE [235, 236]. It has many advantages. It saves the expenditures related to traditional firewalls. It is re-programmable with fewer errors. It can be deployed in the network elements.

To reach our goal, we first discuss traditional firewalls (Section 5.2) and our motivations for SDN firewalls (Section 5.3). Then, we introduce the main ideas of our proposal and its conceptual foundations (Section 5.4). We implement our proposition using python language to prove our concept (Section 5.5). Finally, we evaluate the performance of our solution in different test scenarios (Section 5.6). We evaluate it against a legacy firewall (Netfilter) and under DDoS attacks to demonstrate its advantages. We highlight the cases where it has better performance compared to legacy firewalls.

5.2 Conventional Firewalls

A legacy firewall is a mechanism that enforces the access control of a trusted network (internal network domain) from external access. It accepts, denies or quarantines network traffic in the perimeter of an internal network according to a set of security policies [256, 257]. The legacy firewall considers the network in its perimeter as secured because it assumes that the potential unauthorized accesses come only from outside the perimeter. The network traffic can compounds the following concepts:

1. **Network Packet**: a packet is a structured record that can be transported in the network. It compounds a payload (the useful data of the user), and the protocol specifications (header and footer). The latter specifies the properties of the packets according to network protocol specifications:

$$
\begin{aligned}
&\mathrm{P}acket = (\mathrm{T}ype, \mathrm{V}alue)\\
&\forall i, \mathrm{P}acket_i = (\mathrm{T}ype, \mathrm{V}alue)_i \, and \cup \mathrm{P}acket_i = \mathrm{P}acket\\
&\mathrm{T}ype = \{\forall t, t \in \mathrm{H}eader\_Fields \cup \mathrm{P}ayload \cup \mathrm{F}ooter\_Fields\}\\
&\mathrm{V}alue = \{\forall v, v \in \zeta \, and \, \zeta \, is \, the \, set \, of \, the \, alphanumerical \, values\}
\end{aligned}
\tag{5.1}
$$

Where:

$$\begin{aligned} &Header\_Fields = Source\_Fields \cup Destination\_Fields \cup Other\_Fields \\ &\{IP_{src}, PORT_{src}\} \in Source\_Fields \\ &\{IP_{dst}, PORT_{dst}\} \in Destination\_Fields \\ &\{ETH\_Type, IP\_Proto, TCP\_Flags\} \in Other\_Fields \\ &\{User\_Data\} \in Payload \\ &\{CRC\} \in Footer\_Fields \end{aligned} \tag{5.2}$$

2. **Network stream**: It is a set of unidirectional packets that share their common header fields such as ETH_T*ype* (Network Protocol Type), IP_P*roto* (Network Protocol Code), IP_S*ource*, IP_D*estination* and some other fields. A stream that contains k packets is defined as follows:

$$Stream = Packet_1 \cup ... \cup Packet_k \tag{5.3}$$

Where:

$$\begin{aligned} &Source\_Fields_1 = ... = Source\_Fields_k \\ &Destination\_Fields_1 = ... = Destination\_Fields_k \\ &Other\_Fields_1 \cap ... \cap Other\_Fields_k \neq \emptyset \end{aligned} \tag{5.4}$$

3. **Network Connection**: It is a set of bidirectional related streams. A combination between an outbound stream and its corresponding inbound stream:

$$Connection_k^l = Stream_k \cup Stream_l \tag{5.5}$$

Where:

$$\begin{aligned} (Source\_Fields_k = Destination\_Fields_l) \wedge (Destination\_Fields_k = Source\_Fields_l) \\ \wedge (Other\_Fields_k \cap Other\_Fields_l \neq \emptyset) \end{aligned} \tag{5.6}$$

4. **Connection State**: it describes the configuration of the connection according to the values of its components.

A security policy consists of a set of filtering rules [258]. Each filtering rule consists of a number of criteria with a corresponding action. The firewall matches the packets with the filtering rules as follows:

- Π is the set of policies:

$$\Pi = \{P_1, P_2, ..., P_n\}\ ,\ (n >= 1) \tag{5.7}$$

- A policy P_i is defined as follows:

$$\forall P_i \in \Pi,\ P_i = \{r_1, r_2, ..., r_m\}\ ,\ (m >= 1) \tag{5.8}$$

- Each rule r_j is defined by a set of matching fields and an action as follows:

$$\begin{aligned} &R = \{r_j, \forall j\} \\ &r_j = Matching\_Fields \cup Action \end{aligned} \tag{5.9}$$

- M*atching_Fields* are packet information that correspond to network protocols specifications: M*atching_Fields* = (T*ype*,V*alue*) . A*ction* specifies the behavior of the firewall on the packet:

$$Action = accept \vee Deny \tag{5.10}$$

- A firewall matches a filtering rule on a packet. Then, it applies the corresponding action as follows:

$$r_j \Vdash \mathrm{P}ac\mathit{ket}_k \Leftrightarrow ((\forall \mathrm{T}y\mathit{pe} \in \mathrm{M}at\mathit{ching\_F}i\mathit{elds} \wedge \forall \mathrm{T}y\mathit{pe} \in \mathrm{P}ac\mathit{ket}_k), \\ (\mathrm{T}y\mathit{pe}, \mathrm{V}al\mathit{ue})_j = (\mathrm{T}y\mathit{pe}, \mathrm{V}al\mathit{ue})_k \Rightarrow \mathrm{A}p\mathit{ply}(\mathrm{A}c\mathit{tion})) \tag{5.11}$$

There are six categories of firewalls [259–261]:

1. **Bridge Firewalls**: they operate on the data link of the OSI model. They filter $\mathrm{P}ac\mathit{ket}_k$ based on its $\mathrm{H}ea\mathit{der\_F}i\mathit{elds}_k$ that belongs to the data link layer such as MAC addresses, physical ports, and other information. Bridge firewalls can be placed everywhere in switches; however, they can not secure network access. Their Π cannot control the access correctly. For example, if a policy accepts the packets of a $\mathrm{S}our\mathit{ce\_M}ac = x1$, then all the connections that contain x1 will be allowed to pass including the illegitimate connections. The attacker can forge bogus streams with x1 without being denied. In the opposite case, if the policy denies the packets of a $\mathrm{S}our\mathit{ce\_M}ac = x1$, then all the connections that contain x1 will be rejected including the legitimate one.

2. **Stateless firewalls**: they process $\mathrm{S}tr\mathit{eam}_k$ and $\mathrm{S}tr\mathit{eam}_l$ on network level according to Π. They filter $\mathrm{P}ac\mathit{ket}_k$ and $\mathrm{P}ac\mathit{ket}_l$ based on their $\mathrm{H}ea\mathit{der\_F}i\mathit{elds}_k$ and $\mathrm{H}ea\mathit{der\_F}i\mathit{elds}_l$ that belongs to the network layer such as IP addresses, Virtual ports, and other information. They neither filter $\mathrm{C}on\mathit{nection}_k^l$ nor track its state, nor manage the dynamic network information such as dynamic port allocation. As a result, an attacker can forge bogus packets belonging to the streams without being denied.

3. **Stateful firewalls**: in addition to perform stateless filtering, they operate on the transport level. They process $\mathrm{C}on\mathit{nection}_k^l$ and they track its $\mathrm{S}ta\mathit{te}_{\mathrm{C}on\mathit{nection}_k^l}$ according to Π.

4. **Circuit Level Gateways**: they use a stateful firewall and network proxies to secure the communications between hosts. The stateful firewall enables the gateway to verify the legitimacy of a connection in the handshaking phase. The proxy mechanism enables the protection of the internal host. After validating the synchronization step between the external and internal host, the gateway creates a session separately for each host. Communications between the two hosts go through this circuit. The gateway maintains a connection table that contains the information about the connection such as its IP addresses, ports and connection states. Then, it checks incoming packets against the entries of the table. The gateway relays the packets from the origin session to the destination session, if a packet belongs to a connection in the table. It drops it in the opposite case.

5. **Application Level Gateways**: they are software packages that are installed on servers to protect them from malicious code or bad data. These gateways process application records in $\mathrm{P}ac\mathit{ket}_k$ that belong to the application layer of the OSI model). The purpose of the processing is to determine if the packets are permitted to access the resources of servers. Besides, they control the connections between servers and their clients. They receive the requests from the clients. Then, they perform deep packet inspection on the packets. If they do not detect anomalies, they accept the packet to reach the service. Deep packet inspection extends stateful firewalls. In addition to process connection states, they also inspect the payloads of packets on the application layer to detect malicious code and bad data.

6. **Stateful multilayer inspection firewalls**: they combine the 5 firewall types. They use bridge firewall, a stateless firewall, a stateful firewall, circuit level gateways and application level gateways at each corresponding OSI level. If the packet passes all the firewall

levels; it is permitted to reach its destination. Otherwise, it is dropped or quarantined. The advantage of these firewalls compared to stateful firewalls is their ability to dynamically open and close ports during a connection while stateful firewalls can not. However, they are costly because they require a dedicated hardware and software [262].

5.3 Motivation for an SDN Firewall

We propose an SDN stateful firewall to enforce the network access control. Our primary motivation is to introduce SDN benefits for security into the design of legacy firewalls to improve them. SDN firewalls offer many advantages thanks to the simplicity, agility, global knowledge, interoperability, automation, and orchestration (see Section 3.4 of Chapter 3). They are cost effective because they enable the evaluation of the data plane with the firewall behaviors. They are also flexible since the controller can at any time update their rules according to the network state. They offer a management function for administrators that simplify their tasks. They enforce the firewall policies in the data plane using the holistic knowledge of SDN. Finally, they perform the de-perimeterization of security by pervading firewall rules in all the network elements next to packets sources with different granular levels of security.

5.3.1 Firewall expenditure reduction

Legacy network firewalls are expensive; the Total Cost of Ownership (TCO) of a legacy firewall can reach a price around 1000 k$ [263]. Furthermore, legacy firewalls do not always run on commodity equipment. A customer needs to purchase dedicated hardware that runs the firewall. Besides, legacy firewalls are proprietary technologies. Their upgrades depend on their vendor. The owner can not access or modify their source code because it is locked. As a result, the prices of these firewalls and their maintenance are high.

We think that an SDN firewall enables a company to save the money related to the expenditures of a legacy firewall because buying a dedicated hardware firewall is no more needed. In fact, SDN firewalls deploy their filtering rules on network elements. As a result, these elements behave like firewalls by matching the filtering rule with packets. Besides, SDN controllers are open-source and enable the development of SDN applications that interact with them via the northbound API. Thus, an SDN firewall application can be developed in the application layer. Then, IT uses the controller to deploy its policies on network elements. Its maintenance can be performed without affecting the code of the controller. What is more, the upgrade of the controller code does not also affect the firewall application.

The ability of SDN to turn network elements to firewalls, automatize the deployment of firewall policies and control them from a software (firewall application) has another advantage; SDN firewalls pervade security everywhere in the network close to the hosts. They filter the packet on different granular levels that depend on the capacity of packet classifiers; each port of a network element can be transformed into an ingress firewall to filter the packets between network links, internal hosts, and nodes. As a result, each device connected to a network element port will have its dedicated firewall. For example, OpenFlow network elements enable bridge and stateless firewalls. The pervasiveness of firewall in the data plane layer improves the performance because it distributes the traffic loads on all network elements while in the legacy network the load is concentrated in a centralized firewall that becomes a bottleneck.

5.3.2 Enforcement of security policies in the whole network

Due to their rigid and hardware structure, traditional firewalls are hard to manage and complex to deploy. Besides, they are error-prone because of misconfigurations, overload and hardware

failures. More the number of legacy firewalls increases more it is complicated and burdensome to manage them all. Besides, they are proprietary appliances. They are agnostic to the intern network communications because, most of the time, they are located on the network edge. Therefore, they neither handle ingress packets between the internal links nor they can coordinate their operations. This coordination enables them to collaborate and avoid errors due to racing and policy conflicts. For example, packets that have been previously processed by a legacy firewall can be again re-processed many times by other legacy firewalls that are on the inter-domain path of these packets until they reach their destinations. Another can reject packets that have been authorized by a legacy firewall. The duplication of processing and policy inconsistencies affect network performance and security.

The other concern is related to the deployment of legacy firewalls. What is the best place to deploy them? How can we migrate and adapt them when the network changes? Thus, these questions introduce more complexity and costs in the network again.

SDN firewalls bring vital solutions to the issues mentioned above. The holistic view of the controllers reinforces firewall policies. The controller verifies the consistency of the policies, corrects them, interprets them dynamically as filtering rules and installs them on the network elements. Also, SDN automatizes the management of firewall policies. The controller adapts them dynamically according to network changes. As a result, human intervention, errors and misconfigurations are reduced. Furthermore, thanks to the holistic view, the controllers can at any time distribute the load between the firewalls to keep network performance and prevent overloading a firewall. Administrators no longer need to maintain the policies manually. They push them once, and the controllers interpret, deploy and adapt them according to the network context. Thus, the security perimeter progressively pervades all the system while producing into the network a complete erosion of the security policies [264].

5.3.3 Security de-perimeterization

Many known computer attacks are instigated by insiders that compromise the security of the system [265] because legacy firewalls are based on the concept of perimeter security. They are founded on the assumption that the internal network is trusted and the attacks can only come from outside the trusted perimeter. However, this assumption is wrong because insider attackers compromise access policies inside the perimeter. Legacy firewalls do not counter insiders that abuse authorized access because they can not filter their packets [266].

There are two possibilities to resolve this issue with firewalls. The first one is to forward all the internal packets to the centralized firewall to filter them. Then, the firewall recirculates the permitted packets to the internal network. However, this solution has many issues; the centralized firewall becomes a bottleneck and a single point of failure. It decreases the QoS of the network. The second solution is to use distributed firewalls inside the perimeter. Many firewalls are added in some locations. The internal packets are distributed to them. The drawbacks of this solution are related to the cost of the armada of firewalls, their complex management, their interoperability, their impacts on network performance and the conflicts between their policies.

SDN enables firewalls to filter insider packets next to their sources because it transforms each network element to a firewall. Besides, thanks to the holistic knowledge of the controller, it corrects policies conflicts. It reinforces policies into filtering rules that control the access on different granularity levels. It does not add new hardware because it uses network elements as firewalls by deploying in them the filtering rules. Besides, the controller can share information between the different SDN firewalls. For example, if an insider is blocked on a network element by a firewall application because of an attempt of an attack (such as SYN flooding). This information is shared on all the rest of the network elements which, in turn, block the insider.

5.4 Key Concepts

In this section, we describe in details all the aspects of our SDN stateful Firewall. We discuss its design foundations, its architecture and its algorithms. Also, we give an example of our firewall with TCP (Transmission Control Protocol) [267]. We analyze its performance and scalability theoretically.

5.4.1 General Concepts

The SDN stateful firewall is integrated into the SDN architecture. The firewall application runs in the application layer. It expresses the logic of the firewall and interprets the firewall policies into SDN rules. The controller ensures the installation of the firewall rules as OpenFlow rules in the network elements. These rules express the firewall policies according to OpenFlow. Besides, the firewall application enables the user to express its policies without worrying about their installation and maintenance. Above the firewall application, we propose an orchestrator in the management layer. The orchestrator provides a global view of the network to the administrator. It centralizes all the firewall policies and distributes them on the different firewall applications while ensuring their consistency.

Our solution (see Figure 5.1) is distributed into 3 levels. The higher level is an orchestrator. It is integrated into the management plane. The middle level contains firewall applications which are integrated into the application layer and controllers. This level is responsible for processing the states of the connections, interpreting the policies to SDN rules and filtering connection requests according to the security policies, interpreting the latter to OpenFlow rules and installing them into the network elements. Finally, the last level is found in each network element (in the data plane layer). Network elements filter the packets according to many types of OpenFlow rules that express the firewall behavior.

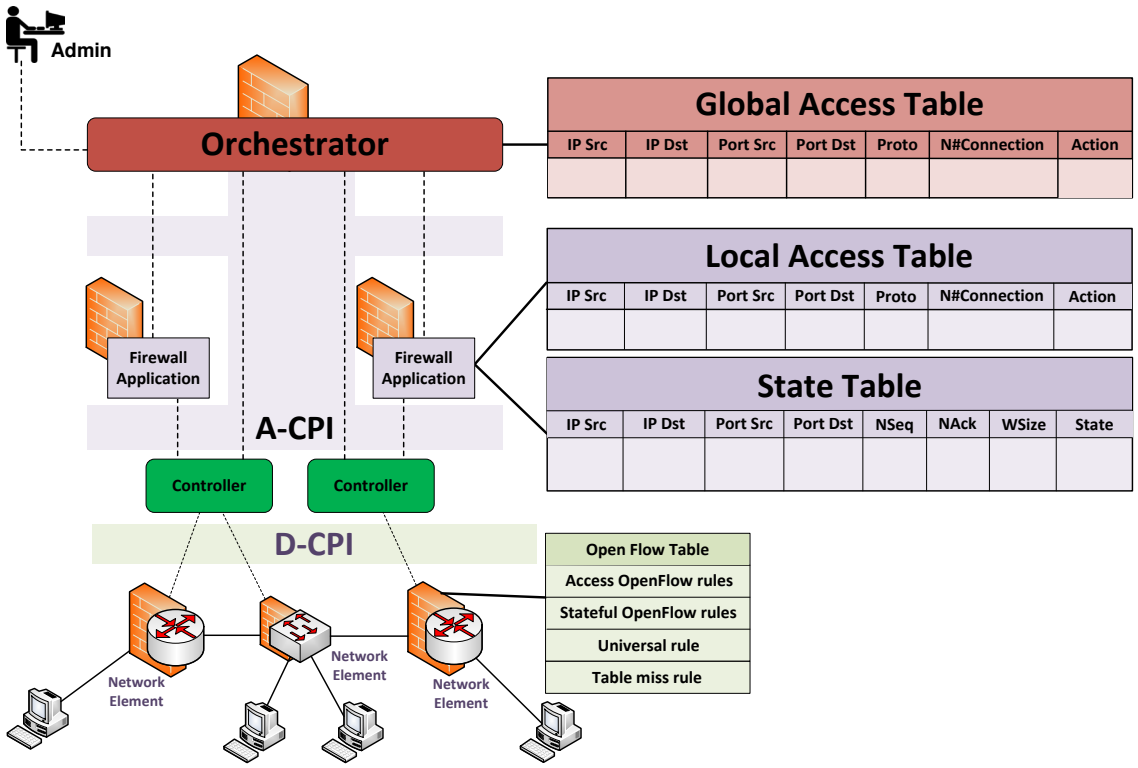
The orchestrator offers a management interface that simplifies the expression of the security policies. The orchestrator has a convening role because it collects the security policies and centralizes. Then, it propagates the policies to the different firewall applications according to the flows that pass their network domains. The orchestrator offers a graphical interface to the administrators to enable them to deploy the security policies and access to the global view of the SDN network. For this matter, the orchestrator includes a policy security editor.

As a mastermind, the orchestrator collects all the information related to the network topology, SDN events, logs and errors from all the controllers and the firewall applications. It manages this information in its database. It collects the domain knowledge of each controller. Then, it federates the whole in a global knowledge of the network (topology, network events, connection states, policy states, and other information). Besides, it keeps all the policies in a global access table that contains all the stateful and stateless security policies specified by the administrator. The orchestrator uses this table to propagate and reinforce the security policies of the network.

The orchestrator (see Figure 5.2) observes the events that come from the administrator (through its graphical interface and from the network. When an administrator updates the orchestrator policies, the orchestrator propagates the updates to the appropriate firewall applications. Besides, it observes network events such as the connection of new controllers or firewall application requests. It sends it the preliminary security policies that are related to the domain of its controller. These security policies are recorded in the local access table of the firewall application.

The orchestrator can also receive requests from the firewall applications according to two communication modes (see Section 5.4.5). For example, the firewall application can request access rules. In this case (see Figure 5.2), the orchestrator reads its global access table. Then,

Figure 5.1 – SDN Stateful Firewall General Architecture



it looks for new updates that affect the firewall application. If it found them, it sends the new access policies to the firewall application.

Furthermore, the orchestrator monitors the operations of the firewall applications. It collects from the controller information about its interactions with its firewall application and the network events in the data plane layer. This information is kept to prevent repudiation attacks in the firewall application (see Section 4.5 of Chapter 4) that exploit the vulnerabilities V14 (Application Function Repudiation) and V24 (Application Content Repudiation).

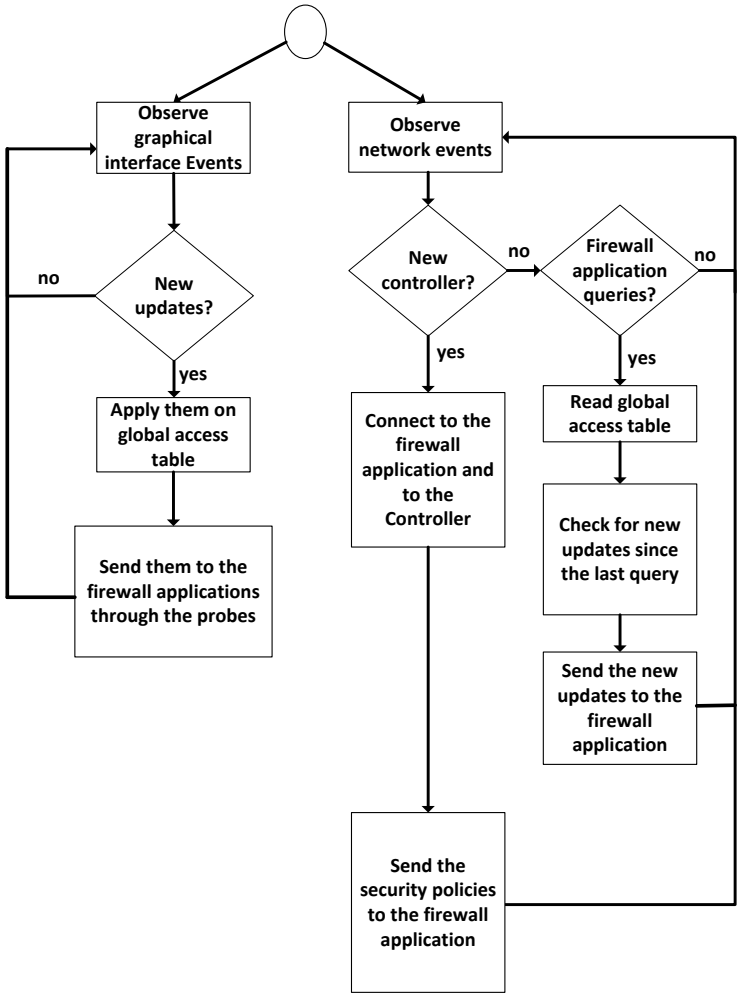
The orchestrator can also configure the behavior of the firewall applications dynamically. When the orchestrator receives a new configuration from the administrator, it updates the corresponding firewall application. Such configurations options are the behavior mode: stateful or stateless, the event mode: periodic (according to a timer) or instantaneous (according to a trigger) and the topology discovering mode: static (the user provides the topology), dynamic (by learning the topology dynamically) or hybrid (a mix between static and dynamic).

The firewall applications are stateful firewall functions that operate as SDN applications. Each firewall application uses a state table to manage the Finite State Machine (FSM) of the connections. It records the connections states, their transitions, and their attributes; the application uses this table to track the connection, its state and its possible transitions to the next states. Besides, the firewall application uses the state table to create State OpenFlow rules. The latter restricts the traffic only to the packets that correspond to the actual state and its valid transitions. This mechanism guarantees that the controller receives only the events that trigger the transitions from the actual state of the active connection.

The controller provides the orchestrator with the requested network information and guarantees the communication with the network elements. It interprets the northbound rules of the firewall application to OpenFlow rules. Then, it installs them on the appropriate network elements. Also, it notifies the firewall applications when it observes network events.

When a new network element joins the controller, the firewall application asks the controller to install the table-miss rule and OpenFlow universal rules on the new network element.

Figure 5.2 – SDN Orchestrator Behavior



These rules depend on the behavior type of the firewall application. The firewall application also configures network elements by setting their table miss entry according to its behavior. The actions of these OpenFlow rules enable network elements to accept or reject packets according to the security policies.

Network elements keep in their OpenFlow tables all the firewall rules. As a result, they perform firewalling behaviors by running the above OF rules. The firewall application reprograms them according to its security policies, and the orchestrator ensures their synchronization using its global knowledge.

5.4.2 Table management

The components of the solution maintain specific tables. At each level, one table is in relation with another located on the lower level. The firewall manages the following tables:

Orchestrator table

The orchestrator proposes a management interface to the administrator to gather the security policies. Once a security policy is entered in the management interface, it is recorded inside a global access table. The latter collects and centralizes all the security policies in an orchestrator database. The management of the global access table is given in Figure 5.2. It comprises two simultaneous threads that manage the behavior of the orchestrator on the table.

The first thread notifies the firewall applications with new policies updates when the administrator updates the global access table. In this case, the orchestrator records the updates in its table. Then, it processes the policy with its global knowledge. It sends the resulting policy to the corresponding firewall applications. The second thread expresses how the orchestrator manages its global access table when new controllers connect to it or when the firewall applications demand new policies. In the former situation, the orchestrator defines the policies from its access table that are related to the domain of the new controller. Then, it sends them to the firewall application that is attached to this new controller. In the latter case (when the firewall application asks policies), the orchestrator checks if the changes of the global view with its access table impact the policies of the firewall application or if its access table has been updated. If it is the case, it sends the new updates to the firewall application.

Firewall application tables

The firewall application uses two tables: a local access table and a state table. The access table contains administrator security policies according to a white list approach. It is part of the global access table of the orchestrator. The orchestrator federates all the network security policies of the network while the local access table convenes the security policies of the controller domain. When the firewall application receives the policies, it stores the policies in its access table to be able to spread them each time new network elements join the controller domain. Besides, it interprets them into northbound rules and sends them to its controller.

The state table keeps the records that enable the firewall application to track network connections. Each entry of the state table identifies a connection. For example, the entry for $Connection_{c1}^{S1}$ between a client (C1) and an Http_server (S1) contains at least the following information:

- Packet header information of the client such as $Source\_Fields_{c1}$.
- Packet header information of the server such as $Source\_Fields_{s1}$
- Other packet fields that can be found in both the streams of the client and the streams of the server: $Other\_Fields_{c1} \cap Other\_Fields_{s1}$
- The state of $Connection_{c1}^{S1}$ according to the FSM of the communication protocol between the client and the server: $State_{c1}^{s1}$

The firewall application uses the state table to track the state of the connection, to verify the legitimacy of a packet and to create state OpenFlow rules.

OpenFlow table

In each network element, a set of OpenFlow rules expresses the security policies according to the OpenFlow standard. Each network element maintains them in an OpenFlow Table. The controller installs in the tables the interpreted security policies. Following these rules, network elements change their behavior as follows. They authorize only the entities allowed by the security policies to communicate. They force each connection to comply with the legitimate behavior as specified in their standard communication protocol. Any packet that does not conform with the state of its connection is rejected. The solution produces three types of firewall OpenFlow rules:

- Table miss: It drops all the unexpected packets
- Universal OpenFlow rule: It forwards all the initialization connection packets to the firewall application

- Access OpenFlow rules: It is created from the access table entries
- Stateful OpenFlow rules: It enables the reception of only valid packets that trigger the transitions from the actual state of the active connections in the State table

5.4.3 SDN Equivalent Finite State Machine

The firewall application takes decisions by processing the Finite State Machine (FSM) of connection oriented network protocols. It takes as entries the preconditions, states, actions transitions and post-conditions, apply to them an SDN function and generates as an outcome SDN Equivalent Finite State Machine (SEFSM). The latter adapts the elements of the concept of the FSM to SDN context. The FSM represents all the possible configurations of the connection with all the transitions between these configurations, their triggers and the changes they induce on the connection. Based on the definition of FSM in [268], we represent an FSM for any stateful network protocol x as follows:

$$\mathrm{FSM}_x = (\mathrm{C}, \mathrm{S}, \delta, \mathrm{S}_0, \mathrm{F}) \tag{5.12}$$

Where:

1. C is the set of conditions that trigger the state transitions. A condition compounds the data of the packet such as its header fields or network meta information such as timers, QoS and other network data.

$$\begin{aligned} &\mathrm{C} = \mathrm{M}atching\_Fields \cup \mathrm{N}etwork\_\mathrm{M}eta\_information \\ &\mathrm{C} \neq \emptyset \end{aligned} \tag{5.13}$$

   C_i^j is the subset of C that triggers the transition from the state s_i to the state s_j.

$$\begin{aligned} &\forall c \in \mathrm{C}_i^j, c \in \mathrm{P}acket_i \vee c \in \mathrm{N}etwork\_\mathrm{M}eta\_information_i \\ &\mathrm{C}_i^j \subseteq \mathrm{C} \end{aligned} \tag{5.14}$$

2. S is the set of all the states of the FSM. $\mathrm{S}tate_{Connection_k^l}$ is the set S for the $\mathrm{C}onnection_k^l$.

$$\begin{aligned} &\mathrm{S} \neq \emptyset \\ &\mathrm{S}_0 \text{ is the set of initial states} \\ &\mathrm{S}_0 \subset \mathrm{S} \text{ and } \mathrm{S}_0 \neq \emptyset \\ &\mathrm{F} \text{ is the set of final states} \\ &\mathrm{F} \subset \mathrm{S} \text{ and } \mathrm{F} \neq \emptyset \\ &\mathrm{S}_i^j = \{s_i, s_j\} \text{ and } \mathrm{S}_i^j \subset \mathrm{S} \end{aligned} \tag{5.15}$$

3. A is the set of actions that are applied to the network when a transition occurs. A_i^j is a subset of A that is applied when the state s_i changes to s_j.

$$\mathrm{A}_i^j \subseteq \mathrm{A} \tag{5.16}$$

4. δ is the state transition function.

$$\begin{aligned} &\delta : \mathrm{C} \times \mathrm{S} \rightarrow \mathrm{A} \times \mathrm{S} \\ &\forall s_i \in \mathrm{S}\ \delta(s_i, \mathrm{C}_i^j) = (\mathrm{A}_i^j, s_j) \end{aligned} \tag{5.17}$$

The firewall application transforms the FSM_x; it maps the conditions and actions of FSM_x with their SDN counterparts while preserving the states and transitions. It produces an SDN oriented FSM that we call $SEFSM_x$ (SDN Equivalent FSM). It performs the following formulas:

1. $SEFSM_x$ transforms FSM_x according to the mapping relation $\Im$:

$$\begin{aligned} SEFSM_x &= \Im(FSM_x) \\ SEFSM_x &= (\gamma, S, \varrho, S_0, F) \end{aligned} \tag{5.18}$$

2. $\Im$ is the application of the two functions γ and ϱ that operate on the conditions and actions of FSM_x in order to generate $SEFSM_x$:

$$\Im : \gamma \times \varrho \tag{5.19}$$

3. γ is the transformation function that operates on the condition set C. It adds to the pre-condition SDN meta information such as global knowledge:

$$\forall c \subset C, \gamma(c) = (c, I_c) \tag{5.20}$$

4. I_c is the set of SDN Meta Information for the condition c. For example, if c is packet fields then:

$$I_c = \{OpenFlow, Version1.3, Packet-in\} \tag{5.21}$$

5. ϱ is the state transition function:

$$\varrho : \gamma \times S \rightarrow \sigma \times S \tag{5.22}$$

6. σ interprets γ, and the actions to functions call in the firewall application and to firewall rules that can be understood by the controller.

$$\sigma : \gamma \times A \rightarrow f \times R^{SDN\_FW} \tag{5.23}$$

7. f is an SDN firewall application function. It can take γ as arguments. An example of f for TCP is the Syn_Flooding_Counter function for the Synchronization packet. f takes the packet header (c), the Syn-flooding threshold (I_c) and the current Syn-flooding counter (I_c) as arguments. f with argument is defined as follows:

$$\forall c \in C, f(\gamma(c)) = f((c, I_c)) \tag{5.24}$$

8. R^{SDN_FW} is the set of SDN firewall rules. It is a subset of R within SDN (R is the set of firewall rules, see Section 5.2). It transforms the conditions and actions into SDN API rules:

$$\begin{aligned} &\forall c \in C_i^j \ \forall a \in A_i^j \ \exists r \in R^{SDN\_FW} \ r = \upsilon(c, a) \\ &\upsilon \text{ is the mapping function of the conditions and actions with SDN API rules} \end{aligned} \tag{5.25}$$

5.4.4 Firewall behaviors

Our SDN Firewall integrates two types of behaviors that have been derived from SDN paradigm. These behaviors defines the way of applying the $SEFSM_x$ to the connections. The reactive behavior generates the $SEFSM_x$ by transition each time the conditions that corresponds to the current state are true for each active connection. This behavior applies to the current state by generating the corresponding firewall rules and calling the proper firewall functions.

The proactive mode generates the SEFSM$_x$ entirely in one time for each active connection. It produces all the corresponding firewall rules to each transition. Then, it sends them to the controller which installs them on the proper network elements. The network elements handle the transitions and apply to each packet the corresponding stateful OpenFlow rules. Each time, the network element receives a packet that matches the conditions of the current state, it maps them to their OpenFlow rules, and at the same time, it notifies the controller with a copy of these conditions. The controller forwards the notification to the firewall application. When the firewall application receives the copy of the conditions, it updates the state in its state table and calls the corresponding firewall functions corresponding to the transition.

