Policy-driven autonomic cyberdefense using software-defined networking

Rishikesh Sahay

THÈSE DE DOCTORAT CONJOINT TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité : Informatique et Réseaux

École doctorale : Informatique, Télécommunications et Électronique de Paris

Présentée par

Rishikesh Sahay

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

Policy-Driven Autonomic Cyberdefense using Software-Defined Networking

---

Soutenue le 14 Novembre devant le jury composé de :

Guillaume URVOY-KELLER
Professeur – Université Nice Sophia Antipolis / *Rapporteur*
Michaël HAUSPIE
Maître de conférences, HDR – Université de Lille / *Rapporteur*
Isabelle CHRISMENT
Professeure – TELECOM Nancy, Université de Lorraine / *Examinateur*
Houda LABIOD
Professeure – Télécom ParisTech / *Examinateur*
Guillaume DOYEN
Maître de conférences – Université de Technologie de Troyes / *Examinateur*
Hervé DEBAR
Professeur – Télécom SudParis / *Directeur de thèse*
Zonghua ZHANG
HDR – IMT Lille Douai / *Co-Encadrant de thèse*
Gregory BLANC
Maître de conférences – Télécom SudParis / *Co-Encadrant de thèse*

Thèse No :2017TELE0022

To my beloved family…

Acknowledgements

There are quite a few people that have contributed one way or another to the accomplishment of this work. Some of these people even come unexpectedly to our lives to give us a word of courage or just to listen to us when we are down, or when we do not find an answer to our multiple questions. I would like to thank all of you from very deep inside.

Foremost, I would like to express my gratitude to Prof. Hervé Debar my thesis director for the continuous support of my Ph.D study and research, for his comments, motivation, enthusiasm, and immense knowledge. It is absolutely difficult to succeed in the process of finding and developing an idea without the help of a specialist in the domain. I found in my director not only the source of wonderful ideas to develop, but also the support that a PhD student needs.

I would like to thank Zonghua Zhang and Gregory Blanc my thesis co-supervisors for their continuous guidance and support during my thesis. Thanks a lot to both of you for your time, dedication, your remarks -always appropriate and direct to the point. Thanks for your advices at the different stages of my thesis which helped me to complete my work.

Thanks to the European project 'NECOMA' (Nippon-European Cyberdefense-Oriented Multi-layer threat Analysis) and 'SUPERCLOUD' and all the partners with whom we worked together. Thanks to Vladimir Dombrovski for his implementation of my framework in the platform. Thanks as well to Mohammed El Barbori for his help during the implementation of the framework in mininet.

I would also like to thank Prof. Guillaume Urvoy-Keller and Dr. Michael Hauspie who, as reporters and members of the jury, had the hard task of reading my thesis and giving their advice in order to improve its content. Thanks a lot to Prof. Isabelle Chrisment, Dr. Guillaume Doyen, Prof. Houda Labiod for their interest, involvement and for being part of the jury of my thesis.

Acknowledgements

I would also like to thank Khalifa Toumi to help me during my last contribution of my thesis. Your suggestions have been very helpful in completing my work.

Thanks to my colleagues Jose, Nesrine, Fabien, Ibrahim, Yasir for all the time spent together. It has been a great experience having you all around.

I am eternally indebted to my loving parents who supported me at every stage of my life, my wife Saumya, brother, my brother's wife and my mother-in-law. Thanks for always showing the pride in their faces while referring to me and my achievements. A special thanks to my lovely wife Saumya who supported me during the last stage of my thesis. You deserve a special thanks, there are 1001 reasons for it.

I can not conclude this acknowledgement without thanking Francoise ABAD for her love and her effort in making sure that our missions were treated properly, and for always helping us in finding a solution to our flights and hotel problems.

Thanks a lot, Merci beaucoup to everybody that contributed directly and indirectly to the realization of this dissertation. Enjoy your reading ! ! !

Abstract

Cyber attacks cause significant loss not only to end-users, but also Internet Service Providers (ISP). Recently, customers of the ISP have been the number one target of the cyber attacks such as Distributed Denial of Service attacks (DDoS). These attacks are encouraged by the widespread availability of tools to launch the attacks. So, there is a crucial need to counter these attacks (DDoS, botnet attacks, etc.) by effective defense mechanisms.

Researchers have devoted huge efforts on protecting the network from cyber attacks. Defense methodologies first contains a detection process, completed by mitigation. Lack of automation in the whole cycle of detection to mitigation increase the damage caused by cyber attacks. It requires manual configurations of devices by the administrator to mitigate the attacks which cause the network downtime. Therefore, it is necessary to close the security loop with an efficient mechanism to automate the mitigation process.

In this thesis, we propose an autonomic mitigation framework to mitigate attacks that target the network resources. Our framework provides a collaborative mitigation strategy between the ISP and its customers. The implementation relies on Software-Defined Networking (SDN) technology to deploy the mitigation framework. The contribution of our framework can be summarized as follows: first the customers detect the attacks and share the threat information with its ISP to perform the on-demand mitigation. We further develop the system to improve the management aspect of the framework at the ISP side. This system performs the alert extraction, adaptation and device configurations. We develop a policy language to define the high level policy which is translated into OpenFlow rules.

Finally, we show the applicability of the framework through simulation as well as testbed validation. We evaluate different QoS and QoE (quality of user experience) metrics in SDN networks. The application of the framework demonstrates its effectiveness in not only mitigating attacks for the victim, but also reducing the damage caused to traffic of other customers of the ISP.

Résumé

Les attaques cybernétiques causent une perte importante non seulement pour les utilisateurs finaux, mais aussi pour les fournisseurs de services Internet (FAI). Récemment, les clients des FAI ont été la cible numéro un de cyber-attaques telles que les attaques par déni de service distribué (DDoS). Ces attaques sont favorisées par la disponibilité généralisée outils pour lancer les attaques. Il y a donc un besoin crucial de contrer ces attaques par des mécanismes de défense efficaces.

Les chercheurs ont consacré d'énormes efforts à la protection du réseau contre les cyber-attaques. Les méthodes de défense contiennent d'abord un processus de détection, complété par l'atténuation. Le manque d'automatisation dans tout le cycle de détection à l'atténuation augmente les dégâts causés par les cyber-attaques. Cela provoque des configurations manuelles de périphériques l'administrateur pour atténuer les attaques affectent la disponibilité du réseau. Par conséquent, il est nécessaire de compléter la boucle de sécurité avec un mécanisme efficace pour automatiser l'atténuation.

Dans cette thèse, nous proposons un cadre d'atténuation autonome pour atténuer les attaques réseau qui visent les ressources du réseau, comme par les attaques exemple DDoS. Notre cadre fournit une atténuation collaborative entre le FAI et ses clients. Nous utilisons la technologie SDN (Software-Defined Networking) pour déployer le cadre d'atténuation. Le but de notre cadre peut se résumer comme suit : d'abord, les clients détectent les attaques et partagent les informations sur les menaces avec son fournisseur de services Internet pour effectuer l'atténuation à la demande. Nous développons davantage le système pour améliorer l'aspect gestion du cadre au niveau l'ISP. Ce système effectue l'extraction d'alertes, l'adaptation et les configurations d'appareils. Nous développons un langage de politique pour définir la politique de haut niveau qui se traduit par des règles OpenFlow.

Enfin, nous montrons l'applicabilité du cadre par la simulation ainsi que la validation des tests. Nous avons évalué différentes métriques QoS et QoE (qualité de l'expérience utilisateur) dans les réseaux SDN. L'application du cadre démontre son efficacité non seulement en atténuant les attaques pour la victime, mais aussi en réduisant les dommages causés au trafic autres clients du FAI.

Contents

List of Figures

List of Tables

1 Introduction

Cyber attacks are defended by two mechanisms: detection and mitigation. The attack diagnosis relies on the detection mechanism. Mitigation is the response mechanism that tries to reduce the impact of ongoing attacks on the network. Research on mitigation approaches have received less attention compared to detection, due to the complexity involved in the deployment of automated mitigation mechanisms. Moreover, existing mitigation mechanisms require the deployment of special hardware or software devices, which makes it complex and expensive for the deployment.

Furthermore, network or cyber attacks do not only cause problems to enterprise networks or end users, but also for Internet Service Providers (ISPs). Generally, enterprise networks and end users get the Internet services from ISPs, as they subscribe to the ISP to get network connectivity. If the attackers target end users then it also causes collateral damage to ISPs as the traffic traverses through their network before reaching to end users. Therefore, the ISPs have become important players in cyber security and their intervention is key in mitigating the cyber attacks and their impacts. However, it is difficult for the ISP to detect the attacks going towards end users or enterprise networks. Specifically because, ISPs have a large customer base, and if they do the detection on behalf of their customers then it causes huge processing overhead. Therefore, it is important to design a mitigation technique that can coexist between ISPs and end users. So, it is crucial for the end-users to share the adequate information to service provider for mitigating cyber attacks.

Therefore, there is a need for cooperative and large scale defense mechanism among different partners involved [51]. Such cooperative mechanisms have scalability concern at the technical and business level. These cooperative mechanisms should overcome the both technical and business challenges, and should benefit all the partners involved.

The management of such cooperative defense mechanism is a difficult task, as it has various challenges: (1) the timely and automated sharing of alert informations between different managed domains, (2) the deployment of network and security policies depend on the Service Level Agreements (SLAs) with different network domain without any policy violation, (3) the configuration of low-level device specific rules on the heterogeneous network devices should not depend on the

manual work of network administrators, (4) the deployment of such defense mechanism should not rely on the specific hardware or software installation. Therefore, it is necessary to have an automated defense mechanism, which respond to network changes and security alerts without intervention from network administrators.

Automated defense framework should have a closed loop between different components of the defense mechanisms from monitoring, analysis to response. As shown in Fig. 1.1, the framework essentially contains a loop which consists of monitor, analysis, countermeasures and reaction. The network status is managed by controller or a centralized manager which uses the analysis results provided by the *Analysis module*, on the collected network traffic by the *Monitor module*, from the data plane devices. The analysis results assist the controller in selecting the countermeasure, and execute the reaction policies in the data plane devices. The network administrator can specify the high

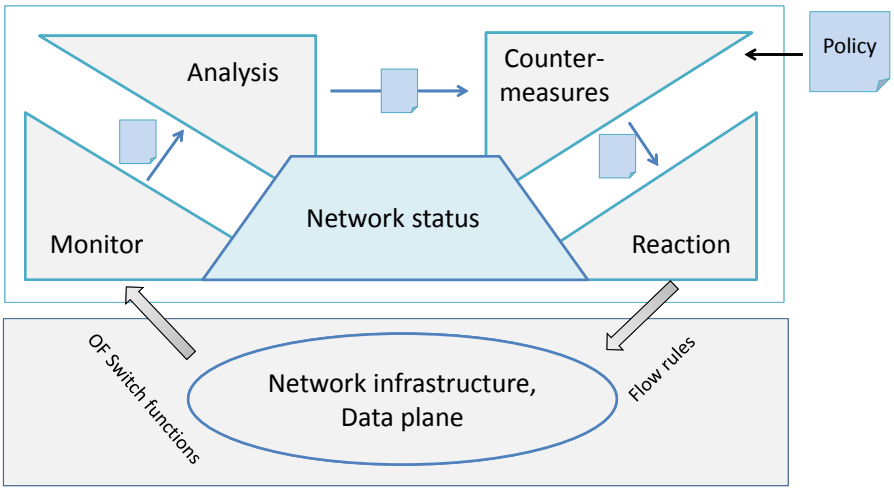

Figure 1.1 – Reference model of autonomic defense framework

1.1 Background and Challenges

Despite the tremendous efforts from both industry and academia, Cyber attacks continue emerging with unprecedented amount and varieties. A clear gap between the detection of security events and incident response still remains. For security professionals, it's desirable to have a system which can automatically detect and defend against cyber attacks in real time, which is, however, mission impossible in practice today. According to the Forbes magazine report [84], three years ago the cost of cyber crime in the U.S. was 100 billion dollars. Many attacks goes undetected because of dependency of security systems on human intervention. IT personnel are no match to the intensive and sustained cyber attacks in the network. According to ABI research [96], in the future enterprises would not act manually to mitigate the cyber attacks after they happen. Specially, Distributed Denial of Service (DDoS) attacks are the most prevalent attack in the

Internet today, which causes huge damage to the enterprise as well as to the ISPs [11]. The existing mechanisms require the presence of network administrators to mitigate the DDoS attacks which impacts the services in the network. Security automation would help the enterprises to mitigate the attacks without any human intervention. Automation provides speed in responding to the high-volume attacks in the network. Therefore, in this thesis we focus on mitigating the DDoS attack with automation of security mechanism.

However, there are some challenges in providing the needed automation for cyber defense. There is clear gap among the different components from monitoring and analysis to response. Automation should provide the control of the network according to different conditions. Different components in an automated defense should communicate effectively with each other for overall defense of the system. It should not increase the deployment cost for the enterprise network in terms special software and hardware. Since, if it requires some special devices or software which requires a lot of modification in the network, then enterprises would be reluctant to deploy these kind of automated mechanisms. Currently, the automated defense mechanisms are hardwired with the hardware devices which makes there deployment and management a complex and daunting task. Furthermore, the automated defense system should be robust and immune to new forms of attacks on the system. So, these challenges still need to be addressed to make the defense system in the network automated and operational.

1.2 Contributions

The objective of this thesis is to study the feasibility and effectiveness of building autonomic cyberdefense mechanism using Software-Defined Networking (SDN) technology. We leverage the unique features of SDN mainly programmability and global visibility over the entire network, and centralized controllability to design autonomic defense mechanism. In another word, thesis is intended to study how and to what extent the autonomic cyberdefense can be achieved. More specifically, the following objectives are identified,

1. Design a collaborative and autonomic defense mechanism which integrates the security functions ranging from anomaly detection in the customer network to the analysis and adoption of countermeasures in the ISP together into a pipeline.

2. A gap between high-level policies and low level rules always remain, motivating us to design such a policy representation and translation mechanism, which can effectively translate high-level security and network policies into the actual actions enforced on networking and security devices.

3. Considering the fact that attack on one customer can potentially affect many customer. So, a dynamic path computation is important for effective traffic engineering. This motivates us to design a policy management and enforcement framework for the ISP network which considers multiple factors such as congestion level in the path for policy enforcement.

The proposed framework relies on managing legitimate, suspicious and malicious flows between the ISP and its customers using SDN technology. SDN is a new networking paradigm in which control plane is decoupled from the data plane. Control plane is a centralized intelligence point and it provides a global visibility of the network. We propose to use SDN for providing security and QoS services to the customers of ISPs. Mitigation and QoS services are achieved by sharing the security alerts and flow information with its ISP by the customers. The information shared by the customer allows the ISP to establish different paths to process the flows.

To achieve the above objectives, we need to develop a framework, as well as designing efficient algorithms, methods, and approaches. The contributions delivered in this dissertation are summarized as follows.

1. Proposing an autonomic defense framework by leveraging SDN paradigm which can systematically integrate key security functions together: anomaly detection, policy specification & countermeasures selection, and response.

2. Designing a SDN based policy representation and translation technique that automatically translate the high-level security and network policy into low-level OpenFlow rules for enforcement in the network and security devices.

3. Providing a dynamic and adaptive policy management and enforcement framework for the ISP network. It handles attack mitigation and QoS requests of different customers and enforce appropriate security policies according to particular demands customers and overall network status of the ISP. A prototype of the framework has also been developed and evaluated via simulation to show the effectiveness of our proposal.

1.3 Outline of Dissertation

This dissertation is organized as follows:

Chapter 2 - State of the Art. This chapter firstly surveys the research efforts on building autonomic cyberdefense mechanisms. Then SDN based security architecture and mechanisms are extensively discussed, including network monitoring, anomaly detection, traffic engineering, attack mitigation, as well as security policy management in SDN. The final part is focused on DDoS attacks, one of the largest Internet threat that has been studied for more than two decades.

Chapter 3 - an SDN based Autonomic Cyberdefense Framework. This chapter presents an autonomic cyberdefense framework which leverages the characteristics of SDN to systematically integrate several essential security functions together, ranging from network monitoring and anomaly detection to the adoption of countermeasures and incident response. A use case about DDoS attack mitigation is developed for testing the feasibility and effectiveness of the proposed framework.

Chapter 4 - SDN-Oriented Policy Representation and Refinement. This chapter presents a technique to represent and translates the network and security policies. It allows the network administrator to define the global network and security policies without knowing how these policies are enforced in the data plane. Upon receiving the security alerts these high-level policies are instantiated and translated into low-level OpenFlow rules for deployment in the OpenFlow switches.

Chapter 5 - a Novel SDN based Policy Management System. This chapter deals with the policy management framework in the ISP network. It enables the ISP to express the high-level security and network policies. These high-level policies in the ISP network are triggered upon receiving the security alert or QoS request from the customers. The framework extracts data from alerts and computes the path and deploy the corresponding low level rules in the OpenFlow switches in an automated way. This policy management and enforcement framework is validated by experimental simulations with an ISP and multiple customers scenario.

Chapter - Conclusion and Future Work This chapter concludes the dissertation with a summary of contributions and presents the perspective for future work.