The sealing mechanism in both behaviors is different. In the reactive behavior, it prevents the destination from receiving any packets (or other conditions) not matching the actual conditions of the current state. As a result, only the packets with the current conditions reach their destination after being processed by the firewall functions and firewall rules. In the proactive behavior, all the packets that match the conditions of all the states can reach their destination for each type of packet because their stateful rules are installed in the network elements. The proactive behavior separates between the forwarding decisions which are executed by the network elements and the firewall functions that process the condition to take further decisions. In both behaviors, the firewall application performs the transition one time which means that the packets that correspond to the current state cannot be accepted again when the firewall application Changes the state to the new state except if it has a counter in the reactive that authorizes it for n times. The universal rule drops all the packets that do not match the transitions from the current state.

We think that the advantage of the proactive behavior is about the performance of the packet delivery time because it avoids the feedback loop between the network elements, the controller, and the firewall applications. Whenever the firewall application changes the state, the network element will only accept the conditions that are in the ESFSM path from the current state to the next states. The details of the firewall application behaviors are as follows:

Reactive behavior

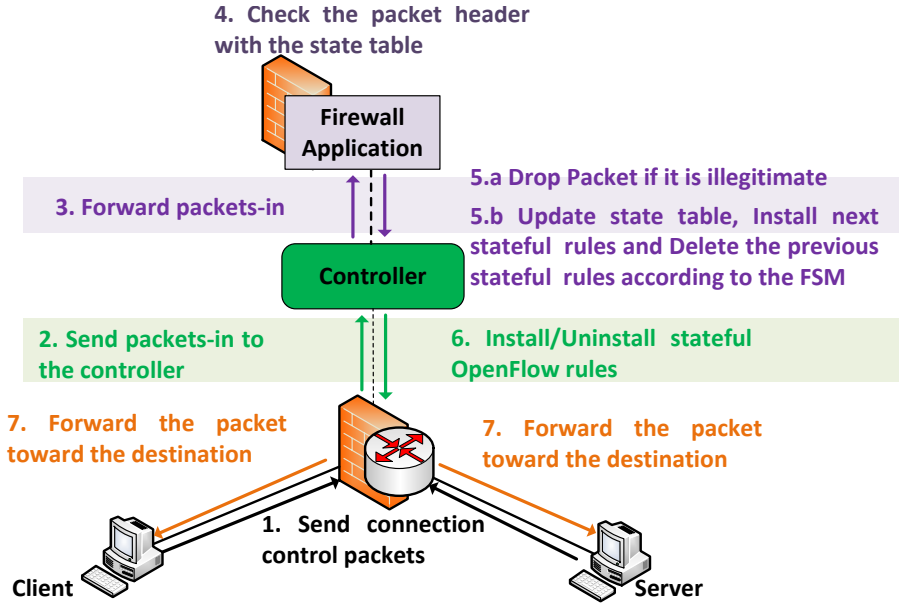
In the reactive behavior (see Figure 5.3), the firewall application processes all the connections at the application level. When the corresponding conditions trigger a transition, the firewall application produces the corresponding rules and executes the corresponding functions. For example, in the case of TCP when it receives the packet that triggers the transitions from the actual state, it executes the firewall functions: $Verify\_Packet\_Legitimacy$, $Delete\_Previous\_Rules$ and $Syn\_Flooding\_Protection$. Besides, it produces the corresponding stateful rules (such as adding new OpenFlow rules). The reactive behavior is based on a blacklist logic. It means that the unauthorized packets are rejected after they are verified with the entries of the local access table. The administrator needs to specify the policies that ban the packets.

Proposition 5.4.1. *The reactive behavior of the firewall application reacts only to the conditions that trigger the transition from the current state. If the conditions exists for a transition from the actual state, then, it produces the corresponding firewall rules and it executes the proper firewall*

functions according to $\Im(\mathrm{FSM}_x)$. *The reactive behavior is defined by the following statement:*

$$
\begin{aligned}
&\exists s_i^j \in \mathrm{S}\ \exists \mathrm{C}_i^j \subseteq \mathrm{C}\ \forall a \in \mathrm{A}_i^j,\ (Execute(f(\gamma(\mathrm{C}_i^j))) \wedge Produce(\upsilon(\mathrm{C}_i^j, a)))\\
&f(\gamma(\mathrm{C}_i^j))\ \textit{is the firewall application functions for the condition set}\ \mathrm{C}_i^j\\
&\upsilon(\mathrm{C}_i^j, a)\ \textit{is the set of firewall rules for the action } a \textit{ and}\ \mathrm{C}_i^j\\
&Execute\ \textit{and}\ Produce\ \textit{are two predicates}\\
&Produce\ \textit{generates DELETE rule if}\ \mathrm{A}_i^j = \{\}\\
&Produce\ \textit{generates ADD rule or MODIFY rule if}\ \mathrm{A}_i^j \neq \{\}
\end{aligned}
\tag{5.26}
$$

Figure 5.3 – Overview of firewall application reactive behavior



The firewall performs the following steps :

1. It installs a table-miss rule that drops all the packets by default. This rule locks the access of all connections.

2. It installs a universal rule that forwards all the connection control packets (initialization and termination) to the controller each time the hosts send them. The network elements send all these packets as Packets-in to the controller (Steps 1 and 2 of Figure 5.3).

3. The controller forwards theses packets-in to the firewall application (Step 3 of Figure 5.3)

4. The firewall application verifies the access rights of the connection according to the security policies. It examines the entries of its local access table and maps them with the header fields of the packet (Step 4 of Figure 5.3).

5. If the connection is accepted and it is new, the firewall application activates the connection in its state table; the firewall application creates a new entry in its state table with the corresponding state using the FSM. In case the connection is rejected, the firewall application performs the last step (Step 5.a of Figure5.3) and ignores the following intermediary steps.

6. It sends to the controller two requests. The first one asks the controller to install the stateful OpenFlow rules that correspond to the transitions from the actual states. The second

request asks the controller to reprogram the network element to forward the connection initialization packet to its destination (Step 6 of Figure5.3).

7. The network element installs the corresponding stateful OpenFlow rules and then forwards the initialization packet to its destination (Step 7 of Figure5.3).

8. For each active connection in the state table, the network elements forward to the controller all the packets that trigger the transitions from the actual state of the connection. It drops all the packets that do not fulfill the potential transitions.

9. Each time the firewall application receives a packet-in that belongs to an active connection, it performs the following sub-steps (Step 4.b of Figure 5.3):

   (a) It updates its state table by moving to the next state.

   (b) It asks the controller to delete the previous stateful OpenFlow rules that enable all the transitions to the new state.

   (c) It sends the stateful OpenFlow firewall rules that will allow the transitions from this new state.

   (d) It asks the controller to program the network element for forwarding the packet to its destination.

10. When the firewall application receives the last connection termination packet, it performs the following steps:

    (a) It asks the controller to delete the previous stateful OpenFlow rules that enable the transition to the last state.

    (b) It asks the controller to program the network element for forwarding the packet to its destination.

    (c) It updates its state table by deactivating the connection.

11. The firewall application rejects the packet-in and asks the controller to program the network element to drop the packet.

The reactive behavior is suitable when the network elements tables start to become full because it filters the connection by the state in the firewall application and it installs in the network element only the OpenFlow rules that enable the transitions from the actual state. Besides, it frees the network element tables each time the connection state changes to a new state by deleting the previous old rules. As a result, the reactive mode prevents network elements disruption attacks (see Section 4.5 of Chapter 4) due to the vulnerabilities V76 (Data Processing Engine Disruption), V156 (D-CPI Agent) and V96 (Data Source Disruption). For example, in the case of TCP, it installs a maximum of four stateful OpenFlow rules for each connection. The maximum number of OpenFlow rules is in the data transfer phase (2 OpenFlow rules for ACK by the stream, one for RST packet and the other for FIN packet).

However, the reactive behavior is vulnerable to disruption attacks on the southbound API and the controller such as V56 (C-Agent Disruption) and V146 (D-CPI Disruption) because the behavior needs the full involvement of the controller and especially the data-control channel. An attacker can perform disruption attacks on the controller (i.e., he can flood the controller with new connection initialization packets). To mitigate such Disruption attacks, we add an OpenFlow meter on the universal rule that limits the throughput of initialization packets-in for all connections. Besides, it needs high bandwidth between the data plane and the control plane layers. It introduces a delivery delay because it requires the firewall application decision to determine the fate of packets.

Proactive behavior

We propose the proactive behavior (see Figure 5.4) to prevent disruption attacks exploiting V56 and V146 and to improve the packet delivery time. In this case, the firewall application delegates the filtering to network elements. It leverages its processing in a higher granular level; it controls the access by connection unlike in the reactive mode where it operates on each transition of a connection. For each authorized connection, it generates the SDN equivalent FSM one time. Then, it installs all the corresponding firewall rules on the network elements that enable the network elements to filter the connections.

The network elements filter all the packets without the involvement of the firewall application because it has all the rules for all the life cycle of the connection in its tables. The firewall application receives only copies of the events observed by the network elements. It uses such information to update the state of the connection and to execute firewall function for the triggered transition. For example, in the case of TCP when it receives the packet that triggers the transitions from the actual state, It executes the firewall functions: D$elete_Rules$ and S$yn_Flooding_Protection$. The proactive behavior is based on a white list logic. It means that the firewall application installs stateful OpenFlow rules for only the accepted packets. The other packets are denied by default because the administrator needs only to specify the connections that are authorized.

Proposition 5.4.2. *The proactive behavior of the firewall application processes all the conditions that trigger the transitions in two phases. When it receives the conditions of the initial state, it produces all the firewall rules according to* $\Im(\mathrm{FSM}_x)$. *When it receives other conditions that correspond to the current state (not the initial state) it executes the proper firewall functions according to* $\Im(\mathrm{FSM}_x)$. *The proactive behavior is defined by the following statement:*

$$
\begin{aligned}
&(\ \exists s_0 \exists \mathrm{C}_0 \forall s_i^j \in \mathrm{S}\ \forall \mathrm{C}_i^j \subseteq \mathrm{C}\ \forall a \in \mathrm{A}_i^j,\ (\\
&(\mathrm{E}xecute(f(\gamma(\mathrm{C}_0))) \wedge \mathrm{P}roduce(\upsilon(\mathrm{C}_i^j, a))\ )\ )\\
&\vee\ (\ \exists s_{i\neq 0}^j \in \mathrm{S}\ \exists \mathrm{C}_{i\neq 0}^j \subseteq \mathrm{C}\ \exists a \in \mathrm{A}_{i\neq 0}^j \wedge \mathrm{A}_{i\neq 0}^j = \{\},\\
&(\mathrm{E}xecute(f(\gamma(\mathrm{C}_i^j))) \wedge \mathrm{P}roduce(\upsilon(\mathrm{C}_i^j, a)))\\
&f(\gamma(\mathrm{C}_i^j))) \textit{ is the firewall application functions for the condition set } \mathrm{C}_i^j\\
&\upsilon(\mathrm{C}_i^j, a) \textit{ is the set of firewall rules for the action } a \textit{ and } \mathrm{C}_i^j\\
&\mathrm{E}xecute \textit{ and } \mathrm{P}roduce \textit{ are two predicates}\\
&\mathrm{P}roduce \textit{ generates DELETE rule if } \mathrm{A}_i^j = \{\}\\
&\mathrm{P}roduce \textit{ generates ADD rule or MODIFY rule if } \mathrm{A}_i^j \neq \{\}
\end{aligned}
\tag{5.27}
$$

The firewall application in the proactive behavior performs the following steps:

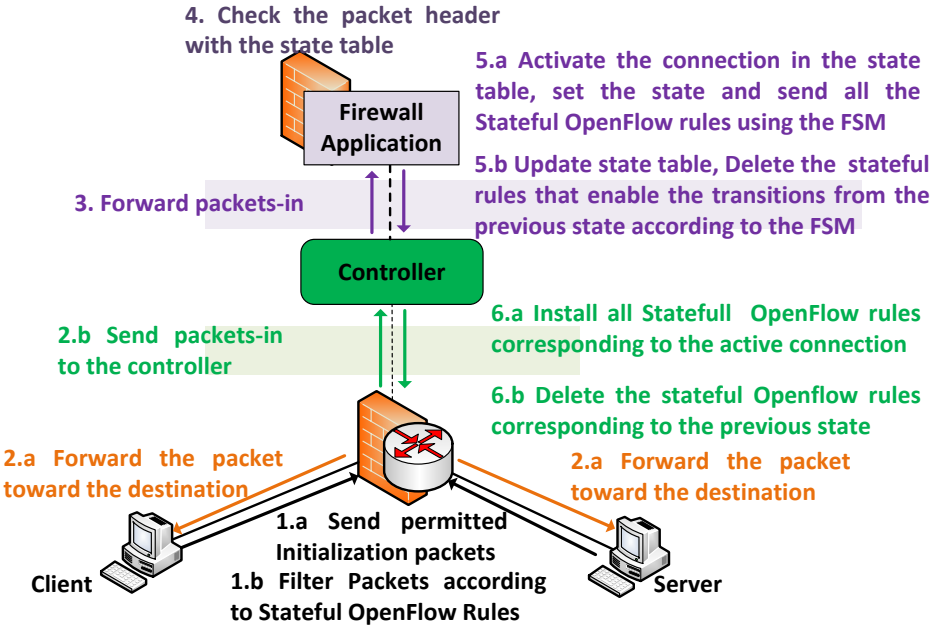
1. It installs a table-miss rule that drops all the packets by default. This rule locks the access of all connections.

2. For each entry of the local access table, the firewall application asks the controller to install a stateful OpenFlow rule in order to receive only the initialization packets of the authorized connections These packets will activate the connection in the state table.

3. When the firewall application receives a connection initialization packet (through steps 1.a, 2.b and 3 of Figure 5.4), it performs the following sub-steps :

   (a) It activates the connection and generates all its stateful rules according to the FSM of the connection (step 5.a of Figure 5.4). Each rule forwards the traffic to its destination and at the same time, it notifies the controller with the header of the packet

Figure 5.4 – Overview of firewall application proactive behavior



(steps 2.a and 2.b of Figure 5.4 will be performed in parallel by the network element except for the initialization packet).

(b) It asks the controller to install all these Stateful OpenFlow rules in a bunch on the appropriate network element.

(c) The controller installs them and notifies the firewall application of the success of the operation (step 6.a of Figure 5.4).

4. The network element forwards the initialization packet to its destination (step 2.a of Figure 5.4).

5. When the network element receives a packet, it filters it with its OpenFlow stateful rules (step 1.b of Figure 5.4). If the packet matches with a rule, then it is forwarded to its destination and its header is notified to the controller (steps 2.a and 2.b of Figure 5.4 are performed in parallel by the network element). If there is not a rule for the packet, it is dropped by default using table-miss rule

6. When the controller receives the header of the packet, it forwards it to the firewall application (step 3 of Figure 5.4).

7. The firewall application updates the stateful table using the information of the notification; it Changes the state to the next state (step 5.b of Figure 5.4).

8. It asks the controller to delete the stateful OpenFlow rules that enable all the transitions to the new state (step 5.b of Figure 5.4).

9. The controller deletes these stateful OpenFlow rules and notifies the firewall application (step 6.b of Figure 5.4).

10. When the firewall application receives the last connection termination packet, it performs the following steps (step 5.b of Figure 5.4):

(a) It asks the controller to delete all the stateful OpenFlow rules for the connection. It keeps the rule of the initialization, if the access table still authorizes the connection.

(b) It updates its state table.

11. The controller deletes these stateful Openflow rules and notify the firewall application (step 6.b of Figure 5.4).

12. The firewall application deactivates the connection.

The proactive behavior makes the network elements entirely responsible for the connection filtering process. The firewall application triggers the filtering, steers it and terminates it for each connection. As a result, the proactive mode reduces the packet delivery delay because it avoids that the packets pass through the circuit between the network element and the firewall application. The only delay that can be introduced will be related to the processing capacities of the network elements. Besides, the behavior prevents disruption attacks using V56 and V146, unlike the reactive mode which can only mitigate them. It can also mitigate other disruption attacks on the application layer (such as V16) since it reduces the load on the firewall application.

Furthermore, this behavior prevents the attacker from exploiting V96 to perform flooding (unlike in the reactive mode) because it works on a whitelist logic and it controls the number of initialization packets attempts per time. As a result, an attacker can not flood the controller with an unknown connection initialization packet(the network elements will drop its packets). Besides, the attacker can not also flood it with authorized initialization packets because of two defense mechanisms in the firewall application. The first one is the limitation of the number of unsuccessful initialization requests per time interval for each connection. When the connection exceeds this threshold, it is blocked for a specific time in the network element by the firewall application. The second defense mechanism is related to the FSM processing in the firewall application. When the firewall moves to the next state, it is not possible for the attacker to send initialization packets as they do not correspond to standard triggers for the potential transitions from the state (This mechanism is also present in the reactive mode).

5.4.5 Firewall modes

Through the controller, the firewall application is constantly listening to network events such as new connections, OpenFlow rule updates, OpenFlow rule expiration, disconnections, topology updates, and other events. It propagates its initial configuration (Universal rule, Table-miss rule, and the Access OpenFlow rules) to the lower layer. For each new connection, it asks the controller to install the OpenFlow Stateful rules when the state transitions are triggered. The firewall application observes networks events according to two modes:

On-demand mode

This mode is a bottom-up synchronous process. It operates in a pull sequence as follows:

1. The firewall application sets a timer to observe periodically network events coming from the controller and the orchestrator.

2. When the timer reaches its threshold, it sends a request to the controller and to the orchestrator to check if any new events have happened in its domain.

3. Once an event is observed, the firewall application generates the corresponding rules and applies the new configurations (such as change the behavior, load a specific FSM and other configuration requests).

4. it asks the controller to install the rules as OpenFlow rules on the appropriate network elements.

The On-demand mode is useful in an environment where the resources of the orchestrator and the firewall applications are loaded. In this context, the mode reduces the load by sending the requests each interval of time (a tolerable delay) instead of transmitting them instantaneously.

Independent mode

The independent mode is a top-down asynchronous process. It works in push sequence as follows:

1. The firewall application registers a probe in the orchestrator and the controller.
2. The probe connects to the event catchers in the orchestrator and controller to collect specific events such as policy updates, topology changes, firewall behavior change, new FSM and other events.
3. Each time a specific event occurs, the probes notify the firewall applications
4. The firewall application applies the updates. It generates the corresponding rules and the configurations
5. It asks the controller to install the rules on the appropriate network elements.

5.4.6 Firewall application General Algorithm

The firewall application processes the FSM of each network protocol and produces a set of decisions (firewall rules and functions) to enforce the access of the connections. Figure 5.5 provides an overview of the general algorithm of the firewall application. This algorithm focuses on processing packets as conditions for generating the SEFSM.
We assume in this algorithm the following premises:

- The firewall behavior and mode are activated.
- The access rules are installed in the network elements.
- The universal and table miss rules are installed in the network elements.
- The packet that is received by the firewall application is valid.

The firewall uses the on-demand mode or the independent mode to collect network events coming from the orchestrator or the controller. Let us suppose that the controller notifies the firewall application with a packet-in. Thus, when the firewall application receives this packet-in, it checks if the packet belongs to an active connection in the state table. If the entry does not exist, the firewall application creates an active entry of the new connection. Otherwise, it uses the packet to identify the entry of the connection and its actual state. It verifies if the conditions contained in the packet trigger any of the available transitions from the actual state of the connection. If the conditions are valid, the state is initial, and the behavior is proactive. The firewall application generates all the SEFSM and asks the controller to install the corresponding rules in the network elements. In any other case, the firewall application processes the conditions to trigger the proper transition of the FSM.

When the transition is selected, the firewall application executes its functions that correspond to the transition. Besides, it produces the corresponding firewall rules if its behavior is reactive. If some functions are only available in the controller (for example, update the topology), it sends the firewall rules and controller function requests to the controller with the proper arguments. Both the firewall rules and the controller functions requests are firewall decisions that are processed by the controller to decide on the fate of packets. The firewall application updates the entry of the connection in the state table with the new state until the connection reaches its final state.

5.4.7 TCP Example

We have applied the FSM transformation formalism to TCP. We obtain a $SEFSM_{TCP}$ for the reactive behavior and another one for the proactive behavior. $SEFSM_{TCP}$ keeps the states of the TCP FSM, it transforms the conditions, transitions, and actions to their equivalent in SDN. It relies on the transformation function of the formalism and on the general algorithm of the firewall application to create the equivalent concepts for TCP.

TCP is a standardized stateful network Protocol that enables the hosts to communicate reliably in an interconnected network. The TCP protocol the connection of two parties: a client that initiates the connection and a server whose the role is to reply to the client requests. It starts with a phase of connection establishment between the client and the server. It uses the principle of the three steps handshaking; the client sends a connection initiation message to the server to initialize a connection. The server confirms this connection initiation with an acknowledgment message that contains data to negotiate the connection parameters. Finally, an acknowledgment that contains the negotiated connection parameters is then sent by the client to the server. In this phase, each host allocates memory space for the connection, and it agrees with the other side on the properties to use for the communication such as sequence numbers to identify the segments, segment size, window size to limit the number of segments without acknowledgment. Also, each segment contains many flag fields to enable the sender to specify to the receiver which kind of segment it has sent and in which state of the TCP protocol the connection is.

When they perform the first phase successfully, the protocol allows them to transfer their data. During the data transfer phase, the two entities send segments that contain data with acknowledgment information of the previously received data. They can also send an unexpected connection interruption message to reset the session when a problem is detected and cannot be resolved, or a connection termination segment to end the communication.

Finally, the last phase is the termination of the connection. Each side sends a connection termination segment when it is ready to terminate the communication and waits to receive from the other side a termination acknowledgment message to close its connection.

The firewall application processes the TCP protocol to control the access of connections. The generated $EFSM_{TCP}$ for the establishment phase are shown in Figure 5.6a for the reactive behavior and in 5.6b for the proactive behavior. In the connection establishment phase, the client tries to open a connection with the server. It sends a connection initiation packet (SYN packet) with a value of the field Flag = SYN. The header of the packet is the condition. It is matched in the data plane device with the available OpenFlow matching fields. If the packet header does not match with the rules of the network element, it is dropped. In the reactive behavior, it is the firewall application which verifies the SYN packet with the access rule instead of the network element (proactive mode). If the SYN packet is accepted in both behaviors, the firewall application activates its entry in its state table with a state value equal to LISTENING.

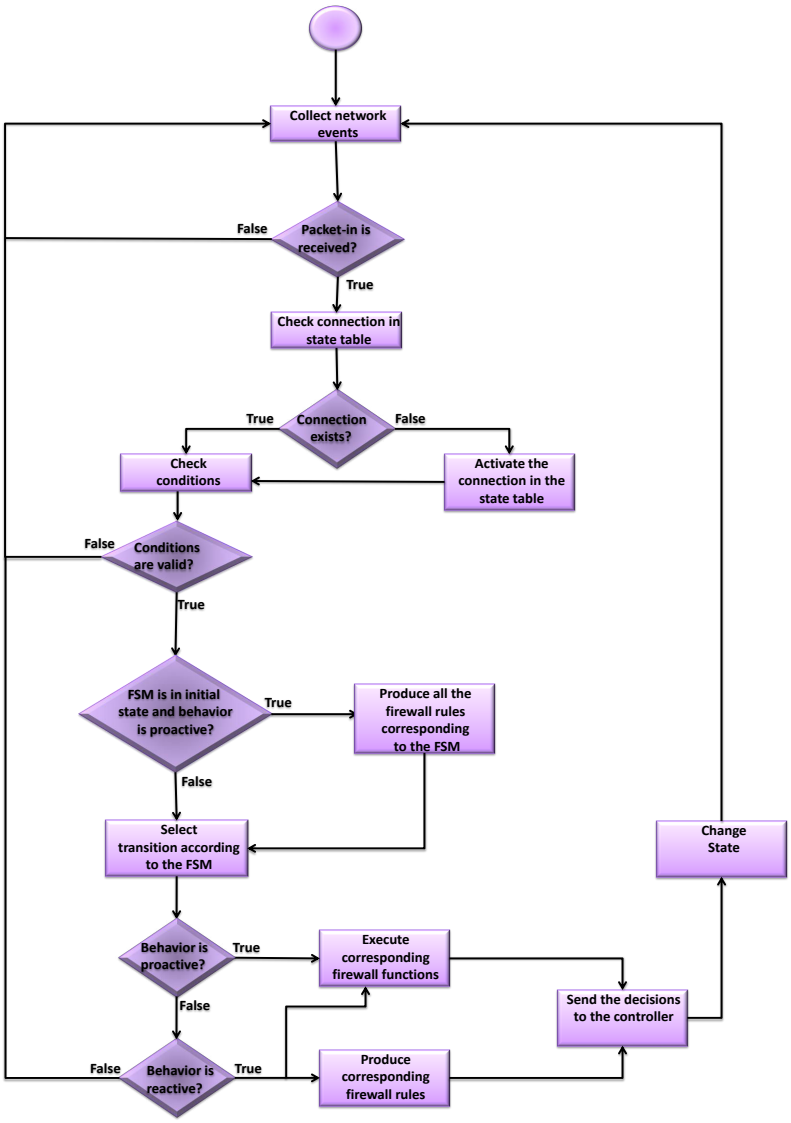
If its behavior is reactive, it transforms the conditions and actions for all the transitions from the actual state to their SDN equivalent (see Figure 5.6a). In the proactive mode however, it transforms all the transitions of TCP. Then, in each state, it deletes or deactivates the elements of the previous transitions (see Figure 5.6b). For example, in the reactive behavior, in the LISTENING state, the firewall application performs the following transformations (see Figure 5.6a):

1. It transforms the condition $c_{\text{LISTENING}} = \{packet\_header, packet\_Flag = \text{SYN}\}$ to: $\gamma(c_{\text{SYN}}) = (\{packet\_header, packet\_Flag = \text{SYN}\}, \text{NBR\_SYN} = +1)$

2. It transforms the action $Send\_\text{SYN}$ as follows:

   (a) It calls the following Functions:

Figure 5.5 – SDN firewall generic algorithm



- $Verify\_Packet\_Legitimacy(packet\_header)$ to verify if the packet is accepted and is not a bogus packet.
- $Syn\_Flooding\_Protection(packet\_header, \text{NBR\_SYN})$ to verify if the packet is not a syn flooding attack.
- $Activate\_Entry(\{packet\_header, packet\_Flag = \text{SYN}\}, \text{NBR\_SYN})$ to set an entry for the connection in the state table.
- $Resubmit(Packet)$ to forward the packet to the controller. The controller will interpret it to Packet_out. This function is not called in the proactive behavior.
- $Update\_State(\text{SYN\_RCV})$ to set the new state to SYN_RCV

(b) It produces the following rules:

- Send (State Rule(SYN=1, Priority=+1, Hard_Timeout=T1, Deny)): it drops any further connection initialization messages for the same connection and it mitigates SYN Flooding attacks. If the SYN flooding function is called, the rule is produced when the connection reached the Syn threshold. T1 is the TCP timeout. Deny will be interpreted by the controller to drop.
- Send (State Rule(RST=1, Priority=+1, Hard_Timeout=T1, Accept)): it allows the client to reset the connection using a RST packet. Accept will be interpreted by

the controller to Forward to the controller.

- Send (State Rule(SYN=1, ACK=1, Priority=+1, Hard_Timeout=T1, Idle_Timeout=T2, Accept)). T2 is the TCP timeout.

When the firewall application receives the conditions for the actual state SYN_RCV, it transforms the actions and conditions of all the transitions if its behavior is reactive (see Figure 5.6a). However, if it is proactive, it calls the functions that delete the rules of the previous transitions. It also deactivates some called functions (such as $Syn\_Flooding\_Protection$). For example, in the case of the proactive behavior and the SYN_RCV state, the firewall application performs the following transformations (see Figure 5.6b):

1. It transforms the condition $c_{\text{SYN\_RCV}}$= {$packet\_header, packet\_Flag$ = SYNACK} to: $\gamma(c_{\text{SYN\_RCV}})$ = ({$packet\_header, packet\_Flag$ = SYNACK})

2. It transforms the action $Send$_SYNACK to the following functions:

   - $Check\_Seq\_Number()$ to verify that the sequence and acknowledge numbers are correct.
   - $Delete(StateOFRule(\text{SYN} = 1, \text{ACK} = 1, Accept))$: to delete the rule that authorizes the server to answer with an acknowledgment to the synchronization request.
   - $Update\_State$(SYN_ACK_RCV) to set the new state value to SYN_ACK_RCV

Likewise, if the client answers with a correct acknowledgment packet before one of the timeouts elapses, the firewall application performs the appropriate transformations to finish the handshaking phase. Up to this step, in the reactive mode all the installed State OF rules forward the packets only to the controller while in the proactive mode they forward to the controller and the destination at the same time. When the state is updated to ESTABLISHED, the connection moves to the data transfer phase. The firewall application transforms the action $Send$_ACK to a call of the function $Check\_Seq\_Number()$. If the latter returns false, the firewall application denies the packet (see Figure 5.7).

The termination phase starts when the client or the server sends a FIN packet. When the firewall application receives this packet, it changes the state to the termination phase. For example, in the case of the reactive behavior in the ESTABLISHED state, the firewall application performs the following transformations (see Figure 5.7):

1. It transforms the condition $c_{\text{ESTABLISHED}}$= {$packet\_header, packet\_Flag$ = FIN} to: $\gamma(c_{\text{ESTABLISHED}})$ = ({$packet\_header, packet\_Flag$ = FIN})

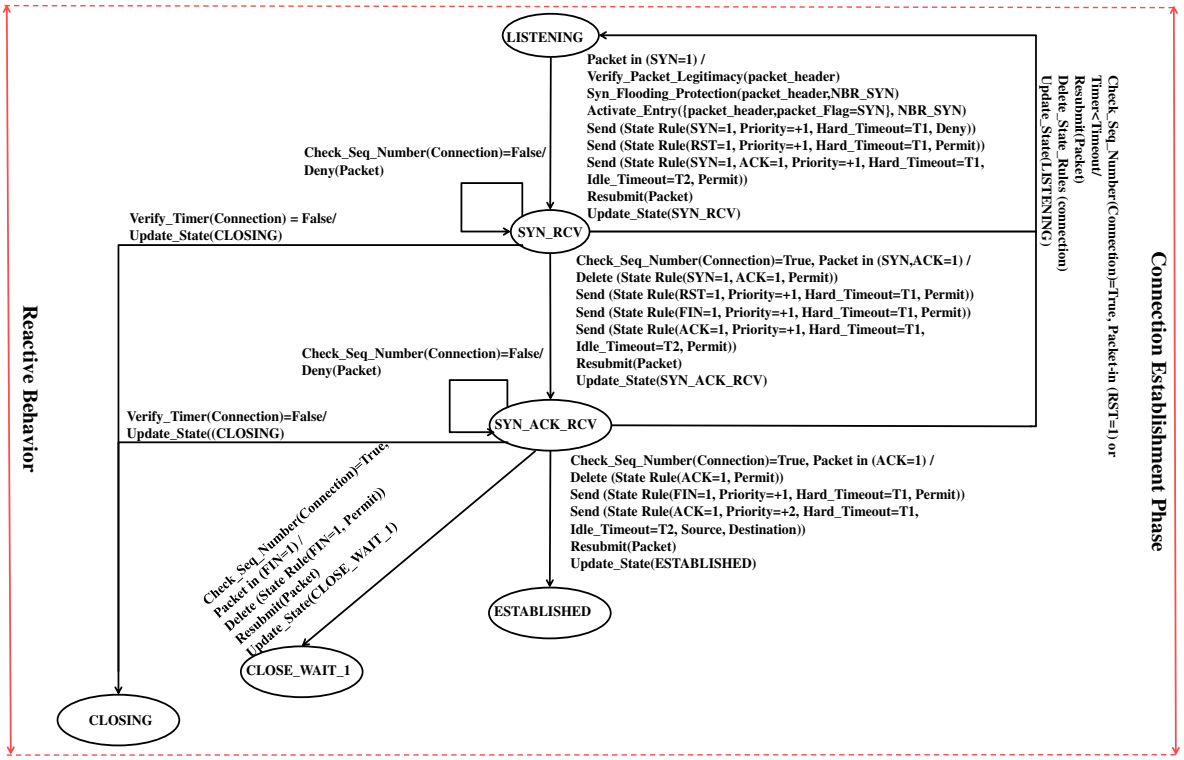
2. It transforms the action $Send$_FIN as follows:

   (a) It calls the following functions:

   - $Check\_Seq\_Number()$
   - $Delete(StateRule(\text{FIN} = 1, Accept))$ to delete the state rule that authorizes the reception of the first FIN packet.
   - $Resubmit(Packet)$ to forward the packet to the controller. The controller will interpret it to Packet_out. This function is not called in the proactive behavior.
   - $Update\_State$(CLOSE_WAIT_1) to set the new state to CLOSE_WAIT_1.

   (b) It produces one rule that enables the destination to acknowledge the FIN packet that it will receive:
   Send (State Rule(FIN=1, ACK=1, Priority=+1, Hard_Timeout=T1, Idle_Timeout=T2, Destination, Accept)).

(a) EFSM for TCP handshaking in reactive mode



(b) EFSM for TCP handshaking in proactive mode

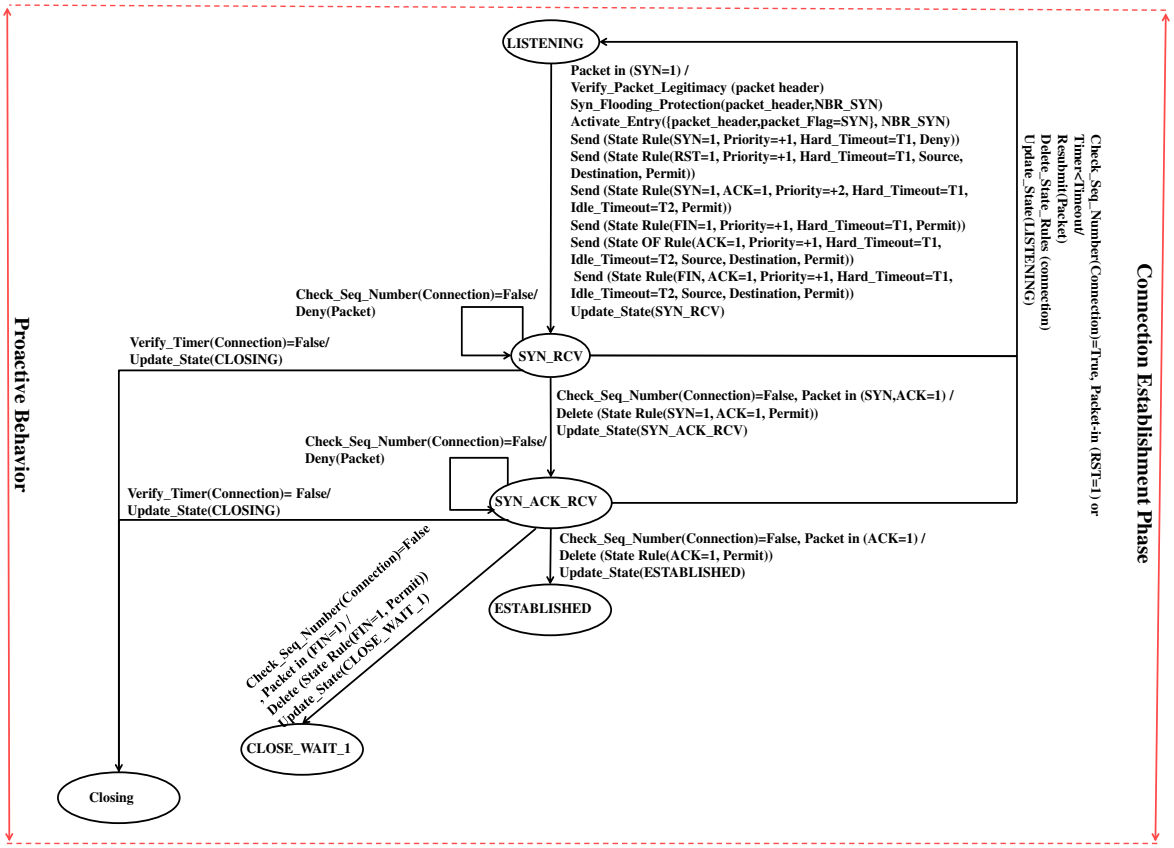


Figure 5.6 – EFSM for TCP handshaking phase

When all the transitions of the TCP termination phase are performed (see Figure 5.7), the firewall application moves to the state CLOSING. It triggers a termination timer according to the TCP protocol, and it waits for its end. When the timer elapses, it asks the controller to delete the rest of the State OF Rules related to the connection. It removes the connection entry from the connection table and deactivates the connection.

5.4.8 Performance Analysis for Transport Protocols

The performance of the firewall application is evaluated regarding scalability and time of packet delivery. The scalability is a critical parameter. It represents the maximal number of connections that can be processed by the firewall application without affecting the network performance. The two metrics used to evaluate the scalability when the firewall application processes a transport protocol are:

- **Number of packets sent to the firewall application**: This parameter represents the load on the firewall application, regarding the number of packets to process per connection. The parameter must be minimized because the processing time per connection is related to it.
- **Maximum number of connections supported by a network element**: Each network element has a limited number of tables. Furthermore, each table is limited by a maximum size (i.e., the maximum number of entries in the OpenFlow table). This parameter has to be maximized to process the maximum number of connections that is tolerated by the network element.

A network element forwards a packet-in each time the records of a packet header trigger a transition from one state to another. Hence, the number of packets sent to the controller is bounded by the number of states of the protocol FSM. The maximum number of packets sent to the controller by a network element i for the connection K is given by the following equations:

$$Number\_Packetin_{i,Connection_k} <= Number\_State_{Connection_k} \tag{5.28}$$

$$Max\_Number\_Rules_{i,Connection_k} <= Number\_State_{Connection_k} \tag{5.29}$$

where $Number\_State_{Connection_k}$ represents the maximum number of states of connection K that the firewall application processes. $Max\_Number\_Rules_{i,Connection_k}$ is the maximum number of OpenFlow rules that will be installed on the network element i for the connection K. This number is related to the transport protocol specification.

If we take the illustration of the TCP protocol, each network element sends at most 7 packets-in (3 packets during the connection establishment phase – SYN, SYN-ACK and ACK, 1 packet during the data transfer phase – FIN and 3 packets during the termination phase – FIN-ACK, FIN, FIN-ACK) to the firewall application. This number is low in comparison to the total number of states per connection which is 11 for TCP.

The maximum number of connections supported by a network element is related to the maximum number of rules installed by the firewall applications. So, the maximum number of connections supported by a network element is defined according to the following equation:

$$MaxConnections^i = \frac{OpenFlow\_Table\_Size_i}{Max\_Number\_Rules_{i,Connection}} \tag{5.30}$$

where $OpenFlow\_Table\_Size_i$ is the size of the OpenFlow table in the network element i.

For TCP connections, the maximum number of OpenFlow rules that are installed per connection is the highest in its proactive behavior. It is equal to 11 stateful OpenFlow rules per

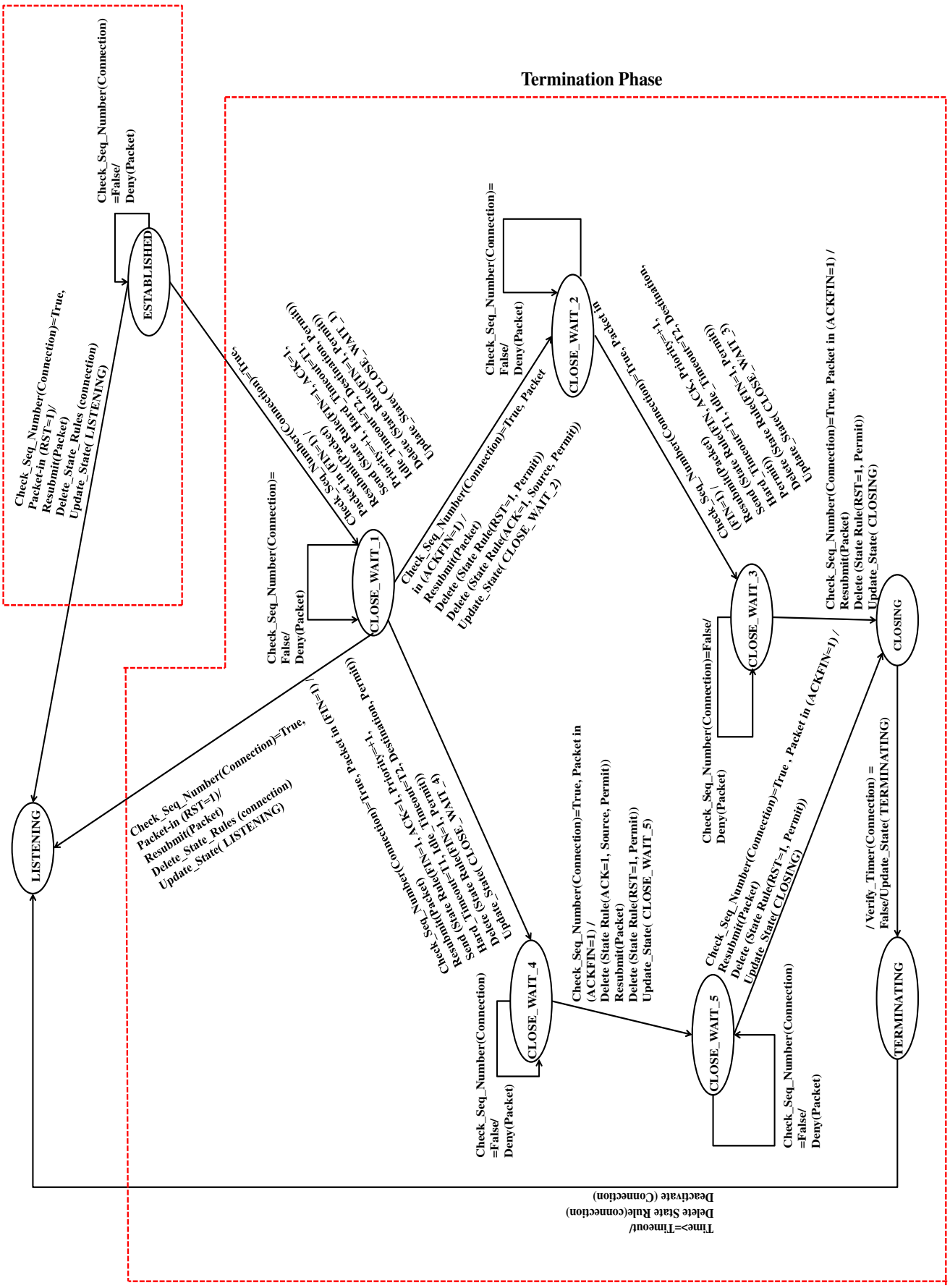


Figure 5.7 – EFSM for TCP data transfer and termination phase in reactive mode

connection (i.e., 3 OpenFlow rules for the handshaking, 2 OpenFlow rules for the reset, 2 OpenFlow rules for data transfer and 4 OpenFlow rules for the termination). For example, let us take the hardware network element HP-5700. It can support $2^{16}-1$ OpenFlow firewall rules [269]. Thus, in the proactive behavior, HP-5700 supports up to 5957 TCP connections. Besides, in the reactive mode, the maximum number of OpenFlow rules that are installed per connection is equal to 4 (i.e., 2 OpenFlow rules for the reset, 2 OpenFlow rules for data transfer). Thus, in the reactive behavior, HP-5700 supports up to 16383 TCP connections (the universal and table miss rules have been taken into consideration).

If the network elements have different tables sizes and all the connections pass through them, then the number of connections supported by the firewall is the lowest value of the maximum number of supported connections in each network element. It can be expressed with the following equation:

$$\forall i \in \text{NE},\ Max\_connections_{\text{FW}} = Min(Max\_connections_i) \tag{5.31}$$

where NE is the set of all the network elements that are protected by the SDN firewall application.

The performance of the SDN firewall solution affects the time of packet delivery (PDT). The latter is a metric for Quality of Experience (QoE) of the end users. The extra delay added to the packet delivery by our solution should be transparent to end users. Our solution has to achieve a tolerable QoE compared to an SDN network without a firewall. PDT starts from the moment the packet leaves the port of the network element to its destination until the moment the network element receives a reply from the destination.

The SDN Firewall Time (SDNFT) includes the Network Element Forwarding Time (NEFT) and the Feedback Loop Time (FLT). The former is the time taken by the packet inside the network element. It includes two times. Packet Processing Time(PPT) which is the time needed by the network element to match the packet with the OpenFlow rules and execute the corresponding action. The other time is the OpenFlow Rules Installation Time (ORIT) which is the time of installing the OpenFlow rules after receiving them from the controller. The feedback loop time starts from the moment the packet-in leaves the network element until the last moment the network element receives the last rule or packet-out for this packet-in. SDNFT must be inferior to PDT to guarantee tolerable QoE.

The behavior of the firewall application affects SDNFT. In the proactive behavior, this time is lower than in the reactive behavior because FLT in the proactive behavior happens only in the Synchronization phase while in the reactive behavior it happens in each transition. As a result, the following equations express the different times:

$$\text{SDNFT} = \begin{cases} \text{PPT} + \text{FLT\_}Reactive + \text{ORIT} & \text{(if the firewall application behavior is reactive)} \\ \text{PPT} + \text{FLT\_}Proactive + \text{ORIT} & \text{(if the firewall application behavior is proactive)} \end{cases} \tag{5.32}$$

Where: FLT_$Proactive$ < FLT_$Reactive$ Thus, from the equation 5.32, we understand that the SDN firewall must take into account the following conditions:

$$\begin{aligned} &\text{SDNFT} < \text{PDT} \\ &\text{More FLT is high, more the delivery time increases} \end{aligned} \tag{5.33}$$

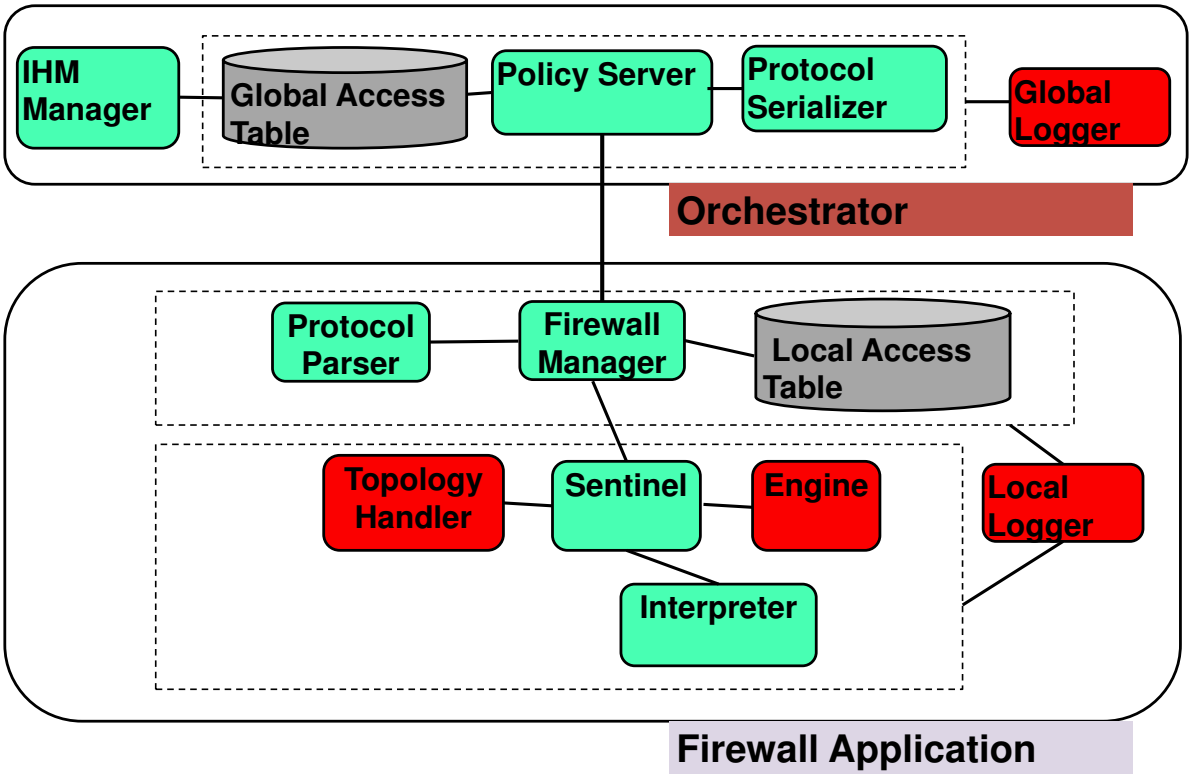
5.5 Implementation

We have implemented the firewall application and the orchestrator in python language. They run on the top of the RYU controller. RYU is a Python open source component-based SDN framework. It manages network traffic by handling and processing network events through

different standardized APIs. It also offers developers the possibility to implement their network applications and to run them on the controller. RYU offers to SDN applications many services that handle and process network events, parse and serialize packets or interact with the network elements through southbound APIs such as OpenFlow.

Figure 5.8 shows the software architecture that we have implemented. It compounds two packages that contain a set of modules. The details of the implementation are as follows:

Figure 5.8 – Implemented Firewall Software Architecture



1. **Orchestrator Package**: It runs in the management layer. It offers a General User Interface to manage the security policies and the OF rules. It allows adding, modifying and displaying the security policies and manages the static topology information. It offers to the administrator the possibility to configure many parameters of the firewall such as event modes, behavior modes. It keeps open sockets with all the Firewall Application instances to communicate with them. Through these canals, it sends management commands and collects network events. The orchestrator has the following modules:

    - **IHM Manager**: It offers a user interface to create, manage and view the firewall policies. Besides, it displays network events that have been collected from firewall applications.
    - **Policy Server**: It manages the policies in the global access table. besides, it manages the interactions with the firewall applications (such as sending new policy updates, collecting events and configuring the firewall applications).
    - **Protocol Serializer**: It transforms the policies according to the communication protocol established between the orchestrator and the firewall application.
    - **Global Logger**: It collects events coming from the firewall applications, statistics from data plane devices and connections information. It sends them to the GUI to

be displayed. Some of the data are also stored in a persistent database to provide the administrator with logs about all the events within the firewall architecture and the SDN Network.

2. **Firewall Application Package**: It integrates the general firewall algorithm. It processes the FSM of network protocols to generate their equivalent EFSM. It handles the interactions with the controller and collects from it network events. It has the following modules:

   - **Protocol Parser**: It parses the received messages from the orchestrator, and it extracts the necessary information for the firewall application modules.
   - **Firewall Manager**: It handles the communications between the firewall application modules and the Policy Server by observing and redirecting all the observed events to/from the orchestrator.
   - **Topology Handler**: It processes the network topology that the firewall protects. It proposes two modes. A static mode where the topology is specified by the orchestrator (via the administrator). A dynamic mode where the firewall discovers the network elements and nodes using the controller.
   - **Sentinel**: It plays a role of an intermediary between the other firewall modules and the controller. It sends the firewall rules to the controller and collects from it any observed event to distribute it to the appropriate firewall module. Sentinel is also responsible for triggering and configuring the other firewall modules according to the information collected from the controller or the orchestrator. For example, it instantiates an engine module for each connection. It configures it with the appropriate parameters that were specified by the orchestrator such as the firewall behavior and the protocol FSM, and it generates the initial configuration of the network element.
   - **Engine**: It expresses the main behavior of the firewall by handling all the phases of a stateful connection and by processing the communication between the client and the server. This behavior is based on the specialization of the general algorithm and the transformation of the FSM into EFSM using the formalism that we have explained in Section 5.4.3. The engine module grants also the flexibility of the architecture since it is possible to coexist with many other Engines modules that handle other stateful network protocols with the Engine of TCP.
   - **Local Logger**: It collects the events coming from the controller and the ones that occur in the firewall application. Then, it sends them to the Orchestrator's Logger.
   - **Interpreter**: It interprets the firewall rules to access OpenFlow rules. The module uses mapping information to translate the security policies into OpenFlow rules. Therefore, any change in the OpenFlow standard affects only the Interpreter module.

5.6 Evaluation

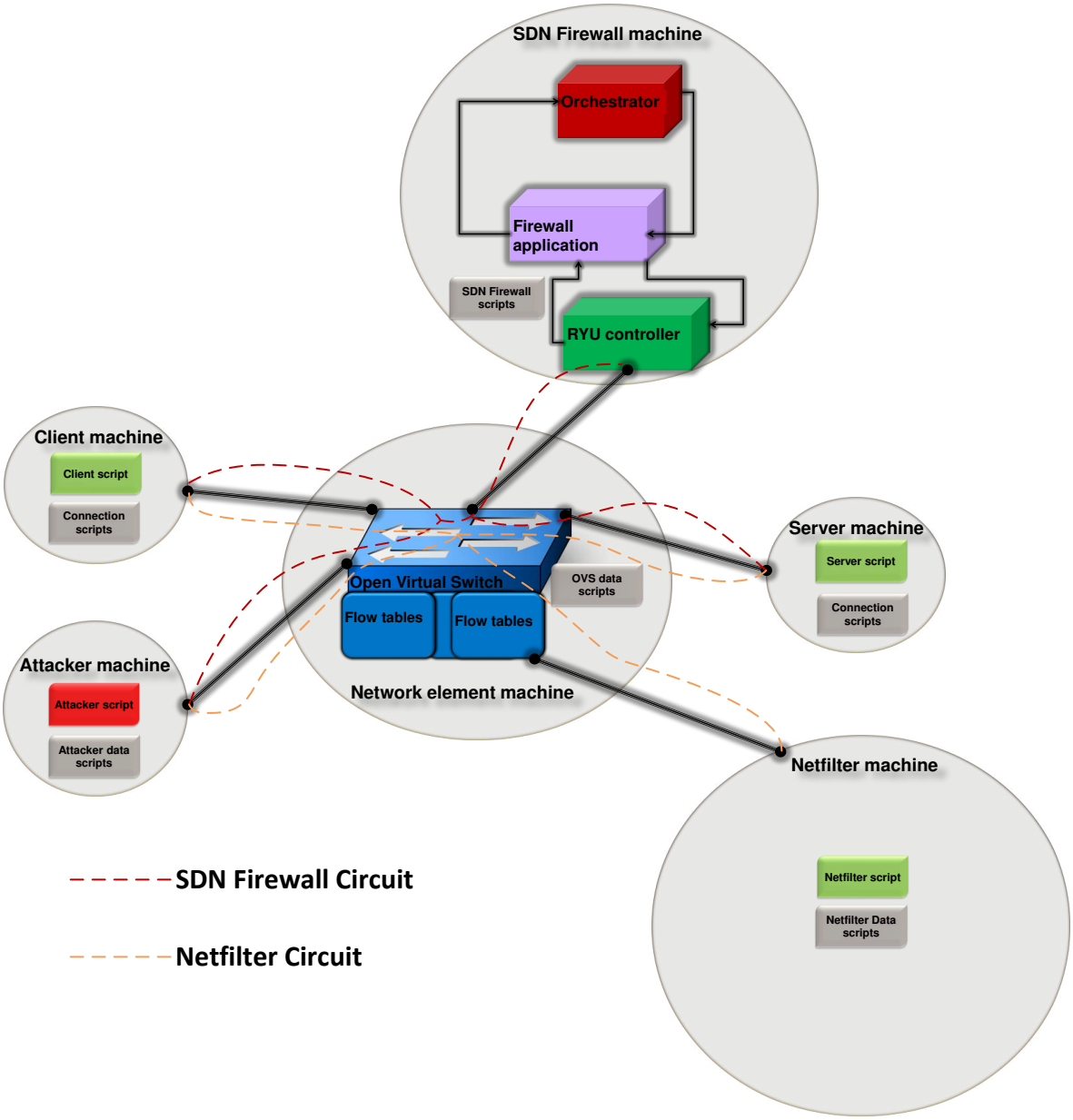
The objective of our evaluation is to estimate the performance of the SDN firewall and its capacity to resist to Syn flooding attacks. We compare the performances of the reactive behavior with the proactive behavior and with the legacy firewall NetFilter. For this matter, we set an evaluation environment, and we evaluate our SDN firewall through many experiments.

5.6.1 Test Bed

We deploy our solution in the B-Secure platform of B<>COM [270]. It is a dedicated evaluation platform to test SDN security solutions and attacks. Figure 5.9 shows the platform that we have deployed to evaluate our solution. The platform consists of the following components:

1. **SDN firewall machine**: It is a physical machine which is characterized by 16 Gb of RAM, Intel i7 processor and one physical network interfaces of 1 Gb/s. The orchestrator, the firewall application and RUY controller run inside this machine. We implement a set of scripts in this machine to monitor the SDN firewall machine:
    - **SDN Firewall scripts**: They monitor the performance of the SDN firewall and capture the network traffic (including OpenFlow) in its network interface.
2. **Network element machine**: It is a physical machine which is characterized by 16 Gb of RAM, Intel i7 processor and five physical network interfaces of 1 Gb/s. OVS runs in this machine as a network element. We implement a set of scripts in this machine to monitor the network element machine:
    - **OVS Data scripts**: They monitor the performance of the network element and capture the network traffic (including OpenFlow) in its network interface.
3. **NetFilter machine**: It is a physical machine which is characterized by 16 Gb of RAM, Intel i7 processor and one physical network interface of 1 Gb/s. NetFilter firewall runs inside the machine. We implement two categories of scripts in this machine to configure and monitor NetFilter:
    - **NetFilter script**: It installs the firewall rules to control the access to the connections between the client machine and the server machine. The script also activates or deactivates the Syn flooding protection module of NetFilter.
    - **NetFilter Data scripts**: They monitor the performance of NetFilter and capture the network traffic in its network interface.
4. **Client machine**: It is a virtual machine which is characterized by 4 Gb of RAM, Intel i3 processor and a dedicated physical network interface of 1 Gb/s. We implement 2 categories of scripts in this machine to configure and monitor the client:
    - **Client script**: It generates and manages a number of TCP connections over a time interval. The script runs from 100 to 500 processes. Each process runs 300 threads that generates and manages simultaneously TCP connections. For example a configuration of 100 processes will generates 30.10^3 TCP connections.
    - **Connection scripts**: They monitor the performance of the client and capture the network traffic in the network interface.
5. **Server machine**: It is a virtual machine which is characterized by 4 Gb of RAM, Intel i3 processor and a dedicated physical network interface of 1 Gb/s. We implement two categories of scripts in this machine to configure and monitor the client:
    - **Server script**: It configures the Apache server for TCP connections. The script creates 500 Apache HTTP processes. Each one of them manages a different port and is connected to one process of the client. The script configures the size of the data that each Apache process offers to its client process.
    - **Connection scripts**: They monitor the performance of the server and capture the network traffic in its network interface.

Figure 5.9 – SDN Firewall Testbed



6. **Attacker machine**: It is a virtual machine that is characterized by 4 Gb of RAM, Intel i3 processor and a dedicated physical network interface of 1 Gb/s.

   - **Attacker script**: It programs the behavior of a DoS attacker that performs Syn flooding attacks. The script can configure different rates of Syn flooding attacks.
   - **Attacker data scripts**: They monitor the performance of the attacker and capture the network traffic in its network interface.

5.6.2 Evaluation Experiments

We set three scenarios in our evaluation. Each one of them is affected by different values of 3 variables. The combination of both the scenarios and variables values gives us 14 different experiments that are performed in our evaluation platform. In all the experiments the client starts with 100 HTTP processes (30.10^3 TCP connections) with the server. It ends with 500

HTTP processes (150.10^3 TCP connections). The scenarios define which entity controls the access of the communication between the client and server. The details of the three scenarios are as follows:

1. SDN proactive firewall: In this scenario, the SDN firewall is configured in the reactive mode. The NetFilter machine is deactivated. The reactive mode transforms the TCP FSM to its SEFSM. It installs in the network element machine the OpenFlow rules to control the access of the communications between the client and the server. We suppose that the topology is known. The firewall is in the static topology mode. It receives the topology data from the orchestrator; we deactivate the dynamic topology learning module of the firewall. The SDN firewall controls all the connections through the SDN Firewall Circuit shown in Figure 5.9.

2. SDN proactive firewall: in this scenario, the SDN firewall is configured in the proactive mode. The rest of the configurations are the like in the previous scenario.

3. NetFilter firewall: In this scenario, we activate the NetFilter machine. The SDN firewall is deactivated. We install in NetFilter the firewall rules to control the access between the client and the server. Besides, in the network element machine, we install two types of OpenFlow rules. The first one (Forward to NetFilter) send all the traffic coming from the client or the server to NetFilter machine. The second one (Forward From NetFilter) sends the traffic that originates from the NetFilter machine to its destination. The latter is the Client machine or the Server machine. NetFilter controls all the connection through the NetFilter Circuit shown in Figure 5.9.

Three variables characterize the scenarios mentioned above. Each time only one variable is active and the two others stay in their initial values. The details of the variables are as follows:

1. Access rules: This variable enables us to evaluate the impact of the size of access rules in the three scenarios. The behavior of each scenario is different from the others. In the proactive mode, the access rules are executed by the network element. In the reactive mode, they are executed by the firewall application. In NetFilter, the access rules are executed by the NetFilter rule engine. We evaluate the scenarios with one access rule (initial value) then with 10^3 access rules.

2. Data size: This variable enables us to evaluate the impact of the size of the transmitted data between the client and the server. The behavior of each scenario is different from the others. In the SDN Firewall, all the data pass by the network element. The SDN firewall receives only the conditions to move from a state to another. In NetFilter, all the data pass through NetFilter Machine to check the actual state of their connections and the possible transitions. We evaluate the scenarios by configuring the server processes. Each one is configured with a file of 1 kB that is requested by each client connection (initial value). Then, the size of the file is increased to 1 MB.

3. Syn Flooding attacks: this variable enables us to evaluate the impact of syn flooding rate on the performance of the three scenarios. We activate the Attacker machine. It injects syn flooding attacks with different ports towards the server. We authorize the attacker to communicate with the server in the three scenarios. In all the scenarios, we set the Syn flooding threshold to 1 for the attacker-server connections. When the threshold is reached, the behavior of each scenario is different from the others. The proactive mode blocks the syn flooding connection in the network element. The network element drops the Syn flooding traffic by preventing it from reaching the controller. The reactive mode drops the Syn Flooding traffic in the firewall application. NetFilter drops the Syn flooding

traffic using its Syn flooding protection module. The initial state of the variable is an attack rate of 0 (attacker is deactivated). Then, the attacker sends 10^3 Syn flooding per second in the second step along the test. The last value of the variable is an attacker that sends 16.10^3 Syn flooding per second; it exploits all its available bandwidth with the network element.

As a result, we perform 14 experiments by combining the three scenarios with the values of 3 variables. In each scenario, we have only one variable that is not in the initial state while the two others are in their initial values. The different experiments are as follows :

1. Reactive behavior with 1 access rule, 1 KB data and 0 Syn flooding attacks (Reactive-1R-1K-0SF).
2. Proactive behavior with 1 access rule, 1 KB data and 0 Syn flooding attacks (Proactive-1R-1K-0SF).
3. NetFilter with 1 access rule, 1 KB data and 0 Syn flooding attacks (NetFilter-1R-1K-0SF).
4. Reactive behavior with 10^3 access rule, 1 KB data and 0 Syn flooding attacks (Reactive-1000R-1K-0SF).
5. Proactive behavior with 10^3 access rule, 1 KB data and 0 Syn flooding attacks (Proactive-1000R-1K-0SF).
6. Reactive behavior with 1 access rule, 1 MB data and 0 Syn flooding attacks (Reactive-1R-1M-0SF).
7. Proactive behavior with 1 access rule, 1 MB data and 0 Syn flooding attacks (Proactive-1R-1M-0SF).
8. NetFilter with 1 access rule, 1 MB data and 0 Syn flooding attacks (NetFilter-1R-1M-0SF).
9. Reactive behavior with 1 access rule, 1 KB data and 10^3 Syn flooding attacks (Reactive-1R-1K-1KSF).
10. Proactive behavior with 1 access rule, 1 KB data and 10^3 Syn flooding attacks (Proactive-1R-1K-1KSF).
11. NetFilter with 1 access rule, 1 KB data and 10^3 Syn flooding attacks (NetFilter-1R-1K-1KSF).
12. Reactive behavior with 1 access rule, 1 KB data and 16.10^3 Syn flooding attacks (Reactive-1R-1K-16KSF).
13. Proactive behavior with 1 access rule, 1 KB data and 16.10^3 Syn flooding attacks (Proactive-1R-1K-16KSF).
14. NetFilter with 1 access rule, 1 KB data and 16.10^3 Syn flooding attacks (NetFilter-1R-1K-16KSF).

5.6.3 Evaluation Results

In each experiment, we evaluate the following metrics:

1. Connection Time (CT): it is the time that takes each connection between the client and the server. We calculate for each experiment the Average (Avg(CT)) , the Maximum (Max(CT)), the Minimum (Min(CT)), the Median (Med(CT) and the Standard Deviation (SD(CT)).

2. Packet Processing Time (PPT): We calculate the time of processing TCP control packets in the firewall application and NetFilter. We calculate for each experiment the Average (Avg(PPT)), the Maximum(Max(PPT)), the Minimum (Min(PPT)), the Median (Med(PPT)) and the Standard Deviation (SD(PPT)).

3. Percentage of Packet Re-transmissions (PPR): We calculate the percentage of TCP control packets that have been re-transmitted between the client and the server. The values are the ratio between the number of TCP control re-transmissions and the total number of TCP control packets.

Figure 5.10 shows the different values of the average connection times (Avg(CT)) according to the impacts of the experiments mentioned above. Figure 5.10a shows the values of Avg(CT) under initial conditions (1 firewall rule, 1 KB of data per connection and the attacker is deactivated). Figure 5.10b shows the impacts of the size of firewall rules on the values of Avg(CT). Figure 5.10c shows the impacts of the size of data transmitted in each connection on the values of Avg(CT). Figure 5.10d shows the impacts of Syn Flooding attacks on the values of Avg(CT). The rate of the attack exploits 10% of the bandwidth with OVS. Figure 5.10e shows the impacts of Syn Flooding attacks on the values of Avg(CT). The rate of the attack exploits all the bandwidth with OVS.