2 State of The Art

Today, Internet offers a wide variety of services in the cloud such as video streaming, online banking, VoIP services, etc. These services demands protection from malicious users and attacks. This is yet a very difficult task; thus protecting these services in the network is known as cyber defense. This term refers to the mechanisms devised for protecting the network and its resources from cyber attacks (Malware spreading, distributed denial of service attacks, etc.) performed by the attackers. However, protecting the network and services from the cyber attacks is very difficult task because of the non trivial manual work involved by the network administrator (e.g. configuring the devices, deployment of the rules, etc.).

The survey report of AlgoSec [7] as shown in Fig. 2.1 shows, that due to the lack of automation 20 percent of organizations experienced security threats. 42 percent of organizations had network outage because of the network mis-configurations caused by the security related tasks. According the the survey many organizations experienced security breach because of the manual security configurations. Report says that for 19 percent of organizations it takes one full working day or more to resolve the security breach because of the manual work involved by the network administrators.

However, with the increased cyber threats, it is required to automate the defense mechanisms. Attackers use automated tools to target the network resources so automated defense is required to tackle these attacks. Next, we present a survey and state of the art on the autonomic defense mechanisms. Then, we discuss security mechanisms proposed using Software-Defined Networking (SDN) technology to which can enables us to create an architecture for automated defense. Then, we present the current trends in the cyber attacks and some proposals to tackle the cyber attacks with SDN technology and then we conclude.

2.1 Autonomic Cyberdefense

There have been efforts to automate the security mechanisms due to large number of cyber attacks that have been occurred in the past. When cyber attacks have been successful they have

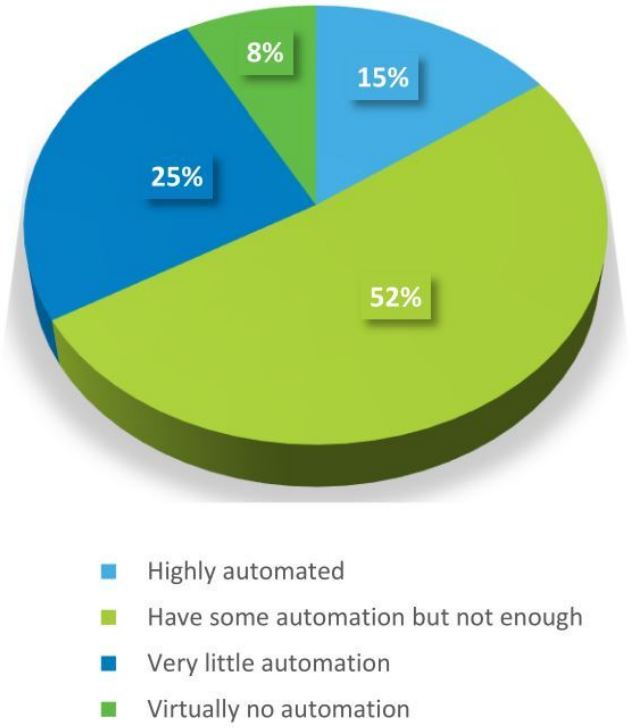

Figure 2.1 – State of Automation in Security.

caused huge financial loss to organizations. Therefore, to make the networks more secure and independent of human intervention, it is necessary to make them automated like Autonomic Computing (AC) system. AC systems are able to dynamically adapt to changes in accordance with the high-level defined policies. AC systems have mechanisms to monitor, analyse, plan and execute themselves based-on the changing environment according to the defined policies.

A rule based automated anomaly detection framework is proposed in [91] to protect the Building Automation and Control (BAC) networks. Intrusion Detection System (IDS) is developed by training the system with the data flows collected by the FireAlarm system testbed using the monitoring module. Rules are collected from the offline data mining techniques to detect the attacks. Cognitive Cyber Defense (CCD) [1] solution proposed by IBM contains a set of cyber security applications which output threat indicators based on analysis of different data. It builds real-time and long-term profiles of different entities (hosts, external users, etc.) by analyzing different data sources which enables to classify and detect anomalous activities. These mechanisms perform offline analysis and then the rules are updated by the network administrators.

ISDS [117] was developed as a framework for automated security based on autonomic computing. The aim of this software is to provide intelligence to its components to dynamically adapt according to the security conditions of the network. The main idea of the framework is to analyse the informations received from the network and adjust dynamically.

AUTONOMIA [29] framework uses the self-configuration approach to control and manage the security systems in the network. It configures the system and modifies the policies dynamically to protect the network. The framework has two modules: Component Management Interface (CMI) and Component Runtime Manager (CRM). The CMI is used to define the high-level policies for each component in the network whether it is software or hardware. The CRM manages the components using the policies defined in the CMI.

The work reported in [94] classifies the traffic as normal, abnormal, and uncertain to prioritize the traffic to provide the quality of service. A tool reported in [116] update the firewall rules based on the information of traffic to a honeynet. It contains a module that analyzes the traffic logs generated by the honeynet and uses data mining mechanisms to generate the new firewall rules for the deployment.

These frameworks require special software or hardware for their deployment. It makes these systems difficult for the deployment. Moreover, these system are limited to offline analysis of data and making a long term profiles for decision making such as in [91], [1], [117], [116] and [94]. They do not focus on the deployment of policies to dynamically adapt the mechanism to provide security. AUTONOMIA [29] framework proposes to modify the policies to protect the network. However, it requires different software and hardware components to be deployed to realize the framework. Thanks to the emergence of Software-Defined Networking (SDN) [88] enables us to revisit the defense mechanism to make them automated. Next, we present about SDN and the security mechanisms proposed using SDN technology.

2.2 Software Defined Networking Architecture

Software-Defined Networking (SDN) is a new networking paradigm which separates the control and data planes as shown in Fig. 2.2 [88]. The separation of control from the data plane enables to easily add new network functions based on the current network requirements. The term SDN was coined at Stanford University to represent the work on OpenFlow [80]. Since then, it has attracted a lot of attention from both academia and industry. Many novel ideas based-on SDN have been proposed [41, 66, 73, 82, 101]. In industry, SDN is considered as a technology which can reduce the operational cost and strengthen their network.

Due to the strong coupling between control and forwarding plane in traditional networks, it becomes difficult to develop and deploy new network applications, since it would require modification in the control plane of all network devices through some hardware upgrades. Therefore, new network features are provided in the network through the introduction of middleboxes such as Intrusion Detection Systems, firewalls, load balancer, etc. These middleboxes need to be placed statically at some key locations in the network which makes it difficult to dynamically reconfigure them at runtime depending on the requirements. The main reason, for researchers to have interests in SDN is mainly because it provides centralized controller where policies can be defined for changing network environment and these policies can be enforced in the data plane

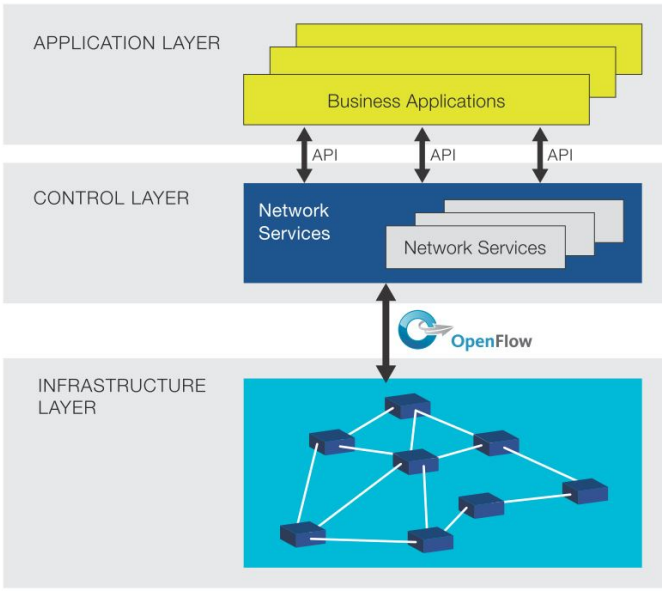

Figure 2.2 – SDN Architecture

through southbound API.

The Software-Defined Networking architecture consists of three distinct layers as follows:

- The application layer consists of the end-user business applications that consume the SDN communications services. The control layer is accessed through the application layer by the northbound API.
- The control layer provides the functionality that supervises the network forwarding behavior through an open interface.
- The infrastructure layer consists of the network elements (NE) and devices that provide packet switching and forwarding.

According to this model, an SDN architecture is characterized by the following key attributes:

- *Logically centralized controller and Network-Wide Visibility:* In SDN, all data plane devices are connected to a centralized controller. The controller can send control messages to data plane devices. The controller can also send queries to data plane devices to get the statistics to infer the network status. With a centralized controller and the knowledge of global network status, decision making is facilitated, as opposed to legacy networks where nodes are unaware of the overall state of the network.
- *Programmability:* SDN networks are controlled by software, which may be provided by the vendors or network operators themselves. Data plane devices can be controlled by

the applications deployed at the controller. It provides the approach to introduce new network functions. Several programming languages have been proposed to enable this feature [46, 114, 115].

- *Abstraction:* The business applications that use SDN services are abstracted from the underlying network technologies. Network devices are also abstracted from SDN control layer to ensure portability. Data plane has very simple logic to forward and drop the flow based on the installed rules. The simplified data plane provides the flexibility to add new features, like DevoFlow [32], NetFPGA [86], [6].

- *Flow-based Management:* Forwarding decisions in the switches are taken per flow. Rules in the switches are installed on per-flow basis. The basic characteristics of SDN is to forward the flow to the controller, when the network devices do not have the rules to handle these flows. This feature allows to deploy the rules as and when it is required. Moreover, it enables the network operator to handle the flow dynamically based on the varying conditions in the network [6].

- *Dynamicity* SDN provides flexibility to accommodate changes for more dynamicity. Devices in the network can be reconfigured easily depending on the varying conditions in the network. It enables to deploy the applications for bandwidth on-demand services in the data centers and service provider network.

SDN Controller

The controller is the core component of the SDN network since all the intelligence is centralized at the controller which generates the network configuration based on the policies defined the network operator. It abstracts the low level implementation details of the data plane configuration. There are various SDN controllers proposed. In Table 2.1, we have clustered some features of different SDN controllers. Features include architecture, northbound API, programming language, and OpenFlow version they support. Architecture of the SDN controllers are either centralized or distributed. A centralized SDN controller is the single entity which manages and configures all the data plane devices in the network based on the policies defined by the administrator. It also becomes a single point of failure. Moreover, a centralized controller may not be a good choice to manage the large networks. Northbound API enables the developers to deploy the application above and abstract the low-level details of the devices. SDN controllers support different types of programming languages to write the applications. Moreover, SDN controllers have also been grouped based on the OpenFlow versions they support.

As we can see in Table 2.1, most of the controllers are centralized and multi-threaded. A centralized SDN controller is the single entity which manages and configures all the data plane devices in the network based on the policies defined by the administrator. Centralized multi-threaded controllers leverage the parallelism of multicore computer architecture. Centralized controllers such as RYU, NOX-MT, Trema, POX have been designed for data centers, and carrier

Table 2.1 – SDN Controllers

SDN Controller	Architecture	Northbound API	Programming Language	OF Version
NOX	Centralized	Ad-hoc API	C++	v1.0
NOX-MT	Centralized multi-threaded	Ad-hoc API	C++	v1.0
OpenDaylight	Distributed	REST, RESTCONF	Java	v(1.0,1.3)
Floodlight	Centralized multi-threaded	REST API	Java	v1.1
POX	Centralized	Ad-hoc API	Python	v1.0
Trema	Centralized multi-threaded	Ad-hoc API	C, Ruby	v1.0
ONOS	Distributed	REST API	Java	v1.0
RYU	Centralized multi-threaded	REST API	Python	v(1.0,1.2,1.3,1.4,1.5)

grade networks. Distributed controllers such as ONOS, OpenDaylight can be scaled to manage the needs of small to large networks. The advantage of distributed controller is that it provides fault-tolerance. When one node fails, then another node can take the duties of the failed node.

Currently, SDN controllers support a wide variety of northbound APIs such as REST APIs and ad-hoc APIs. As shown in Table 2.1, RYU, floodlight, OpenDaylight, and ONOS support the RESTful API for communication through northbound interface. However, NOX, POX, and Trema have their own ad-hoc APIs. Currently, only RYU SDN controller supports all the five OpenFlow versions.

Communication with SDN Controller:

The core modules of the SDN controller includes the Network Operating System (NOS) and the network device drivers. Switches, network applications like firewalls, Network Address Translator (NAT) are external applications to SDN controller. The SDN controller communicates with the external applications in the network with three different APIs.

1. *Southbound API:* This API enables the communication between the controller and OpenFlow switches. OpenFlow is a standard protocol used for communication between the controller and the switches. It is a vendor-independent protocol which provides flexibility to network operators to program the controller and data plane according to their requirements.
2. *Northbound API:* Network applications which want to communicate with the controller and underlying network infrastructure communicate through this API. For instance, firewall, Deep Packet Inspection (DPI) and other network applications deployed at the application

layer of the controller uses this API to communicate with the network devices through the SDN controller. Generally, these applications use a REST API for the communication with the controller.

3. *East-West API:* Generally, this API is used to communicate between distributed SDN controllers. For instance, in a network, if the control is distributed between two controllers, then it may require to pass information to another controller. It can also be the case that two controllers reside in different domains, and they share informations related to their domains with each other through east-west bound API. In other scenarios, different controllers in the network may perform different tasks allocated to them. For example, one controller may be allocated to perform network related tasks such as routing, load balancing. While an other controller is allocated to perform security related tasks.

There are different types of flow messages in SDN which uses the APIs described above for communication in SDN network.

1. *Packet-in messages:* The packet-in messages are used for communication between switches and controller. When a flow arrives at the switches for which it does not have a rule, then this flow is forwarded to the controller to decide the action to be enforced. Depending on the high-level policy controller enforce the rule in the switches to take the action on the flow.

2. *Controller-to-Switch Messages* These messages are initiated by the controller for asking the switch features or sending the rules in the switches for forwarding or dropping flows.

3. *Flow Statistics:* These messages are triggered by the controller to collect the flow statistics from the OpenFlow switches.

The salient features of SDN technology enable us to enhance the security of the network. Next in Section 2.4.1, we present how SDN features can provide benefits to security mechanisms.

2.3 Genesis of Software Defined Networking

Though SDN is a recent concept, it builds on the history of ideas on networking. Specifically, it has been developed on the work of programmable networks such as active networks [108], programmable ATM networks [71] and on the idea of control and data plane separation such as routing control platform [25].

We summarize the historical perspective of pre-SDN related works in Table 2.2 into three different categories. Along with the categories, we specified in the second and third column of the table which mentions past initiatives and more recent SDN deployments.

Table 2.2 – History of programmable networks

Category	Pre-SDN Initiative	SDN Developments
Data plane programmability	Active Networks [108]	P4 [23],OpenFlow [81]
Control and data plane decoupling	Routing control platform [25], NCP [53]	Ethane [28], OpenFlow [81]
Network Operating System	Cisco IOS [22]	ONOS [20], NOX [52]

Active networks [108] is one of the early works on building networks on the concept of data plane programmability. It assumes that the switching devices can download the programs with specific instructions for processing of the packets. The second way proposed is to replace the packets with the tiny programs, which are encapsulated in transmission frames and processed at each node in the path.

P4 [23],OpenFlow [81] represent the recent approach of data plane programmability. OpenFlow [81] is one of the early works on the SDN deployment. It relies on configuring the forwarding devices to support flow tables which can be dynamically modified using centralized controller. P4 [23] is a high level language for programming protocol independent packet processors.

The works on separating the data and control plane dates back to 1980s. NCP [53] is one of the early works done on decoupling the control and data plane. It was developed by AT&T to manage its telephone network. Similarly, routing control platform proposed to improve the management in BGP networks.

Recently, Ethane [28] and OpenFlow [81] proposed the decoupling of the control and data planes for Ethernet networks. These mechanisms do not require modifications on forwarding devices. It makes these solutions very attractive for the networking community.

Network operating system abstract the underlying hardware to the network administrator which makes it easier to manage the network resources as well as the deployment of network applications. The work on Network Operating System was restarted with the introduction of OpenFlow based NOSs such as NOX [52], ONOS [20]. CISCO IOS [22] has been the most widely used NOS since 1990s.

2.4 Improving Network Security with Software Defined Networking (SDN)

2.4.1 SDN Enabled Security Mechanisms

We believe that SDN can significantly improve the security research and it can also be integrated with the existing security mechanisms. Decoupling of control and data plane allow to define the global network and security policy at the control plane which can be enforced in the data plane through programmability provided by the SDN. Moreover, global visibility of SDN makes it possible to effectively handle the traffic in the network. In the recent years, there have been some proposals for providing security using SDN technology. In this section, we discuss the SDN enabled security mechanisms.

Attack Detection mechanism in SDN

In [50], authors proposes an anomaly based detection and mitigation mechanism using OpenFlow and sFlow statistics collection mechanism. The framework uses entropy based anomaly detection technique over SDN and compare it with the OpenFlow and sFlow flow statistics collection mechanisms. Flow statistics collection with the native OF (OpenFlow) approach is performed when the SDN controller request for the flow statistics from the OpenFlow switches. In SDN, the controller is informed of the first packet for each flow. This first packet is matched to the network policy at the controller, and then the subsequent packets are collected from the switch and forwarded to the anomaly detection module running at the controller. The disadvantage of OF collection method is that there is no sampling involved in the collection process, so it collects the overall network traffic passing through the switch.