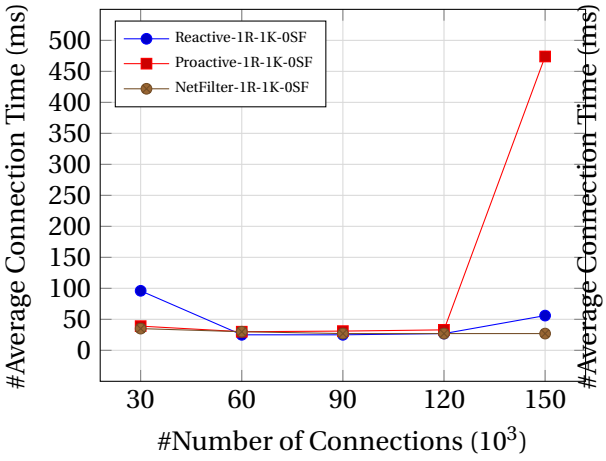
In the initial conditions (see Figure 5.10a), Avg(CT) for the reactive firewall is 96ms with 30.10^3 connections. For the proactive firewall, Avg(CT) is 39ms. For NetFilter, Avg(CT) is 35ms. Avg(CT) in the reactive firewall decreases until 60.10^3 connections while the number of connections raises. After an investigation, we find out that the client generates the connections differently when the load is 30.10^3 connections. It generates the connections according to a fluctuation between 0 and 70 connections each time (the generation of connections is steady in the other experiments). This fluctuation affects the processor occupation in OVS and the installation of rules by the controller. OVS processor is occupied only during 500ms (working cycle) in each cycle of 1ms.The controller installs its rules in this occupation interval. As a result, the processor resources are used in the working cycle then they are freed in the next cycle in each second. Besides, the event queue of RYU is also filled in the working cycle then destroyed in the next cycle in each second. The cost of these fluctuations of resources occupation and freeing is a bigger value of the connection time because in the other experiments the resources are continuously occupied.

From 60.10^3 connections to 120.10^3 connections all the Avg(CT) in the reactive firewall (25, 27, 27)ms, proactive firewall (30, 31, 33)ms and NetFilter (30, 27, 27)ms became steady and close. With 150.10^3 connections, Avg(CT) raises in the proactive firewall to 474ms while in NetFilter, it stays steady and in the reactive firewall it rises slightly to 56ms. The increase in the Proactive firewall is due to the occupation rate of OVS resources. With 150.10^3 connections in the proactive firewall, OVS processor occupation reaches an average of 95% and its memory occupation reaches an average of 84%.

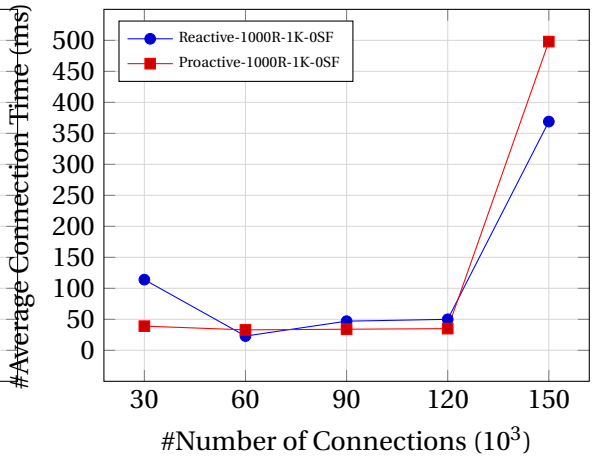
The size of the firewall rules does not impact the Avg(CT) in the proactive firewall in its access table because it delegates to OVS the matching of the traffic with these rules (see Figure 5.10b). In fact, its curve evolves with the rise in the number of connections similar to the initial conditions. Its values are almost similar to the results of the initial conditions. The tiny rise comparing to the initial conditions is around 3ms from 30.10^3 to 120.10^3 connections and 25 ms with 120.10^3 connections. However, the Avg(CT) in the reactive mode is affected by the access table because the firewall application maps the traffic to the rules. With 90.10^3 connections, Avg(CT) increases to 47ms then to 50ms with 120.10^3 connections. The increase becomes important with 150.10^3 connections. At this step, Avg(CT) reaches a value of 369ms. This increase is due to the occupation of the processor in the SDN Firewall Machine. The firewall application is multi-threaded but can only execute its tasks on one processor core because it is

Figure 5.10 – Average Connection Times according to the different experiments

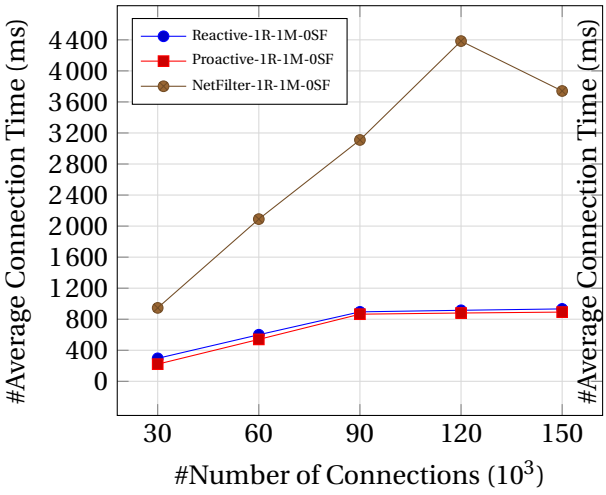
(a) Initial Conditions



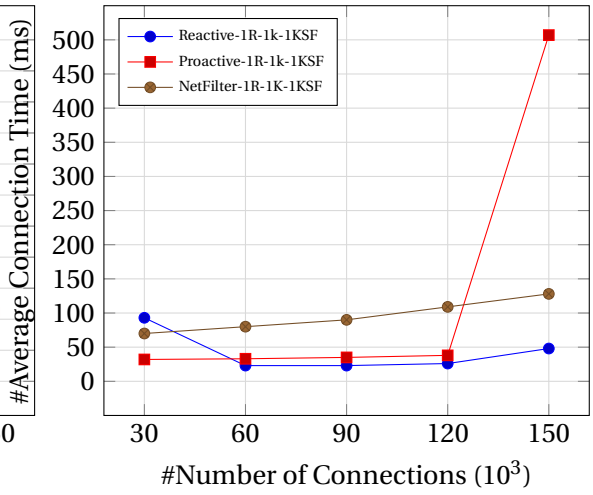
(b) Firawall Rules Impact



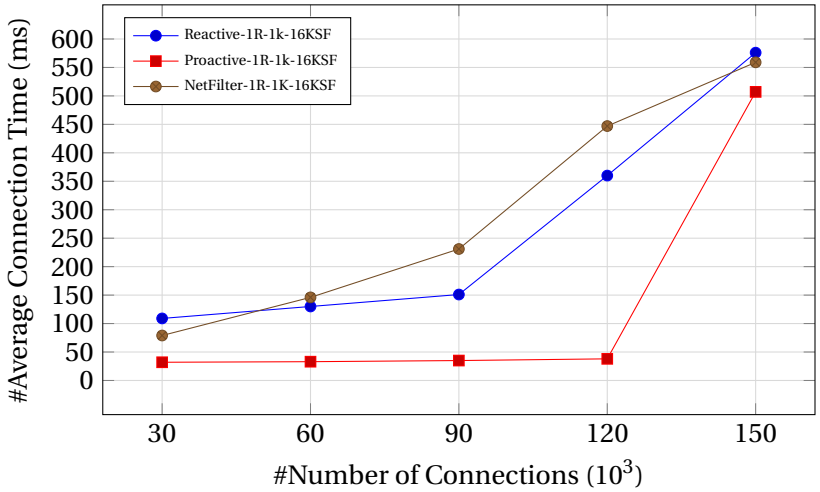
(c) Data Size Impact



(d) 1k Syn Flooding Impact



(e) 16k Syn Flooding Impact



developed in the Python language. Python does not support parallelism for security reasons.

The proactive firewall, the reactive firewall, and NetFilter are impacted by the size of data that are transported in the connections (see Figure 5.10c). However, NetFilter is the most impacted by this variable. Its Avg(CT) increases with a maximum factor of × 162 compared to its Avg(CT) in the initial conditions. With 30.10^3 connections, Avg(CT) with NetFilter is 946ms (increase of × 30 compared to the initial conditions). The values continue to increase linearly until 120.10^3 connections reaching an Avg(CT) value of 4384ms (increase of × 162 compared to the initial conditions). Then, this value decreases to 3741ms. The massive rise in NetFilter is due to its hardware resources occupation. NetFilter uses fully one processor core and an average of 90% of memory with 120.10^3 connections. Then, with 150.10^3 connections, it distributes the load to a second processor core while the memory becomes fully utilized. Besides, OVS memory is fully utilized from 90.10^3 connections. For this reason, Avg(CT) in the proactive and the reactive firewalls increases linearly respectively from 221ms and 296ms (30.10^3 connections) to 865ms and 895 ms (90.10^3 connections). Then the increase becomes tiny in both SDN firewalls until the end; an average increase of 15ms is observed with the proactive firewall and another of 20ms with the reactive firewall.

The proactive firewall is not impacted by the syn flooding attacks (see Figures 5.10d and 5.10e) because it drops the attacker traffic in OVS whether with 10^3 Syn Flooding or 16.10^3 Syn Flooding. In addition, the reactive firewall is not impacted by 10^3 Syn Flooding; the connection times stay steady and close to the values of the initial conditions. However, the values of the connection times increase with 16.10^3 Syn Flooding in the reactive firewall. Avg(CT) increases slightly from 109ms with 30.10^3 connections to 151ms with 90.10^3 connections. Then the rise becomes more important as it increases to 576ms with 150.10^3 connections. This rapid rise is due to the occupation of the memory and the processor in the reactive firewall application and in OVS. NetFilter is the most impacted by SynFlooding attacks whatever their rates. Its Avg(CT) increases from 70ms (with 30.10^3 connections) to 128ms (with 150.10^3 connections) under 10^3 Syn Flooding attacks. In this case, the connection times values reach 4 × their counterparts in the initial conditions. However, the rise becomes important under 16.10^3 Syn Flooding. Avg(CT) starts with 79ms (with 30.10^3 connections) and reaches a value of 559ms (with 150.10^3 connections). This increase is 20 × the value in the initial conditions.

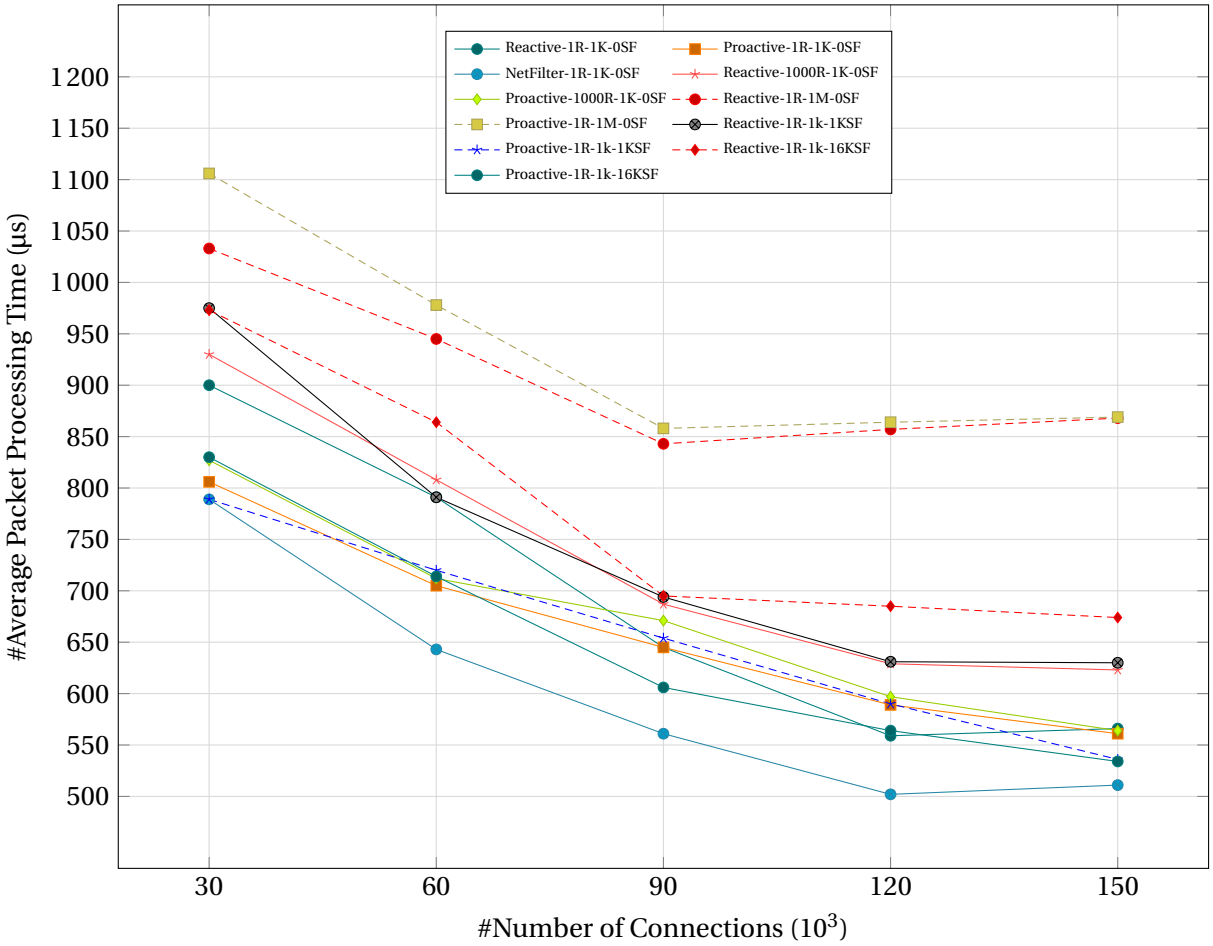
Figure 5.11a displays Avg(PPT) in µs for all the experiments with the proactive firewall and the reactive firewall. The figure includes also Avg(PPT) with NetFilter in the initial conditions. Figure 5.11b displays Avg(PPT) in ms for Syn Flooding attacks with NetFilter. Figure 5.11c displays Avg(PPT) in ms for data transport with NetFilter.

We observe in Figure 5.11a that all the Avg(PPT) values of the reactive firewall are higher than their counterparts with the proactive firewall. The only exception is Avg(PPT) values for data transport with the proactive mode which are the highest. Avg(PPT) is higher in the reactive firewall because it performs more processing in the firewall application than in the proactive firewall. In the initial conditions, Avg(PPT) values for the 3 Firewalls are close. They are between 900µs and 566µs for the reactive firewall, between 806µs and 561µs for the proactive firewall and between between 789µs and 511µs for NetFilter. These values slightly rise for the reactive and the proactive firewalls; the highest values for both firewalls belong to the experiments with data transport.

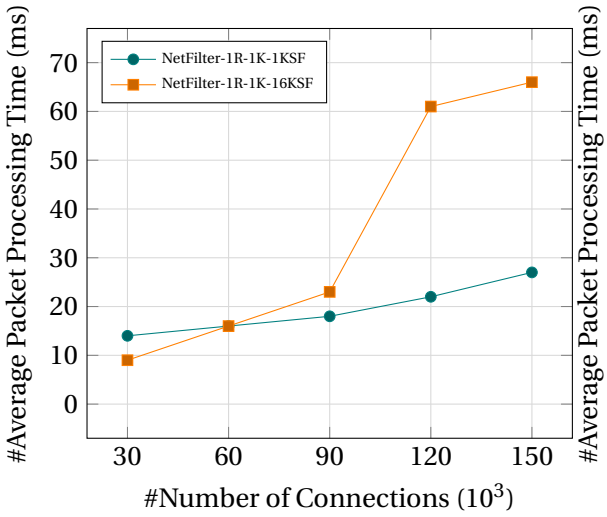
However, Avg(PPT) values for NetFilter are the highest outside the initial conditions. They increase highly to reach values in ms under Syn Flooding attacks and Data transport. In Figure 5.11a, we observe that both Avg(PPT) under 1k Syn Flooding and 16k Syn Flooding increase with the rise in the number of connections. The highest scores are in the case of 16k Syn Flooding attacks from 60.10^3 connections. The values reach a score of 66ms with 150.10^3 connections. These values are equivalent to 75 × the values of Avg(PPT) for the reactive and the proactive firewalls under the same experiment. Besides, Avg(PPT) values with data transport reach

Figure 5.11 – Average Packet Processing Times according to the different experiments

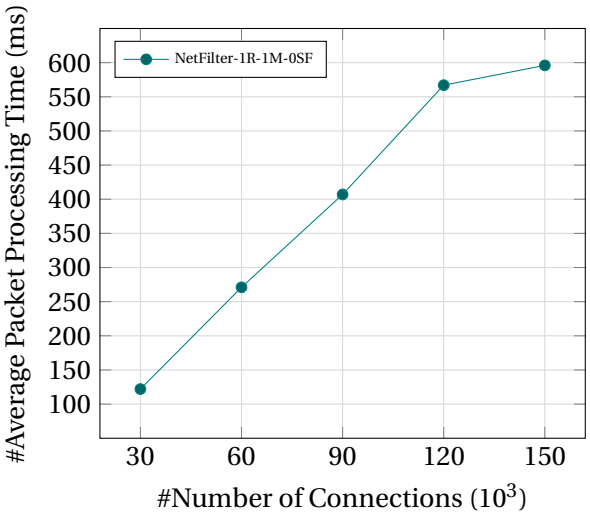
(a) TCP Control Packet Processing Time Part 1



(b) TCP Control PPT Part 2



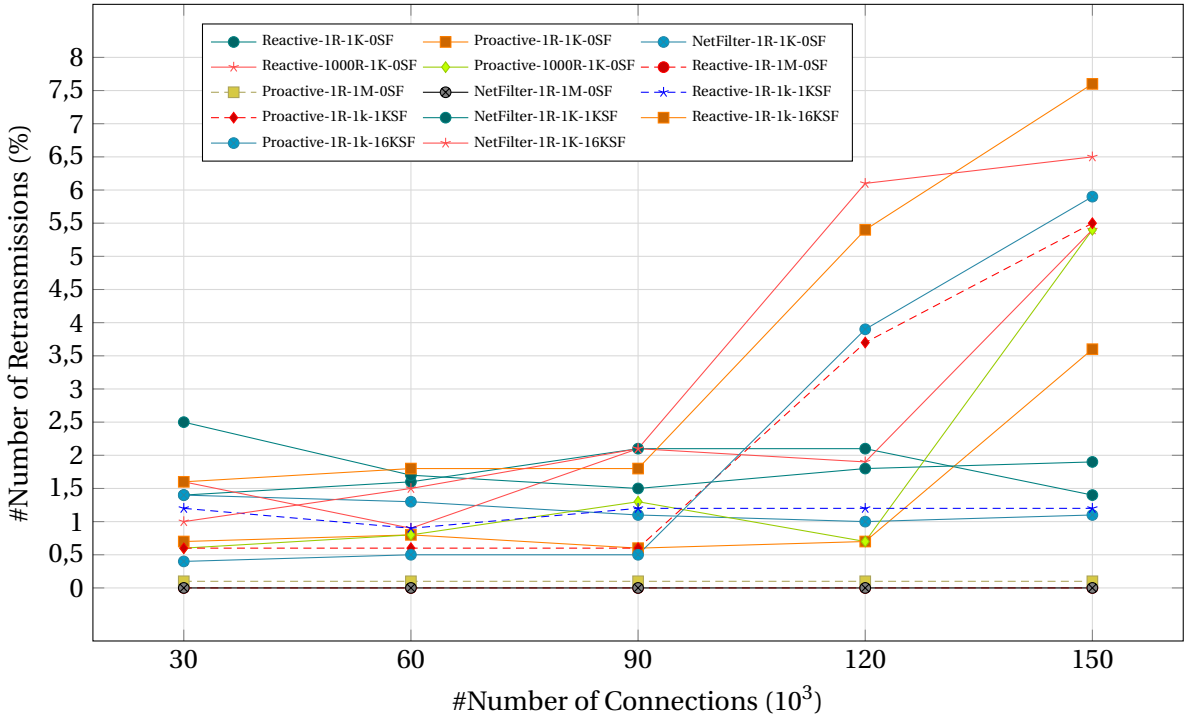
(c) TCP Control PPT Part 3



the maximum values for NetFilter. The performance of NetFilter is the lowest in this experiment. The values increase continuously from 122ms with 30.10^3 connections to reach 596ms with 150.10^3 connections. Avg(PPT) for NetFilter under Syn Flooding attacks and with data transport is one of the reasons while the connection times of NetFilter in the same conditions increase. NetFilter passes more time to process each control traffic. As a consequence, Avg(CT) increases in these experiments.

Furthermore, we observe that almost all Avg(PPT) of Figure 5.11a decrease along the rise in the number of connections. After an investigation, we find out that the number of control packets re-transmissions is one of the reasons for this decrease. The three firewalls process re-transmission packets more rapidly than new packets. They do not re-install new rules for them. They only check their legitimacy by matching them with the access table and the state table. Figure 5.12 shows the ratio of re-transmission packets compared to the total transmitted TCP control packets. We observe that all the firewalls in data transport experiments have the lowest ratio of re-transmissions; NetFilter and the reactive firewall do not comprehend any control packet re-transmission while in the proactive firewall, the ratio of retransmissions is 0.1% along the increase of the number of connections. Besides, we observe that the ratios of re-transmissions are almost steady for all the experiments between 30.10^3 and 90.10^3 connections. In this interval, the reactive mode under initial conditions has the highest ratio (2.5%) with 30.10^3. From 90.10^3 connections, the re-transmissions with NetFilter and the reactive firewall under 16k Syn Flooding and those of the proactive Firewall under 1k and 16k Syn Flooding overgrows. The retransmissions under 16k Syn Flooding for NetFilter (6.5%) and the reactive firewall (7.6%) are the highest among all the re-transmissions ratios. These results also fulfill the increase in the connection times for these experiments in Figure 5.10e. The increase of the ratios of the re-transmissions for the proactive Firewall under Syn Flooding attacks is because of the load on OVS. Its resources are highly occupied under Syn Flooding attacks which results in delaying the transmission of control packets.

Figure 5.12 – The ratios of TCP Control re-transmission packets



5.7 Discussion

We have discussed in this chapter our SDN statefull firewall solution. We have formalized all its concepts. Then, we have presented its design in terms of architecture and behavior. We presented two behviors. A reactive behavior that processes the FSM of a network protocol only from the actual state and in the firewall application before installing the firewall rules. Besides, we have presented the proactive behavior. In this behavior, the firewall application processes the FSM and pre-installs all the corresponding rules in the network elements. Besides, we have implemented our solution in the Python language. We have evaluated the performance and resistance of our firewall and NetFilter under different loads, experiments and attacks.

The conclusions are as follows. In the initial conditions, NetFilter and the reactive firewall have better performances while those of the proactive firewall starts to decrease from 120.10^3 connections. The number of rules impacts the reactive Firewall in its tables, and the rate of Syn Flooding attacks when it is 16.10^3/s because it processes all the events in the firewall application. The proactive firewall is neither impacted by the rates of Syn Flooding attacks nor by the number of rules because it delegates the processing to the network element. NetFilter is the most impacted by the rates of Syn Flooding attacks. All the Firewalls are impacted by the size of the data, especially NetFilter. In the case of our firewall, the reason is related to the increase of OVS processor and memory occupations. Besides, our Firewall processes TCP control packets during periods that are in µs. Apart from the initial state where NetFilter PPT is slightly better, the proactive firewall PTT is the better in the other experiments. NetFilter PPT is the highest in other experiments. It reaches values in ms. NetFilter re-transmission ratio also is the higher regarding re-transmission ratio because Syn Flooding attacks impact it.

In the next chapter, we focus on the management plane. We enhance the orchestrator of our solution by improving its programmability in the context of cloud computing. We integrate into the orchestrator a policy model that supports security policy expression, selection and negotiation for Network Service Providers (the owner of the SDN assets) and Network Service Customers (the entity that requires SDN assets).

Chapter 6

SDN Firewall Orchestration

" I have always believed that technology should do the hard work - discovery, organization, communication - so users can do what makes them happiest: living and loving, not messing with annoying computers!
That means making our products work together seamlessly. "

Larry Page

Contents

6.1 Introduction

The evolution of the digital world drives cloud computing as a key infrastructure for data and services. This breakthrough is transforming SDN into a Software as a Service because of its advantages such as automation, simplicity, agility and global knowledge [271]. As a result, many Network Service Providers (NSPs) select SDN as a cloud network service and offer it to their customers. However, due to the rising number of NSPs and their security obligations, Network Service Customers (NSCs) strive to find the best provider candidate who satisfies their security requirements.

In this context, we integrate to our orchestrator a policy orchestration framework for SDN firewall policies. Our enhanced orchestrator enables an NSC and SDN NSPs to express their firewall policies, asses them and to negotiate them. It enforces these security requirements using the holistic view of the SDN controllers, and it deploys the generated firewall policies to the firewall applications. We collaborate with another Ph.D. [272] to enhance our orchestrator.

The rest of the chapter is organized as follows: Section 6.2 speaks about the context of our contribution and its objectives. Section 6.3 describes all the processes of our SDN firewall policy model. It presents our policy language, the policy assessment process, the contract establishment process and the policy deployment process. These processes cover the firewall policy expression until the policies deployment on the network elements. Section 6.4 presents the implementation of the model into the orchestrator. Section 6.5 evaluates the performance of our solution.

6.2 Context and Objectives

Suppose we have an NSC that requires many firewall policies and an SDN service in the cloud to run its business. Besides, suppose we have many NSPs that offer different SDN deployments and different firewall capabilities. In this case, one needs to choose the best NSP that satisfies the NSC requirements. Then, the NSC needs to agree with the chosen NSP by a mutual contract on the properties of the SDN resources and the firewall policies. Besides, the cloud service needs to provide the SDN resources and the agreed firewall policies. Let us assume that there is in the cloud a well-established mechanism to provide the SDN resources. In our case, we focus on only the firewall policies as the variable that determines the expression of the requirements and the obligations, the selection of the best NSP, the negotiation of the contract and the deployment of the contract in the SDN resources of the chosen NSP. Thus our work integrates these expression, selection, negotiation and deployment processes into the orchestrator to satisfy the NSC regarding firewall policies.

The interactions between the NSC, the NSPs, and the orchestrator are as follows:

1. The NSC specifies its firewall policies requirements using the policy model offered by the orchestrator.

2. The NSPs specifies their firewall policies obligations using the policy model offered by the orchestrator.

3. The orchestrator assesses the requirements and the obligations to select the best NSP that satisfies the requirements.

4. The orchestrator starts a negotiation process between the selected NSP and the NSC to establish an agreement (Contract) on the firewall policies.

5. The orchestrator generates an agreement in the form of firewall policies.
6. The orchestrator saves the firewall policies into its global access table.
7. The orchestrator deploys the firewall policies on the SDN firewall applications.
8. The firewall applications interpret the policies and deploy them on the network elements.

6.3 SDN firewall policy model

Our solution tackles the lack of policy model that can support firewall policy expression, selection, negotiation, and deployment when dealing with multiple NSPs. Our contribution meets key-functional conditions for SDN as a cloud service. It addresses the security configuration at the management layer. It abstracts the complexity of the infrastructure to simplify the expression of the firewall policies. It provides a unified language to enable NSCs and NSPs to express their requirements and obligations. It classifies different rule relations between NSCs and NSPs to select the best NSP. It provides a negotiation protocol for NSC and NSP to reach an agreement. It interprets the agreements to firewall policies and deploys them on the firewall applications. To the best of our knowledge, no method in the literature considers all these points.

6.3.1 Use case description

We introduce a use case to ease the comprehension of our model and to illustrate each process by concrete examples. The subjects involved in the use case are an NSC, an orchestrator, and 3 NSPs. NSC requires an SDN firewall service and SDN resources that meet its firewall policies (requirements). Each NSP provides a type of an SDN firewall and a set of firewall policies (obligations). The three SDN firewall services are as follows:

1. **NSP1: SDN Reactive Stateful Firewall**: The firewall application processes the connections based on a reactive mode (see Section 5.4.4 of Chapter 5).
2. **NSP2: SDN Proactive Stateful Firewall**: The firewall application processes the connections based on a proactive mode (see Section 5.4.4 of Chapter 5).
3. **NSP3: Stateless SDN Firewall**: The firewall application behaves as a stateless firewall. It relies only on its access table to deny or accept the traffic.

6.3.2 Firewall policy language

We propose an SDN firewall policy language to homogenize NSP's obligations and NSC's requirements. The proposed language is inspired from the Attribute-Based Access Control Model (ABAC) [273–275]. It allows expressing firewall policies based on a common template. This unification guarantees the interoperability between the obligations and the requirements. The grammar of our language is as follows:
Π is the set of all the firewall policies. It describes the access controls within the dynamic environment of the allocated cloud resources: $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$ where $\pi_{i\_1..m}$ are firewall policies.
Θ is a set of obligations. It encompasses all the firewall policies of Π expressed by NSPs. $\Theta = \{\theta_1, \theta_2, ..., \theta_k\}$
Φ is a set of requirements. It encompasses all the firewall policies of Π expressed by NSC. $\Phi = \{\phi_1, \phi_2, ..., \phi_j\}$
Where $\Pi = \Theta \cap \Phi$

Each firewall policy π_i is formed by many atomic elements $\varepsilon_{i_{=1..n}}$:
$\pi_{i_{=1..m}} \equiv \varepsilon_1 \wedge \varepsilon_2 \wedge ... \wedge \varepsilon_n$
$\varepsilon_{i_{=1..n}}$ is defined by a preposition of predicates. Each predicates defines a propriety of the $element$.

Proposition 6.3.1. $A(\varepsilon_i)$ *and* $B(\varepsilon_i)$ *are two predicates defining* ε_i *proprieties. Predicates equivalence is determined by the preposition :* $A(\varepsilon_i) \in \Omega, B(\varepsilon_i) \in \Psi \mid (\Psi = \Omega) \rightarrow (A(\varepsilon_i) \equiv B(\varepsilon_i))$.

The atomic rule element $\varepsilon_{i_{=1..n}}$ is formed by the following predicates :

1. **Type:** $type(\varepsilon_i) \equiv subject(\varepsilon_i) \vee action(\varepsilon_i) \vee object(\varepsilon_i) \vee context(\varepsilon_i)$

2. **Domain:** $domain(type(\varepsilon_i)) \in \{protocol, time...\}$. Domain restricts the unit of an element.

3. **Value:** $value(type(\varepsilon_i)) \equiv variable(type(\varepsilon_i)) \vee non\text{-}variable(type(\varepsilon_i))$.
   $variable$ has not an assigned value while $non\text{-}variable$ has an already assigned value. Both $variable$ and $non\text{-}variable$ can be assigned by three kinds of data types:

   (a) **constant:** numeric or semantic value, ex. $value(type(\varepsilon_i)) = \text{TCP}$.

   (b) **interval**: numeric interval, ex. $value(type(\varepsilon_i)) = [8:00,\ 20:00]$

   (c) **set**: a collection of values, ex. $value(type(\varepsilon_i)) = \{15:00, 16:00\}$

   For simplification, we use x_i to present a $variable$, $x_i \equiv variable(type(\varepsilon_i))$

4. **Scope**: it defines the access to the values of a $variable$. It can be:

   (a) **Public preference**: $\mathbf{pub_{pre}}(x_i)$ a public preference $variable$ is accessible as public information.

   (b) **Private preference**: $\mathbf{pri_{pre}}(x_i)$ a private preference $variable$ is a local configuration that can not be disclosed.

If $context$ is not specified in a policy we add a universal context element $\top$. It indicates that all the obligations for the context are acceptable.
$(context(\varepsilon_i) \equiv \top) \rightarrow ((domain(context(\varepsilon_i)) \equiv \top) \wedge (value(context(\varepsilon_i)) \equiv \top))$
Finally we write:
$\varepsilon_i \equiv type(\varepsilon_i) \wedge domain(type(\varepsilon_i)) \wedge value(type(\varepsilon_i)) \wedge (\mathbf{pub_{pre}}(x_i) \vee \mathbf{pri_{pre}}(x_i))$
When the scope is not defined: $\varepsilon_i \equiv type(\varepsilon_i) \wedge domain(type(\varepsilon_i)) \wedge value(type(\varepsilon_i))$
The firewall policies given in 6.3.2 using our language are defined in Table 6.1.

6.3.3 Firewall policy assessment

The assessment of firewall policies is based on matching the obligations with the requirements in order to determine which NSPs' policies satisfy NSC's requests. This process is used by the selection algorithm to determine which NSP satisfies the NSC. Besides, it depends on two level of relationships. Element-Element relation which relies on mapping the predicates of the firewall policies elements. The second level (Policy-Policy relation) focuses on finding the relationship between the matched elements.

Table 6.1 – Firewall Policy Expression for NSC, NSP1, NSP2 and NSP3

NSC	**Requirement** ϕ_1				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	time
	value	{NSC, NSP}	pass	{HTTP, TCP, ICMP}	[0:00,24:00]
	Scope	-	-	-	-
	Requirement ϕ_2				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	connection_exc
	value	{NSC, NSP}	x_2	TCP	TCP_failed_Time >3
	Scope	-	**pri**$_{\textbf{pre}}$ ({quarantine, block, alert})	-	-
	Requirement ϕ_3				
	Element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	attack_detection
	value	{NSC, NSP}	block	ICMP	DoS_detection
	Scope	-	-	-	-
NSP1	**Obligation** θ_1				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	time
	value	NSP	x_2	x_3	x_4
	Scope	-	**pub**$_{\textbf{pre}}$({pass, block})	**pub**$_{\textbf{pre}}$({HTTP, TCP, ICMP})	$\top$
	Obligation θ_2				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	connection_exc
	value	NSP	x_2	x_3	x_4
	Scope	-	**pri**$_{\textbf{pre}}$({block, alert})	**pub**$_{\textbf{pre}}$({TCP, HTTP, SSH, ICMP})	$\top$
NSP2	**Obligation** θ_1 **(same policy as NSP1)**				
	Obligation θ_2 **(same policy as NSP1)**				
	Obligation θ_3				
	Element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	attack_detection
	value	NSP	block	x_3	x_4
	Scope	-	-	**pub**$_{\textbf{pre}}$({HTTP, TCP,SSH, ICMP})	**pub**$_{\textbf{pre}}$({Poisoning, DoS_detection})
NSP3	**Obligation** θ_1 **(same policy as NSP1)**				
	Obligation θ_4				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	IP_address	$\top$
	value	NSP	x_2	x_3	x_4
	Scope	-	**pub**$_{\textbf{pre}}$({pass, block})	$\top$	$\top$

Element-Element relation

There are five relations between the elements:

1. **inconsistent**: $(\ type(\varepsilon_i) \not\equiv type(\varepsilon_j)) \rightarrow (\varepsilon_i \dashv\vdash \varepsilon_j)$. If two rule elements ε_i and ε_j have not equivalent $type$ predicates then they are in $inconsistent$ relation denoted: $\varepsilon_i \dashv\vdash \varepsilon_j$. For example, in Table 6.1, $\phi_1.\varepsilon_1 \dashv\vdash \theta_1.\varepsilon_2$ because $subject(\phi_1.\varepsilon_1) \not\equiv action(\theta_1.\varepsilon_2)$

   Proposition 6.3.2. *Not equivalence of $type$ is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \not\subseteq \Psi) \wedge (\Psi \not\subseteq \Omega)) \rightarrow (type(\varepsilon_i) \not\equiv type(\varepsilon_j))$

2. **comparable**: $((type(\varepsilon_i) \equiv type(\varepsilon_j)) \wedge (domain(\varepsilon_i) \cong domain(\varepsilon_j))) \rightarrow (\varepsilon_i \sim \varepsilon_j)$. If two rule elements ε_i and ε_j have equivalent $type$ predicates and their domain predicates are in congruence, then they are in $comparable$ relation. It is denoted with $\varepsilon_i \sim \varepsilon_j$. For example, in Table 6.1, $\phi_1.\varepsilon_2 \sim \theta_1.\varepsilon_2$ because their $subject$ predicates are equivalent and their $domain$ predicates are congruent.

   Proposition 6.3.3. *Domain congruence is defined as follows:*
   $domain(\varepsilon_i) \in \Omega, domain(\varepsilon_j) \in \Psi \mid ((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (domain(\varepsilon_i) \cong domain(\varepsilon_j))$

3. **equal**: $((\varepsilon_i \sim \varepsilon_j) \wedge (value(type(\varepsilon_i)) \cong value(type(\varepsilon_j)))) \rightarrow (\varepsilon_i = \varepsilon_j)$. If two rule elements ε_i and ε_j are $comparable$ and their values predicates are in congruence, then they are in $equal$ relation denoted with $\varepsilon_i = \varepsilon_j$. For example, in Table 6.1, $\phi_2.\varepsilon_3 = \theta_3.\varepsilon_3$ because both elements are $comparable$ and their $value$ predicates are congruent ($\{TCP\} \subseteq \{HTTP, TCP, SSH, ICMP\}$).

   Proposition 6.3.4. *Value congruence is defined as follows:*
   $value(type(\varepsilon_i)) \in \Omega, value(type(\varepsilon_j)) \in \Psi \mid$
   $((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (value(type(\varepsilon_i)) \cong value(type(\varepsilon_j)))$

4. **unequal**: $((\varepsilon_i \sim \varepsilon_j) \wedge (value(type(\varepsilon_i)) \not\equiv value(type(\varepsilon_j)))) \rightarrow (\varepsilon_i \neq \varepsilon_j)$. If two rule elements ε_i and ε_j are $comparable$ but do not have equivalent value, they are in $unequal$ relation denoted with $\varepsilon_i \neq \varepsilon_j$. For example, in Table 6.1, $\phi_1.\varepsilon_2 \neq \theta_3.\varepsilon_2$ because both are comparable however they have not equivalent values ($pass \not\equiv block$).

   Proposition 6.3.5. *Not equivalence of $value$ is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \not\subseteq \Psi) \wedge (\Psi \not\subseteq \Omega)) \rightarrow (value(type(\varepsilon_i)) \not\equiv value(type(\varepsilon_j)))$

5. **incomparable**: $((type(\varepsilon_i) \cong type(\varepsilon_j)) \wedge (domain(\varepsilon_i) \not\equiv domain(\varepsilon_j))) \rightarrow (\varepsilon_i \nsim \varepsilon_j)$. If two rule elements ε_i and ε_j have equivalent $type$ predicates and not equivalent $domain$ predicate, then they have $incomparable$ relation denoted with $\varepsilon_i \nsim \varepsilon_j$. For example, in Table 6.1, $\phi_1.\varepsilon_3 \nsim \theta_4.\varepsilon_3$ because they have congruent $type$ predicates but their domains are not equivalent ($protocol \not\equiv IP\_address$).

   Proposition 6.3.6. *Congruence of $type$ is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (type(\varepsilon_i) \cong type(\varepsilon_j))$

Policy-Policy relations

We derive from Element-Element relations, three relations between policies. These relations are as follows:

1. **match**: $(P_1 \wedge P_2 \wedge ... \wedge P_n) \rightarrow (\pi_\alpha \bowtie \pi_\beta)$
   $P_i \equiv ((\pi_\alpha.\varepsilon_i = \pi_\beta.\varepsilon_1) \vee ... \vee (\pi_\alpha.\varepsilon_i = \pi_\beta.\varepsilon_n)) \mid \pi_\alpha, \pi_\beta \in \Pi,\ i = 1..n$
   If any $element$ of a policy α is in equal relation with another element of a policy β, then the two policies are in $match$ relation denoted with $\pi_\alpha \bowtie \pi_\beta$. For example, in Table 6.1, $\phi_3 \bowtie \theta_3$ because $(\phi_3.\varepsilon_1 = \theta_3.\varepsilon_1) \wedge (\phi_3.\varepsilon_2 = \theta_3.\varepsilon_2) \wedge (\phi_3.\varepsilon_3 = \theta_3.\varepsilon_3) \wedge (\phi_3.\varepsilon_4 = \theta_3.\varepsilon_4)$.

2. **mismatch**: $\exists\varepsilon_i \exists\varepsilon_j(\pi_\alpha.\varepsilon_i \nsim \pi_\beta.\varepsilon_j) \rightarrow (\pi_\alpha \asymp \pi_\beta) \mid \pi_\alpha, \pi_\beta \in \Pi$. If there are at least $incomparable$ elements ε_i and ε_j from two policies π_α and π_β, then the two policies have $mismatch$ relation denoted with $\pi_\alpha \asymp \pi_\beta$. For example, in Table 6.1, $\phi_3 \asymp \theta_2$ because $\phi_3.\varepsilon_4 \nsim \theta_2.\varepsilon_4$.

3. **potential match**: $((\forall\varepsilon_i \forall\varepsilon_j(\pi_\alpha.\varepsilon_i \sim \pi_\beta.\varepsilon_j)) \wedge (\exists\varepsilon_k \exists\varepsilon_l(\pi_\alpha.\varepsilon_k \neq \pi_\beta.\varepsilon_l))) \rightarrow (\pi_\alpha \propto \pi_\beta) \mid \pi_\alpha, \pi_\beta \in \Pi$. If all the elements of the policies π_α and π_β are $comparable$ but it exists at least an $unequal$ relation between two of their respective elements then the two policies are in a $potential\ match$ relation denoted $\pi_\alpha \propto \pi_\beta$. For example, in Table 6.1, $\phi_1 \propto \theta_1$ because $((\phi_1.\varepsilon_1 \sim \theta_1.\varepsilon_1) \wedge (\phi_1.\varepsilon_2 \sim \theta_1.\varepsilon_2) \wedge (\phi_1.\varepsilon_3 \sim \theta_1.\varepsilon_3) \wedge (\phi_1.\varepsilon_4 \sim \theta_1.\varepsilon_4)) \wedge (\phi_1.\varepsilon_1 \neq \theta_1.\varepsilon_1)$.

NSP selection

The orchestrator ranks each NSP based on the relations between the $requirements$ and the $obligations$ (see Section 6.3.2). This process enables the selection of the most compliant NSP according to the algorithm 1. Our Algorithm is simple. A more sophisticated version is proposed by work of [276] based on computing the similarity between the NSP and THE NSC. This part is integrated in the PHD thesis of our contributor [272].

Algorithm 1 NSP Ranking

```
rank_list is Empty {Initial ranking list is Empty}
for All NSPs do
    rank_table[Length(Requirement)] ← 0 {Default ranking Table initialized with 0 values}
    noncompliant ← False {To test if the NSP is not compliant}
    silver ← False {To test if the NSP is silver}
    gold ← False {To test if the NSP is gold}
    for i=0, i≤Length(Requirement)-1, i++ do
        for j=0, j≤Length(Obligation)-1, j++ do
            if Match(Requirement[i], Obligation[j]) == True then
                rank_table[i] ← 2
                Break from the current For
            else if Potential_Match(Requirement[i], Obligation[j]) == True then
                rank_table[i] ← 1
            end if
        end for
    end for
    for i=0, i≤Length(rank_table)-1, i++ do
        if rank_table[i] == 2 then
            gold ← True
        else if rank_table[i] == 1 then
            silver ← True
        else if rank_table[i] == 0 then
            noncompliant ← True
        end if
    end for
    if noncompliant == True then
        print NSP is not compliant with NSC, it will not be add to the ranking list
    else if silver == True then
        Add (rank_list, (NSP,NSP_silver)){Tag NSP as NSP silver and add it to the ranking list}
    else if gold == True then
        Add (rank_list, (NSP, NSP_gold)){Tag NSP as NSP gold and add it to the ranking list}
    end if
end for
return rank_list
```

The results of the assessment and selection processes for the example of Section 6.3.1 are as follows. The orchestrator finds $potential\ match$ relations between the pairs: (ϕ_1, θ_1), (ϕ_2, θ_2) and (ϕ_3, θ_3). NSP1 does not meet the firewall requirement of ϕ_3 and NSP3 does not fulfill ϕ_2 and ϕ_3. As a consequence, the resulting relations are $mismatch$. The orchestrator puts NSP2 into the ranking list.

6.3.4 Contract establishment

The orchestrator generates the contract between the chosen NSP and NSC directly if the former is gold. When NSP is not gold, it starts a negotiation process between the NSC and the first ranked NSP. Upon the negotiation, the orchestrator can generate a new contract if both

parties accept the negotiated terms. Algorithm 2 describes these steps. It illustrates the contract building process conducted by the orchestrator. The latter chooses the NSP_{top} in the top of $rank\_list$. The orchestrator accepts directly without negotiation NSP_{gold} and establishes a contract with NSC. While for NSP_{silver}, it starts a negotiation process with NSC by executing the proposed negotiation protocol (see Table 6.2). It transforms $potential\ match$ relations into $match$ relations. If the negotiation fails NSP is deleted from $rank\_list$ and the negotiation process is re-conducted.

Algorithm 2 Establishment of contract

```
1: rank_list← call (NSP Ranking) {Making NSP Ranking List}
2: while NSP in rank_list do
3:     Best_NSP = NSP_top {Choose NSP_top from rank_list}
4:     if Best_NSP is NSP_gold then
5:         Accept (Best_NSP.Obligation()) { Accept NSP_top offer }
6:         Contract =Generate_Contract (Best_NSP, NSC) {Establish contract with NSC}
7:         return
8:     else
9:         Negociate (Best_NSP.Obligation()), NSC.Requirement()) {Start negotiation with NSC}
10:        negotiation_Result = RENP (potential match) {Execute negotiation protocol between
           potential match rule pairs}
11:        if negotiation_Result = Accepted then
12:            Contract =Generate_Contract (NSP, NSC) {Establish contract with NSC}
13:            return
14:        else
15:            Delete (NSP_top, rank_list) { Delete NSP_top from rank_list }
16:        end if
17:    end if
18: end while
```

Negotiation protocol

When an agreement is not reached between the peers, the orchestrator negotiates the offers of the chosen NSP with NSC. We propose the Rule-Element Based Negotiation Protocol (RENP) in order to manage the negotiation process. Our protocol specifies for each element the next action regarding the proposed values (v_{rec}) of NSP and the local configuration (v_{loc}) of NSC. The protocol contains three types of actions:

1. **accept**: it indicates that the proposed value is agreed.
2. **refuse**: it indicates that the proposed value is aborted.
3. **propose**: it generates a counter-offer .

Table 6.2 presents RENP. (vp) is the proposed variable upon negotiation. The details of the negotiation protocol are as follows:

1. **Row 1: Local value is a non-variable.** This case indicates that the local value (v_{loc}) is unchangeable and not negotiable.
   (a) **Column 1**: The received non-variable is compared directly with the local one. As non-variable does not hold negotiation possibility, only equal value leads to $accept$ action.
   (b) **Column 2**: The received value is a variable with public preference $pub_{pre_{rec}}$, when (v_{loc}) is the subset of $pub_{pre_{rec}}$, the receiver ensures that the received value can be accepted by the sender. Then it sends v_{loc} in order to obtain the acceptance confirmation.
   (c) **Column 3**: The received variable indicates that the sender has its private constraint. The receiver proposes its local value v_{loc}.

Table 6.2 – RENP protocol

v_{loc} / v_{rec}	non variable	variable **pub$_{pre}$**	variable **pri$_{pre}$**	proposed value (**vp**)
non variable	$(v_{rec} = v_{loc}) \rightarrow accept(v_{rec})$ $(v_{rec} \neq v_{loc}) \rightarrow refuse$	$(\{v_{loc}\} \subseteq \{pub_{pre_{rec}}\}) \rightarrow propose(v_{loc})$ $(\{v_{loc}\} \nsubseteq \{pub_{pre_{rec}}\}) \rightarrow refuse$	$propose(v_{loc})$	-
variable **pub$_{pre}$**	$(\{v_{rec}\} \subseteq \{pub_{pre_{loc}}\}) \rightarrow accept(v_{rec})$ $(\{v_{rec}\} \nsubseteq \{pub_{pre_{loc}}\}) \rightarrow refuse$	$((\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) \neq \emptyset) \rightarrow propose(x)$ $x = (\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\})$ $((\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) = \emptyset) \rightarrow refuse$	$propose(x)$ $x = pub_{pre_{loc}}$	$(\{vp_{rec}\} \subseteq \{pub_{pre_{loc}}\}) \rightarrow accept(vp_{rec})$ $(\{vp_{rec}\} \nsubseteq \{pub_{pre_{loc}}\}) \rightarrow refuse$
variable **pri$_{pre}$**	$(\{v_{rec}\} \subseteq \{pri_{pre_{loc}}\}) \rightarrow accept(v_{rec})$ $(\{v_{rec}\} \nsubseteq \{pri_{pre_{loc}}\}) \rightarrow refuse$	$((\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) \neq \emptyset) \rightarrow propose(x)$ $x = (\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\})$ $((\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) = \emptyset) \rightarrow refuse$	$propose(x)$ $x \in \{pri_{pre_{loc}}\}$	$(\{vp_{rec}\} \subseteq \{pri_{pre_{loc}}\}) \rightarrow accept(vp_{rec})$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) = \emptyset) \wedge \neg negotiate) \rightarrow refuse$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) = \emptyset) \wedge negotiate) \rightarrow propose(x)$ $x \in \{pri_{pre_{loc}}\}$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) \neq \emptyset) \wedge (\{vp_{rec}\} \nsubseteq \{pri_{pre_{loc}}\}) \wedge negotiate) \rightarrow propose(x)$ $x \in (\{vp_{rec}\} \cap \{pri_{pre_{loc}}\})$

(d) **Column 4**: the case does not exist in RENP protocol because the local variable is not negotiable.

2. **Row 2: Local value is a variable with public preference.** This case indicates that the local variable should be assigned by negotiation.

   (a) **Column 1**: The received non-variable is accepted if it is a subset of the local public preference $pub_{prec_{loc}}$.

   (b) **Column 2**: If $pub_{prec_{rec}}$ has intersection with $pub_{prec_{loc}}$, the intersection is sent as a proposition in order to obtain confirmation.

   (c) **Column 3**: Upon receiving a variable, $pub_{prec_{loc}}$ is sent as a proposition.

   (d) **Column 4**: The proposed value vp is accepted if it is a subset of $pub_{prec_{loc}}$.

3. **Row 3: Local value is a variable with private preference**. Instead of disclosing private preference $pri_{pre_{loc}}$, the local variable is assigned by negotiation process.

   (a) **Column 1**: The received non-variable is accepted if it is a subset of $pri_{pre_{loc}}$.

   (b) **Column 2**: If received public preference $pub_{pre_{rec}}$ has intersection with $pri_{pre_{loc}}$, then their intersection is sent as a proposition in order to obtain confirmation.

   (c) **Column 3**: Upon receiving a variable, as $pri_{pre_{loc}}$ can not be completely disclosed, a value belongs to $pri_{pre_{loc}}$ is generated according to a local strategy. Then, it is sent.

   (d) **Column 4**: The proposed value vp_{rec} is accepted if it is a subset of $pri_{pre_{loc}}$. It is refused if its intersection with $pri_{pre_{loc}}$ is empty and the local negotiation condition

does not hold at the same time. In the opposite case, another value that belongs to $pri_{pre_{loc}}$ is generated according to the local strategy. It is sent as a counter offer. Otherwise, the intersection between vp_{rec} and $pri_{pre_{loc}}$ is proposed.

The orchestrator in our use case conducts policy negotiation with NSC and NSP2. The orchestrator accepts the obligations θ_1 for ϕ_1 and θ_3 for ϕ_3. However, it proposes a counter offer for ϕ_2. This case corresponds to column 5 in row 4 of table 6.2 because the received value (in ϕ_2) $vp_{rec} : quarantine$ has no intersection with $pri_{pre_{loc}} : \{block, alert\}$ (in θ_2). Thus, the orchestrator chooses another value = $block$ in $pri_{pre_{loc}}$ as a new proposition. After receiving the proposition, NSC accepts the new value because $block$ belongs to the local private configuration of ϕ_2. Then the orchestrator establishes the contract between NSC and NSP2 (see Table 6.3).

Table 6.3 – Final agreement between NSP2 and NSC

NSC	**Policy 1**				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	time
	value	NSP	pass	{HTTP, TCP, ICMP}	[0:00,24:00]
	Policy 2				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	connection_exc
	value	NSP	block	TCP	TCP_failed_Time>3
	Policy 3				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	attack_detection
	value	NSP	block	ICMP	DoS_detection

6.3.5 Firewall policies deployment

SDN NSP contains two levels of policy abstraction: (1) a service level abstraction which defines the business logic. Administrators and tenants express this high level. (2) An OpenFlow level which interprets the high-level abstraction into infrastructure specific rules. The abstraction at the service level hides the details of the network configuration and service deployment. It simplifies the expression of the service policies while the OpenFlow level ensures deploying the policies into the network elements according to the network state.

The orchestrator sends the high-level policies to SDN firewall applications. Each one interprets the high-level policies into OpenFlow rules and sends them to the controller.

The interpretation process is based on mapping the elements of the high-level policy model into OpenFlow elements. A high-level policy can be interpreted into more than one OpenFlow rule.

OpenFlow is based on flow rules. Mainly, it structures policies into six parts:

1. OF.Type can be *Flow ADD rules*, *Flow MODIFY rules* and *Flow DELETE rules*.

2. $Matching\ Fields$ define the characteristics of the traffic. They describe the header of a packet to identify network flows.

3. A$ctions$ specify the operations on the traffic. These actions can be Drop traffic, F$orward\ to\ controller$, F$orward\ to$ Port.

4. T$imers$ indicate the lifetime of the rule (*Hard_Timeout*) or the ejection time if the rule does not match for the time interval (Idle_T$imeout$).

5. M$etadata$ can be used to save any extra information.

6. C$ounters$ enable the controller to specify OpenFlow rules based on traffic statistics.

Table 6.4 shows the interpretation of the final agreement (Table 6.3) into OF rules. We applied the following mappings.

1. O$bject$ corresponds to OF M$atching$ F$ields$. It is the first element which is mapped to its OF counterparts. In our example: O$bject$ = $protocol$ = {UDP, ICMP} corresponds to two dependent OF M$atching$ F$ields$: ETH_Type and IP_P$roto$.
UDP corresponds to (ETH_Type = 2048, IP_P$roto$ = 17).
ICMP corresponds to (ETH_Type = 2048, IP_P$roto$ = 1). The interpretation of the object element will generate at least an OF rule for each object value.

2. A$ction$ of the high-level policy corresponds to OpenFlow A$ction$ F$ield$. OpenFlow also offers the possibility to express many actions (ACTION_List) and to associate them with the same OF rule. The orchestrator verifies that there is no contradiction between actions (for example $block$ and $allow$) in the same policy. For example, $block$ corresponds to the OF action: DROP.

3. C$ontext$ element is mapped to OpenFlow components such as TIME_OUTs and OpenFlow C$ounters$ but also to firewall specific functions. The interpretation of the firewall function triggers a condition to execute the OpenFlow rule of the C$ontext$.
For example, the context: TCP__F$ailed_Time$ > 3 in $policy2$ triggers TCP connection counting function and when it exceeds 3 connections Synchronization attempts, it installs the corresponding rule for $policy2$.

4. NSP is mapped to the topology of the service provider. For each link between the nodes, the interpretation module generates the corresponding OF rules taking the three mappings mentioned above. The OF Matching fields that correspond to the topology are at least IP_{src}, IP_{dst}, $PORT_{src}$, and $PORT_{dst}$. If the topology is not provided, the firewall application installs the OF rules without specifying the topology matching fields.

5. The default type of OF rule is ADD. The firewall application verifies firstly that the rule is not a duplicate of a previous interpreted rule by comparing both parts. If only the contexts are different, then the OF rule type is set to MODIFY. If the firewall application receives from the Controller an error upon sending a MODIFY rule, the firewall application changes the MODIFY rule to ADD rule and re-sends it to the controller.

Finally, the agreed contract in our use case (see Table 6.3) is interpreted to OpenFlow rules by the firewall application (see Table 6.4). Then, it is deployed on the network elements. **Policy 1** is interpreted to two Flow Mod ADD OpenFlow rules. **Policy 2 and Policy 3** are interpreted to two Flow Mod Modify OpenFlow rules. Besides, **Policy 2** generates a call for the Syn Flooding Function. **Policy 3** generates also a call to a function that alerts SNORT IDS if a DDOS will happen.

Table 6.4 – Interpretation of the Final Agreement into OpenFlow Rules

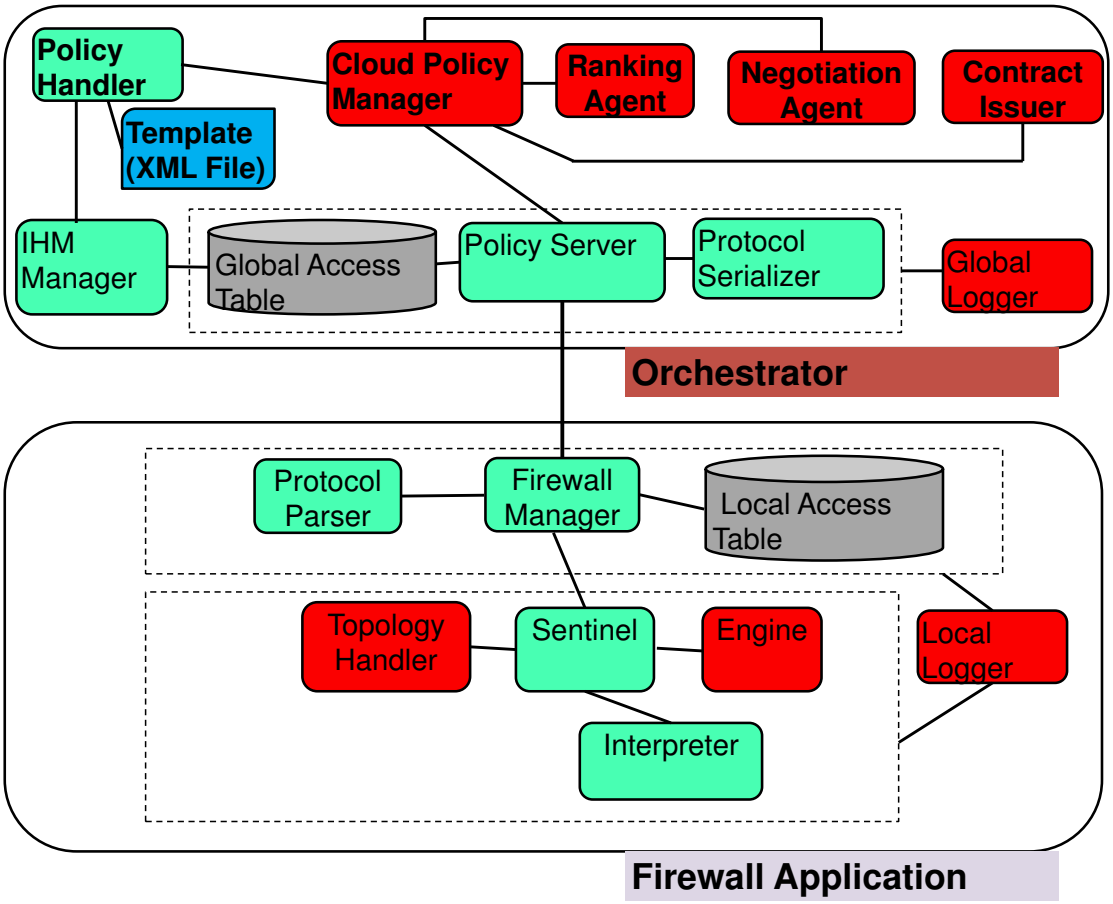
	OF Type	**Matching Field**	**Action**	**Timer**	**Firewall Function**
Policy 1	ADD	ETH_Type=2048 IP_Proto=6	Forward Controller	Idle_Timeout=0 Hard _Timeout=0	
Policy 1	ADD	ETH_Type=2048 IP_Proto=1	Forward Controller	Idle_Timeout=0 Hard _Timeout=0	
Policy 2	MODIFY	ETH_Type=2048 IP_Proto=6	DROP		TCP_Count(3)
Policy 3	MODIFY	ETH_Type=2048 IP_Proto=1	DROP		SNORT.ALERT =DDOS

6.4 Implementation

We implement the proposed policy model into our orchestrator solution (see Sections 5.4 and 5.5 of Chapter 5). The new added classes are implemented in Python language. They have been integrated mainly into the orchestrator part. Figure 6.1 describes the software architecture of the implemented solution. The details of the new architecture are as follows:

- *Policy Handler:* it enables the NSC and the NSPs to express their security requirements and obligations. It issues a policy template, and it sends it to the IHM Manager. The policy template is created using a Template database. NSCs and NSPs fill the templates in the IHM Manager. Then, the filled templates are transmitted to the Policy Handler. The Policy Handler verifies the contents, aligns them with the policy model and distributes them to the Cloud Policy Manager.
- *Cloud Policy Manager:* it compares the policy templates of the NSC and NSPs according to the assessment rules. It manages the interaction between the different processes of the policy model.
- *Ranking Agent:* it determines the best NSCs candidates using the ranking rules.
- *Negotiation Agent:* it negotiates the security requirements with NSC and the chosen NSP according to our proposed RENP Protocol. The Cloud Policy Manager triggers the negotiation process.
- *Contract Issuer:* it issues a contract between NSC and NSPs according to the policy template and the negotiation terms. The contract is sent to the corresponding peers and saved in the Policy Data Base (Global Policy DB). The Cloud Policy Manager also triggers this process.
- *Policy Server:* It transforms the contract into high-level security policies in ABAC and saves the results into the Global Policy DB. At the same time, it manages the interactions with the firewall applications.

Figure 6.1 – The Policy Model Implementation



6.5 Evaluation

The objective of our evaluation is to measure the performance of our policy model regarding processing the number of requirements. Besides, we measure the global performance of our solution from the expression of the requirements until the installation of the corresponding OpenFlow rules in the network elements.

6.5.1 Testbed

We deploy our solution in the B-Secure platform (see Section 5.6.1 of Chapter 5). We install the new architecture in the SDN firewall machine. In the data plane device machine, we run OVS.

In the central machine, we deploy NSC and NSP2 of the use case (see Section 6.3.1). The orchestrator generates the contract and sends the high-level policies to the SDN firewall application. Then, the latter interprets the policies to OpenFlow rules and asks the RYU controller to install them on OVS. We vary the number of NSC's *requirements* from 1 to 2500 policies. We measure the following performance metrics:

1. *Policy Processing Time (PPT)* is the time that the orchestrator needs to process the policy expression.

2. *Orchestrator Processing Time (OPT)* is the total time taken by the orchestrator from the first policy expression to sending the last policy.

3. *Firewall Application Processing Time (FAPT)* is the total time taken by the firewall application to process the policies.

4. *Controller Processing Time (CPT)* is the total time taken by the controller to send all the OpenFlow rules.

5. *Infrastructure Processing Time (IPT)* is the total time that the infrastructure (Controller-OVS link and OVS) needs to install all the OF rules.

6. *Policy Processing Total Time (PPTT)* is the total policy provisioning time.

7. *Orchestrator Setup Rate (OSR)* is the speed of the orchestrator:

$$\mathrm{OSR} = \mathrm{N}umber\,of\,\mathrm{P}olicies/\mathrm{OPT} \tag{6.1}$$

8. *Firewall Setup Rate (FSR)* is the speed of the firewall Application:

$$\mathrm{FSR} = \mathrm{N}umber\,of\,\mathrm{P}olicies/\mathrm{FAPT} \tag{6.2}$$

9. *Controller Setup Rate (CSR)* is the speed of the Controller:

$$\mathrm{CSR} = \mathrm{N}umber\,of\,\mathrm{O}penflow\ \mathrm{R}ules/\mathrm{CPT} \tag{6.3}$$

10. *Infrastructure Setup Rate (ISR)* is the speed of the Infrastructure:

$$\mathrm{ISR} = \mathrm{N}umber\,of\,\mathrm{O}penflow\ \mathrm{R}ules/\mathrm{IPT} \tag{6.4}$$

6.5.2 Evaluation Results

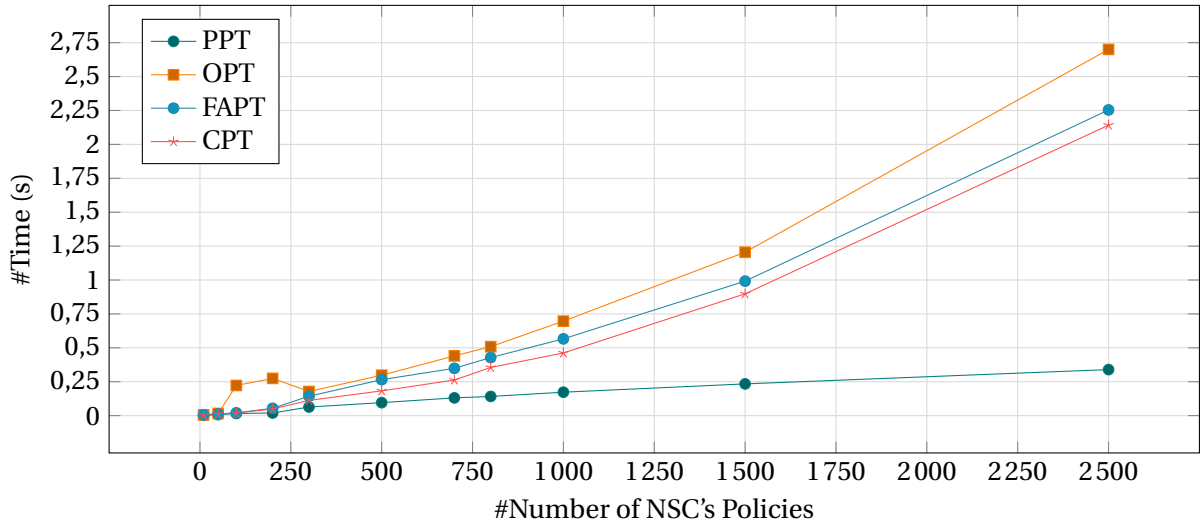
Figure 6.2a and Figure 6.2b display the different measured processing times during the experiment. The processing times in all the figures increase with the rise of rule number. In Figure 6.2a, we observe that PPT increases slowly with a starting value of $0.00236s$ for 10 policies to a maximal value of $0.6421s$ for 2500 policies. In addition, PPT's values are the lowest among all processing times. FAPT is slightly higher than CPT. Moreover, OPT is slower than both FAPT and CPT. For example, the values for 2500 policies are: $2.700s$, $2.254s$ and $2.141s$. However, the largest processing times are those of IPT as shown in Figure 6.2b ($(10, 0.0034s)$ and $(2500, 11.025s)$).