The framework also proposes a sFlow based statistics collection mechanism. In this method, sampling process is involved which enables to construct the flow definition and collects the flow statistics. This results in a significant reduction in the required number of flow entries. The proposed system compares the number of flow entries required for detection and CPU usage, with entropy based detection and Threshold random walk with credit based connection rate limiting(TRW-CB) algorithms. Results show that the average number of flows required under sFlow collection mechanism with entropy based method is 217 flows. However, with the OF based approach the required number of flows are 5184 flows. The same is the case with TRW-CB algorithm. Average number of flows required are 217 with sFlow mechanism;however, with the OF based approach the number of flows required are above 2000. The CPU usage under sFlow and TRW-CB algorithm is 32 percent. However, under OF based flow collection mechanism the CPU usages for entropy based and TRW-CB algorithm is 47 and 42 percent respectively. Therefore, authors shows that the sFlow traffic collection mechanism is better than the native OF (OpenFlow) traffic collection mechanism.

The Lightweight DDoS detection using OpenFlow [24] proposes to use NOX platform using Self

Organizing Maps (SOM) to detect the attacks. SOM is an unsupervised artificial neural network method. Authors used SOM method to classify the network traffic as normal or abnormal. In this technique, NOX controller periodically interacts with the switches and collects the traffic samples from the switches. It extracts the features of interest such as average of packet per flow, average of bytes per flow etc. from flow entries from all switches, and then the extracted features are forwarded to the classifier module for further processing. The classifier module analyzes whether traffic is DDoS flooding attack or legitimate traffic.

In [82, 83],multiple anomaly detection algorithms are used in the experimentation in order to validate their usability in small environments. Authors propose the idea that the anomaly detection algorithm can run at the border router of the home network. Four anomaly detection algorithms were implemented over the NOX SDN controller. These algorithms are Threshold Random Walk with Credit based Rate Limiting (TRW-CB), Rate Limiting, Maximum Entropy Detector, and NETAD. In the experimentation, they have used low network traffic rates ranging from 60 to $12,000$ packets per second as they have considered the home network. Results indicate that the algorithms perform well in the small enterprise network environment. However, when it was tested at the ISP level algorithms were unable to perform well. It also shows that the detection is far more accurate and efficient in the home network in comparison to the ISP network because of the high volume of traffic in the ISP network.

In NetFuse [118], a proxy device is deployed between the switches and the controller. The proxy monitors the network load, and instructs switches to reroute any overloading flows to NetFuse devices. This technique requires multiple SDN controllers in hierarchy. The proxy device is deployed in between the SDN controller and the data plane. It monitors the network and also communicates with the centralized controller. It uses passively collected data to monitor active flows in the network.

Industrial solutions have also been proposed to detect the attacks. Radware [103] has developed DefenseFlow a solution to detect the Distributed Denial of Service (DDoS) attacks. Radware's anti-DDoS application instructs OpenFlow controller to program OpenFlow switches to behave as OpenFlow counters, which collects the flow statistics. It makes a baseline profile of the network traffic and then monitors for the anomalies related to DDoS attacks. If the attack is detected then, it notifies the controller to reroute the attack traffic to specialized devices deployed in the network to process the attack traffic. Radware has also developed an open source version defense4all of its commercial product DefenseFlow. It has been developed on the OpenDaylight controller.

In [126] a two-stage rate based detection mechanism is used to detect the DDoS attacks. Threshold is defined by the network administrator, and once the traffic exceed the threshold then second stage detection is activated. For instance, authors in their experimentation set 3000 packets for every 5 seconds as a first stage detection. Once the traffic exceed this threshold then second stage threshold of 800 Packets Per Second (PPS) is activated. If the traffic continues at the 800 PPS for 5 times, then DDoS defender application starts dropping the traffic.

An anomaly-based detection and mitigation mechanism based on SDN is introduced in [49]. It uses a bidirectional count sketch algorithm to detect the attacks. Furthermore, it uses packet count on sFlow samples based on destination IP for attack detection. It identifies the destination IP addresses with high asymmetric communication pattern. It uses a threshold value to detect the asymmetry in communication. It removes the IP addresses which do not cross the threshold value. Once the controller identifies the malicious flows, it instructs the switch to drop the malicious flows. Legitimate traffic is still forwarded to victim as compared to Remote Triggered Black Hole (RTBH) in which victim becomes unreachable for benign traffic. The authors in the paper proposed a modular approach for data collection, anomaly detection and victim identification and attack mitigation. The proposed mechanism is deployed at the ingress point of the network so that once the malicious flow is identified, then it can be blocked at the entry point in the network which saves the network resources inside the network from collateral damage.

Table 2.3 shows the DDoS detection application deployed using SDN technology, their methodology, deployment location and the SDN controllers they have used for the application deployment. Detection methodology specifies the algorithm or mechanism used to detect the attack. Deployment location provides the switch or router location from where data is collected to detect the attacks. For instance,mechanisms such as [50], [83], [118], and [49] are deployed at the border routers in the network. Mechanisms [24] and [103] collect information from all the switches in the network to detect the attacks. Moreover, in Table 2.3 columns containing controller tells the SDN controller used to deploy the mechanism. For example, detection mechanisms [50], [83], and [24] are implemented using NOX controller. POX controller is used to deploy Remote Triggered Black Hole [49] mechanism, while Radware's solution [103] is deployed using OpenDaylight controller. NetFuse [118] has not given the controller used to deploy its mechanism.

Traffic Monitoring in SDN Environment

In the legacy network the traffic monitoring tool requires a separate hardware or special software configuration making it tedious and expensive task for deployment and management. Moreover, precise measurement of traffic matrix in the large IP network is difficult due to high number of switches and other networking components. However, traffic matrix estimation problem can be minimized in the SDN environment. By leveraging the global visibility of SDN controller and streamlined flow operations in the SDN switches which allows to query for flow statistics, OpenTM provides a traffic estimator mechanism. OpenTM [109], provides a traffic estimator mechanism for origin-destination (OD) pairs in the network. It keeps account of the active flows in the switches and gets the routing information from the OpenFlow controller's routing application and periodically polls the switches for the packet count counters to get the statistics. By using the informations available to the SDN controller, OpenTM create different types of TM for sources and destinations. In this mechanism the flow statistics is collected from the ingress and egress switches in the network because they are close to the source and destination.

OpenNetMon [112] also provides a mechanism to monitor the network in the SDN environment.

Table 2.3 – DDoS Detection Applications over SDN

Detection Application	Detection Method	Deployment Location	Controller
Combining OpenFlow and sFlow [50]	Compares the OF flow statistics collection with sFlow based collection approach on entropy based method	Egress port of the egress switch	NOX
Lightweight DDoS Detection [24]	Self-organizing map with traffic features is used to detect the attack	All OF switches in the network	NOX
Traffic Anomaly Detection using SDN [83]	Threshold Random Walk,Rate limiting,Maximum entropy detector and NETAD algorithms were tested	Border routers of the home network is the deployment location	NOX
Remote Triggered Black Hole (RTBH) [49]	Uses bidirectional count sketch algorithm to detect the attacks and asymmetric communication pattern to detect the attacks based on threshold	Detect DDoS attacks at the ingress point of the network	POX
NetFuse [118]	Uses multi dimensional flow aggregation to combine the flow for detection and safeguard the network from traffic overloading	Ingress switch	Not given
Radware [103]	Makes a baseline profile of network traffic and then monitors the network for anomalies	Distributed detection probes	OpenDaylight

Authors in the paper pointed out that currently, ISP over-provision network resources to meet the QoS requirements of their customers. It polls edge switches at an adaptive rate which increases when flow rate differ between samples and decreases when flows stabilize. It minimizes the number of queries between switch and controller. It continuously monitors all the flows between pre defined link on throughput, packet loss, and delay. The monitoring of end-to-end QoS parameters helps traffic engineering mechanisms to compute the appropriate paths for routing the traffic.

The OpenSAFE [14] framework suggest to distribute the traffic to monitoring applications. It leverages the fact that every first packet of a flow needs to be forwarded to the controller. Then SDN controller forwards the new flows to the traffic monitoring applications which analyzes it

with IDS. However, OpenSAFE requires expensive hardware to perform the monitoring.

OpenSketch [124] proposed an SDN-based monitoring mechanism. It separates the measurement data plane from the control plane. It provides a three stage pipeline (hashing, filtering, counting) which can be implemented with the switches and they can support different measurement tasks. The control plane of OpenSketch provides the measurement library which automatically configures the pipeline and assigned resources to varied measurement task. However, the new protocol proposed by OpenSketch somehow requires the upgrade in the network devices. Moreover, it also requires the standardization of the new protocol. Therefore, it makes ISP reluctant to deploy this mechanism.

The Passive Flow Monitoring (PaFloMon) [12] aims at providing the slice based monitoring in the OpenFlow networks. in this mechanism network is segmented into different slices. It is motivated by the OFELIA [2], which is the OpenFlow testbed in the Europe. PaFloMon aims to enrich the OFELIA framework with slice aware monitoring. The sFlow monitoring tool is integrated with the OFELIA. The aim of the OFELIA is to safeguard the experiments from the intrusive traffic that active measurements causes during experimentation.

Traffic monitoring applications are grouped in Table 2.4 according to the methodology, deployment location and SDN controller used to deploy these applications. OpenTM [109] collects the flow statistics from the switches closer to destination; OpenNetMon [112] uses edge switches in the network to collect the flow statistics; while OpenSAFE [14] collects the flow statistics from middleboxes deployed in the network path to monitor the traffic. OpenTM [109] and OpenSAFE [14] use the NOX controller while OpenNetMon [112] uses POX controller to deploy the mechanism.

Traffic Engineering (TE) using SDN

In traditional networks, possible ways to perform traffic engineering is to manually modify the rules or pre-deploy the rules in network devices. It is a very intensive and daunting task for the network operators. Furthermore, in traditional network the traffic engineering mainly contains IP-based TE and MPLS-based TE. IP-based TE solves the problem of load balancing in multi path environment by optimizing the IP routing algorithm. But, IP-based routing has some drawbacks. For example, if OSPF link weights are used to control the routing of traffic, then it is difficult to split the traffic randomly. Additionally, if the link fails or topology changes then it takes time to converge. Likewise, MPLS-based TE causes performance overhead in the network. In summary, the tight-coupling between the control and forwarding planes, in addition to distributed network control, makes the traffic management a difficult task in traditional networks. The separation of control and data planes in SDN allows network operators to define a global network policy at the centralized controller which can be dynamically enforced. Moreover, the global visibility of SDN enables network operators to deploy the monitoring tool which can monitor and collect the network status information in the real time.

Table 2.4 – Traffic Monitoring Applications over SDN

Traffic Monitoring Applications	Methodology	Deployment location	Controller
OpenTM [109]	provides the traffic estimation for source and destination pair in a network	Switches closer to destination is used for querying the flow statistics	NOX
OpenNetMon [112]	It provides monitoring to determine whether end-to-end QoS parameters are achieved and forward the input to TE to compute the paths	Edge switches in the network are used to collect the statistics	POX
OpenSAFE [14]	Forwards the first packet of flow to the monitoring application to get the information about the flow	Middleboxes are deployed in the path to monitor the traffic	NOX
OpenSketch [124]	Provides 3-stage pipeline (hashing, filtering, counting) and measurement library to configure resources for varied measurement tasks	No details given	No details given
PaFloMon [12]	Provides a slice based monitoring in the SDN network	No details given	No details given

Shu et.al [102] proposed a framework for traffic engineering in an SDN environment. The framework consists of two parts: (i) traffic measurement and (ii) traffic management. Traffic measurement mainly includes monitoring the network and collecting the network status in real time for managing the traffic. The network status information includes end-to-end network latency, topology connection status, bandwidth utilization, etc. Depending on these informations, the current network status is validated whether it is correct or not. Depending on the traffic measurement information, traffic in the network is scheduled to satisfy the user needs in terms of QoS.

Cao et.al [26] proposed mechanism for planing and online routing of traffic based on SDN. The SDN controller in the mechanism performs the policy lookup, flow steering, and route installation. The policy table at the controller determines the logical sequence of middleboxes the packets need to traverse. The Controller maps the logical sequence of middleboxes to the physical topology. Once the logical to physical path is mapped, then the SDN controller installs the rules in the flow table of the switches to forward the flow. The authors proposed an optimization algorithm to optimize the resources such as the middlebox capacity to handle the projected traffic. They also provided an optimization technique for online traffic steering. The aim of this algorithm is to determine the optimal allocation of resources for the flow upon arrival.

Agarwal et.al [3] proposed a traffic engineering technique in an SDN environment. It believes that the SDN can be helpful in better network resource utilization and it can improve the network performance metrics like packet loss and delay. The objective of this traffic engineering mechanism is to adaptively and dynamically manage the network traffic according to network operators requirement. It uses the Fully Polynomial Time Approximation Scheme (FPTAS) to compute the route at the centralized controller. It considers the partial deployment of SDN forwarding element (OpenFlow switches) to route the traffic in the network.

2.4.2 SDN based Security Architecture and Services

In this section, we discuss the security architecture and framework based on SDN to provide the mitigation, middlebox deployment and middlebox chaining to process the traffic through middleboxes in an SDN environment.

Attack Mitigation

In Drawbridge [73], the customers can subscribe to traffic engineering services provided by ISPs. It is based on the assumption that the customer's controller can communicate with the ISP's controller for the deployment of the mitigation rules. The objective of this proposal is to avoid the unwanted dropping of customers' traffic by their ISP. Since, in case of congestion ISP may drop legitimate traffic traversing to the customer network. In the architecture, authors assumed that the end-hosts perform the detection and send the rules to the ISP controller for mitigation. ISP controller checks the validity of the rules and enforce the rules in its switches or send the rules to the upstream ISP for the enforcement. This framework, provides the flexibility for the customers to be adaptive in dealing with the traffic based on its dynamics. The customer can decide at runtime rules to be established at the controller. However, in this mechanism ISP needs to share its Policy Enforcement Points (PEP) with their customers.

The system presented by SENSS [125] provides an interface to request for attack traffic mitigation. Once the attack is detected by the victim network, then it sends a message to the ISP requesting the concerned traffic details and its route. The victim network can also requests the ISP to filter or modify routes for the traffic coming to its network. SENSS system enables the customer to ask the ISPs during no attack period, about the amount of traffic that it receives from their neighbors and route to the network which is sending heavy traffic towards it. During DDoS attacks, the customer compares the traffic distribution before the attack and during attack and identifies the upstream ISP that is sending a lot of traffic. It assumes that these ISPs may be sending attack traffic, then the customer network issues a filtering message to these ISPs to mitigate the attack traffic. The important features of SENSS are:(1) Victim oriented: The victim has the incentive to do the attack detection and mitigation. SENSS proposes a system to enable the victim to directly request for the security services from remote ISPs. Victim can request the ISP for mitigating the traffic which belongs to the victim's address space; (2) It provides a simple interface for victims to request for the mitigation services from ISPs. Victim can request services like traffic

filtering, rerouting and quality of service guarantees; (3) Through the interface provided by the ISPs victims can request multiple ISPs to traceback the attack and then it can issue the commands for the ISPs to take the action to mitigate the attack. This framework requires multiple ISPs to collaborate with each others.

The Bohatei [41] leverages SDN and network function virtualization (NFV) capability to provide elastic DDoS defense. In this technique, authors leverages the NFV approach to instantiate the defense VM (Virtual Machine) at the required location in the network. Bohatei leverages the SDN paradigm to steer traffic towards instantiated VMs in the network. It provides an ISP centric deployment model, where an ISP offers DDoS-defense-as-a-service to its customers. The ISP can monetize these services. It assumes that the ISP uses some anomaly detection system to detect whether a customer is under DDoS attack. Then attack estimation is done for the suspicious traffic. The Resource manager in Bohatei uses the estimates to determine the type, number and location of VMs needed to be instantiated. The Resource manager tries to optimize the network resources using two algorithms: (1) Data center selection problem; (2) Server selection problem in the data center (SSP). DSP is a greedy algorithm to select the data center for steering of suspicious traffic. The algorithm provides two outputs. Firstly, it provides the what fraction of suspicious traffic should be forwarded to each data center. Secondly, it provides a physical graph corresponding to attack type defense to be deployed by the data center. SSP algorithm greedily tries to assign the servers with the higher capacities to process the suspicious traffic. Then, forwarding rule is set up to forward the traffic to defense VM. Evaluation showed that the Bohatei system is able to handle attacks of hundreds of Gbps. It also enables the response for DDoS traffic in less than 1 minutes. However, this framework requires the ISP to perform the detection on behalf of their customer which can cause processing overhead on the ISP. Since, ISPs have hundreds and thousands of customers to manage.

The DBA [74] proposes SDN-based DDoS blocking mechanism to mitigate botnet based attacks in the network. It uses the standard OpenFlow APIs to deploy the mechanism in the SDN environment. In this mechanism, authors propose to protect a server from botnets. Protected server inside the network establishes a secure communication channel with the DDoS Blocking Application (DBA). When an attack is detected by the server, it notifies the DBA about the attack. Then, the DBA provides a redirected address to server. The DBA maintains a pool of public IP addresses which can be used for redirection. Upon receiving the new IP address, the server notifies the legitimate clients to use the new address to access the service. The new address needs not to be physically replicated, but may be in the same subnet. Redirection address information provided to the clients require some computation cost to protect these services from bots. In their scheme, authors used CAPTCHA [4] to protect the services running at the server from bots. However, in this mechanism, bots are poorly programmed as it may not be the case in the practical scenario. Moreover, it has been assumed that the bots are not using IP spoofing. Additionally, in this technique of redirection, the server needs to cooperate with the DBA for redirection.

A report from the Open Networking Foundation (ONF) [87], discusses the advantages of SDN

in providing security in the data center. It provides a case study where an Automated Malware Quarantine (AMQ) detects and isolates the infected and insecure network devices before it can impact the network. Upon identifying a threat, it dynamically applies the policies to mitigate the threat. Once the threat is mitigated, it allows the network devices to join the network. It takes advantage of centralized intelligence and global network visibility for mitigation. In contrast to this approach, the current approach in the network is static and inflexible in its nature. In the traditional network, AMQ is deployed in the proprietary hardware devices, which requires the expert network administrator for operation.

The FRESCO [101] introduced by Shin et.al, provides a programming framework for rapid design of detection and mitigation modules. Detection and mitigation logic are programmed and linked as modular libraries to provide a complete defense in the network. Upon detection of a threat in a network by detection modules, FRESCO mitigation module generate flow rules to mitigate the threat. The framework consists of application layer (to compose security applications) and a security enforcement kernel which implements the action from the security application. The application layer modules of FRESCO are developed in Python and run over the NOX OpenFlow controller. These modules run as an event driven applications. The Security Enforcement Kernel (SEK) is directly integrated with the NOX OpenFlow controller. FRESCO SEK provides features upon which FRESCO relies for the enforcement of the rules. It prioritizes the rules derived from security applications over non-security applications. The prototype works on NOX version 0.5.0 using OpenFlow version 1.1.0.

Application of framework is illustrated with two examples. In the first example, authors have shown the working of honeynet to detect the malicious scanner. To show this proposal two modules are composed together. Scan detection module first detect an active malicious scanner and then redirect all the malicious flow towards a honeypot for processing. Then, honeypot sends packets back to the scanner, which is unaware that all of its flows are processed by the honeypot. In the second example, authors illustrate that legacy security devices like BotHunter, DPI can be integrated with the FRESCO. These security devices monitor network to identify malicious traffic. Alerts generated from these security applications are forwarded to the mitigation module which generates the OpenFlow rules to mitigate the attacks. Messages from the legacy security applications are forwarded to a module as - MESSAGE_LEGACY either in (i) FRESCO format or in (ii) other standard format such as Intrusion Detection Message Exchange Format (IDMEF) [44]. For instance, if IDMEF format is used then alert is specified as MESSAGE_LEGACY:IDMEF.

Table 2.3 shows the DDoS mitigation applications their mitigation method, mitigation location and the SDN controller used to deploy these applications. Mitigation methods have already been described earlier. So, we describe the mitigation location and the controller used for the deployment of mitigation applications. Drawbridge [73], Bohatei [41] and SENSS [125] are deployed in the ISP network of the customers. However, FRESCO [101], Automated Malware Quarantine (AMQ) [87], and DDoS Blocking Application [74] provides the mitigation at the ingress point in the network without collaborating with upstream networks. FRESCO [101] uses NOX controller; OpenDaylight controller is used by Bohatei [41]; and DDoS Blocking

Application [74] uses POX controller for the deployment purpose.

Table 2.5 – DDoS Mitigation Applications over SDN

Attack Mitigation applications	Mitigation Method	Mitigation Location	Controller
Drawbridge [73]	sharing of mitigation rules by customer with their ISPs	ISP of customer or upstream ISP	No Implementation
SENSS [125]	Provides interface on the SDN controller which enables the victim to request for mitigation and rerouting service from the ISPs	Upstream ISP of victim network	No details given
FRESCO [101]	Provides a framework for detection and mitigation module	OpenFlow switches in the network	NOX
Bohatei [41]	Provides DDoS defense service and instantiate the VMs in the network to steer the suspicious traffic to these VMs	ISP network	OpenDaylight
DDoS Blocking Application [74]	It provides redirected address to server which is used to redirect the legitimate connection upon detection of attacks	It protect the server from botnets by redirecting traffic from ingress switch	POX
Automated Malware Quarantine (AMQ) [87]	Upon malware detection isolates the infected devices in the network	It is deployed at the edge of the network	No detail given

SDN-based Middlebox Management

Middleboxes are devices deployed in the network to process the traffic before forwarding it towards the destination. For example, firewalls, NAT (Network Address Translator), DPI (Deep Packet Inspection) are middleboxes frequently used in the network. Dynamic traffic modification performed by these middleboxes in the network makes it difficult to correlate the flows for access control and forensics. It creates conflict between the policy at the controller and at the data plane. The conflict between the policy at the controller and at the data plane devices makes it difficult to integrate the middleboxes in the SDN network. FlowTags [42] architecture handles this problem. It introduced the Flowtags-capable middleboxes, which can request the SDN controller to generate and provide the tag for processing the flow. The Tag API is provided at the controller. Middleboxes request the controller to provide the tag for the flow for insertion before processing. This tag insertion in the flow by the middleboxes before processing ensures: (i) the

binding between the packet and its origin, (ii) Second, it assures that the packet follows the path determined by the central policy.

Networks rely on middleboxes (e.g., firewalls, NATs, DPI, WAN optimizer, Intrusion Detection System) to provide critical services such as security and load balancing. But, in the traditional network settings, it requires careful design of network topology and intensive manual work from network operators to set up the rules in the middleboxes and in the switches to steer the traffic through a sequence of middleboxes. However, the advent of SDN technology provides a promising alternative for steering traffic through a sequence of middleboxes from a centralized control point. In this regard, SIMPLE [93] introduced a system which allows network administrators to specify a policy to route the traffic through a logical sequence of middleboxes, which is automatically translated into forwarding rules considering the physical topology and middlebox resource constraints. The SIMPLE architecture has three key components at the controller:

1. The ResMgr component takes as input a network traffic matrix, the topology of the network and policy requirements, to output a set of middleboxes for processing the traffic.
2. The DynHandler module maps the incoming and outgoing traffic from the middleboxes. It uses a payload similarity algorithm to correlate the packets after modification from the middleboxes.
3. The RuleGen module takes the output from the ResMgr and DynHandler modules and generates the data plane configuration to steer the traffic through a sequence of middleboxes.

Aaron et.al. [48] discussed in their paper the fact that SDN can be useful for managing the different middleboxes in the network. They pointed out some major challenges in managing the middleboxes. Firstly, interpreting the middlebox state that what state exist. For example, in case of Firewall it can be connection established. Secondly, manipulating the middlebox state. For example, it can be specified that a packets of the flow must traverse the same IPS for the duration of the flow. because it is required for the proper attack detection. Moreover, they discussed how informed state control decisions can be taken by a separate control logic at the SDN controller. In the proposal authors advocated for the SDN-based middlebox networking framework for flexible and unified control on the deployed middleboxes in the network.

Slick [10] proposed a programmable middlebox architecture to overcome the challenges like placement and scaling. In the paper, the authors pointed out that, in the existing network, administrator need to plan in advance the deployment of the middleboxes at some choke points in the topology. This static deployment makes it difficult for the network operators to dynamically reconfigure the middleboxes based on demands. Moreover, it makes the network inefficient as the network operator needs to over provision middlebox resources. To overcome this problem, the authors proposed a *Slick* middlebox architecture. The main aim of the Slick architecture is to provide a means by which network operator can easily implement and and efficiently deploy policies in the network. The *Slick* architecture achieve this objectives by three main

components:(i) a protocol that handles a coordination between controller and middleboxes, (ii) Single policy is split into multiple executables that can run on multiple middleboxes, and (iii) an optimization algorithm run at the *Slick* controller which dynamically places the middlebox functions and route the traffic.

Service Chaining

Service chaining is performed to steer the traffic through a set of middleboxes deployed in the network. It enables the network operator to specify the processing of the traffic through a chain of middleboxes before reaching to the destination. However, due to static configuration of middleboxes in the legacy network it is difficult to steer the traffic dynamically for processing through these middleboxes.

The StEERING [127] pointed out that managing middleboxes (also known as inline services) such as Network Address Translation (NAT), firewalls in data center or enterprise network is still a daunting task. Because network operators still need to rely on the low-level complex and manual configurations to manage these devices. Therefore, network operators either forward all the traffic through middleboxes or set-up extra tunnels in the network to by-pass it. Therefore, StEERING is a framework proposed using SDN technology for inline service chaining, and it allows to steer traffic at the granularity of service provider network and traffic types using simple policies defined at the SDN controller. StEERING framework consists of the data plane and control plane module. There are two types of switches in the data plane modules. The perimeter or border OF switches reside at the entry point of the service delivery network. These switches work as *classifiers* and classify the incoming traffic and forward it towards the next service in the service chain. The *core* switches in the network process the traffic based on the L2 layer. The core switches need not to be necessarily OF switches. The control plane of StEERING consists of two modules. The first module is the SDN controller, which is responsible to manage the switches in the network and install the flow rules in the OF switches. The second module of control plane is an optimization algorithm which periodically determines the best locations for the services to be deployed. However, StEERING does not address the identification and handling of packets from service instances where packet header informations are modified.

J.Blendin et.al [21] proposed service chaining architecture based on SDN. They pointed out limitations in the existing service chain approach. Authors argue that the existing pre-configured and static service chains are useful but they rely on fixed hardware and software configuration. Additionally, changing existing service chains require intensive manual work from expert network administrator and makes it time consuming for on-demand services. They point out that dynamic service chaining is achievable with the SDN technology, which can improve the situation in two ways: First, service chains can easily be created from existing service functions. Second, service functions can be dynamically created during runtime. The proposed service chaining architecture based-on SDN consists of three layers: Forwarding layer, control layer, and application layer. The forwarding layer contains the actual OpenFlow devices. The control layer contains the Service

Function Chaining Router (SFCR). The Service Function Chaining Controller (SFCC) resides at the application layer. SFCC is responsible for controlling all the components, service instance allocation and offering high-level API. When a new flow arrives in the network it is provided a default service instance by the SFCC. SFCC instructs the SFCR to install the network flow rules. SFCC provides high-level API which enables network operators to define, modify and remove the service chains dynamically. If a service chain is modified, e.g., by adding or removing a service function, then for each corresponding service chain an appropriate service instance is inserted or removed. SFCR is notified by the SFCC to update the service chain routing by updating the OpenFlow flow rules at the ingress and egress router of the service chain.

Eder et.al [98] proposed a policy based architecture based on SDN technology for dynamic service chaining. It enables network operators to define policies that govern the chaining of VNFs. Based-on the set of available VNFs in the infrastructure, the proposed framework creates a graph that represents the Service Chaining (SC). This architecture uses a controlled natural language to write the policies, so that network operators do not need to be familiar with the low-level programming language. The aim of this architecture is to ease the tasks of network operators when specifying the service chain from low-level details.

Table 2.6 describes service chaining applications their main purpose and the controllers that have been used for their deployment. StEERING [127] aims at chaining the middleboxes in the network and steer the traffic to these middleboxes for processing. It uses NOX controller for the implemented purpose. The [21] supports dynamic service chaining through APIs at the controller to add, remove or modify the service chain. The Policy based Dynamic Service Chaining [98] uses service chaining graphs in the policy-based management system to provide service chaining in the network. [21] and [98] do not provide the details of SDN controller used for the implementation purpose.

2.4.3 Security Policy Management in SDN

The centralized control plane in the SDN makes it possible to define the high-level abstract policies at the controller which can be deployed in the switches dynamically through a southbound API. It abstracts the low-level implementation details from the administrator. In this section, we discuss policy based management system using SDN to protect the network.

The Hierarchical Flow Table (HFT) [45] framework uses hierarchical policies to define the context in which network resources can be used and shared among multiple entities. In HFT, policies are defined as a trees, and each node in the tree can determine the action to perform on the packets. Policy tree consists of policy nodes and which contains policy atoms. Policy atom comprises of match and action pair. In the case of conflict policy tree resolve conflicts by using a user defined conflict resolution operator. Hierarchical policies provides flexibility to designate the management of network resources to different entities. For instance, network operator may assign the task of mitigating the attack traffic to security expert in the network. The main focus

Table 2.6 – Service Chaining

Service Chaining	Main Purpose	Short description	Controller
StEERING [127]	aims at providing in-line service chaining	allows to chain the middleboxes in the network and steer the traffic at the granularity of the service provider and traffic types based on the policies defined at the controller	NOX
SDN Service Chaining [21]	Provides a service chaining architecture based-on SDN	Offers dynamic service chain and provides the API to add, remove and modify the service chain	No details given
Policy based Dynamic Service Chaining [98]	aims at providing dynamic service chaining	Uses service chaining graphs over policy based management system to achieve the dynamic service chaining in the network	No details given

of this work is to resolve the conflicts among the policies rather than translating the high-level security and network policies into low-level OpenFlow rules.

The Policy refinement toolkit [77] enables network operators to define Service Level Agreements (SLAs) without delving into configuration details of low-level network devices. The toolkit automatically translates the high-level policies into a set of low-level rules to be deployed in the network devices. The policy refinement is performed in two stages: (i) In the first phase, also known as bottom-up, SDN controller collects the network informations (e.g., delay, jitter) from the forwarding-plane and store them in the policy repository. In the second phase, which is top-down approach, framework translates the high-level SLAs into Service Level Objectives (SLOs). Depending on the informations collected in the bottom-up phase, the framework provides the best possible configuration for the high-level SLA. The main objective of this framework is to translate the high-level goals specified in the SLA into low-level rules.

In [47] authors proposed an SDN-based policy deployment framework. Its a three layer framework which provides a mechanism for policy deployment. Three main components of the framework are:(i) Policy engine; (ii) Job scheduler; and (iii) Device manager. Policy engine in the framework takes the high-level policies and network topology as inputs and transforms the policies into specific low-level security device configuration. Job scheduler maintains the real-time configuration to be deployed in the future. Furthermore, the device manager module gets the configurations of the security devices from the job scheduler module and deploys the configuration in these devices.

The EnforSDN [18] provides a mechanism to integrate the network appliances such as firewall, Intrusion Prevention System (IPS), IDSs into SDN networks. It decouples the policy resolution layer from policy enforcement layer and centralizes the policy resolution layer. High-level policies are specified at the policy resolution layer. To decouple policy resolution instances from policy enforcement instances, resolution instances are connected to the SDN controller, through an application called $EnforSDN$ manager. $EnforSDN$ manager is deployed at the SDN controller. The SDN controller configures the data plane devices such that the flow is routed through appropriate instances of network appliances in the network. The aim of EnforSDN is to centralize the policy resolution layer and steer the flow correctly through appropriate set of network appliances such as firewall, IDS, etc.

Tripathy et.al [110] proposed a policy management system based on SDN to ensure that policies are enforced by a certified server after detecting and resolving conflicts among heterogeneous policies. Their policy framework provides three network functions:Trust_Verify, Policy_Conflict_Resolve, and Policy_Consistency_Check. Trust_verify function identifies the compromised applications and applies the appropriate measures for control. Policy_Conflict_Resolve function analyzes the possible conflicts among the policies and resolves them for policy enforcement. Policy_Consistency_Check function checks whether the existing flow table entries in the switches are in accordance with the high-level policies. These three functions in the framework guarantee security, correctness and adaptability for on-demand changes in the policy rules. The focus of this work is to resolve the conflicts among policies before deploying them into network devices rather than specifying the high-level policies and translating them into low-level device specific rules.

In the PolicyCop [16] authors argue that the traditional Policy-based QoS management mechanisms have number of issues. For example, most of the mechanisms are based on DiffServ or MPLS-DiffServ which have problems. They provide static traffic classes for different QoS levels. Moreover, they require specialized software or hardware devices in the network for policy refinement and enforcement. PolicyCop leverages the network programmability and separate control and data plane in the SDN to achieve dynamic configuration in their autonomic QoS policy enforcement framework. It provides an interface for specifying QoS policies and leverages the northbound API of SDN controller to enforce those policies. Advantages provided by the PolicyCop over the traditional autonomic QoS framework are listed below:

1. It provides per flow control and dynamic flow aggregation in the network in contrast to aggregation based on Type of Service (TOS) field in legacy network.
2. It offers layered architecture and the APIs between the layers are defined using JSON and REST API which facilitates to use different programming languages in different layers.
3. It provides dynamically configurable traffic classes.
4. Traditional QoS mechanisms require different protocols to run for performing the task like routing, MPLS label exchange etc. However, PolicyCop avoids this problem by using

OpenFlow protocol for communication between network services and network devices.

PolicyCop mainly focuses on providing QoS management to traffic in the network. It does not provide any policy language to express the high-level network or security policies.

Table 2.7 provides the short description of SDN-based policy management frameworks and the policy language they support. From Table 2.7 it is clear that only [45], [77] and [47] provides a grammar to define the high-level policies.