The reasons are due to the amount and nature of the processing that each layer performs. The orchestrator runs many processes to generate the final agreement. The firewall application performs the interpretation of high-level policies to OpenFlow rules, and the controller deploys the generated OF rules in the network element. The performance of OVS impacts the infrastructure. Our solution accumulates a processing time of $7.96s$ with 2500 rules, while the infrastructure processing time is 1.5 times higher $(2500, 11.02s)$. Around 50% (in Average) of PPTT is taken by the infrastructure to deploy the rules. For example, for 2500 rules, it takes $18.12s$ from the time of releasing the initial policy request in the orchestrator to the time of deploying the final OpenFlow rule in the network element. 60% of this time is taken by the infrastructure alone (see Figure 6.2b).

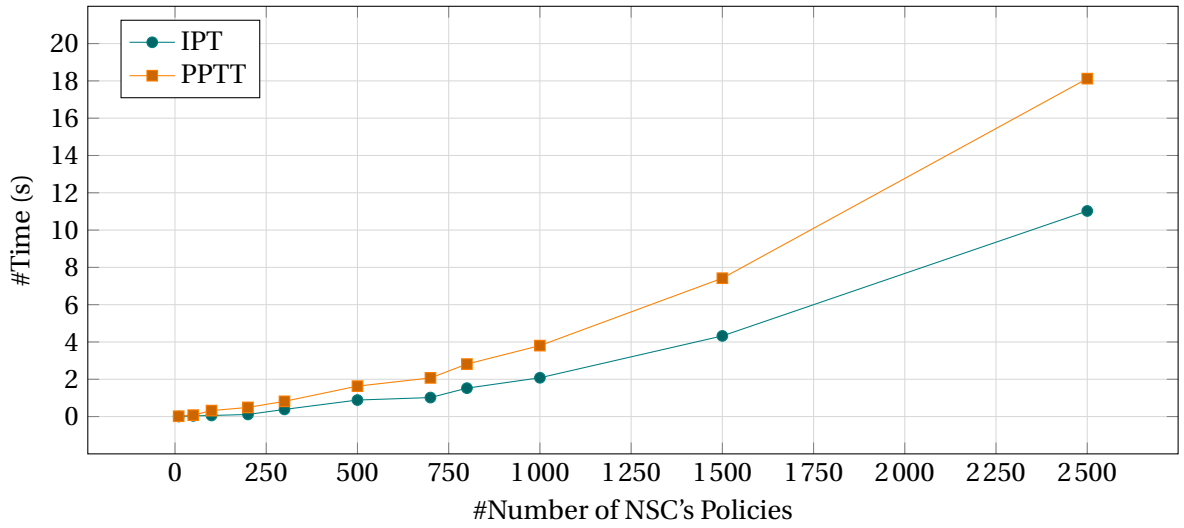
Figure 6.2c displays the different policy setup rates in the infrastructure. We observe three different states. In the first state, CSR, FSR, OSR and ISR increase to reach their top values respectively $(100, 5186)$, $(100, 4838)$, $(50, 3150)$, $(10, 2941)$. In the second stage, all the rates decrease rapidly. The diminution is linear for CSR, FSR and OSR while fluctuating for ISR. In the third stage, we observe that all the rates reduce with the increase in the number of policies. The rates of our solution reach a value of around 1000 $\mathrm{P}olicies/s$ at 2500 while ISR continues

Figure 6.2 – Average Packet Processing Times according to the different experiments

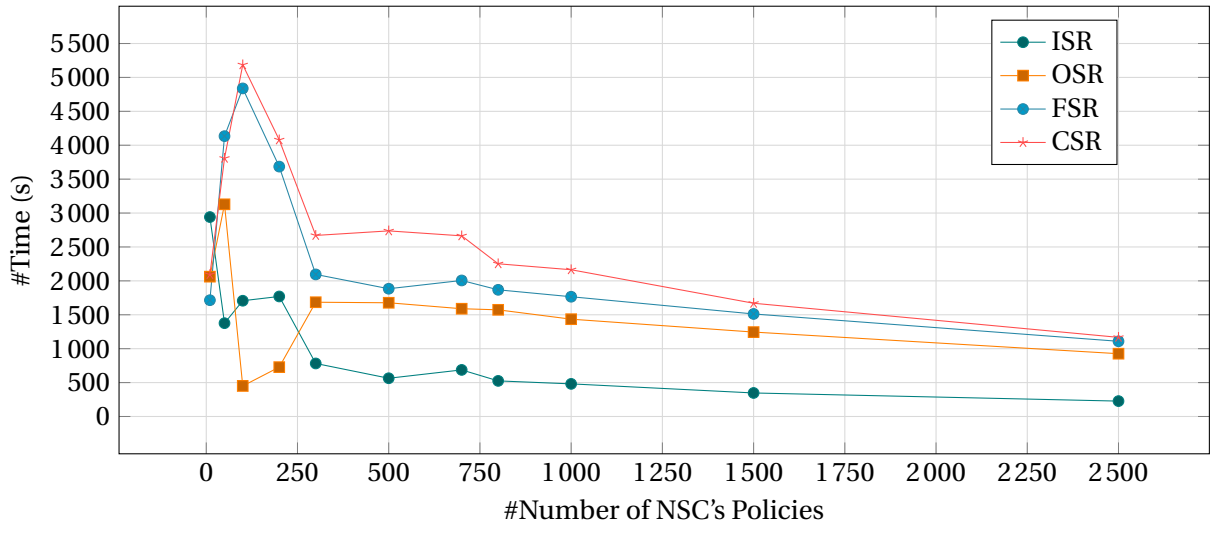
(a) Policy Processing Times



(b) Policy Processing Times



(c) Policy Processing Rates



to hold lower values. This observation comforts the previous results. The load on OVS causes ISR low rates. The latter spends an important time to install the OpenFlow rules. We observe that the orchestrator has lower performance than the firewall application. This observation consolidates the explanations provided previously.

Our solution has good performance with around 1000 policies/s. In practice, the number of firewall rules depends on the size of the topology and the granularity of each rule. Furthermore, policies changes do not need the repetition of all the processes because OpenFlow enables the update of the installed rules.

6.6 Discussion

In this chapter, we have proposed a cloud policy model to express firewall policies, assess different NSCs' requirements and NSPs' obligations, select the best NSC candidate, negotiate and agree on a common policy contract then deploy the agreement in an SDN platform. We have also integrated the solution into our SDN firewall architecture. Moreover, we have deployed our solution in a physical environment, and we have evaluated its performance.

Our framework brings many advantages. It offers interoperability between different NSCs and NSPs through a unified language that simplifies administrator's tasks. It abstracts the complexity of the network by hiding the infrastructure details. Besides, it automatizes firewall policies orchestration.

Besides, the evaluation shows promising results regarding the deployment rate. Our solution can deploy 1000 firewall policies/s from the expression phase to the installation of the OpenFlow rules in the network element. Moreover, we have observed that between 50% and 60% of the total policy time is taken by the network element to install the corresponding OpenFlow rules.

Part IV

Conclusion

Chapter 7

Conclusion

*“ Every once in a while,
a new technology, an old problem,
and a big idea turn into an
innovation. ”*

Dean Kamen

Contents

7.1 General Conclusion

The work presented in this thesis contributed to identifying SDN security vulnerabilities and their severity, to improving firewalls with SDN features and to proving their feasibility by comparing their performance and resistance to a conventional firewall, and to enhancing the firewall policy management in the case of SDN as a service. Our research work has been performed based on three ideas. The first one is the possibility of using AHP to integrate it into CVSS in the context of SDN. We improve the computation of SDN vulnerabilities severity thanks to this integration. The combination of AHP with CVSS helps us to identify the attack surface of SDN. The second idea is that integrating SDN into firewalls helps to improve their resistance and performance. Our SDN firewall has better or acceptable performance when compared to NetFilter. The third idea is using SDN orchestration for firewall policy management to provide firewall policies in SDN as a service.

In this thesis, we have explored, studied and developed the relationship between SDN and security. We have presented in chapter 2, SDN principles, concepts, architecture, its challenges, and benefits. We have also introduced the features of OpenFlow. Then, we have studied the literature on the relationship between SDN and security in chapter 3. We have shown on the one hand that SDN features enlarge the attack surface. As a result, SDN assets are vulnerable and can even be even used as attack vectors to attack other SDN assets. We showed that the controller is a single point of failure because it concentrates all the SDN features at the same time. On the other hand, we have seen that SDN improves security applications thanks to its advantages such as simplification, independence, agility, global knowledge, convergence, automation, and orchestration. We showed that SDN is an enabler for security applications. It improves monitoring, detection, prevention, mitigation and defense mechanisms.

The studies that we have performed in chapter 2 and 3 lead us to address two aspects of the relationship between SDN and security. The first aspect that we have explored is related to security for SDN. We have provided in chapter 4 an analysis of SDN vulnerabilities to understand the security of its assets and SDN attack surface. The second aspect of the relationship that we have addressed concerns security for SDN. In this aspect, chapter 5 has introduced SDN concepts to firewalls to improve their performance and resistance. Besides, chapter 6 has used SDN orchestration to enhance policy management in the case of SDN as a cloud service.

7.2 Contribution Summary

The work in this thesis contributed to SDN and security by identifying and quantifying SDN vulnerabilities, by improving firewalls with SDN simplification, independence, convergence, automation, and orchestration.

7.2.1 SDN Vulnerability analysis

Chapter 4 addresses a vulnerability analysis method that integrates CVSS with AHP to compute the severity of SDN assets. The process consists firstly of identifying SDN vulnerabilities by combining a model of SDN assets with Open Access, Non-Identification, Non-Secrecy, Repudiation, Alterability, and Disruption. As a result, we build 120 new SDN vulnerabilities that cover all SDN potential vulnerabilities. We compute the severity of these vulnerabilities using CVSS. We have chosen the options of the CVSS metrics based on many assumptions related to our knowledge of SDN and security. We have calculated the severity in the base, temporal and the

environmental groups. The preliminary results that we have obtained showed that the vulnerability surface is between 8.3 (high severity) and 5.2 (Medium Severity) in the base group. The different interfaces and their agents have the highest scores because they enlarge the attack scope and are more attractive to being exploited by an attacker than the other assets. In the temporal group, the vulnerability surface reduces between 7.2 and 4.3 due to the unavailability of mature malicious code and confirmed attacker methods that ease the exploitation of SDN vulnerabilities. Likewise, the vulnerability surface decreases in the environment group due to the potential integration of TLS in the interfaces and their agents. For this reason, disruption attacks become more severe than Open Access attacks. These CVSS results indicate a problem which is the inadequacy of CVSS to take into account SDN specific features. In fact, we observe that the average severity scores of controller assets are equal to those of network element assets. We observe the same equality between the different interfaces in the control layer and their respective agents in the application layer and the infrastructure layer. However, the severity of controller assets should be higher than their counterparts in the other SDN layers.

We enhance the vulnerability severity scores by introducing AHP weights into the computations. AHP helps to measure the importance of the impact of each asset on SDN security. We defined the criteria and alternatives for our AHP. We choose SDN features such as programmability, centralization, federation, and externalization as criteria. Besides, we define SDN assets as alternatives to find the most important asset. The objective of the AHP is to find the most SDN assets whose criteria impact the security. We perform the following calculations to determine the importance of each assets impact on security:

1. **Criteria Weights**: we calculate the pairwise comparison matrix of the criteria based on many assumptions related to the impact of SDN features on security. The matrix determines the importance of each criterion to the other. We calculate the weight vector to determine the weights of each criterion. We find that centralization has the highest impact on security with a weight of 0.5694. We performed consistency calculation to verify the quality of our calculation, and we obtain a consistency at 98% .

2. **Alternatives Weights**: we compute the pairwise comparison matrix of the alternatives in each criteria dimension based on many assumptions related to the influence of the criteria in determining the impact of each asset on SDN security. We obtained four matrices. Each matrix determines the importance of the impact of an SDN asset on security according to each criterion. We calculate the weights of each asset in each criteria dimension. We find that Controller Function and C-Agent have the most important weights in each dimension. We also compute the consistency ratio to check the quality of our computations. We obtained a consistency level between 97% and 100%.

3. **Consolidation of the alternatives with the criteria**: we integrate the alternatives weights to the criteria weights to determine the importance of the impact of each SDN asset on its security. As a result, we observed that controller layer assets have the highest importance especially the controller agent and controller function.

We integrate the AHP weights to the CVSS results to adapt the severity computations to SDN specific features. We enhance the scores as follows. The vulnerability surface increases due to the impact of SDN features. It becomes in the base group between 9 and 5.7. The controller layer vulnerabilities increase and become the most severe comparing to their counterparts in other layers. As a result, the introduction of AHP into CVSS breaks the severity equality relationship between controller assets and the rest.

The main advantages of our approach are as follows:

1. We rely on a computation model based on mathematics to determine the severity of SDN vulnerabilities.

2. We determine the importance of the impact of each SDN asset on security using another mathematical model for decision making.

3. All our AHP computation are consistent with at most 3% errors.

4. We are the first to compute the severity of SDN vulnerabilities and quantify its vulnerability surface.

However, our work has a limitation related to the assumptions we have used for CVSS and AHP computations. These assumptions have been built according to our expertise. They can be different from one expert to another.

7.2.2 SDN stateful firewall

Chapter 5 describes our SDN stateful firewall. The objective of this chapter was to prove that the softwarization of firewalls using SDN features is advantageous for network security and the performance of firewalls. We incorporate SDN programmability, centralization, federation and externalization into the design of firewalls, to resolve firewall issues related to complexity, cost, perimeter security and policy reinforcement. Besides, we develop a proof of concept of our proposition and evaluate it to determine its performance and resistance. We perform the following steps:

1. We propose an SDN firewall architecture and formalism to design our solution. The firewall architecture has principally two software layers. In the management layer, we propose an orchestration that is responsible for the interaction with the administrator and the centralization of firewall policies. In the application layer, we propose an SDN firewall application that integrates a stateful filter to operate. The firewall application receives as an entry the FSM of a network protocol and produces as an output a SEFSM. It transforms the conditions and actions of the FSM into two classes. The first one is the firewall and controller functions calls. The second class is OpenFlow rules. Indeed both transformations determine the ability of our firewall to automatize its behavior and reprogram the network elements with firewall rules. We propose and formalize two types of behaviors for the firewall application as follows:

   (a) The reactive behavior generates the SEFSM each time the conditions corresponding to a possible transition are met for each current transition. As a result, it installs the corresponding OpenFlow rules and calls the corresponding functions.

   (b) The proactive behavior generates the SEFSM completely in one time for each active connection. It installs all the corresponding firewall rules on the network elements. It receives a copy of the conditions to call the corresponding functions for each transition.

2. We apply the SEFSM formalism on TCP FSM. We generate two SEFSMs. The first one integrates the reactive behavior and generates the OpenFlow rules and function calls in each TCP transition. The second SEFSM installs all the firewall rules that correspond to TCP transitions. It receives a copy of the TCP packets, and it verifies their legitimacy. If it does not find an anomaly, it deletes the previous OpenFlow rules that do not correspond to a possible transition and calls the corresponding functions.

3. We analyze the performance of our SDN firewall theoretically. We find that the number of packets-in and the maximum number of rules are bounded by the number of the states of the FSM. If the number of states increases, it is likely that both the number of the packets-in and the maximum number of rules will increase. The latter metric affects the

scalability of our firewall. Besides, we show that the relationship between the OpenFlow table size and the maximum number of rules supported by a network element determine the maximum number of connections that are supported by our firewall. Finally, we determine that the firewall processing time is impacted by the feedback loop time in both the reactive and proactive behaviors. The more the firewall feedback time increases the more the packet delivery time increases. One needs to reduce the feedback loop time to improve the packet delivery time.

4. We implement our solution in Python language, and we have evaluated it in a data center platform. The objective of the evaluation was to estimate the performance of the SDN firewall and its capacity to resist to SYN-Flooding attacks. We have compared the performance and resistance values of the reactive SDN firewall with the proactive SDN firewall and with those of NetFilter. We have performed 14 different experiments that injects a traffic of 30.10^3 to 150.10^3 connections. We take into account three variables with each type of firewall. These variables are as follows:

   (a) The number of access rules to determine the impact of the size of firewall rules.

   (b) The size of the transferred data to evaluate the impact of the data on the firewall.

   (c) The flow rate of SYN-Flooding attacks to determine the resistance of the firewalls against SYN-Flooding attacks and their impacts on the firewall performance.

   As a result, we find out that NetFilter and the reactive firewall perform better under the initial conditions. The performance of the proactive behavior starts to decrease from 120.10^3. The number of access rules impacts the reactive behavior because it performs the filtering in the firewall application. The size of the data impacts all three firewalls. NetFilter is the most heavily impacted while the proactive firewall is the least impacted. The rates of SYN-Flooding impact both the reactive firewall and NetFilter. These impacts on NetFilter increase its packet processing times and the number of re-transmissions. We have also noticed that the decrease in the performance of the proactive mode is related to the load on the OVS resources.

The main advantages of our solution are as follows:

1. It turns network elements to firewalls. This feature can reduce the cost of legacy firewalls.

2. It automatizes the firewall behavior from software by reprogramming the network element using OpenFlow and calling the firewall and controller functions.

3. It pervades the firewall rules throughout the whole network without needing to install firewall devices or any manual intervention.

4. It filters packets on different granular levels.

5. Thanks to its federation, it can be run on any SDN Controller.

6. It improves firewall performance and resistance in case of large data transfers and SYN-Flooding attacks.

However, our work is limited by the capacity of OpenFlow specification and implementations. In fact, we handle firewall functions in the firewall application, and we can not perform firewall operations using OpenFlow such as counting the number of SYN, learning the state of the connection, comparing header fields in the network element. The main reason is related to the limitations of OpenFlow.

7.2.3 SDN policy orchestration

Chapter 6 presents an orchestration framework for firewall policies management in the context of SDN as service. The framework is the first work that tackles the life cycle of firewall policies from their expression to their deployment of network elements completely. The orchestrator offers an expression language for NSC's obligations and NSP's offers. It evaluates the different NSP to select the best one that meets the requirements of NSC. It negotiates the policies between NSC and the selected NSP. Then, it builds a common agreement between them based on the negotiation results. The orchestrator sends this contract to the firewall application. The latter deploys it as OpenFlow rules in the network elements. We perform the following steps:

1. **Unified policy language**: we present a formalism to unify the expression of the obligations and the offers. The objective of the formalism is to federate the interaction between NSPs and NSCs.

2. **Assessment of policies**: The goal of this step is to map the obligations with the offers to determine the NSP that best satisfies the NSC. We have compared the obligations and the offers based on the relations between the elements of the policies. Then we have derived and formalized the relationships between the policies. These relationships determine the selection of the best NSP. We outline three relations as follows:

   (a) Match relationship determines the relationship by which all the elements of the two policies are equal. If all the policies have match relationship, the NSP rank is gold.

   (b) Mismatch relationship determines the relationship in which there are at least two elements that are incomparable. If two policies are in Mismatch, the NSP is rejected.

   (c) Potential match relationship determines the relationship of which all the elements are incomparable, and there are at least two of them that have an unequal relationship. If two policies are in a Potential match, the NSP rank is silver.

3. **Negotiation of policies**: we propose a negotiation protocol (RENP) to negotiate the policies when the NSP is ranked as silver. RENP processes the different states that can be taken by the values of the elements of NSP regarding the local values of the elements of NSC. Depending on the cases, the RENP accepts the NSP value, refuses it or proposes a new one that meets both the NSC and NSP. Thanks to RENP, the NCP and NSC can reach an agreement.

4. **Deployment of firewall policies**: we present a deployment model based on mapping policy element with OpenFlow rules.

5. **Implementation of the Framework and evaluation**: We implement a prototype of the framework in the orchestrator of our firewall solution. We evaluate its performance by varying the number of obligations from 1 to 2500 policies. We have found that the processing times of all the different processes of our solution, the controller and the network element rise with the increase of the number of policies. When we compare the processing time, we observe that 60% of the total processing time is taken by the network element alone because it spends more time installing the OpenFlow rules when the number of policies increases. We show that our framework can process 1000 policies per second.

The main advantages of our framework are as follows:

1. It is the first to offer a complete process that orchestrates the whole life cycle of firewall policies in SDN.

2. It unifies the obligations and offers based on the same expression model. This unification simplifies the expression of policies and reduces policy inconsistencies.

3. It pervades the firewall policies on all the firewall applications while reinforcing them according to their scopes (in the case the topology is known).

4. It transforms SDN firewalls as an enabler for a secure SDN as a service.

5. It processes the firewall policies with a rate of 1000 policies per second in all the SDN.

However, our framework is limited by two aspects. NSCs and NSPs must express their obligations and offers using the proposed language to reach an agreement. The selection algorithm does not tackle the case in which two or more NSPs have the same ranking.

7.3 Perspectives

The work presented in this thesis has addressed issues regarding SDN vulnerability analysis, SDN firewall design, and development and Firewall policy orchestration. We plan to improve the vulnerability analysis, to enhance the performance of the SDN firewall and the processing of the orchestrator.

Regarding the work presented in chapter 4, our next objective is to employ the SDN vulnerabilities and their severity scores to calculate the severity of SDN security risks and their likelihood. The idea is to define the path to an SDN security risk based on the relationship between SDN vulnerabilities. A forest of possibilities can define risk. Each possibility is a tree of vulnerability. The risks are the branches of the tree. Their severity and likelihood are the leaves. In each tree, the relations between the vulnerabilities is a logical AND. The relations between the trees is a logical OR. We will work on a solution that automatically generates such forests and then calculates the severity of the risk and likelihood using the relations between the branches and between the trees.

For future directions regarding the work presented in chapter 5, we propose to work on the issue of the feedback loop time. One direction that we have started in a master internship is to delegate the firewall application as a container next to the network element with a dedicated controller. The delegated firewall application and its controller manage all the local states of the network elements. They notify the root firewall application about the state of the connections to enable it to construct a domain state awareness. This knowledge concentrates all the notifications coming from each delegated firewall application. Another proposition is to reinforce the packet-in processing in the firewall application. Suppose the following case arises; we have a connection between a client and the server and their packets pass through many network elements in intra-domain and inter-domain scopes. In the actual case, each time a network element receives the packet it forwards it to the firewall application. This behavior is repeated even if the packet has already been processed in the first network element that is linked to the source of the packet. The idea is to deploy OpenFlow rules on all the network elements that belong to the path to make them forward the packet to the next hop directly while avoiding the feedback loop. When the packet is processed far the first time, this mechanism is executed to improve the packet delivery time.

Besides, we need to investigate in-depth the evaluation results, especially, the performance of OVS. We need to understand the impact of OpenFlow ADD and DELETE rules on the performance of OVS. Besides, we need to understand the behavior of OVS while dealing at the same time with an essential load of packet-in, packet-out and flow mod while its buffer and tables are occupied. This investigation will help us understand the behavior of the proactive mode in the initial conditions.

The work of chapter 6 can be extended with context awareness using SDN global knowledge. The purpose of such improvement is to enable the orchestrator to deploy the firewall policies intelligently and to adapt them to the changes happening in the network. This enhancement needs to modify the expression language with new concepts such as user groups, application types, and reputation. A user with an IP address can have different firewall policies according to its membership in each user group because they have different roles in each group. An application can behave differently in different contexts. For example, it solicits a network element port when it needs to send user data, and it solicits another port when it needs to send its commands. In this case, the context awareness enables the orchestrator to adapt the policies for each application behavior. Besides, the orchestrator can deny or permit access to other resources using their reputations. It adapts the firewall rules according to the updates of the reputation. For example, if the reputation of a server becomes terrible, the orchestrator will change the firewall policies to deny or limit communications with this server. The integration of context awareness into the orchestrator is essential to keeping the consistency of firewall policies and improve security. The orchestrator will receive domain knowledge from all its controllers. Each control includes information such as resource utilization, QOS, topology, and other information in its knowledge context. The orchestrator will construct holistic knowledge of the network. Then, it extracts information from the holistic knowledge of the global context. It combines them with the other context information that the administrator submitted using the orchestrator expression language. As a result, the orchestrator dynamically generates new policies that take the context into consideration.

The last but not least idea is related to the future of SDN. According to our knowledge, there are two significant areas of SDN that will determine its fate.

The first area is controller scalability. It is a hot subject. The controller must keep acceptable performances when its load increases. We think that controller delegation and separating the local control function from the global control function can improve the controller scalability. The idea is to delegate (as aforementioned) all the local decision to a simpler controller that can be executed as a VNF (Virtual Network Function) next to the network element. This delegate will be in charge of all the local control operations and will report its local view to the global controller. As a result, the scalability of the controller will improve. Another advantage of this solution is the protection of the controller by its delegate. We can hide the identity of the controller and expose only those of the delegates. In the case of an attack, the delegates play a shield to protect the controller.

The second area is security for SDN. The research community needs to work on protecting SDN assets especially those of the controller. We propose many improvements. The first one is to put access control mechanisms in the controller on the different interfaces. The access control will authorize or deny the resource requests and data of SDN applications, network elements, and other controllers. It determines their different roles and rights through these interactions in the interfaces. The second idea is to encrypt and sign all the interactions in all the interfaces to avoid confidentiality and integrity violations.

Part V

Appendix

.1 Appendix AHP Computations

In this section, we detail the computation of the AHP alternative weights as follows:

A Programmability

Pairwise comparisons among Assets	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Manager	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	1,00	3,00	1,00	3,00	0,50	2,00	4,00	4,00	4,00	4,00	2,00	2,00	2,00	2,00	2,00	2,00	4,00	3,00	3,00	3,00
Application Content	0,33	1,00	0,33	1,00	0,33	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	0,50	0,50	0,50	1,00	0,50	0,50	0,50
Controler Function	1,00	3,00	1,00	3,00	0,50	2,00	4,00	4,00	4,00	4,00	2,00	2,00	2,00	2,00	2,00	2,00	4,00	3,00	3,00	3,00
Controler Content	0,33	1,00	0,33	1,00	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	0,50	0,50	0,50	1,00	0,50	0,50	0,50
C-Agent	2,00	3,00	2,00	2,00	1,00	2,00	4,00	4,00	4,00	4,00	2,00	2,00	2,00	2,00	2,00	2,00	4,00	3,00	3,00	3,00
Controler RDB	0,50	1,00	0,50	1,00	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	0,50	0,50	0,50	1,00	0,50	0,50	0,50
Data Processing Engine	0,25	1,00	0,25	1,00	0,25	1,00	1,00	1,00	1,00	1,00	0,25	0,25	0,25	0,25	0,25	0,25	0,50	0,25	0,25	0,25
Data Source	0,25	1,00	0,25	1,00	0,25	1,00	1,00	1,00	1,00	1,00	0,25	0,25	0,25	0,25	0,25	0,25	0,50	0,25	0,25	0,25
Data Sink	0,25	1,00	0,25	1,00	0,25	1,00	1,00	1,00	1,00	1,00	0,25	0,25	0,25	0,25	0,25	0,25	0,50	0,25	0,25	0,25
Network Element RDB	0,25	1,00	0,25	1,00	0,25	1,00	1,00	1,00	1,00	1,00	0,25	0,25	0,25	0,25	0,25	0,25	0,50	0,25	0,25	0,25
A-CPI Agent	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	1,00	1,00	0,50	2,00	2,00	2,00	4,00	2,00	2,00	2,00
A-CPI	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	1,00	1,00	0,50	2,00	2,00	2,00	4,00	2,00	2,00	2,00
C-CPI	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	2,00	2,00	1,00	3,00	3,00	2,00	4,00	2,00	2,00	2,00
D-CPI	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,33	1,00	1,00	2,00	4,00	2,00	2,00	2,00
D-CPI Agent	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,33	1,00	1,00	2,00	4,00	2,00	2,00	2,00
Management Function	0,50	2,00	0,50	2,00	0,50	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,50	0,50	0,50	1,00	2,00	1,00	1,00	1,00
Management Content	0,25	1,00	0,25	1,00	0,25	1,00	2,00	2,00	2,00	2,00	0,25	0,25	0,25	0,25	0,25	0,50	1,00	0,50	0,50	0,50
Application Coordinator	0,33	2,00	0,33	2,00	0,33	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,50	0,50	0,50	1,00	2,00	1,00	1,00	1,00
Controler Coordinator	0,33	2,00	0,33	2,00	0,33	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,50	0,50	0,50	1,00	2,00	1,00	1,00	1,00
Network Element Coordinator	0,33	2,00	0,33	2,00	0,33	2,00	4,00	4,00	4,00	4,00	0,50	0,50	0,50	0,50	0,50	1,00	2,00	1,00	1,00	1,00

B Programmability

Normalized Matrix	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Manager	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	0,0960	0,0857	0,0960	0,0882	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,1270	0,1270	0,1491	0,1013	0,1013	0,0870	0,0870	0,1154	0,1154	0,1154
Application Content	0,0320	0,0286	0,0320	0,0294	0,0388	0,0313	0,0175	0,0175	0,0175	0,0175	0,0317	0,0317	0,0373	0,0253	0,0253	0,0217	0,0217	0,0192	0,0192	0,0192
Controler Function	0,0960	0,0857	0,0960	0,0882	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,1270	0,1270	0,1491	0,1013	0,1013	0,0870	0,0870	0,1154	0,1154	0,1154
Controler Content	0,0320	0,0286	0,0320	0,0294	0,0583	0,0313	0,0175	0,0175	0,0175	0,0175	0,0317	0,0317	0,0373	0,0253	0,0253	0,0217	0,0217	0,0192	0,0192	0,0192
C-Agent	0,1920	0,0857	0,1920	0,0588	0,1165	0,0625	0,0702	0,0702	0,0702	0,0702	0,1270	0,1270	0,1491	0,1013	0,1013	0,0870	0,0870	0,1154	0,1154	0,1154
Controler RDB	0,0480	0,0286	0,0480	0,0294	0,0583	0,0313	0,0175	0,0175	0,0175	0,0175	0,0317	0,0317	0,0373	0,0253	0,0253	0,0217	0,0217	0,0192	0,0192	0,0192
Data Processing Engine	0,0240	0,0286	0,0240	0,0294	0,0291	0,0313	0,0175	0,0175	0,0175	0,0175	0,0159	0,0159	0,0186	0,0127	0,0127	0,0109	0,0109	0,0096	0,0096	0,0096
Data Source	0,0240	0,0286	0,0240	0,0294	0,0291	0,0313	0,0175	0,0175	0,0175	0,0175	0,0159	0,0159	0,0186	0,0127	0,0127	0,0109	0,0109	0,0096	0,0096	0,0096
Data Sink	0,0240	0,0286	0,0240	0,0294	0,0291	0,0313	0,0175	0,0175	0,0175	0,0175	0,0159	0,0159	0,0186	0,0127	0,0127	0,0109	0,0109	0,0096	0,0096	0,0096
Network Element RDB	0,0240	0,0286	0,0240	0,0294	0,0291	0,0313	0,0175	0,0175	0,0175	0,0175	0,0159	0,0159	0,0186	0,0127	0,0127	0,0109	0,0109	0,0096	0,0096	0,0096
A-CPI Agent	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,0635	0,0635	0,0373	0,1013	0,1013	0,0870	0,0870	0,0769	0,0769	0,0769
A-CPI	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,0635	0,0635	0,0373	0,1013	0,1013	0,0870	0,0870	0,0769	0,0769	0,0769
C-CPI	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,1270	0,1270	0,0745	0,1519	0,1519	0,0870	0,0870	0,0769	0,0769	0,0769
D-CPI	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0248	0,0506	0,0506	0,0870	0,0870	0,0769	0,0769	0,0769
D-CPI Agent	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0248	0,0506	0,0506	0,0870	0,0870	0,0769	0,0769	0,0769
Management Function	0,0480	0,0571	0,0480	0,0588	0,0583	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0373	0,0253	0,0253	0,0435	0,0435	0,0385	0,0385	0,0385
Management Content	0,0240	0,0286	0,0240	0,0294	0,0291	0,0313	0,0351	0,0351	0,0351	0,0351	0,0159	0,0159	0,0186	0,0127	0,0127	0,0217	0,0217	0,0192	0,0192	0,0192
Application Coordinator	0,0320	0,0571	0,0320	0,0588	0,0388	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0373	0,0253	0,0253	0,0435	0,0435	0,0385	0,0385	0,0385
Controler Coordinator	0,0320	0,0571	0,0320	0,0588	0,0388	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0373	0,0253	0,0253	0,0435	0,0435	0,0385	0,0385	0,0385
Network Element Coordinator	0,0320	0,0571	0,0320	0,0588	0,0388	0,0625	0,0702	0,0702	0,0702	0,0702	0,0317	0,0317	0,0373	0,0253	0,0253	0,0435	0,0435	0,0385	0,0385	0,0385