Table 2.7 – SDN-based Policy Management

Policy Management	Short description	Policy Language
Hierarchical Flow Table (HFT) [45]	provides a framework to realize the hierarchical policies in the SDN	Policies are defined as a tree in the framework
Policy refinement toolkit [77]	facilitates to define the SLA without knowing the implementation details	Controlled natural language is used to define the high-level policies
EnforSDN [18]	Decouples policy resolution layer from policy enforcement layer	No policy specification language is given
Computer Network defense Policy [47]	provides policy based network management to achieve security in the network	Provides Computer Network Defense Policy (CNDP) policy specification
Policy Management Framework [110]	Provides security, correctness and adaptability for on-demand changes in the policies and ensures that policies are enforced by a certified server and conflicts are resolved before enforcement	No policy specification language is given
PolicyCop [16]	Provides policy-based QoS management using SDN	No policy specification language is given

2.5 Attack Models and Taxonomy

Nowadays, cyber attacks are increasing in the Internet in scale and also in terms of severity. The digital world provides a good ground for cyber attacks. The most prevalent attack in the Internet is still the Distributed Denial of Service attack (DDoS) [11]. The aim of DDoS attack is to prevent the legitimate use of a service in the target network. In DDoS attack multiple attacking

entities target the same end host or network. It is widely used to congest the service provider or enterprise network from providing the services such as video streaming, online gaming, etc. to their customers. Nowadays, the rate of the DDoS attacks have increased. Specially the attacks against the service providers and their customers have increased [11]. Major categories of DDoS attacks are:

1. **Volumetric Attacks:** The aim of the volumetric attack is to congest the network so that legitimate users can not use the services provided by the service providers. The main objective is to consume the bandwidth in the network so that legitimate users can not get fair share of the bandwidth in the network.

2. **TCP State-Exhaustion Attacks:** The purpose of the TCP state-exhaustion attack is to consume the connections state tables in the security devices such as firewall, loadbalancer etc. It can also consume the resources of the server so that response from the server becomes slow.

3. **Application Layer Attacks:** These attacks exploit the vulnerabilities in the applications at layer 7. These attacks are very sophisticated as they can be launched from a single machine to take down the server. Moreover, it is difficult to detect these kinds of attack, as they can be very slow.

As shown in Fig. 2.3 [11], according to the leading organization Arbor, in detecting and mitigating the DDoS attacks, volumetric DDoS attacks have increased in the Internet as compared to the TCP state exhaustion and application layer attacks.

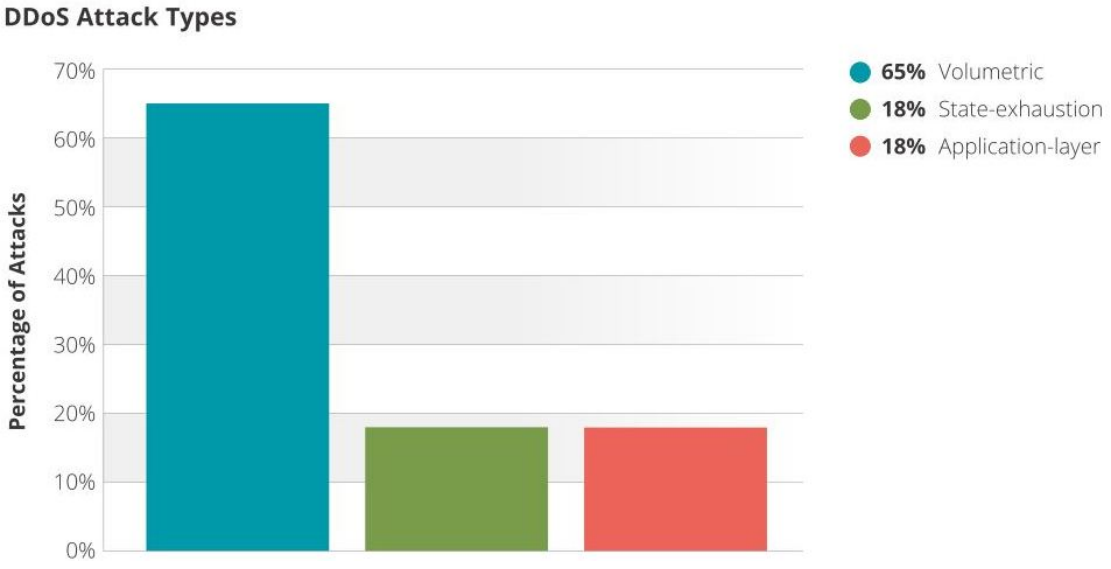

Figure 2.3 – Current trend in the DDoS Attack

2.6 Distributed Denial of Services (DDoS) Attacks and Mechanisms

To date, a large variety of DDoS or DoS mitigation mechanisms have been proposed, covering the entire security life-cycle from prevention and detection to characterization and response. However, few of the existing mechanisms have been considered for widespread deployment due to undesirable overhead and complexity resulting from the life-cycle management. This Section aims at conducting an in-depth comparative study on several representative state-of-the-art schemes, and highlighting the key observations. The selected schemes cover five categories: capability-based, congestion control, traceback, cooperative detection and reaction, and traffic engineering.

2.6.1 Capability-based techniques

In general, capability-based techniques require the communication parties to establish a privileged channel [75, 121, 122]. This means the sender of a message must obtain the permission, e.g., a capability token, from the receiver, in order to send the message. Then, the sender inserts the capability token in the subsequent packets to be sent to the receiver. In doing so, the traffic is divided into two classes: the *privileged* traffic made of packets bearing the token and the *unprivileged* one without token. In the presence of a DDoS attack, the privileged traffic is given priority over the unprivileged traffic.

In the SIFF architecture [121] for instance, the sender is required to complete a three-way handshake to get the capability token that will allow the transmission of the traffic. Part of the capability token is distributed among routers in the network. When the sender inserts the acquired capability token in the traffic, the token is verified by the routers along the path to the receiver. Clearly, the traversed routers need to maintain the state of the flow and part of the capability token, leading to processing overhead and operational complexity. Moreover, SIFF is vulnerable to two types of attacks, one on the capability setup channel and other with the already acquired capability token. Additionally, the scheme may incur latency in launching communications, since the sender is always required to get the capability token from the receiver before sending any message.

2.6.2 Congestion control mechanism

The basic idea of congestion control can be exemplified by the Pushback mechanism [59]: when congestion increases in the network, a router, close to the traffic bottleneck, sends messages to the routers upstream, asking for filtering the traffic based on a congestion signature, ensuring that legitimate traffic can flow through the network. To implement this mechanism, however, Pushback devices need to be deployed at each router in the network.

In [57], a DDoS mitigation framework called STRIDE is proposed, which aims at providing end-to-end data delivery even in the presence of DDoS attacks. In particular, STRIDE is based

on the trust framework in which root authority is responsible to establish the path between hosts in different autonomous domains(ADs). In STRIDE, state is maintained, per flow, at the access routers, while intermediate routers do not need to maintain states. However, extra latency is still incurred due to establishing an up-path to the Path server, as well as a down path from the Path server to the destination.

2.6.3 Traceback techniques

The purpose of IP traceback mechanism is to filter the attack packets as close to the attack source as possible [8, 13, 17, 97], by reconstructing the actual path from the victim to the attacker, even in the presence of spoofed IP packets. To achieve this goal, two techniques can be used: packet sampling and packet marking. An example of the former technique is *Source Path Isolation Engine* (SPIE) [104], which presents an audit trail method that uses a cryptographic hash function to generate lightweight packet digests stored at network routers on the path, allowing to trace the origin of the attack source. As for packet marking, Yaar et al. proposed the *Path identification* (Pi) [120] technique that identifies the path taken by attack packets thanks to a mark insert by the traversed router. Specifically, a path identifier is imprinted by a router in the 16-bit IP identification field of each packet, enabling the receiver to classify the packets. Once the attack traffic is identified as coming from a specific path, then all the subsequent packets coming from that path are dropped. Due to the fact that different paths may end up with the same Path ID the false positive rate is very high. Also, routers need to keep in memory the marks which are to be inserted in the packets.

As described previously, in Section 2.2.3 SENSS allows to traceback the attack traffic in the SDN-enables network. However, it also require that multiple ISPs collaborate with each other to effectively trace the source of the attack traffic.

2.6.4 Cooperative detection and reaction

Intuitively, cooperative detection and reaction architectures rely on the collaboration between different communities or administrative domains, which usually have *a prior* trust relationship. In these domains, security information or detection reports are periodically exchanged for achieving broader detection coverage and more reliable reaction. Based on this principle, a Cooperative Detection and Reaction Architecture was presented in [62]. Every participating domain is assigned a Cooperative Counter-DDoS Entity, which is a modular software platform that automatically exchange security information in the form of alerts and heartbeat messages, using the XML-based Intrusion Detection Message Exchange Format (IDMEF) [35]. Also, the communication within the framework requires a multicast router which listens to the notification from other domains, and maintains the multicast tree containing the information of the subscribers. Clearly, the effectiveness of this architecture depends on the extent of the participation from the different domains. Further, the multicast router is potentially vulnerable to DDoS attacks which

may halt the exchange of the security information between the partners.

A similar architecture called CITRA was reported in [99], which aims at automating intrusion analysis and response tasks that are usually performed by expert network administrators. CITRA also relies on the concept of communities (administrative domains), with an objective to integrate independent security components, e.g., IDS, firewalls, routers. These devices send attack reports to their CITRA neighbors to trace the attack path and initiate the response. In particular, a *boundary controller* in the administrative domain, a role usually played by an access router, uses the attack reports to track whether the attack packets have crossed through its network. Responses are then taken at each CITRA enabled device. It is obvious that the effectiveness of CITRA depends on the collaboration of different domains, which rely on the boundary controller to disseminate and exchange security information.

Mitigation mechanisms proposed using SDN such as Bohatei [41] and Drawbridge [73] provides a collaborative mitigation. However, in Bohatei ISP is required to perform the detection for their customer which naturally cause the processing overhead for the ISP as it may have thousands of customers. Moreover, in Drawbridge it requires the ISP to share its policy enforcement point with their customers which can cause policy violation in the ISP network.

2.6.5 Traffic engineering

The general idea of this approach is to enforce a differentiated processing to different classes of flows *legitimate*, *suspicious* or *anomalous* classes are usually considered, in order to optimize attack response. For example, anomalous traffic can be directed to special purpose devices or paths. To achieve this goal, the networking infrastructure must be modified in certain ways. For example, CenterTrack [107] is a proposed overlay-based architecture, in which special tracking routers are attached to the edge routers, and IP tunnels are pre-established from edge routers to the special tracking routers. The identified malicious traffic is redirected from the edge routers to the special tracking routers. Clearly, as an overlay network needs to be established in advance, operational complexity and additional administrative tasks will be introduced.

Another example is dFence [78], which proposes to redirect suspicious traffic to middleboxes deployed in the core of the ISP network at the time the customer experiences congestion. One potential issue is that the suspicious traffic is broadcasted to different middleboxes due to which it may reach a middlebox which is not designed for its processing, thereby causing the congestion in the core of the ISP network.

2.7 Concluding Remarks

In this chapter, we discussed about different categories of applications which can be helpful in designing an automated defense framework to mitigate the attacks. First, we presented the current state of autonomic cyberdefense and what makes these mechanisms complex for deployment

purpose. Then, we introduced about a new networking paradigm SDN and its features which can be helpful for the deployment of network and security applications. After that, we discussed about the existing attack detection and traffic monitoring applications developed using SDN. The discussion about these applications enables us to take assumptions in our framework that attack traffic can be detected by the applications deployed in SDN. Moreover, discussion on traffic monitoring applications allows to assume that the network status can be monitored continuously to route the traffic through different paths in the network.

We also presented works on SDN-based security architecture performing mitigation in the network. These mechanisms perform mitigation in a single network domain. After that, we introduced works on policy-based management system in SDN to enforce the policies in the forwarding plane. These policy-based management systems focus on resolving conflicts among the policies or steering the flows in the network. In the last part of this chapter we presented the current trends in the cyber attacks. We found that the most prevalent attack in the Internet is the DDoS attack. Then, we provided a discussion on the different categories of DDoS attack mitigation mechanisms.

These initial works on traffic monitoring, attack detection, mitigation of attack and policy-based management systems in the SDN motivate us to design a collaborative mitigation framework to mitigate the attacks in an automated way. In this regard, we also propose a policy language to define the high-level network and security policies in the network which are automatically translated into low-level OpenFlow rules for deployment in their PEPs. Then, we design a policy-based management system for the ISP network which allows them to provide mitigation and QoS services to its customers.

In the next, chapter 3, we propose an autonomic mitigation framework to mitigate the DDoS attacks in an automated way. we evaluate the framework in the testbed as well as with simulation while mitigating the DDoS attacks.

3 An SDN based Autonomic DDoS Defense Framework

We leverage the characteristics of SDN in order to design a collaborative attack mitigation framework to mitigate the attacks. Our framework enables the ISP and its customer to collaborate for mitigating the attack traffic and its impact on legitimate traffic. ISP provides the mitigation service to the customers who are subscribed for this service. In the framework, customer performs the detection in its network and share the security alerts with the ISP for mitigation. Depending on the security alert ISP provides different types of processing to the flow either redirect the flow through low bandwidth path or to the middleboxes, or block the flows. ISP provides a security API to its customers for sharing the security alerts.

The rest of the chapter is structured as follows. An architectural description of the framework, *ArOMA*, together with the implementation details of key components are presented in Section 3.2. A use case that illustrates the applicability of *ArOMA* (given in Section 3.3). A report on the development and implementation of an *ArOMA* prototype using a RYU SDN controller over hardware and software OpenFlow switches provided in Section 3.4, describing the set of experiments conducted to test the performance of *ArOMA*. The evaluation shows that ArOMA is capable of defending against different types of DDoS flooding attacks up to $250,000$ packets per second.

3.1 Limitations of Existing Solutions and Motivations

A large variety of DDoS mitigation mechanisms have been proposed, which cover detection as well as mitigation phases. However, none of the existing mechanisms have been considered for widespread deployment due to the complexity involved in the network management tasks.

In Table 3.1, we highlight the characteristics of the existing mitigation framework and discuss the key observations that motivate us to develop *ArOMA*, the autonomic defense framework. The selected schemes cover five categories described in chapter 2 : capability-based, congestion control, traceback, cooperative detection and reaction, and traffic engineering.

Table 3.1 – Comparison between existing DDoS mitigation schemes

Mitigation Scheme	Detection Point	Threat Information Exchange	Collaboration	Mitigation Scope	Dedicated Software and Hardware	Modification of Original Routing Device
Capability-based	End host	Not addressed	No	Single Site	No	Yes
Congestion-based	End host	Addressed	Yes	Single Site	Yes	No
Traceback	End host	Addressed	Yes	Distributed	No	Yes
Cooperative Detection and Reaction	End host	Addressed	Yes	Distributed	Yes	Yes
Traffic Engineering	End host	Not addressed	No	Distributed	Yes	No
ArOMA	End host	Addressed	Yes	Distributed	No	No

Survey of the existing mechanisms is summarized in Table 3.1 based on the DDoS mitigation techniques discussed in chapter 2, which provides the following observations:

- Threat information exchange is not necessarily applied, depending on the design principle and architecture of the mechanism. Usually, the threat information exchange relies on either dedicated software and hardware add-ons (e.g., congestion-based mechanisms), or the modification of original routing devices (e.g., traceback techniques). Sometimes both (e.g., cooperative detection and reaction mechanisms). All these lead to additional operational complexity, as well as computing and communication overheads as most of them need to be operational all the time. Therefore, effective information exchange is required between the different parties involved (i.e., ISPs and their customers), so that a customer can send its threat information to the ISP at a very early stage of the DDoS attack;

- One of the desirable features is that DDoS mitigation should be distributed, to make the countermeasures more effective and reduce collateral damage. While some of the existing ones have this feature, they require special purpose software or hardware devices. Therefore, there is a need for a mechanism which supports easy deployment and avoids the use of special-purpose software or hardware devices. Moreover, the ISP must timely handle the requests from its customers and enforce appropriate DDoS mitigation policies;

- Most of the mechanisms rely on the manual configuration and real-time intervention of a network administrator, mainly because the mitigation policies need to be statically enforced, introducing latency to the reaction process. Therefore, there is a compelling need to significantly automate the whole life-cycle management of DDoS mitigation schemes.

3.1.1 Threat Model and Objectives

In this section, we first describe the threat model that is addressed by our framework and the main objective of our framework.

3.1.2 Threat Models

We are concerned with DDoS attacks against the customer of an ISP. Attackers may choose multiple locations in the ISP network with multi vector attacks to flood different customer networks. However, here we focus on a single customer network, which can be extended to multiple customers scenario. Also, we restrict our focus to DDoS flooding attacks. In particular, two common DDoS flooding attacks which cause collateral damage to the ISP and its other customers are explained below.

1. *Target flooding attacks*: an adversary sends a huge amount of traffic to the customer network to rattle up the customer's communication. Generally, an adversary could use the UDP flood, TCP SYN flood and ICMP flood traffic to disrupt the services provided by the customer network.
2. *Bandwidth flooding attacks*: the adversary's objective is to flood the link between the ISP and customer networks, ultimately disrupting the legitimate traffic which share the same link. These types of attacks causes collateral damage to other customers of the ISP, as it floods the links of the ISP. UDP flood and ICMP flood are widely used for launching such attacks.

With the key observations in mind, we designed an autonomic DDoS mitigation framework that can overcome the identified limitations and achieve the desired properties. More specifically, our mechanism, *ArOMA*, bears the following properties:

- Broad protection coverage: protecting both end-host networks (e.g., end-users, CDNs) and communication links between ISPs and customers.
- Easy deployment: the modification of routing devices or the installation of special-purpose software or hardware are unnecessary. The threat information between the ISP and its customers is exchanged via a well-defined security API.
- Dynamic and optimized mitigation: ensuring QoS of victim customers in the presence of DDoS attacks, while avoiding damage to the traffic of other customers.
- Automated policy enforcement: achieving dynamic policy deployment through event-based reaction to security alerts and network state changes, alleviating the burden of human operators.

3.2 Proposed Autonomic Cyberdefense Framework

Our analysis shows that most of the existing techniques perform well in a single network environment, but they are often limited by the lack of automated cooperation among different domains for effective mitigation. By leveraging the characteristics of SDN technology, we

propose a framework which satisfies the requirements for effective DDoS mitigation. Our framework is built on the following assumptions:

1. DDoS mitigation is provided to the customers who are subscribed to the ISP before hand, and a prior trust relationship is assumed between the ISP and its customers for the mitigation of attack traffic;

2. DDoS detection engine runs in the customer network and generates security alerts for mitigation;

3. Customer and ISP controllers are authenticated to each other in a reliable and secure way, and they are immune from attacks. We don not focus on securing the SDN controller rather than our aim is to provide security using SDN controller.

In particular, to develop this framework, we need to tackle the following challenges,

1. **Scalable flow management:** The reactive nature of the SDN makes it difficult for an ISP to provide DDoS mitigation as a service to their customers. Every time an OpenFlow switch receives a flow for which it does not have the entry in its flow table it forwards that flow to the SDN controller to get the policy for forwarding that flow. Since it causes processing overhead on the SDN controller, it becomes difficult for the ISP to provide DDoS mitigation as a service to its customer.

2. **Consistency of forwarding policy:** NAT devices, which are widely used in networks, involve packet header information modification. Such alteration may have an impact on how flow packets are forwarded, violating the ISP network's forwarding policy for these flows.

3. **Controller-to-controller communication:** Communication between the ISP controller and the customer controller is required for timely and effective DDoS mitigation. We assume that a subscription based business model can be applied here, i.e., the ISP controller only accepts the messages from customers which have subscribed to the mitigation services.

4. **Dynamic deployment of policies:** Based on the request from the customer controller, the ISP controller needs to provide dynamic deployment of mitigation policies.

In the remainder of this section, we present our proposed framework: *ArOMA*. First, an overview of the architecture is presented, along with the operational workflow of the framework, then we specify the key components of our design.

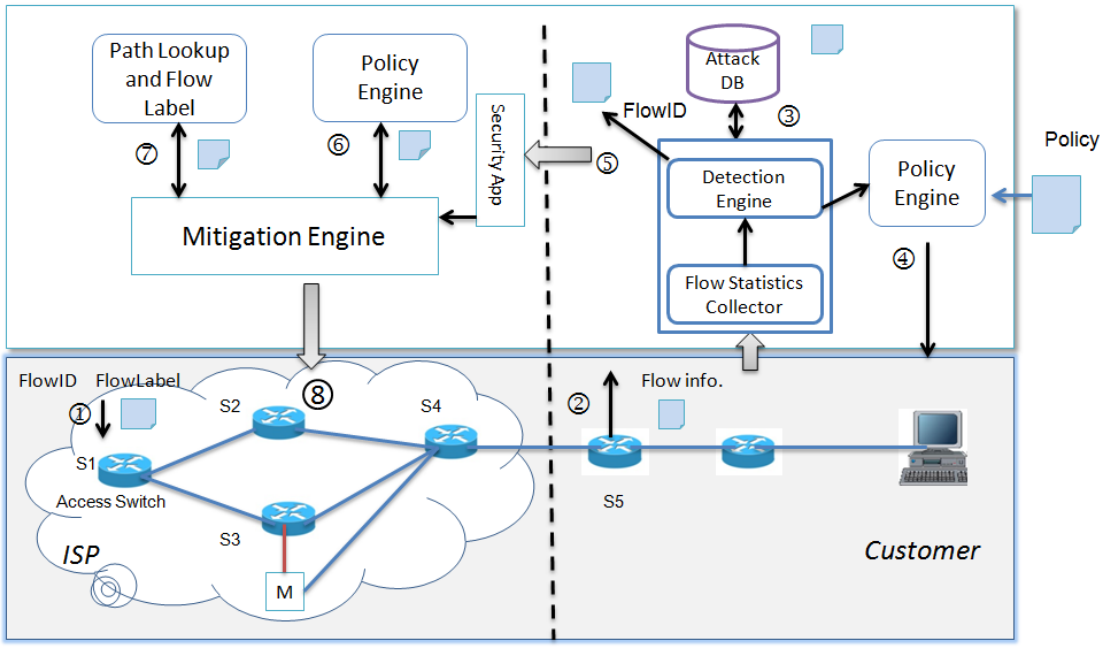

Figure 3.1 – SDN-enabled DDoS mitigation framework

3.2.1 Design Components

The essential design components of *ArOMA*, as shown in Figure 3.1, are described below. We take a bottom-up approach by first covering the data plane components, and then the control plane components.

Data Plane Components

While the data plane is mostly comprised of packet-forwarding devices, mainly the OpenFlow switches, packet-processing functions supported by the middleboxes are also deployed in the network.

OpenFlow Switches. According to the OpenFlow specifications [89], OpenFlow-enabled switches maintain flow tables to perform packet lookups and forwarding. Basically, flow entries consist of match fields, counters, and actions to be applied to the matching flows. Upon reception of a packet, OpenFlow switches perform a lookup operation in the flow table: if it does not have a flow entry for that packet, then it is considered the first packet of the flow, and its header information is forwarded to the controller using a `PACKET-IN` message.

Middlebox (*M*). The term, as used here, generally refers to the devices that host the security functions able to mitigate the DDoS attacks. These devices are deployed in the network for processing and filtering the suspicious and malicious traffic in the network. As shown in

Figure 3.1, a middlebox M is deployed in the ISP network, along with switch $S3$. In our framework, we assume that M enforces pre-defined security policies for mitigating DDoS attacks, such as UDP flood, TCP SYN flood, ICMP flood. For instance, middlebox monitors the source IP address of the suspicious flow which sends UDP packets. If the source sends a huge amount of packets (above a given threshold) then it is categorized as malicious and dropped, otherwise it is treated as legitimate. Apparently, more sophisticated policies dealing with amplification and redirection attacks can be specified as well.

Control Plane Components

Components in the control plane span across the customer and the ISP networks to complete the autonomic operational loop, in which the customer provides traffic monitoring and detection, while the ISP provides mitigation countermeasures. For easier representation, we use C_i to denote the components belonging to the customer, and S_i to represent the ones deployed in ISP network.

C_1 **– Monitoring Component** is composed of two modules to perform the task of collecting incoming traffic and detecting attacks in the customer network.

- *Flow statistics Collector:* flow statistics are periodically requested by the customer controller from the OpenFlow switches and forwarded to the *detection engine* for further analysis and processing. Some flow statistics collection techniques have already been experimented with SDN technology in [24, 82]. In the proposed framework, the OpenFlow collector (OF collector) has been used for collecting the statistics.

- *Detection Engine:* taking flow statistics as inputs, it generates security alerts in the presence of anomalous traffic, based on common anomaly-detection algorithms or using signatures from the *attack database*. The alerts then trigger the local *policy engine* for incoming flows to be processed accordingly, as a first layer of defense. Proposing a new detection algorithm is outside the scope of this paper. However, for the implementation purpose, we assume that the customer network uses a detection technique to flag whether it is under attack or not based on traffic rate threshold [41, 78].

C_2 **– Attack Database**. We assume such a database, which contains previously-seen anomalous traffic and their associated signatures [79]. It is maintained by the customer network and possibly updated on a subscription basis at security vendors and other trusted third parties. The detection engine then compares the characteristics of anomalous flows with the flow features in the database to detect attacks. Especially, we envision that such a database can interact with other threat intelligence mechanisms (e.g., NECOMAtter [58]) using a common interface such as n6 [61].

C_3 **– Policy Engine**. The policy engine enables the ISP and the customer to select and deploy security policies in their respective network. The policy engine at the customer controller generates the local rules to tackle the identified anomalous flows based on the alerts issued by

the *detection engine*, and possibly characterized thanks to the *attack database*. Security alerts are formatted using IDMEF [44] (a description of the format is given in Sect. 3.3) and contain the source and destination IP address, as well as the flow identifier `FlowID`, a `Security_Class` which provides classification information on the detected flow, and the `Impact_Severity` of the detected flow on the customer network. `Impact_Severity` is categorized into three categories: `Low`, `Medium` and `High`. Depending on the information mentioned in the alert received from the customer controller, the *policy engine* at the ISP side enforces the policy to mitigate the attacks.

As for the `Security_Class`, we distinguish legitimate traffic from suspicious and malicious classes of traffic. The motivation is to minimize the impact of countermeasures applied on the flows in the ISP network by considering an intermediate class of traffic, *suspicious flows*, on which we cannot decide. Suspicious flows are not blocked in the ISP network, but may be redirected through paths with a low quality of service. In the ISP network, different paths are provisioned from the ingress point to the the egress points (end-to-end) for different suspicious traffic based on its impact severity. Thus, we are able to reduce the collateral damages on the legitimate traffic, instead of blindly dropping the target flows. The mechanism to classify the flows is out of the scope of this thesis, and we assume that the customer uses some detection mechanism to classify the flows as suspicious and malicious [24]. Our focus is about enforcing the policy to mitigate the attacks once it is detected.

Listing 3.1 – A Sample Policy File: Redirection of Suspicious Traffic

```
1  <Policy PolicyID="Security_policy">
2      <Event attack="ICMP-Flood">
3      </Event>
4      <Condition>
5          <flow class="suspicious"/>
6          <Impact severity="low"/>
7      </Condition>
8      <Actions action="Low_Bandwidth_Path"/>
9      </Actions>
10 </Policy>
```

Moreover, policies are defined as an event, condition and action paradigm [33]. Attack type is used to define the event in our policy file. Flow class and impact severity are used to specify the condition of the policy. High-level action such as drop and low_bandwidth_path are defined in the action part of the policy as shown in the Listing 3.1 and Listing 3.2. For example, if the alert specifies the security class as suspicious, impact severity as low, and attack type as ICMP-Flood then the policy engine uses these informations to check the policy file to enforces the action on the flow. As shown in the Listing 3.1, the policy specifies to provide the low bandwidth path to the flow with low impact severity and suspicious in nature. Similarly, if the security class is malicious then the high level action drop is specified as shown in the Listing 3.2, according to the policy at the ISP controller.

Table 3.2 – Security API overview

Request	Arguments	Response
GET url/redirect	id=FlowID,Security Class, Impact Severity, attack type	status = 200 OK
GET url/block	id=FlowID,Security Class	status = 200 OK

Listing 3.2 – Mitigation of Malicious Traffic Policy

```
<Policy PolicyID="Security_policy">
    <Event attack="ICMP-Flood">
    </Event>
    <Condition>
        <flow class="malicious"/>
    </Condition>
    <Actions action="Drop"/>
    </Actions>
</Policy>
```

S_1 **– Security API**. We envision that the ISP provides DDoS mitigation as an on-demand service via a security API provided by its SDN controller. Through this security API, the customer can request the ISP to deploy middleboxes to filter suspicious and malicious flows, and to assign a higher priority to legitimate flows. In particular, this Security API is implemented at the ISP controller as a REST API, which receives mitigation requests from the customer controller. The detection engine at the customer controller sends the security alert in the standard IDMEF message format. Upon receiving the alert, it extracts the alert information and forwards the information to the mitigation engine at the ISP controller.

Such security service interface is being currently discussed at the IETF under the working group I2NSF [40] which aims at standardizing these security functions, prompting them to be available for widespread use in coming years. Moreover, the security API enables the customers and ISPs to share the threat informations with different ISPs and customers which enables the *ArOMA* to be extended for multiple customer and multiple ISP scenario. Table 3.2 shows the HTTP requests the security API handles with the specified parameters, e.g., FlowID, Security Class. In particular, the security API listens to the request from the customer. It then extracts the parameters from an incoming request and forwards them to the policy engine, which finally takes the appropriate action.

S_2 **– Flow Label Module**. To tackle the two challenges identified at the beginning of this Section, i.e., the *scalable flow management* and the *consistency of the forwarding policy*, *ArOMA* uses labels to identify the path from the ingress switch to the egress switch (end-to-end path) of the ISP network. Labels are used for fast switching and rerouting, as core switches in the ISP network simply need to check the label and forward the packets. Additionally, by using labels, the path policy is preserved through the network, as the modification of the packet header, e.g., by a NAT

device, does not cause a PACKET-IN [89] event at subsequent switches, since only the label is checked for processing. Another advantage of using labels is to reduce the flow table entries in the core switches of the ISP network [123], resolving the concern of scalability.

Algorithm 1 $FlowLabel(packet) \rightarrow labeled\_packet$

```
for each incoming packet P do
    if Packet is in the flow table then
        forward(P, out_port)
    else
        Check the destination IP and destination port
        P.VLANID ← Push(Legitimate_label)
        forward(P, out_port)
    end if
end for
```

Algorithm 1 illustrates the flow label assignment process. Specifically, the label is inserted in the 12 bits VLAN ID field using the *Push* method of OpenFlow [89]. At the ingress switch of the ISP, flow rules are pre-installed based on the destination IP address of the customer network. When a flow arrives in the ISP network, it is checked whether the flow is present in the flow table. If the flow table contains the flow then it is forwarded through an output port. Otherwise, flow informations are parsed and its destination IP and port informations are checked and then a label is assigned to the flow, and it is forwarded through an output port of the ingress switch. As such, the switch-controller interactions to assign labels to flows can be significantly reduced, alleviating overhead on the controller as well.

The reason of using the VLAN ID field to assign labels is that it can provide $2^{12} = 4096$ different labels in its single domain, which are sufficient for a single domain of ISPs, which usually has a small number of links connected with their customers through its single domain, e.g., 500 links in Ebone [106], [119].

S_3 **– Path lookup**. This module provides the ability to the ISP to maintain a list of end-to-end paths at its controller. Paths in the ISP network spanning from the ingress switch to the egress switch are computed using an all-pairs shortest path algorithm [36]. In *ArOMA*, a pool of paths is maintained by the ISP according to different bandwidths, link loss probability and delay to accommodate different types of flows which are categorized into legitimate and suspicious ones. In doing so, the ISP can provide differentiated traffic engineering to its customers, as for example, suspicious flows would be transported to paths providing low QoS with a higher number of hops. On the contrary, legitimate flows would be flown along paths providing high QoS with a small number of hops. Specifically, it receives the input in the form of middleboxes to traverse or type of path (i.e., low bandwidth path) with impact severity(i.e., suspicious or legitimate) from the mitigation engine and returns any matching path and its associated label. It allows the ISP controller to redirect the flow through a policy-based path rather than broadcasting the traffic towards security devices [78].

S_4 **– Flow Identifier**. Aggregating the flows at the ingress switch of the ISP network, protects

these switches from limited flow table entries. *ArOMA* uses `FlowID` to identify and aggregate the flows at the ingress switch of the ISP network and share them between the ISP and the customer controllers (see Alg. 2 for `FlowID` generation). Specifically, a `FlowID` is computed using the IP-4 tuple (source IP, destination IP, source port, destination port) as input to the SHA1 algorithm. Flows from the same source to the same destination are aggregated under the same identifier. `FlowID` is inserted into the 64-bit cookie field set by the controller and is maintained at the ingress switch. When the flow reaches the customer network then its controller requests the ISP controller for the corresponding `FlowID` by sending the source and destination IP addresses of the flow.

Additionally, it protects the ingress switches from state exhaustion attacks due to the limited TCAM entries, however still we need to maintain the flow states at the ingress switch. As the size of flow tables is increasing to meet the requirements of SDN deployments, we are now able to use software switches (e.g., OVS switches) (widely used in the datacenters as well) to avoid the forwarding state limitation of hardware switches [65]. Moreover, switching devices with high performances chips (e.g, EZchip NP-4) can provide optimized TCAM memory that supports from 125,000 up to 1,000,000 flow table entries [65].

Algorithm 2 `FlowID`$(flow) \rightarrow flow\_identifier$

for each incoming flow f **do**
  $f \leftarrow Flow(f.IP\_src, f.Port\_src, f.IP\_dst, f.Port\_dst)$
  if $f.IP\_dst \in Ingress\_switch.FlowTable$ **then**
    $f.cookie \leftarrow$ `FlowID`
    $forward(f, out\_port)$
  else
    $controller(f)$ // Flow is forwarded to controller for `FlowID`
    new_`FlowID` $\leftarrow hash(f)$ // new `FlowID` is assigned
    $f.Cookie \leftarrow Push(new$_`FlowID`$)$ // the new `FlowID` is inserted in the Cookie field
  end if
end for

S_5 **– Mitigation Engine**. This module communicates with other modules deployed at the application layer of the ISP controller in order to enforce the final mitigation countermeasures. Upon receiving information extracted from the alert message (i.e.,`FlowID`, `Security_Class`, `Impact_Severity`, source IP, destination IP) from the Security API, it forwards these informations to the *policy engine* at the ISP side, which provides high-level policies (e.g., Redirect, Drop) that are enforced. When it receives a high-level action from the policy engine, the mitigation engine also checks the *Path lookup* module for getting the appropriate path and the associated label to be assigned to the flow.