	Weights Wp	Products	Ratio		
Application Function	0,0947	2,0355	21,5046		
Application Content	0,0257	0,5396	20,9627		
Controler Function	0,0947	2,0355	21,5046		
Controler Content	0,0267	0,5572	20,8602		
C-Agent	0,1057	2,2509	21,2963		
Controler RDB	0,0283	0,5888	20,7958		
Data Processing Engine	0,0181	0,3710	20,4533		
Data Source	0,0181	0,3710	20,4533		
Data Sink	0,0181	0,3710	20,4533		
Network Element RDB	0,0181	0,3710	20,4533		
A-CPI Agent	0,0692	1,4887	21,4995		
A-CPI	0,0692	1,4887	21,4995		
C-CPI	0,0825	1,7892	21,6824		
D-CPI	0,0604	1,2850	21,2792		
D-CPI Agent	0,0604	1,2850	21,2792		
management function	0,0484	1,0042	20,7670		
Management Content	0,0242	0,5021	20,7670		
Application Coordinator	0,0458	0,9551	20,8592		
Controler Coordinator	0,0458	0,9551	20,8592		
Network Element Coordinator	0,0458	0,9551	20,8592		
		CI	0,0529	CI/RI	0,03

A Centralization

Pairwise comparisons among Assets	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	1,00	1,00	0,20	0,25	0,20	0,25	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,25	0,33	1,00	0,33	1,00
Application Content	1,00	1,00	0,20	0,25	0,20	0,25	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,25	0,33	1,00	0,33	1,00
Controler Function	5,00	5,00	1,00	1,00	0,50	2,00	7,00	7,00	7,00	5,00	4,00	2,00	2,00	2,00	4,00	2,00	2,00	6,00	2,00	6,00
Controler Content	4,00	4,00	1,00	1,00	0,50	1,00	7,00	7,00	7,00	5,00	4,00	2,00	2,00	2,00	4,00	2,00	2,00	6,00	2,00	6,00
C-Agent	5,00	5,00	2,00	2,00	1,00	2,00	7,00	7,00	7,00	5,00	4,00	2,00	2,00	2,00	4,00	2,00	2,00	6,00	2,00	6,00
Controler RDB	4,00	4,00	0,50	1,00	0,50	1,00	7,00	7,00	7,00	5,00	4,00	2,00	2,00	2,00	4,00	2,00	2,00	6,00	2,00	6,00
Data Processing Engine	1,00	1,00	0,14	0,14	0,14	0,14	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
Data Source	1,00	1,00	0,14	0,14	0,14	0,14	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
Data Sink	1,00	1,00	0,14	0,14	0,14	0,14	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
Network Element RDB	1,00	1,00	0,20	0,20	0,20	0,20	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
A-CPI Agent	1,00	1,00	0,25	0,25	0,25	0,25	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
A-CPI	3,00	3,00	0,50	0,50	0,50	0,50	3,00	3,00	3,00	3,00	3,00	1,00	1,00	1,00	3,00	2,00	2,00	3,00	1,00	3,00
C-CPI	3,00	3,00	0,50	0,50	0,50	0,50	3,00	3,00	3,00	3,00	3,00	1,00	1,00	1,00	3,00	2,00	2,00	3,00	1,00	3,00
D-CPI	3,00	3,00	0,50	0,50	0,50	0,50	3,00	3,00	3,00	3,00	3,00	1,00	1,00	1,00	3,00	2,00	2,00	3,00	1,00	3,00
D-CPI Agent	1,00	1,00	0,25	0,25	0,25	0,25	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,20	0,20	1,00	0,33	1,00
Management Function	4,00	4,00	0,50	0,50	0,50	0,50	5,00	5,00	5,00	5,00	5,00	0,50	0,50	0,50	5,00	1,00	1,00	3,00	1,00	3,00
Management Content	3,00	3,00	0,50	0,50	0,50	0,50	5,00	5,00	5,00	5,00	5,00	0,50	0,50	0,50	5,00	1,00	1,00	3,00	1,00	3,00
Application Coordinator	1,00	1,00	0,17	0,17	0,17	0,17	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,33	0,33	1,00	0,33	1,00
Controler Coordinator	3,00	3,00	0,50	0,50	0,50	0,50	3,00	3,00	3,00	3,00	3,00	1,00	1,00	1,00	3,00	1,00	1,00	3,00	1,00	3,00
Network Element Coordinator	1,00	1,00	0,17	0,17	0,17	0,17	1,00	1,00	1,00	1,00	1,00	0,33	0,33	0,33	1,00	0,33	0,33	1,00	0,33	1,00

B Centralization

Normalized Matrix	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Element Coordinator
Application Function	0,0213	0,0213	0,0214	0,0251	0,0272	0,0228	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0129	0,0171	0,0192	0,0192	0,0192
Application Content	0,0213	0,0213	0,0214	0,0251	0,0272	0,0228	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0129	0,0171	0,0192	0,0192	0,0192
Controler Function	0,1064	0,1064	0,1068	0,1004	0,0679	0,1825	0,1167	0,1167	0,1167	0,0962	0,0833	0,1224	0,1224	0,1224	0,0833	0,1033	0,1024	0,1154	0,1154	0,1154
Controler Content	0,0851	0,0851	0,1068	0,1004	0,0679	0,0912	0,1167	0,1167	0,1167	0,0962	0,0833	0,1224	0,1224	0,1224	0,0833	0,1033	0,1024	0,1154	0,1154	0,1154
C-Agent	0,1064	0,1064	0,2136	0,2008	0,1358	0,1825	0,1167	0,1167	0,1167	0,0962	0,0833	0,1224	0,1224	0,1224	0,0833	0,1033	0,1024	0,1154	0,1154	0,1154
Controler RDB	0,0851	0,0851	0,0534	0,1004	0,0679	0,0912	0,1167	0,1167	0,1167	0,0962	0,0833	0,1224	0,1224	0,1224	0,0833	0,1033	0,1024	0,1154	0,1154	0,1154
Data Processing Engine	0,0213	0,0213	0,0153	0,0143	0,0194	0,0130	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
Data Source	0,0213	0,0213	0,0153	0,0143	0,0194	0,0130	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
Data Sink	0,0213	0,0213	0,0153	0,0143	0,0194	0,0130	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
Network Element RDB	0,0213	0,0213	0,0214	0,0201	0,0272	0,0182	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
A-CPI Agent	0,0213	0,0213	0,0267	0,0251	0,0340	0,0228	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
A-CPI	0,0638	0,0638	0,0534	0,0502	0,0679	0,0456	0,0500	0,0500	0,0500	0,0577	0,0625	0,0612	0,0612	0,0612	0,0625	0,1033	0,1024	0,0577	0,0577	0,0577
C-CPI	0,0638	0,0638	0,0534	0,0502	0,0679	0,0456	0,0500	0,0500	0,0500	0,0577	0,0625	0,0612	0,0612	0,0612	0,0625	0,1033	0,1024	0,0577	0,0577	0,0577
D-CPI	0,0638	0,0638	0,0534	0,0502	0,0679	0,0456	0,0500	0,0500	0,0500	0,0577	0,0625	0,0612	0,0612	0,0612	0,0625	0,1033	0,1024	0,0577	0,0577	0,0577
D-CPI Agent	0,0213	0,0213	0,0267	0,0251	0,0340	0,0228	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0103	0,0102	0,0192	0,0192	0,0192
Management Function	0,0851	0,0851	0,0534	0,0502	0,0679	0,0456	0,0833	0,0833	0,0833	0,0962	0,1042	0,0306	0,0306	0,0306	0,1042	0,0516	0,0512	0,0577	0,0577	0,0577
Management Content	0,0638	0,0638	0,0534	0,0502	0,0679	0,0456	0,0833	0,0833	0,0833	0,0962	0,1042	0,0306	0,0306	0,0306	0,1042	0,0516	0,0512	0,0577	0,0577	0,0577
Application Coordinator	0,0213	0,0213	0,0178	0,0167	0,0226	0,0152	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0172	0,0171	0,0192	0,0192	0,0192
Controler Coordinator	0,0638	0,0638	0,0534	0,0502	0,0679	0,0456	0,0500	0,0500	0,0500	0,0577	0,0625	0,0612	0,0612	0,0612	0,0625	0,0516	0,0512	0,0577	0,0577	0,0577
Network Element Coordinator	0,0213	0,0213	0,0178	0,0167	0,0226	0,0152	0,0167	0,0167	0,0167	0,0192	0,0208	0,0204	0,0204	0,0204	0,0208	0,0172	0,0171	0,0192	0,0192	0,0192

	Weights WC	Products	Ratio		
Application Function	0,0199	0,4064	20,3845		
Application Content	0,0199	0,4064	20,3845		
Controler Function	0,1101	2,2754	20,6636		
Controler Content	0,1034	2,1348	20,6402		
C-Agent	0,1239	2,5509	20,5928		
Controler RDB	0,1008	2,0797	20,6408		
Data Processing Engine	0,0177	0,3595	20,2536		
Data Source	0,0177	0,3595	20,2536		
Data Sink	0,0177	0,3595	20,2536		
Network Element RDB	0,0190	0,3845	20,2488		
A-CPI Agent	0,0201	0,4064	20,2451		
A-CPI	0,0620	1,2901	20,8095		
C-CPI	0,0620	1,2901	20,8095		
D-CPI	0,0620	1,2901	20,8095		
D-CPI Agent	0,0201	0,4064	20,2451		
Management Function	0,0655	1,3329	20,3562		
Management Content	0,0634	1,2930	20,4104		
Application Coordinator	0,0190	0,3871	20,4253		
Controler Coordinator	0,0569	1,1612	20,4253		
Network Element Coordinator	0,0190	0,3871	20,4253		
		CI	0,0244	CI/RI	0,01

A Federation

Pairwise comparisons among Assets	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	1,00	1,00	0,25	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	1,00	1,00	0,20	0,20	0,20
Application Content	1,00	1,00	0,25	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	1,00	1,00	0,20	0,20	0,20
Controler Function	4,00	4,00	1,00	1,00	0,50	2,00	3,00	3,00	3,00	3,00	0,50	0,50	0,50	0,50	0,50	3,00	4,00	1,00	1,00	1,00
Controler Content	2,00	2,00	1,00	1,00	0,33	1,00	2,00	2,00	2,00	2,00	0,33	0,33	0,33	0,33	0,33	3,00	3,00	0,33	0,33	0,33
C-Agent	5,00	5,00	2,00	3,03	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
Controler RDB	2,00	2,00	0,50	1,00	0,33	1,00	2,00	2,00	2,00	2,00	0,33	0,33	0,33	0,33	0,33	2,00	2,00	0,33	0,33	0,33
Data Processing Engine	1,00	1,00	0,33	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	2,00	2,00	0,20	0,20	0,20
Data Source	1,00	1,00	0,33	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	2,00	2,00	0,20	0,20	0,20
Data Sink	1,00	1,00	0,33	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	2,00	2,00	0,20	0,20	0,20
Network Element RDB	1,00	1,00	0,33	0,50	0,20	0,50	1,00	1,00	1,00	1,00	0,20	0,20	0,20	0,20	0,20	2,00	2,00	0,20	0,20	0,20
A-CPI Agent	5,00	5,00	2,00	3,00	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
A-CPI	5,00	5,00	2,00	3,00	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
C-CPI	5,00	5,00	2,00	3,00	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
D-CPI	5,00	5,00	2,00	3,00	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
D-CPI Agent	5,00	5,00	2,00	3,00	1,00	3,00	5,00	5,00	5,00	5,00	1,00	1,00	1,00	1,00	1,00	7,00	7,00	2,00	2,00	2,00
Management Function	1,00	1,00	0,33	0,33	0,14	0,50	0,50	0,50	0,50	0,50	0,14	0,14	0,14	0,14	0,14	1,00	1,00	0,14	0,14	0,14
Management Content	1,00	1,00	0,25	0,33	0,14	0,50	0,50	0,50	0,50	0,50	0,14	0,14	0,14	0,14	0,14	1,00	1,00	0,14	0,14	0,14
Application Coordinator	5,00	5,00	1,00	3,00	0,50	3,00	5,00	5,00	5,00	5,00	0,50	0,50	0,50	0,50	0,50	7,00	7,00	1,00	1,00	1,00
Controler Coordinator	5,00	5,00	1,00	3,00	0,50	3,00	5,00	5,00	5,00	5,00	0,50	0,50	0,50	0,50	0,50	7,00	7,00	1,00	1,00	1,00
Network Element Coordinator	5,00	5,00	1,00	3,00	0,50	3,00	5,00	5,00	5,00	5,00	0,50	0,50	0,50	0,50	0,50	7,00	7,00	1,00	1,00	1,00

B Federation

Normalized Matrix	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinato
Application Function	**0,0164**	**0,0164**	**0,0126**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0120**	**0,0119**	**0,0110**	**0,0110**	**0,0110**
Application Content	**0,0164**	**0,0164**	**0,0126**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0120**	**0,0119**	**0,0110**	**0,0110**	**0,0110**
Controler Function	**0,0656**	**0,0656**	**0,0502**	**0,0297**	**0,0493**	**0,0571**	**0,0508**	**0,0508**	**0,0508**	**0,0508**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0361**	**0,0476**	**0,0551**	**0,0551**	**0,0551**
Controler Content	**0,0328**	**0,0328**	**0,0502**	**0,0297**	**0,0325**	**0,0286**	**0,0339**	**0,0339**	**0,0339**	**0,0339**	**0,0328**	**0,0328**	**0,0328**	**0,0328**	**0,0328**	**0,0361**	**0,0357**	**0,0184**	**0,0184**	**0,0184**
C-Agent	**0,0820**	**0,0820**	**0,1004**	**0,0899**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
Controler RDB	**0,0328**	**0,0328**	**0,0251**	**0,0297**	**0,0328**	**0,0286**	**0,0339**	**0,0339**	**0,0339**	**0,0339**	**0,0328**	**0,0328**	**0,0328**	**0,0328**	**0,0328**	**0,0241**	**0,0238**	**0,0184**	**0,0184**	**0,0184**
Data Processing Engine	**0,0164**	**0,0164**	**0,0167**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0241**	**0,0238**	**0,0110**	**0,0110**	**0,0110**
Data Source	**0,0164**	**0,0164**	**0,0167**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0241**	**0,0238**	**0,0110**	**0,0110**	**0,0110**
Data Sink	**0,0164**	**0,0164**	**0,0167**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0241**	**0,0238**	**0,0110**	**0,0110**	**0,0110**
Network Element RDB	**0,0164**	**0,0164**	**0,0167**	**0,0148**	**0,0197**	**0,0143**	**0,0169**	**0,0169**	**0,0169**	**0,0169**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0197**	**0,0241**	**0,0238**	**0,0110**	**0,0110**	**0,0110**
A-CPI Agent	**0,0820**	**0,0820**	**0,1004**	**0,0890**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
A-CPI	**0,0820**	**0,0820**	**0,1004**	**0,0890**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
C-CPI	**0,0820**	**0,0820**	**0,1004**	**0,0890**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
D-CPI	**0,0820**	**0,0820**	**0,1004**	**0,0890**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
D-CPI Agent	**0,0820**	**0,0820**	**0,1004**	**0,0890**	**0,0985**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0985**	**0,0843**	**0,0833**	**0,1102**	**0,1102**	**0,1102**
Management Function	**0,0164**	**0,0164**	**0,0167**	**0,0099**	**0,0141**	**0,0143**	**0,0085**	**0,0085**	**0,0085**	**0,0085**	**0,0141**	**0,0141**	**0,0141**	**0,0141**	**0,0141**	**0,0120**	**0,0119**	**0,0079**	**0,0079**	**0,0079**
Management Content	**0,0164**	**0,0164**	**0,0126**	**0,0099**	**0,0141**	**0,0143**	**0,0085**	**0,0085**	**0,0085**	**0,0085**	**0,0141**	**0,0141**	**0,0141**	**0,0141**	**0,0141**	**0,0120**	**0,0119**	**0,0079**	**0,0079**	**0,0079**
Application Coordinator	**0,0820**	**0,0820**	**0,0502**	**0,0890**	**0,0493**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0843**	**0,0833**	**0,0551**	**0,0551**	**0,0551**
Controler Coordinator	**0,0820**	**0,0820**	**0,0502**	**0,0890**	**0,0493**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0843**	**0,0833**	**0,0551**	**0,0551**	**0,0551**
Network Element Coordinator	**0,0820**	**0,0820**	**0,0502**	**0,0890**	**0,0493**	**0,0857**	**0,0847**	**0,0847**	**0,0847**	**0,0847**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0492**	**0,0843**	**0,0833**	**0,0551**	**0,0551**	**0,0551**

	Weights WF	Products	Ratio		
Application Function	**0,0159**	**0,3205**	**20,1909**		
Application Content	**0,0159**	**0,3205**	**20,1909**		
Controler Function	**0,0508**	**1,0419**	**20,5069**		
Controler Content	**0,0317**	**0,6389**	**20,1794**		
C-Agent	**0,0934**	**1,9230**	**20,5862**		
Controler RDB	**0,0292**	**0,5901**	**20,1905**		
Data Processing	**0,0173**	**0,3485**	**20,1665**		
Data Source	**0,0173**	**0,3485**	**20,1665**		
Data Sink	**0,0173**	**0,3485**	**20,1665**		
Network Element RDB	**0,0173**	**0,3485**	**20,1665**		
A-CPI Agent	**0,0934**	**1,9220**	**20,5858**		
A-CPI	**0,0934**	**1,9220**	**20,5858**		
C-CPI	**0,0934**	**1,9220**	**20,5858**		
D-CPI	**0,0934**	**1,9220**	**20,5858**		
D-CPI Agent	**0,0934**	**1,9220**	**20,5858**		
Management Function	**0,0120**	**0,2413**	**20,1389**		
Management Content	**0,0118**	**0,2370**	**20,1372**		
Application Coordinator	**0,0678**	**1,3876**	**20,4617**		
Controler Coordinator	**0,0678**	**1,3876**	**20,4617**		
Network Element	**0,0678**	**1,3876**	**20,4617**		
		CI	**0,01868824**	**CI/RI**	**0,01**

$\mathbf{A}_{\text{Externalization}}$

Pairwise comparisons among Assets	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Application Content	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Controler Function	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
Controler Content	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
C-Agent	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
Controler RDB	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
Data Processing Engine	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Data Source	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Data Sink	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Network Element RDB	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
A-CPI Agent	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
A-CPI	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
C-CPI	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
D-CPI	2,00	2,00	1,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	1,00	1,00	1,00	2,00	2,00	2,00	2,00	2,00	2,00
D-CPI Agent	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Management Function	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Management Content	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Application Coordinator	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Controler Coordinator	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00
Network Element Coordinator	1,00	1,00	0,50	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	0,50	0,50	0,50	1,00	1,00	1,00	1,00	1,00	1,00

B Externalization

Normalized Matrix	Application Function	Application Content	Controler Function	Controler Content	C-Agent	Controler RDB	Data Processing Engine	Data Source	Data Sink	Network Element RDB	A-CPI Agent	A-CPI	C-CPI	D-CPI	D-CPI Agent	Management Function	Management Content	Application Coordinator	Controler Coordinator	Network Element Coordinator
Application Function	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Application Content	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Controler Function	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
Controler Content	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
C-Agent	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
Controler RDB	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
Data Processing Engine	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Data Source	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Data Sink	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Network Element RDB	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
A-CPI Agent	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
A-CPI	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
C-CPI	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
D-CPI	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741	0,0741
D-CPI Agent	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Management Function	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Management Content	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Application Coordinator	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Controler Coordinator	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370
Network Element Coordinator	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370	0,0370

	Weights WE	Products	Ratio		
Application Function	0,0370	0,7407	20,0000		
Application Content	0,0370	0,7407	20,0000		
Controler Function	0,0741	1,4815	20,0000		
Controler Content	0,0741	1,4815	20,0000		
C-Agent	0,0741	1,4815	20,0000		
Controler RDB	0,0741	1,4815	20,0000		
Data Processing Engine	0,0370	0,7407	20,0000		
Data Source	0,0370	0,7407	20,0000		
Data Sink	0,0370	0,7407	20,0000		
Network Element RDB	0,0370	0,7407	20,0000		
A-CPI Agent	0,0370	0,7407	20,0000		
A-CPI	0,0741	1,4815	20,0000		
C-CPI	0,0741	1,4815	20,0000		
D-CPI	0,0741	1,4815	20,0000		
D-CPI Agent	0,0370	0,7407	20,0000		
Management Function	0,0370	0,7407	20,0000		
Management Content	0,0370	0,7407	20,0000		
Application Coordinator	0,0370	0,7407	20,0000		
Controler Coordinator	0,0370	0,7407	20,0000		
Network Element Coordinator	0,0370	0,7407	20,0000		
		CI	0	CI/RI	0

Part VI

References

Bibliography

[1] Lori MacVittie and David Holmes. The new data center firewall paradigm. *F5 Networks, Inc., Seattle*, 2012.

[2] Marc F Körner. *Software Defined Networking based Data Center Services*. PhD thesis, University of Berlin, 2015.

[3] Malcolm Betts, Li Fengkai, Chen Qiaogang, Manuel Paul, Lothar Reith, Luis Miguel Contreras Murillo, Nigel Davis, Sibylle Schaller, Paul Doolan, Fabian Schneider, et al. Sdn architecture. white paper, Open Networking Foundation, 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, 2016.

[4] ONF. Framework for sdn: Scope and requirements. white paper, Open Networking Foundation, 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, 2015.

[5] Thomas Zinner, Michael Jarschel, Tobias Hossfeld, Phuoc Tran-Gia, and Wolfgang Kellerer. A compass through sdn networks. Technical report, Tech. Rep. 488, University of Würzburg, 2013.

[6] Antonio Manzalini, R Saracco, C Buyukkoc, P Chemouil, S Kuklinski, A Gladisch, M Fukui, E Dekel, D Soldani, M Ulema, et al. Software-defined networks for future networks and services. In *White Paper based on the IEEE Workshop SDN4FNS*, 2013.

[7] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[8] Fernando MV Ramos, Diego Kreutz, and Paulo Verissimo. Software-defined networks: On the road to the softwarization of networking. *Cutter IT journal*, 2015.

[9] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, David Walker, et al. Composing software defined networks. In *NSDI*, volume 13, pages 1–13, 2013.

[10] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[11] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2015.

[12] Michael Jarschel, Thomas Zinner, Tobias Hoßfeld, Phuoc Tran-Gia, and Wolfgang Kellerer. Interfaces, attributes, and use cases: A compass for sdn. *IEEE Communications Magazine*, 52(6):210–217, 2014.

[13] Stuart Bailey, Deepak Bansal, Linda Dunbar, Dave Hood, Zoltán Lajos Kis, Ben Mack-Crane, Jeff Maguire, Dan Malek, David Meyer, Manuel Paul, et al. Sdn architecture overview. white paper, Open Networking Foundation, 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, 2013.

[14] Malcolm Betts, Steve Fratini, Nigel Davis, Dave Hood, Rob Dolin, Mandar Joshi, Paul Doolan, Kam Lam, Fabian Schneider, Scott Mansfield, et al. Sdn architecture. white paper, Open Networking Foundation, 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, 2014.

[15] Martin Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Communications of the ACM*, 57(10):86–95, 2014.

[16] Rahim Masoudi and Ali Ghaffari. Software defined networks: A survey. *Journal of Network and Computer Applications*, 67:1–25, 2016.

[17] Wolfgang Braun and Michael Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[18] Andreas Richard Voellmy. *Programmable and Scalable Software-Defined Networking Controllers*. PhD thesis, Yale University, 2014.

[19] K JayachandraBabu, Prasad S.G Raghavendra, and Jitendranath Mungara. Development of openflow soft-switch. *International Journal of Emerging Technology and Advanced Engineering*, 4(3):782–785, 2014.

[20] Joe Stringer. *Contributing to OpenFlow 1.1 Support in Open vSwitch*. PhD thesis, University of Waikato, 2012.

[21] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. Performance characteristics of virtual switching. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 120–125. IEEE, 2014.

[22] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *NSDI*, pages 117–130, 2015.

[23] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtual switching in an era of advanced edges, 2010.

[24] Haoyu Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 127–132. ACM, 2013.

[25] Jingzhou Yu, Xiaozhong Wang, Jian Song, Yuanming Zheng, and Haoyu Song. Forwarding programming in protocol-oblivious instruction set. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 577–582. IEEE, 2014.

[26] B. Pfaff and B. Davie. The open vswitch database management protocol. RFC 7047, RFC Editor, December 2013. http://www.rfc-editor.org/rfc/rfc7047.txt.

[27] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51, 2014.

[28] Bartosz Belter, Artur Binczewski, Krzysztof Dombek, Artur Juszczyk, Lukasz Ogrodowczyk, Damian Parniewicz, Maciej Stroiński, and Iwo Olszewski. Programmable abstraction of datapath. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 7–12. IEEE, 2014.

[29] Damian Parniewicz, Roberto Doriguzzi Corin, Lukasz Ogrodowczyk, Mehdi Rashidi Fard, Jon Matias, Matteo Gerola, Victor Fuentes, Umar Toseef, Adel Zaalouk, Bartosz Belter, et al. Design and implementation of an openflow hardware abstraction layer. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pages 71–76. ACM, 2014.

[30] Alberto Rodriguez-Natal, Marc Portoles-Comeras, Vina Ermagan, Darrel Lewis, Dino Farinacci, Fabio Maino, and Albert Cabellos-Aparicio. Lisp: a southbound sdn protocol? *IEEE Communications Magazine*, 53(7):201–207, 2015.

[31] CISCO. Opflex: An open policy protocol white paper, 2014.

[32] Anders Nygren, Ben Pfaff, Bob Lantz, Brandon Heller, Casey Barker, Curt Beckmann, Dan Cohn, Dan Malek, Dan Talayco, David Erickson, et al. Openflow switch specification version 1.5.1. Specification, Open Networking Foundation, March 2015.

[33] Timothy L Hinrichs, Natasha S Gude, Martin Casado, John C Mitchell, and Scott Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 1–10. ACM, 2009.

[34] Andreas Voellmy and Paul Hudak. Nettle: Taking the sting out of programming network routers. *Practical Aspects of Declarative Languages*, pages 235–249, 2011.

[35] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. In *ACM Sigplan Notices*, volume 46, pages 279–291. ACM, 2011.

[36] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 43–48. ACM, 2012.

[37] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular sdn programming with pyretic. *Technical Reprot of USENIX*, 2013.

[38] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[39] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

[40] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[41] FUJITA Tomonori. Introduction to ryu sdn framework. *Open Networking Summit*, 2013.

[42] H Yin, H Xie, T Tsou, D Lopez, P Aranda, and R Sidi. Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains. *IETF draft, work in progress*, 2012.

[43] Messaoud Aouadj. *AirNet: le modèle de virtualisation «Edge-Fabric» comme plan de contrôle pour les réseaux programmables*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2016.

[44] Minh Pham and Doan B Hoang. Sdn applications-the intent-based northbound interface realisation for extended applications. In *NetSoft Conference and Workshops (NetSoft)*, pages 372–377. IEEE, 2016.

[45] Wei Zhou, Li Li, Min Luo, and Wu Chou. Rest api design patterns for sdn northbound api. In *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, pages 358–365. IEEE, 2014.

[46] Federico M Facca, Elio Salvadori, Holger Karl, Diego R López, Pedro Andrés Aranda Gutiérrez, Dejan Kostic, and Roberto Riggio. Netide: First steps towards an integrated development environment for portable network apps. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 105–110. IEEE, 2013.

[47] R Doriguzzi-Corin, Elio Salvadori, PA Aranda Gutiérrez, Christian Stritzke, Alec Leckey, Kevin Phemius, Elisa Rojas, and Carmen Guerrero. Netide: removing vendor lock-in in sdn. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–2. IEEE, 2015.

[48] Yosr Jarraya, Taous Madi, and Mourad Debbabi. A survey and a layered taxonomy of software-defined networking. *IEEE communications surveys & tutorials*, 16(4):1955–1980, 2014.

[49] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2014.

[50] Saurav Das, Guru Parulkar, and Nick McKeown. Unifying packet and circuit switched networks. In *GLOBECOM Workshops*, pages 1–6. IEEE, 2009.

[51] Roberto Bifulco and Fabian Schneider. Openflow rules interactions: definition and detection. In *SDN forFuture Networks and Services (SDN4FNS)*, pages 1–6. IEEE, 2013.

[52] Mohammed Basheer Al-Somaidai and Estabrak Bassam Yahya. Survey of software components to emulate openflow protocol as an sdn implementation. *American Journal of Software Engineering and Applications*, 3(6):74–82, 2014.

[53] Sahil Sachdeva. Software defined networks. Engineering report, Institute of Development and Research in Banking Technology, Road No. 1, Castle Hills, Masab Tank, Hyderabad-500057, 2014.

[54] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Software-defined networking security: pros and cons. *IEEE Communications Magazine*, 53(6):73–79, 2015.

[55] Kshira Sagar Sahoo, Sagarika Mohanty, Mayank Tiwary, Brojo Kishore Mishra, and Bibhudatta Sahoo. A comprehensive tutorial on software defined network: The driving force for the future internet technology. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, page 114. ACM, 2016.

[56] CloudPassage. What csos need to know about software-defined security, 2015.

[57] Fei Hu, Qi Hao, and Ke Bao. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, 16(4):2181–2206, 2014.

[58] Aaron Yi Ding, Jon Crowcroft, Sasu Tarkoma, and Hannu Flinck. Software defined networking for security enhancement in wireless mobile networks. *Computer Networks*, 66:94–101, 2014.

[59] Tomas Hegr, Leos Bohac, Vojtech Uhlir, and Petr Chlumsky. Openflow deployment and concept analysis. *Advances in Electrical and Electronic Engineering*, 11(5):327, 2013.

[60] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[61] Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, 2013.

[62] Kapil Dhamecha and Bhushan Trivedi. Sdn issues-a survey. *International Journal of Computer Applications*, 73(18), 2013.

[63] Paulo Fonseca and Edjard Mota. A survey on fault management in software-defined networks. *IEEE Communications Surveys & Tutorials*, 2017.

[64] Lav Gupta. Sdn: Development, adoption and research trends. White paper, Washington University in St. Louis, One Brookings Drive St. Louis, Missouri 63130, 2013.

[65] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, pages 1–12. ACM, 2007.

[66] Deepak Kumar and Manu Sood. Software defined networking: A concept and related issues. *International Journal of Advanced Networking and Applications*, 6(2):2233, 2014.

[67] BJ Van Asten. *Increasing robustness of Software-Defined Networks*. PhD thesis, Delft University of Technology, 2014.

[68] Paul Zanna, Sepehr Hosseini, Pj Radcliffe, and Benjamin O'Neill. The challenges of deploying a software defined network. In *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*, pages 111–116. IEEE, 2014.

[69] Guru Parulkar, Timon Sloane, Saurav Das, and Cassandra Blair. Open networking foundation. https://www.opennetworking.org/, 2017. Accessed: 2017-07-04.

[70] Barbara Guttman and Edward A Roback. *An introduction to computer security: the NIST handbook*. DIANE Publishing, 1995.

[71] R. Shirey. Internet security glossary. RFC 2828, RFC Editor, May 2000.

[72] ITUT Recommendation. Security architecture for systems providing end-to-end communications, 2002.

[73] ITUT Recommendation. Overview of cybersecurity, 2008.

[74] Mohammed Alhabeeb, Abdullah Almuhaideb, Phu Dung Le, and Bala Srinivasan. Information security threats classification pyramid. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 208–213. IEEE, 2010.

[75] Anil Bazaz and James D Arthur. Towards a taxonomy of vulnerabilities. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 163a–163a. IEEE, 2007.

[76] Gary Steffen. Chapter 1 vulnerabilities threats and attacks. Course, Indiana University Purdue University Fort Wayne (IPFW), 2101 E. Coliseum Blvd. Fort Wayne, Indiana 46805, 2007.

[77] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. A fair solution to dns amplification attacks. In *Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on*, pages 38–47. IEEE, 2007.

[78] Samaneh Rastegari, M Iqbal Saripan, and Mohd Fadlee A Rasid. Detection of denial of service attacks against domain name system using machine learning classifiers. In *Proceedings of the 18th World Congress on Engineering*, 2010.

[79] Samaneh Rastegari, M Iqbal Saripan, and Mohd Fadlee A Rasid. Detection of denial of service attacks against domain name system using neural networks. *arXiv preprint arXiv:0912.1815*, 2009.

[80] Jisa David and Ciza Thomas. Ddos attack detection using fast entropy approach on flow-based network traffic. *Procedia Computer Science*, 50:30–36, 2015.

[81] Manu Sood et al. A survey on issues of concern in software defined networks. In *Image Information Processing (ICIIP), 2015 Third International Conference on*, pages 295–300. IEEE, 2015.

[82] Lisa Schehlmann, Sebastian Abt, and Harald Baier. Blessing or curse? revisiting security aspects of software-defined networking. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 382–387. IEEE, 2014.

[83] Open Networking Foundation. Principles and practices for securing software-defined networks, 2015.

[84] Margaret Wasserman and Sam Hartman. Security analysis of the open networking foundation (onf) openflow switch specification, 2013.

[85] Rowan Kloti, Vasileios Kotronis, and Paul Smith. Openflow: A security analysis. In *Network Protocols (ICNP), 2013 21st IEEE International Conference*, pages 1–6. IEEE, 2013.

[86] D Romão, N Van Dijkhuizen, S Konstantaras, and G Thessalonikefs. Practical security analysis of openflow. *University of Amsterdam, Amsterdam*, 2013.

[87] Wanqing You, Kai Qian, Xi He, and Ying Qian. Openflow security threat detection and defense services. *International Journal of Advanced Networking and Applications*, 6(3):2347, 2014.

[88] Rowan Kloti. Openflow: A security analysis, 2013.

[89] Maragathavalli Palanivel and Kanmani Selvadurai. Risk-driven security testing using risk analysis with threat modeling approach. *SpringerPlus*, 3(1):754, 2014.

[90] Shibo Luo, Mianxiong Dong, Kaoru Ota, Jun Wu, and Jianhua Li. A security assessment mechanism for software-defined networking-based mobile networks. *Sensors*, 15(12):31843–31858, 2015.

[91] Thomas L Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[92] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.

[93] Ellen Fanning. Software-defined networks, 2014.

[94] Linyuan Yao, Ping Dong, Tao Zheng, Hongke Zhang, Xiaojiang Du, and Mohsen Guizani. Network security analyzing and modeling based on petri net and attack tree for sdn. In *Computing, Networking and Communications (ICNC), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[95] Marc C Dacier, Hartmut König, Radoslaw Cwalinski, Frank Kargl, and Sven Dietrich. Security challenges and opportunities of software-defined networking. *IEEE Security & Privacy*, 15(2):96–100, 2017.

[96] Andres J Gonzalez, Gianfranco Nencioni, Bjarne E Helvik, and Andrzej Kamisinski. A fault-tolerant and consistent sdn controller. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.

[97] Mohamed Azab and José AB Fortes. Towards proactive sdn-controller attack and failure resilience. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 442–448. IEEE, 2017.

[98] Ramanpreet Kaur, Amardeep Singh, Sharanjit Singh, and Shruti Sharma. Security of software defined networks: Taxonomic modeling, key components and open research area. In *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*, pages 2832–2839. IEEE, 2016.

[99] Mudit Saxena and Rakesh Kumar. A recent trends in software defined networking (sdn) security. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, pages 851–855. IEEE, 2016.

[100] Yang Wu, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. Answering why-not queries in software-defined networks with negative provenance. In *Proceedings of the twelfth ACM workshop on hot topics in networks*, page 3. ACM, 2013.

[101] KAIST NSS lab. An overview of misuse attack cases. http://www.sdnsecurity.org/vulnerability/attacks/, 2017. Accessed: 2018-02-20.

[102] Yoshiaki Hori, Seiichiro Mizoguchi, Ryosuke Miyazaki, Akira Yamada, Yaokai Feng, Ayumu Kubota, and Kouichi Sakurai. A comprehensive security analysis checksheet for openflow networks. In *International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 231–242. Springer, 2016.

[103] Mauro Conti, Fabio De Gaspari, and Luigi Vincenzo Mancini. A novel stealthy attack to gather sdn configuration-information. *IEEE Transactions on Emerging Topics in Computing*, 2018.

[104] Roberto Bifulco, Heng Cui, Ghassan O Karame, and Felix Klaedtke. Fingerprinting software-defined networks. In *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*, pages 453–459. IEEE, 2015.

[105] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166. ACM, 2013.

[106] Abdelhadi Azzouni, Othmen Braham, Thi Mai Trang Nguyen, Guy Pujolle, and Raouf Boutaba. Fingerprinting openflow controllers: The first step to attack an sdn control plane. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.

[107] Felix Klaedtke, Ghassan O Karame, Roberto Bifulco, and Heng Cui. Access control for sdn controllers. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 219–220. ACM, 2014.

[108] Dongting Yu, Andrew W Moore, Chris Hall, and Ross Anderson. Authentication for resilience: the case of sdn. In *Cambridge International Workshop on Security Protocols*, pages 39–44. Springer, 2013.

[109] Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 171–172. ACM, 2013.

[110] Sandra Scott-Hayward, Christopher Kane, and Sakir Sezer. Operationcheckpoint: Sdn application control. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 618–623. IEEE, 2014.

[111] eTutorials. The four primary types of network attack. http://etutorials.org/Networking/Cisco+Certified+Security+Professional+Certification/Part+I+Introduction+to+Network+Security/Chapter+1+Understanding+Network+Security+Threats/The+Four+Primary+Types+of+Network+Attack/, 2018. Accessed: 2018-02-26.

[112] V KRISHNA REDDY and D SREENIVASULU. Software-defined networking with ddos attacks in cloud computing. *International Journal of Innovative Technologies*, 4(19):3779–3783, 2016.

[113] Microsoft. Common types of network attacks. https://technet.microsoft.com/en-us/library/cc959354.aspx, 2018. Accessed: 2018-02-26.

[114] Andry Putra Fajar and Tito Waluyo Purboyo. A survey paper of distributed denial-of-service attack in software defined networking (sdn). *International Journal of Applied Engineering Research*, 13(1):476–482, 2018.

[115] Jeremy M Dover. A denial of service attack against the open floodlight sdn controller. *Dover Networks, Tech. Rep.*, 2013.

[116] Rajat Kandoi and Markku Antikainen. Denial-of-service attacks in openflow sdn networks. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1322–1326. IEEE, 2015.

[117] Nhu-Ngoc Dao, Junho Park, Minho Park, and Sungrae Cho. A feasible method to combat against ddos attack in sdn network. In *Information Networking (ICOIN), 2015 International Conference on*, pages 309–311. IEEE, 2015.

[118] Duohe Ma, Zhen Xu, and Dongdai Lin. Defending blind ddos attack on sdn based on moving target defense. In *International Conference on Security and Privacy in Communication Systems*, pages 463–480. Springer, 2014.

[119] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media, 2011.

[120] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[121] Hiep T Nguyen Tri and Kyungbaek Kim. Assessing the impact of resource attack in software defined network. In *Information Networking (ICOIN), 2015 International Conference on*, pages 420–425. IEEE, 2015.

[122] Bin Yuan, Deqing Zou, Shui Yu, Hai Jin, Weizhong Qiang, and Jinan Shen. Defending against flow table overloading attack in software-defined networks. *IEEE Transactions on Services Computing*, 2016.

[123] S Padmaja and V Vetriselvi. Mitigation of switch-dos in software defined network. In *Information Communication and Embedded Systems (ICICES), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[124] Yicong Zhang, Jie Li, Lin Chen, Yusheng Ji, and Feilong Tang. A novel method against the firewall bypass threat in openflow networks. In *Wireless Communications and Signal Processing (WCSP), 2017 9th International Conference on*, pages 1–6. IEEE, 2017.

[125] Rami Ghannam and Anthony Chung. Handling malicious switches in software defined networks. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 1245–1248. IEEE, 2016.

[126] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*, 2015.

[127] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. How to detect a compromised sdn switch. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–6. IEEE, 2015.

[128] David R Miller, Shon Harris, Allen Harper, Stephen VanDyke, and Chris Blask. *Security Information and Event Management (SIEM) Implementation (Network Pro Library)*. McGraw Hill, 2010.

[129] Tzu-Wei Chao, Yu-Ming Ke, Bo-Han Chen, Jhu-Lin Chen, Chen Jung Hsieh, Shao-Chuan Lee, and Hsu-Chun Hsiao. Securing data planes in software-defined networks. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pages 465–470. IEEE, 2016.

[130] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS*, volume 15, pages 8–11, 2015.

[131] Aliyu Lawal Aliyu, Peter Bull, and Ali Abdallah. *Investigating the Security Aspect of Software Defined Networking*. Birmingham City University, 2016.

[132] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyi Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, et al. Leveraging sdn layering to systematically troubleshoot networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 37–42. ACM, 2013.

[133] Wanderson Paim de Jesus, Daniel Alves da Silva, Rafael T Júnior, and Francisco Vitor Lopes da Frota. Analysis of sdn contributions for cloud computing security. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 922–927. IEEE Computer Society, 2014.

[134] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. Enhancing network security through software defined networking (sdn). In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–9. IEEE, 2016.

[135] Open Networking Foundation. Sdn security considerations in the data center, 2013.

[136] RL Smeliansky. Sdn for network security. In *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International*, pages 1–5. IEEE, 2014.

[137] Sytel Reply. Advanced security mechanisms to protect assets and networks: Software-defined security, 2015.

[138] Bob Shaw. Security-centric sdn: A new approach to implement network security that works, 2013.

[139] Roberto Bifulco and Ghassan Karame. Towards a richer set of services in software-defined networks. In *Proceedings of the NDSS Workshop on Security of Emerging Technologies (SENT)*, 2014.

[140] Gabriel Sund and Haroon Ahmed. Security challenges within software defined networks. Technical report, KTH Royal Institute of Technology, 2014.

[141] Madhusanka Liyanage, Ahmed Bux Abro, Mika Ylianttila, and Andrei Gurtov. Opportunities and challenges of software-defined mobile networks in network security. *IEEE Security & Privacy*, 14(4):34–44, 2016.

[142] Radu F Babiceanu and Remzi Seker. Software-defined networking-based models for secure interoperability of manufacturing operations. In *Service Orientation in Holonic and Multi-Agent Manufacturing*, pages 243–252. Springer, 2018.

[143] Kristian Slavov, Migault Daniel, and Pourzandi Makan. Identifying and addressing the vulnerabilities and security issues of sdn. Report, Ericsson, Ericsson SE-164 83 Stockholm, Sweden, 2015.

[144] Wenjuan Li, Weizhi Meng, et al. A survey on openflow-based software defined networks: Security challenges and countermeasures. *Journal of Network and Computer Applications*, 68:126–139, 2016.

[145] Syed Taha Ali, Vijay Sivaraman, Adam Radford, and Sanjay Jha. A survey of securing networks using software defined networking. *IEEE transactions on reliability*, 64(3):1086–1097, 2015.

[146] Ijaz Ahmad, Suneth Namal, Mika Ylianttila, and Andrei Gurtov. Security in software defined networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(4):2317–2346, 2015.

[147] Kubra Kalkan and Sherali Zeadally. Securing internet of things (iot) with software defined networking (sdn). *IEEE Communications Magazine*, 2017.

[148] Krzysztof Cabaj, Jacek Wytrebowicz, Slawomir Kuklinski, Pawel Radziszewski, and Khoa Truong Dinh. Sdn architecture impact on network security. In *FedCSIS position papers*, pages 143–148, 2014.

[149] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *2012 20th IEEE international conference on network protocols (ICNP)*, pages 1–6. IEEE, 2012.

[150] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[151] Seungwon Shin, Phillip Porras, Vinod Yegneswaran, and Guofei Gu. A framework for integrating security services into software-defined networks. *Proceedings of the 2013 open networking summit (Research Track poster paper), ser. ONS*, 13, 2013.

[152] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

[153] Adrian Lara and Byrav Ramamurthy. Opensec: Policy-based security using software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):30–42, 2016.

[154] Yongning Tang, Guang Cheng, Zhiwei Xu, Feng Chen, Khalid Elmansor, and Yangxuan Wu. Automatic belief network modeling via policy inference for sdn fault localization. *Journal of Internet Services and Applications*, 7(1):1, 2016.

[155] B Batista and M Fernandez. Ponderflow: A policy specification language for openflow networks. In *The Thirteenth International Conference on Networks*, pages 204–209, 2014.

[156] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. Springer, 2001.

[157] Yaniv Ben-Itzhak, Katherine Barabash, Rami Cohen, Anna Levin, and Eran Raichstein. Enforsdn: Network policies enforcement with sdn. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 80–88. IEEE, 2015.

[158] Diogo Menezes Ferrazani Mattos and Otto Carlos Muniz Bandeira Duarte. Authflow: authentication and access control mechanism for software defined networking. *Annals of Telecommunications*, 71(11-12):607–615, 2016.

[159] P. Congdon, B. Aboba, A. Smith, G. Zorn, and J. Roese. Ieee 802.1x remote authentication dial in user service (radius) usage guidelines. RFC 3580, RFC Editor, September 2003.

[160] Sadiq T Yakasai and Chris G Guy. Flowidentity: Software-defined network access control. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 115–120. IEEE, 2015.

[161] Almulla Hesham, Fragkiskos Sardis, Stan Wong, Toktam Mahmoodi, and Mallikarjun Tatipamula. A simplified network access control design and implementation for m2m communication using sdn. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2017 IEEE*, pages 1–5. IEEE, 2017.

[162] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.

[163] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[164] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi 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*, pages 127–132. ACM, 2012.

[165] Rupam Kumar Sharma, Hemanta Kumar Kalita, and Biju Issac. Different firewall techniques: A survey. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[166] Michelle Suh, Sae Hyong Park, Byungjoon Lee, and Sunhee Yang. Building firewall over the software-defined network controller. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pages 744–748. IEEE, 2014.

[167] Chaitra N Shivayogimath and NV Uma Reddy. Modification of l2 learning switch code for firewall functionality in pox controller. In *Silicon Photonics & High Performance Computing*, pages 103–114. Springer, 2018.

[168] Wajdy M Othman, Hao Chen, Ammar Al-Moalmi, and Ali N Hadi. Implementation and performance analysis of sdn firewall on pox controller. In *Communication Software and Networks (ICCSN), 2017 IEEE 9th International Conference on*, pages 1461–1466. IEEE, 2017.

[169] Jake Collings and Jun Liu. An openflow-based prototype of sdn-oriented stateful hardware firewalls. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 525–528. IEEE, 2014.

[170] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. Enabling security functions with sdn: A feasibility study. *Computer Networks*, 85:19–35, 2015.

[171] Karamjeet Kaur, Krishan Kumar, Japinder Singh, and Navtej Singh Ghumman. Programmable firewall using software defined networking. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, pages 2125–2129. IEEE, 2015.

[172] Tariq Javid, Tehseen Riaz, and Asad Rasheed. A layer2 firewall for software defined network. In *Information Assurance and Cyber Security (CIACS), 2014 Conference on*, pages 39–42. IEEE, 2014.

[173] Justin Gregory V Pena and William Emmanuel Yu. Development of a distributed firewall using software defined networking technology. In *Information Science and Technology (ICIST), 2014 4th IEEE International Conference on*, pages 449–452. IEEE, 2014.

[174] Thuy Vinh Tran and Heejune Ahn. A network topology-aware selectively distributed firewall control in sdn. In *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*, pages 89–94. IEEE, 2015.

[175] Sergey Morzhov, Igor Alekseev, and Mikhail Nikitinskiy. Firewall application for floodlight sdn controller. In *Control and Communications (SIBCON), 2016 International Siberian Conference on*, pages 1–5. IEEE, 2016.

[176] Nayana Zope, Sanjay Pawar, and Zia Saquib. Firewall and load balancing as an application of sdn. In *Advances in Signal Processing (CASP), Conference on*, pages 354–359. IEEE, 2016.

[177] Mohd Abuzar Sayeed, Mohd Asim Sayeed, and Sharad Saxena. Intrusion detection system based on software defined network firewall. In *Next Generation Computing Technologies (NGCT), 2015 1st International Conference on*, pages 379–382. IEEE, 2015.

[178] Dhaval Satasiya, Rupal Raviya, and Hiresh Kumar. Enhanced sdn security using firewall in a distributed scenario. In *Advanced Communication Control and Computing Technologies (ICACCCT), 2016 International Conference on*, pages 588–592. IEEE, 2016.

[179] Jay Shah. Implementation and performance analysis of firewall on open vswitch. Technical report, Technische Universitat Munchen, 2015.

[180] Deepa Balagopal and X Agnise Kala Rani. Netwatch: Empowering software-defined network switches for packet filtering. In *Applied and Theoretical Computing and Communication Technology (iCATccT), 2015 International Conference on*, pages 837–840. IEEE, 2015.

[181] Deepa Balagopal and X Agnise Kala Rani. Empowering sdn firewall against arp poison routing. *International Journal of Applied Engineering Research*, 12(18):7466–7469, 2017.

[182] Thuy Vinh Tran and Heejune Ahn. Flowtracker: A sdn stateful firewall solution with adaptive connection tracking and minimized controller processing. In *Software Networking (ICSN), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[183] Avinash Kumar and NK Srinath. Implementing a firewall functionality for mesh networks using sdn controller. In *Computation System and Information Technology for Sustainable Solutions (CSITSS), International Conference on*, pages 168–173. IEEE, 2016.

[184] Vipin Gupta, Sukhveer Kaur, and Karamjeet Kaur. Implementation of stateful firewall using pox controller. In *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*, pages 1093–1096. IEEE, 2016.

[185] Amandeep Kaur and Vikramjit Singh. Building l2-l4 firewall using software defined networking. *International Journal of Innovations and Advancement in Computer Science*, 6(6):159–167, 2017.

[186] Mathis Steichen, Stefan Hommes, and Radu State. Chainguard—a firewall for blockchain applications using sdn with openflow. In *Principles, Systems and Applications of IP Telecommunications (IPTComm), 2017*, pages 1–8. IEEE, 2017.

[187] Xinpei Jia and Joseph KH Wang. Distributed firewall for p2p network in data center. In *ICCE-China Workshop (ICCE-China), 2013 IEEE*, pages 15–19. IEEE, 2013.

[188] Karamjeet Kaur, Sukhveer Kaur, and Vipin Gupta. Software defined networking based routing firewall. In *Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on*, pages 267–269. IEEE, 2016.

[189] A Mahesh, Adhiyan Chandrasekaran, R ArunKumar, K SivaKumar, and N Vigneshwaran. Cloud based firewall on openflow sdn network. In *Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), 2017 International Conference on*, pages 1–6. IEEE, 2017.

[190] Jarrod N Bakker, Ian Welch, and Winston KG Seah. Network-wide virtual firewall using sdn/openflow. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pages 62–68. IEEE, 2016.

[191] Alaauddin Shieha. Application layer firewall using openflow. Technical report, University of Colorado at Boulder, 2014.

[192] Andis Arins. Firewall as a service in sdn openflow network. In *Information, Electronic and Electrical Engineering (AIEEE), 2015 IEEE 3rd Workshop on Advances in*, pages 1–5. IEEE, 2015.

[193] M Saad Waheed, M Al Mufarrej, M Sobhieh, A Al Barrak, A Baig, and A Al Mazyad. Implementation of virtual firewall function in sdn (software defined networks). In *9th IEEE-GCC Conference and Exhibition (GCCCE)*. IEEE, 2017.

[194] Dustin Grant, Sandeep Gupta, Sridhar Narahari, and Michael J Satterlee. Methods and apparatus to provide a distributed firewall in a network, August 31 2017. US Patent App. 15/594,010.

[195] Sajad Shirali-Shahreza and Yashar Ganjali. Efficient implementation of security applications in openflow controller with flexam. In *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*, pages 49–54. IEEE, 2013.

[196] Sajad Shirali-Shahreza and Yashar Ganjali. Flexam: flexible sampling extension for monitoring and security applications in openflow. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 167–168. ACM, 2013.

[197] Sajad Shirali-Shahreza and Yashar Ganjali. Empowering software defined network controller with packet-level information. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 1335–1339. IEEE, 2013.

[198] Sajad Shirali-Shahreza and Yashar Ganjali. Protecting home user devices with a sdn-based firewall. *IEEE Transactions on Consumer Electronics*, 2018.

[199] Wonkyu Han, Hongxin Hu, Ziming Zhao, Adam Doupé, Gail-Joon Ahn, Kuang-Ching Wang, and Juan Deng. State-aware network access management for software-defined networks. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, pages 1–11. ACM, 2016.

[200] Hongxin Hu, Gail-Joon Ahn, Wonkyu Han, and Ziming Zhao. Towards a reliable sdn firewall. In *ONS*, 2014.

[201] W Juan, W Jiang, C Shiya, J Hongyang, and K Qianglong. Sdn (self-defending network) firewall state detecting method and system based on openflow protocol. *China Patent CN*, 104104561, 2014.

[202] Justin Pettit and Thomas Graf. Stateful connection tracking & stateful nat, 2014.

[203] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and TV Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2014.

[204] Masoud Moshref, Apoorv Bhargava, Adhip Gupta, Minlan Yu, and Ramesh Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 61–66. ACM, 2014.

[205] Shuyong Zhu, Jun Bi, Chen Sun, Chenhui Wu, and Hongxin Hu. Sdpa: Enhancing stateful forwarding for software-defined networking. In *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*, pages 323–333. IEEE, 2015.

[206] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[207] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[208] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 29–43. ACM, 2016.

[209] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 15–28. ACM, 2016.

[210] Jedidiah McClurg, Hossein Hojjat, Nate Foster, and P Cerny. Specification and compilation of event-driven sdn programs. *arXiv preprint arXiv: 1507.07049*, 2015.

[211] P Ayuso. Netfilter's connection tracking system. *LOGIN: The USENIX magazine*, 31(3), 2006.

[212] Tooska Dargahi, Alberto Caponi, Moreno Ambrosin, Giuseppe Bianchi, and Mauro Conti. A survey on the security of stateful sdn data planes. *IEEE Communications Surveys & Tutorials*, 19(3):1701–1725, 2017.

[213] Juan Deng, Hongxin Hu, Hongda Li, Zhizhong Pan, Kuang-Ching Wang, Gail-Joon Ahn, Jun Bi, and Younghee Park. Vnguard: An nfv/sdn combination framework for provisioning and managing virtual firewalls. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 107–114. IEEE, 2015.

[214] Simeon Miteff and Scott Hazelhurst. Nfshunt: A linux firewall with openflow-enabled hardware bypass. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 100–106. IEEE, 2015.

[215] Casimer Decusatis and Peter Mueller. Virtual firewall performance as a waypoint on a software defined overlay network. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, pages 819–822. IEEE, 2014.

[216] Juan Deng and Hongda Li. On the safety and efficiency of virtual firewall elasticity control. In *24th Network and Distributed System Security Symposium (NDSS 2017)*, 2017.

[217] Antônio J Pinheiro, Ethel B Gondim, and Divanilson R Campelo. An efficient architecture for dynamic middlebox policy enforcement in sdn networks. *Computer Networks*, 122:153–162, 2017.

[218] Simeon Miteff. An sdn-based firewall shunt for data-intensive science applications. Technical report, University of the Witwatersrand, 2016.

[219] Rahul Tajpuriya, Vinit Gupta, and Indr Jeet Rajput. A systematic approach for highly secure framework for virtual firewall. *International Journal For Technological Research In Engineering*, 4(8):2347–4718, 2017.

[220] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. *ACM SIGCOMM computer communication review*, 43(4):27–38, 2013.

[221] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *NSDI*, volume 14, pages 533–546, 2014.

[222] Claas Lorenz, David Hock, Johann Scherer, Raphael Durner, Wolfgang Kellerer, Steffen Gebert, Nicholas Gray, Thomas Zinner, and Phuoc Tran-Gia. An sdn/nfv-enabled enterprise network architecture offering fine-grained security policy enforcement. *IEEE communications magazine*, 55(3):217–223, 2017.

[223] Florian Heimgaertner, Mark Schmidt, David Morgenstern, and Michael Menth. A software-defined firewall bypass for congestion offloading. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[224] Vasaka Visoottiviseth, Suthasinee Lertviriyasawat, Peerada Suppiyatrakoon, Pattarajit Chitkornkitsil, and Nariyoshi Yamai. Reflo: Reactive firewall system with openflow and flow monitoring system. In *Region 10 Conference, TENCON 2017-2017 IEEE*, pages 2273–2278. IEEE, 2017.

[225] Jérôme François, Lautaro Dolberg, Olivier Festor, and Thomas Engel. Network security through software defined networking: a survey. In *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications*, page 6. ACM, 2014.

[226] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.

[227] Sebastian Seeber, Lars Stiemert, and Gabi Dreo Rodosek. Towards an sdn-enabled ids environment. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 751–752. IEEE, 2015.

[228] Nen-Fu Huang, Chuang Wang, I-Ju Liao, Che-Wei Lin, and Chia-Nan Kao. An openflow-based collaborative intrusion prevention system for cloud networking. In *Communication Software and Networks (ICCSN), 2015 IEEE International Conference on*, pages 85–92. IEEE, 2015.

[229] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Proceedings of LISA 99: 13th Systems Administration Conference Lisa*, pages 229–238. USENIX, 1999.

[230] Jack Koziol. *Intrusion detection with Snort*. Sams Publishing, 2003.

[231] Usama Ahmed, Imran Raza, Syed Asad Hussain, Amjad Ali, Muddesar Iqbal, and Xinheng Wang. Modelling cyber security for software-defined networks those grow strong when exposed to threats. *Journal of Reliable Intelligent Environments*, 1(2-4):123–146, 2015.

[232] Adnan Akhunzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Muhammad Imran, and Sghaier Guizani. Securing software defined networks: taxonomy, requirements, and open issues. *IEEE Communications Magazine*, 53(4):36–44, 2015.

[233] Kevin J Soo Hoo. *How much is enough? A risk management approach to computer security*. Stanford University Stanford, Calif, 2000.

[234] Vinay M Igure and Ronald D Williams. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials*, 10(1), 2008.

[235] EUREKA Network. Project sendate, 2018.

[236] SENDATE. About the project, 2018.

[237] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6), 2006.

[238] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-Forum of Incident Response and Security Teams*, volume 1, page 23, 2007.

[239] Mike Schiffman, A Wright, D Ahmad, and G Eschelbeck. The common vulnerability scoring system. *National Infrastructure Advisory Council, Vulnerability Disclosure Working Group, Vulnerability Scoring Subgroup*, 2004.

[240] Peter Mell, Karen Ann Kent, and Sasha Romanosky. *The common vulnerability scoring system (CVSS) and its applicability to federal agency systems*. US Department of Commerce, National Institute of Standards and Technology, 2007.

[241] Kardi Teknomo. Analytic hierarchy process (ahp) tutorial. *Revoledu. com*, pages 1–20, 2006.

[242] Alessio Ishizaka and Philippe Nemery. Analytic hierarchy process. *Multi-Criteria Decision Analysis: Methods and Software*, pages 11–58, 1999.

[243] Thomas L Saaty. How to make a decision: the analytic hierarchy process. *European journal of operational research*, 48(1):9–26, 1990.

[244] Omkarprasad S Vaidya and Sushil Kumar. Analytic hierarchy process: An overview of applications. *European Journal of operational research*, 169(1):1–29, 2006.

[245] Zhihu Wang and Haiwen Zeng. Study on the risk assessment quantitative method of information security. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 6, pages V6–529. IEEE, 2010.

[246] Navneet Bhushan and Kanwal Rai. *Strategic decision making: applying the analytic hierarchy process*. Springer Science & Business Media, 2007.

[247] Thomas L Saaty. Fundamentals of the analytic hierarchy process. In *The analytic hierarchy process in natural resource and environmental decision making*, pages 15–35. Springer, 2001.

[248] FIRST.org. Common vulnerability scoring system v3.0: Specification document. https://www.first.org/cvss/specification-document#1-1-Metrics, 2018. Accessed: 2018-03-23.

[249] Park Foreman. *Vulnerability Management*. CRC Press, 2009.

[250] FIRST.org. Cvss-sig version 2 history. https://www.first.org/cvss/v2/history, 2018. Accessed: 2018-04-23.

[251] Melvin Alexander. Decision-making using the analytic hierarchy process (ahp) and sas/iml. *20th Annual South East SAS Users Group (SESUG) Conference*, pages 1–12, 2012.

[252] Thomas L Saaty. Eigenvector and logarithmic least squares. *European journal of operational research*, 48(1):156–160, 1990.

[253] Jose Antonio Alonso and M Teresa Lamata. Consistency in the analytic hierarchy process: a new approach. *International journal of uncertainty, fuzziness and knowledge-based systems*, 14(04):445–459, 2006.

[254] FIRST.org. Common vulnerability scoring system version 3.0 calculator. https://www.first.org/cvss/calculator/3.0, 2018. Accessed: 2018-03-26.

[255] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. http://www.rfc-editor.org/rfc/rfc5246.txt.

[256] J Bryan Lyles and Christoph L Scuba. A reference model for firewall technology and its implications for connection signaling. 1996.

[257] Thomson. Thomson gateway stateful inspection firewall configuration: Sif r7.4 and higher. Guide, Thomson Telecom Belgium, 2008.

[258] Safaa Zeidan and Zouheir Trabelsi. A survey on firewall's early packet rejection techniques. In *2011 International Conference on Innovations in Information Technology (IIT)*, pages 203–208. IEEE, 2011.

[259] Huang Ling-Fang. The firewall technology study of network perimeter security. In *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, pages 410–413. IEEE, 2012.

[260] Rajesh K. Stateful multilayer inspection firewalls. http://www.excitingip.com/205/what-are-packet-filtering-circuit-level-application-level-and-stateful-multilayer-inspection-firewalls/, 2009. Accessed: 2018-04-03.

[261] Kenneth Ingham and Stephanie Forrest. A history and survey of network firewalls. *University of New Mexico, Tech. Rep*, 2002.

[262] Stephen Woodall. Firewall design principles. *Computer Networks and Computer Security Coursework paper, North Carolina State University, USA*, 2004.

[263] Frank Artes, Thomas Skybakmoen, Bob Walder, Vikram Phatak, and Ryan Liles. Firewall comparative analysis: Total cost of ownership (tco). Repport, NSS Labs, 2013.

[264] Scott Hogg. Is an sdn switch a new form of a firewall? https://www.networkworld.com/article/2905257/sdn/is-an-sdn-switch-a-new-form-of-a-firewall.html, 2015. Accessed: 2018-04-04.

[265] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Series x: Data networks, open system communications and security x.1205. Repport, International Telecommunication Union, 2008.

[266] Habtamu Abie. An overview of firewall technologies. *Telektronikk*, 96(3):47–52, 2000.

[267] Behrouz A Forouzan and Sophia Chung Fegan. *TCP/IP protocol suite*. McGraw-Hill Higher Education, 2002.

[268] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[269] Piotr Rygielski, Marian Seliuchenko, Samuel Kounev, and Mykhailo Klymash. Performance analysis of sdn switches with hardware and software flow tables. In *proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 80–87. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2017.

[270] bcom. A propos de b<>com, 2018.

[271] Jungmin Son and Rajkumar Buyya. A taxonomy of software-defined networking (sdn)-enabled cloud computing. *ACM Computing Surveys (CSUR)*, 51(3):59, 2018.

[272] Yanhuang Li. *Interoperability and Negotiation of Security Policies*. PhD thesis, Ecole Nationale Supérieure des Télécommunications de Bretagne-ENSTB, 2016.

[273] Ed Coyne and Timothy R Weil. Abac and rbac: scalable, flexible, and auditable access management. *IT Professional*, 15(3):14–16, 2013.

[274] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[275] Vincent C Hu, D Richard Kuhn, and David F Ferraiolo. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[276] Yanhuang Li, Nora Cuppens-Boulahia, Jean-Michel Crom, Frédéric Cuppens, Vincent Frey, and Xiaoshu Ji. Similarity measure for security policies in service provider selection. In *International Conference on Information Systems Security*, pages 227–242. Springer, 2015.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UBO
Université de Bretagne Occidentale

Titre : Analyse de sécurité et renforcement de control d'accès à travers les réseaux programmables

Mots clés : Réseaux programmables, cyber sécurité, pare-feu, orchestration, vulnérabilité, control d'accès

Résumé : Les réseaux programmables (SDN) sont un paradigme émergeant qui promet de résoudre les limitations de l'architecture du réseau conventionnel.

Dans cette thèse, nous étudions et explorons deux aspects de la relation entre la cyber sécurité et les réseaux programmables. D'une part, nous étudions la sécurité pour les réseaux programmables en effectuant une analyse de leurs vulnérabilités. Une telle analyse de sécurité est un processus crucial pour identifier les failles de sécurité des réseaux programmables et pour mesurer leurs impacts.

D'autre part, nous explorons l'apport des réseaux programmables à la sécurité. La thèse conçoit et implémente un pare-feu programmable qui transforme la machine à états finis des protocoles réseaux, en une machine à états équivalente pour les réseaux programmables. En outre, la thèse évalue le pare-feu implémenté avec NetFilter dans les aspects de performances et de résistance aux attaques d'inondation par paquets de synchronisation. De plus, la thèse utilise l'orchestration apportée par les réseaux programmables pour renforcer la politique de sécurité dans le Cloud. Elle propose un Framework pour exprimer, évaluer, négocier et déployer les politiques de pare-feu dans le contexte des réseaux programmables sous forme de service dans le Cloud

Title : Security Analysis and Access Control Enforcement through Software Defined Networks

Keywords : Software Defined Networks, Cyber security, Firewall, Orchestration, Vulnerability, Access Control

Abstract : Software Defined Networking (SDN) is an emerging paradigm that promises to resolve the limitations of the conventional network architecture.

SDN and cyber security have a reciprocal relationship. In this thesis, we study and explore two aspects of this relationship. On the one hand, we study security for SDN by performing a vulnerability analysis of SDN. Such security analysis is a crucial process in identifying SDN security flaws and in measuring their impacts. It is necessary for improving SDN security and for understanding its weaknesses.

On the other hand, we explore SDN for security. Such an aspect of the relationship between SDN and security focusses on the advantages that SDN brings into security. The thesis designs and implements an SDN stateful firewall that transforms the Finite State Machine of network protocols to an SDN Equivalent State Machine. Besides, the thesis evaluates SDN stateful firewall and NetFilter regarding their performance and their resistance to Syn Flooding attacks. Furthermore, the thesis uses SDN orchestration for policy enforcement. It proposes a firewall policy framework to express, assess, negotiate and deploy firewall policies in the context of SDN as a Service in the cloud.