For instance, if the security class and impact severity specified for the flow is suspicious and high respectively, then the high-level action is provided as to redirect the flow through lowest bandwidth path. Then, the mitigation engine checks the *Path lookup* module to get the label associated with the lowest bandwidth path and modifies the label of the corresponding flow at the ingress switch to be redirected. Similarly, if the flow is malicious and high level policy is *drop*,

mitigation engine modifies the action for the corresponding flow to be blocked at the ingress switch of the ISP.

3.2.2 Operational Workflow

In the framework, both the ISP and the customers have their own SDN controller running in their respective network and communicating in a secure manner. As shown in Figure 3.1, the framework is distributed across the ISP and customer networks, and the operational workflow (labeled with step numbers) can be described as follows:

1. As the traffic enters the ISP network, the controller assigns a unique flow identifier, `FlowID`, for each new flow, typically by leveraging the `PACKET-IN` message. `FlowID` is used by the customer controller to request for mitigation from the ISP. Labels are also assigned to flows at the ISP's ingress switches for fast forwarding. The label helps preserving the policy applied to the flow to ensure unaltered routing.

2. On the customer side, flow statistics are periodically collected via OpenFlow agents, and forwarded to the detection engine for processing.

3. The detection engine relies on an attack database, which can interact with external databases via certain threat information exchange methods like the n6 API [61].

4. The detection engine generates security alerts in the presence of malicious and suspicious traffic flows, and triggers reaction from a local policy engine, which is in charge of generating and installing policy rules at the ingress switch of the customer network for traffic processing.

5. The `FlowID`s of suspicious and malicious flows are encapsulated into security requests sent to the mitigation engine residing on top of the SDN controller at the ISP side. To that end, a security API is exposed by the ISP's SDN controller. This security API is the main component of the controller-to-controller communication.

6. The mitigation engine requests the policy engine for mitigation actions to handle the detected suspicious or malicious flows.

7. The mitigation engine further checks with the *path lookup* module to infer the best path to route the detected flows according to the mitigation actions.

8. Mitigation actions, e.g, traffic redirection to security middleboxes, are finally taken by updating the labels of the flows of interest. This mitigation process involves automated and dynamic deployment of policies within the ISP network.

3.2.3 Controller to Controller Communication

Another important feature of *ArOMA* is the communication between the controller of the ISP network and the one of the customer network. Controller-to-controller communication enables the customer to share the alert information with the ISP in an automated way. This communication channel actually serves two purposes: sharing the `FlowID` from the ISP to the customer, and alerting the ISP about suspicious and malicious traffics incoming the customer network. Specifically, the way for sharing `FlowID` is described as follows:

1. Initially, when the flow reaches the customer network, the customer controller requests for the `FlowID` of the corresponding flow by sending the IP addresses (source IP, destination IP) of the flow to the Security API at the ISP controller.
2. The `FlowID` is maintained in the cookie field at the ingress switch of the ISP.
3. On receiving the request from the customer controller, the ISP controller checks its ingress switch for the `FlowID` and sends it back to the customer controller.

For example, the controller communication for DDoS mitigation mainly contains the following operations:

1. Upon detecting DDoS traffic at the customer side, the detection engine sends the alert for the detected flows;
2. The security alert is sent using IDMEF (different formats can be used as well).
3. The Security interface (Security API) deployed at the ISP controller is invoked via a REST API by sending the security alert message through the detection engine at the customer controller.
4. The Security API extracts informations such as, the security class, flow informations (source IP, destination IP), impact severity and attack type from the security alert.
5. An acknowledgement message containing the flow information and action enforced is returned to the customer controller after action is enforced on the flow.

3.3 Use Case

For better illustrating and understanding the design purpose and principle of *ArOMA*, we develop a use case, as shown in Figure 3.2, in which one customer and one ISP collaborate with each other to mitigate DDoS attacks. A use case including multiple customers and multiple ISPs will be studied as our future work. In addition, for a single ISP network, it is possible to deploy multiple SDN controllers, each of which interact with one customer network through the security API.

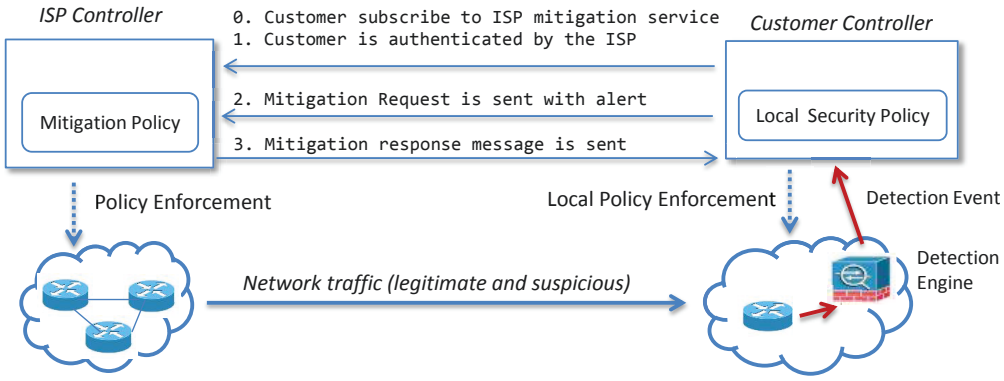

Figure 3.2 – Use case illustrating the application of *ArOMA*.

First of all, the customers that intend to run *ArOMA* need to subscribe to their ISP for this security service, and obtain authentication credentials (Step 0 and Step 1 in Figure 3.2). Meanwhile, an anomaly detection system needs to be set up and configured in the customer network, in order to monitor and detect the unwanted traffic according to its local security policies. It is worth noting that neither dedicated hardware nor special software is required for deploying *ArOMA* in the customer network, except for the fact that the network should be SDN-enabled. This yields significant flexibility for the customers, as they are allowed to run different, even multiple, detection engines and develop specific attack characterization by taking into account their local context, policies and requirements.

For Step 2, we assume a threat information exchange format is well defined between the ISP and customer controllers, so that the alert information can be issued to the ISP as soon as an attack of concern is detected in the customer network. To enable this process, the ISP deploys a security function via a RESTful API at its SDN controller to handle the request received from the customers. Upon receipt of the alert, the security interface verifies the validity of the alert at first, and then extracts the information from the alert and forwards it to the policy engine. Then, the policy engine checks whether a policy matches the received alert and enforces the corresponding rules in its SDN switches. For example, the IDMEF format can be employed for our purpose thanks to its flexibility and compatibility with other formats. An example alert is shown in Listing 3.3, indicating suspicious traffic with a low impact severity. Specifically, the *Classification* and *Assessment* fields specify the type and impact of the attack respectively. The latter indicates the *impact severity* of the detected traffic with a qualitative value among {low, medium, high}. Apart from the IP addresses of the attack source and victim network, two additional fields, the *security class* and `FlowID` are specified in the alert as well.

It is worth noting that the policy engine needs to take into account the status of the ISP network, ensuring the legitimate flows are always assigned routing paths with high QoS, and the collateral

damage on other customer networks can be reduced. A path lookup algorithm can be run here for this purpose. As a result, the ISP notifies the customer about the mitigation process, indicated as Step 3. To complete this process, it is even possible to have an extra step in acknowledging the status of the customer network after mitigation.

Listing 3.3 – Threat Information Exchange Format Between ISP and Customer Network

```
<IDMEF-Message version="1.0">
<Alert>
    <Analyzer analyzerid="CUSTOMER_C"/>
    <Source>
       <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.2</address>
            </Address>
        </Node>
    </Source>
    <Target>
        <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.3</address>
            </Address>
        </Node>
    </Target>
    <Classification attack="UDP-Flood Attack">
    </Classification>
    <Assessment>
        <Impact severity="low"/>
    </Assessment>
    <AdditionalData> type ="string" meaning="security class">
    <string>suspicious</string>
    </AdditionalData>
    <AdditionalData> type ="string" meaning="FlowID">
    <string>196764794928656</string>
    </AdditionalData>
</Alert>
</IDMEF-Message>
```

3.4 Simulation based Experiments

The purpose of our experiments is to validate the effectiveness of our proposed autonomic DDoS mitigation framework *ArOMA*. First, we validate the prototype using both mininet based simulations and later in testbed based experiments. Our experiments are based on the threat model describe in Sub Section 3.1.2. Next, we present the detailed settings and configurations of our experiments, including the implementation details of the prototype.

3.4.1 Network Topology

The prototype is implemented in Python and runs as an OpenFlow application on RYU controller. To perform the simulation, we employ mininet, which provides rapid prototyping environment for the emulation of OpenFlow switches [69]. In particular, we emulate two OpenFlow networks as shown in Figure 3.3, and the traffic is generated by using iperf. We use UDP flood as the attack traffic in this simulation.

We consider a domain of ISP and its customer as shown in Figure 3.3. We use this topology as ISP networks do not have as many switches/routers as enterprise or data center networks [123]. The ISP network has 14 OpenFlow switches, from S_1 to S_{14}, connected with the controller. In particular, ingress switch S_1 is connected to the external network, and the ISP controller applies the policies on the flows coming from the external network at S_1. Also, switch S_5 of the ISP is connected to the customer network, switch S_{15} is connected to the customer controller. Two external hosts are configured as legitimate and malicious hosts to send legitimate and malicious flows respectively. Moreover, we assume that those communication links in the ISP network provide different capacities,

- Link dedicated to legitimate traffic (LT) provides 1 Gbps capacity;
- The link reserved for suspicious traffic with low impact severity ST_l provides the link capacity of 300 Mbps with 4 hops;
- The link reserved for suspicious traffic with medium impact severity is ST_m and the high impact severity is ST_h. These are configured with the link capacities of 200 Mbps and 150 Mbps with 5 hops respectively. Suspicious Traffic of high impact severity causes more damage to the legitimate traffic as compared to suspicious traffic of medium impact severity. So, these traffic are provided the lowest bandwidth path to reduce its impact on other traffic.

3.4.2 Simulation Scenario

Our simulation scenario involves all the components described in Section 3.2.1, as shown in Figure 3.3. In particular, two external hosts, one legitimate (denoted as `L`) and one malicious (`A`) communicate with a host (`C`) in the customer network. SDN controllers in the ISP and the customer networks communicate with each other via REST APIs.

ISP maintains a policy table as shown in Table 3.3, and the ISP controller installs the labels beforehand in the flow table of the ingress switch S_1. Then the traffic between the two external hosts and host (`C`) are provided with labels and `FlowID` when they enter the ISP network. Upon receiving new flows, i.e., flows for which the controller does not have any forwarding policies, the ISP controller considers them as *legitimate* and assigns label '1' or '2' (determined by the load balancing scheme). As a result, all the intermediate switches simply checks the label in the

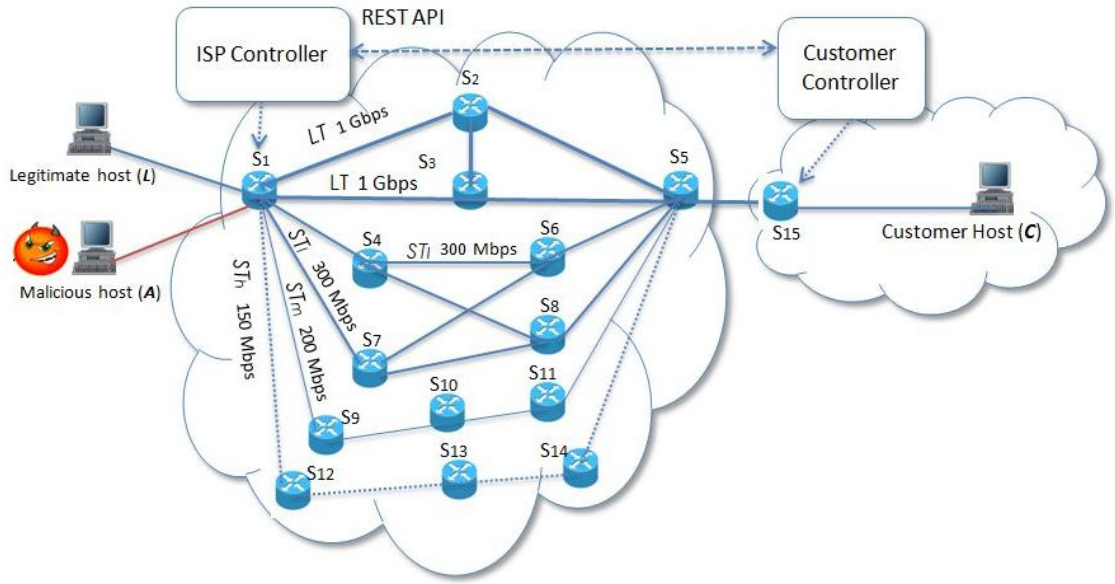

Figure 3.3 – Network topology used for the simulations

Table 3.3 – Mapping between security class, impact severity, policy, bandwidth and associated label

Security class	Impact Severity	Policy	Bandwidth	Path	Label
Legitimate	Empty	Forward	1 Gbps	(S_1, S_2, S_5)	VLAN_ID = 1
Legitimate	Empty	Forward	1 Gbps	(S_1, S_3, S_5)	VLAN_ID = 2
Suspicious	Low	Redirect	300 Mbps	(S_1, S_4, S_6, S_5)	VLAN_ID = 3
Suspicious	Low	Redirect	300 Mbps	(S_1, S_7, S_8, S_5)	VLAN_ID = 4
Suspicious	Medium	Redirect	200 Mbps	$(S_1, S_9, S_{10}, S_{11}, S_5)$	VLAN_ID = 5
Suspicious	High	Redirect	150 Mbps	$(S_1, S_{12}, S_{13}, S_{14}, S_5)$	VLAN_ID = 6

VLAN ID field and forwards the flows through legitimate path. Finally, the egress switch S_5 pops the label from the flows before forwarding them to the customer network.

When the ISP controller receives an alert from the customer controller, it modifies the label accordingly. Referring to the example alert shown in Listing 3.3, as the mitigation response, the ISP controller will enforce the *redirect* policy by inserting label '3' to all the packets of the flow originating from host `A`. Meanwhile, legitimate flows with label '1' originating from host `L` are still forwarded through the path with high QoS.

3.4.3 Simulation Results

We conducted simulations to evaluate the end-to-end effectiveness, throughput and network jitter of the legitimate traffic. We also evaluated the throughput of suspicious traffic when they were processed according to their impact severity. We measured these metrics to know how the DDoS

flows impact the legitimate traffic and how the suspicious flows are impacted when they are redirected through path of low bandwidth. Formally, the following two metrics are carefully studied,

- Throughput and network jitter of legitimate traffic in case the presence of attacks;
- Throughput of suspicious traffic under different impact severity;

End-to-end effectiveness

We evaluated the effectiveness of our prototype *ArOMA* in how quickly it can restore the throughput of legitimate traffic in the presence of given DDoS attacks. The results are shown in Figure 3.4. The X-axis in Figure 3.4 represents the time which varies, Y-axis shows throughput of the flows which changes continuously. Here, we try to measure how quickly throughput of legitimate traffic returns to normal level when the ISP controller receives the security alert from the customer controller for the mitigation of DDoS traffic. Specifically, we launched the UDP attack traffic at the 10th second. Then upon receiving the mitigation request from the customer controller, the ISP controller started the mitigation at around the 35th second. As clearly indicated in the figure, it took less than 10 seconds for the legitimate traffic to recover its throughput after the mitigation started. Thus, we can conclude that *ArOMA* generally provides rapid mitigation response in restoring the performance of the legitimate traffic.

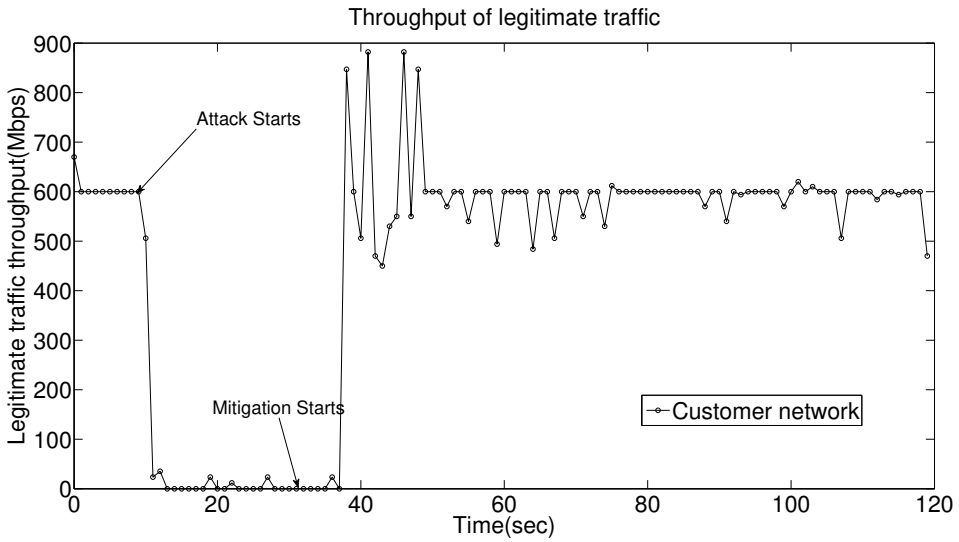

Figure 3.4 – Response of ArOMA and throughput of legitimate traffic

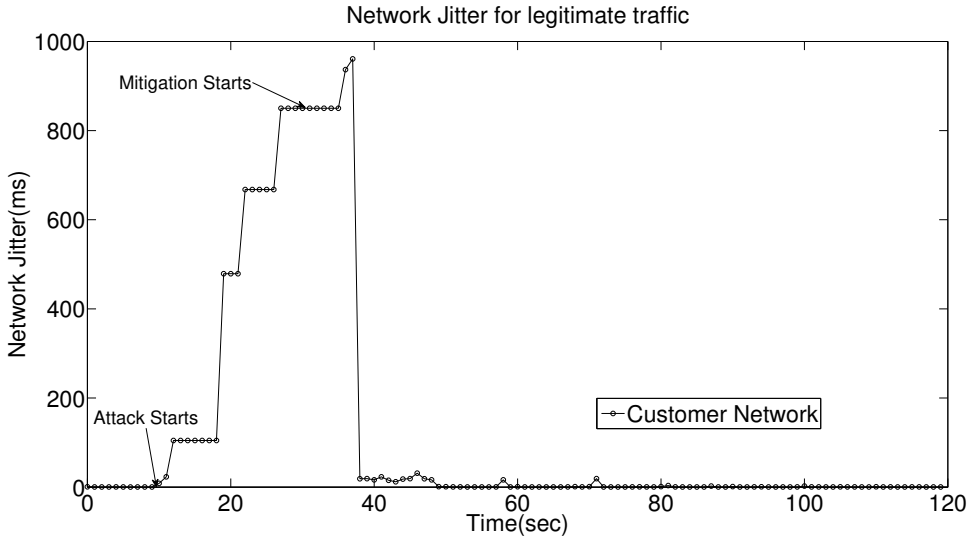

Figure 3.5 – Network Jitter of legitimate traffic going towards customer network

Network Jitter

We measure the network jitter of the legitimate traffic for determining the variation in arrival time between packets, and further understanding the impact of DDoS attack on the quality of service of the legitimate traffic. X-axis in Figure 3.5 shows the time in seconds which is used to note the time when the attack and mitigation starts respectively. Y-axis tells the network jitter in milliseconds which also varies depending on the congestion in the network. As shown in Figure 3.5, the attack traffic was launched at the 10th second, then immediately the network jitter started to increase. Clearly, when the mitigation was activated upon receiving the alert close to 40 second from the customer, the network jitter decreased to around 0.5 ms within 10 second. It shows that the legitimate traffic returns to normal level very quickly.

Throughput of suspicious traffic

Another interesting experiment was to test the throughput of the suspicious traffic, which are classified into three types in terms of its impact severity on the customer network: low, medium and high, as shown in Table 3.3. In the ISP network, suspicious traffic is not dropped as it may be legitimate. So, to avoid collateral damage to these kinds of traffics, the ISP provides them different processing through path with more number of hops and less bandwidth. As the results shown in Figure 3.6, the throughput of the suspicious traffic with *low* impact severity was maintained at around 300 Mbps, while the suspicious traffic with *medium* and *high* impact severity were redirected through the path with larger number of hops, with throughputs close to 200 Mbps and 150 Mbps respectively. Figure 3.6 shows that flows suspicious in nature get the bandwidth provisioned for them in the ISP network rather than getting dropped completely.

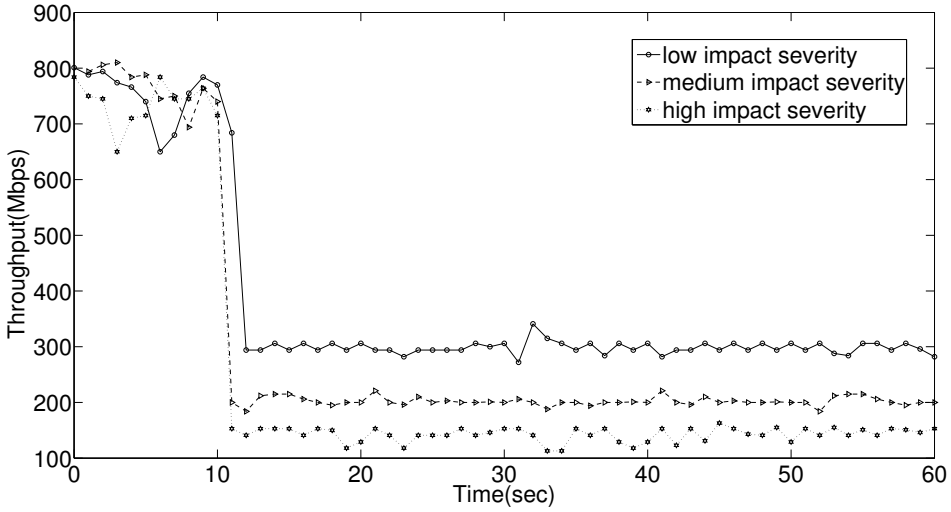

Figure 3.6 – Throughput of suspicious traffic with different impact severity

Table 3.4 – Platform specifications

Component	Quantity	Specification or Configuration	Role
IBM RackSwitch G8052	5	48 Gigabit ports	OF enabled switch
DELL server r620	2	8-core 2.9 GHz CPU, 16GB RAM, 10 Gb/s Ethernet NICs	SDN controller host (ISP, customer)
Hosts	3	Ubuntu server 12.04.2, 6-core 3 GHz CPU, 4GB RAM,20GB HDD	Playing as video streaming server L, attack host A and video streaming client C
SDN Controller	2	Ryu 3.20	Controlling switches in ISP (S_1 to S_4) and customer networks (S_5)

3.5 Testbed based Experiments

In addition to the simulation based experiments, we also experimented with an actual platform to evaluate our prototype *ArOMA*. In the platform, we use the video streaming service as the target to protect, and measure its quality, in the face of DDoS attacks, in terms of Quality of user Experience (QoE) metrics.

3.5.1 Experimental Settings

Platform. The topology of our experimental platform is similar to the data plane configuration shown in Figure 3.1, where the ISP and the customer networks have their own SDN controllers. The components and specifications of the platform are given in Table 3.4. For simplicity, two

Table 3.5 – Defined metrics to measure the impact of DDoS attacks

Metric	Definition	Unit	Impact of attack
Time to rebuffer	the time taken to rebuffer the video when the streaming buffer gets empty due to network congestion	second	increases
Time to start	the time taken to complete the initial buffering to start the video streaming	second	increases
Duration of paused mode	the time player is in the paused mode for the buffer to fill up during the entire video length	second	Paused for more time
Average buffer length	the buffer length of the player	second	increases
Average goodput	rate of data transfer	KB/sec	decreases

routing paths are configured in the platform for the ISP network: one is QoS guaranteed and used exclusively for legitimate traffic (going through switches S_1, S_2, and S_4), while the other is for suspicious or malicious traffic (going through S_1, S_3 and S_4). In the customer network, the OpenFlow switch S_5 is attached to the customer controller. Also, ten host machines are virtualized in the platform, serving as legitimate host, attack host and customer machine.

Traffic generation. We use video as the source of legitimate traffic, considering the fact that nowadays video traffic accounts for more than 70 percent of all consumer Internet traffic [38]. We use *TAPAS* [34], a video streaming tool for video traffic generation between the customer machine *C* and the legitimate host *L*. 10 *TAPAS* video clients generate a legitimate video traffic over HTTP traffic. Moreover, the BotNet Simulator (BoNeSi) tool is used to generate volumetric DDoS attacks with high traffic rates. We generate an attack traffic close to $250,000$ packets per second.

Evaluation metrics. The metrics used to evaluate our prototype are specified in Table 3.5, which are essentially related to these QoE metrics analyzed in [39, 67, 100]. Our hypothesis is that in the presence of DDoS attack, *time to rebuffer* will increase, *average buffer length* will decrease and the *duration of player in the paused mode* will increase. The intuition behind is that attack traffic may deplete the playout buffer of the video player, eventually reducing the quality of user experiences watching the video.

3.5.2 Results and Analysis

We conducted several rounds of tests to comparatively study the given metrics in three cases,

- Video streaming services in normal condition (without attack traffic);
- Under different attacks without mitigation;

- And under attack with our prototype *ArOMA* activated.

The metrics are measured at the video streaming client by collecting and analyzing the log files generated by TAPAS [34].

Time to rebuffer

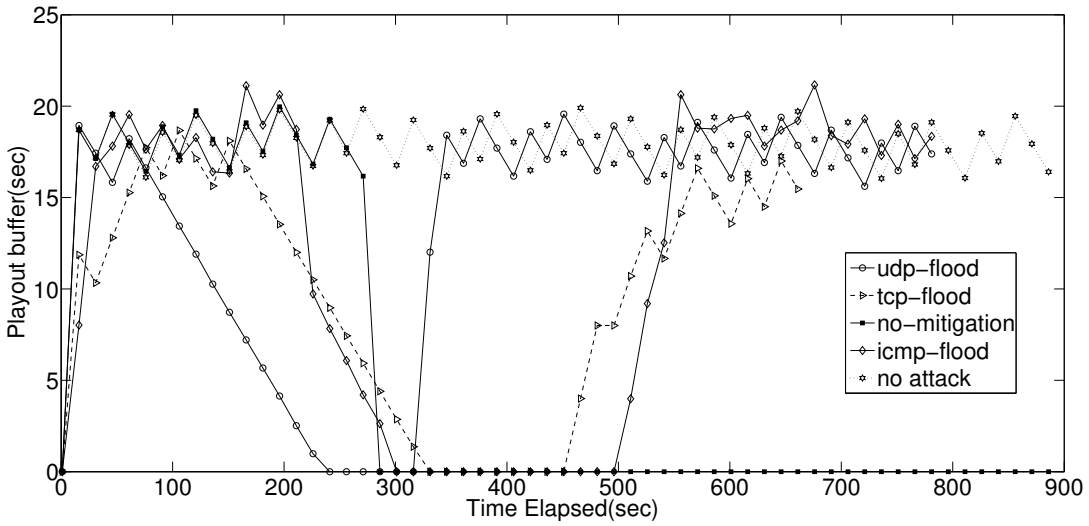

Figure 3.7 – Time to rebuffer

The hypothesis about this metric is that the client is not able to play the video as the streaming buffer gets depleted due to overwhelming attack traffic. If such a condition holds for a long time, the QoE of users will be degraded. Figure 3.7 clearly validates this hypothesis: the *TAPAS* video client was able to complete the video transmission without any interruptions when there was no DDoS traffic, and the buffer length was maintained above 15 seconds. In contrast, under DDoS attack, the video client was not able to buffer the video, leading to a depletion of the playout buffer down to zero. When the mitigation scheme was activated by the ISP upon the request of the customer controller, the playout buffer returned to a normal level, allowing the video client to play the video. When the buffering started, it took 10 to 15 seconds for the playout buffer to return to the normal level.

Average buffer length

We measured the average buffer length, and observed that the buffer length was maintained above 16 seconds when there was no attack. As shown in Figure 3.8, in the presence of UDP flood attack, the buffer length was maintained close to 14 seconds with *ArOMA* activated. Similarly, the buffer length was maintained at 12 and close to 10 seconds, in face of ICMP and TCP

flood attacks respectively. In contrast, the average buffer length was about 4 seconds when the mitigation module was not activated during attacks.

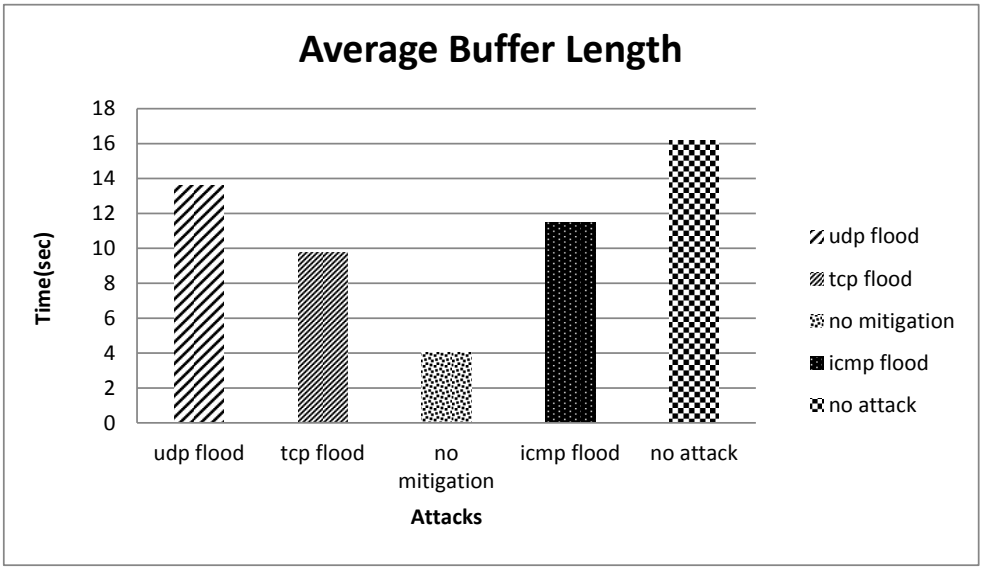

Figure 3.8 – Average buffer length of legitimate and attack traffic

Duration of paused mode

This metric was evaluated for comparing how much time the player is in the paused mode during initial buffering and in the midst of the attack. The results are shown in Figure 3.9, which has the following implications,

- When there was no attack, the total duration that the player was in the paused mode is approximately 2 seconds during the initial buffering phase;
- With *ArOMA* activated, the video player was paused for 12 seconds under UDP flood attack. The values were more than 15 and 40 seconds under TCP SYN flood and ICMP flood attack respectively;
- Noticeably, the player was not able to buffer the video to play if *ArOMA* was not activated.

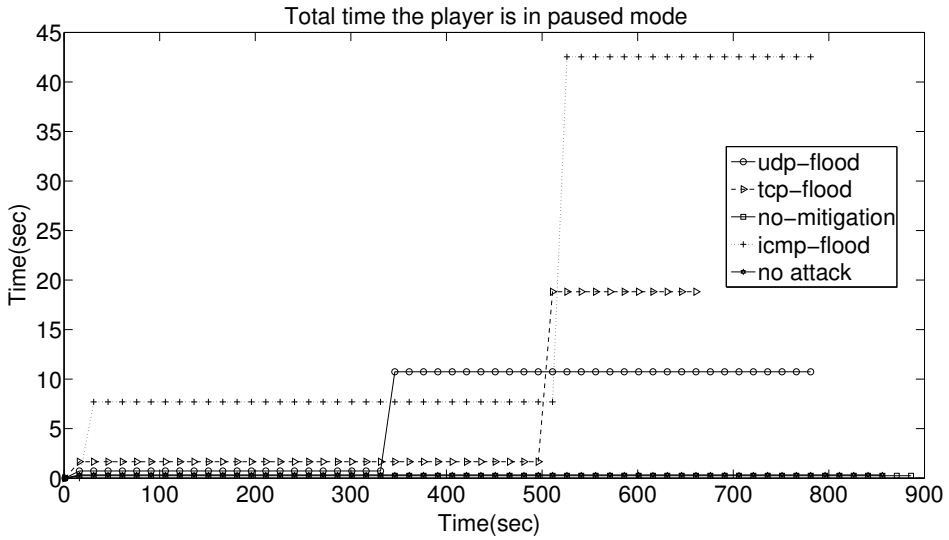

Figure 3.9 – Duration of the player in paused mode

Average Goodput

We also examined the *average goodput*, and observed it was maintained close to 700 KB/sec in the absence of attacks. When DDoS attack was launched and *ArOMA* was activated, the average goodput could be maintained at 550 KB/sec against the UDP flood attack. As for ICMP and TCP SYN flood attacks, it was maintained close to 500 and 450 KB/sec respectively, as indicated in Figure 3.10. In contrast, when *ArOMA* was not activated, the average goodput dropped down below 300 KB/sec and could not return to a normal level.

3.6 Discussion

Both the simulation based and testbed based experimental results demonstrated that *ArOMA* is able to provide quick response in mitigating attack traffic. It is worth noting, however, classifying the flows as suspicious and malicious are out of concern of our proposed framework. We assume that the customers can deploy some detection mechanisms to detect and classify the flows as suspicious and malicious based on their local intelligence and criteria. Nevertheless, *ArOMA* provides a distributed and automated way to distinguish suspicious traffic in terms of their severity, i.e., they will be assigned with different labels and redirected to different paths. Moreover, although the attack traffic used in the experiments are mainly UDP-flood, TCP-flood and ICMP-flood, *ArOMA* can also be applied to mitigate other types of attacks as long as the victim customers can provide sufficient flow features and attack characteristics.

One may argue that in *ArOMA* the communication channel between the ISP controller and the customer controller is vulnerable to attacks. While we have to admit this fact, the current

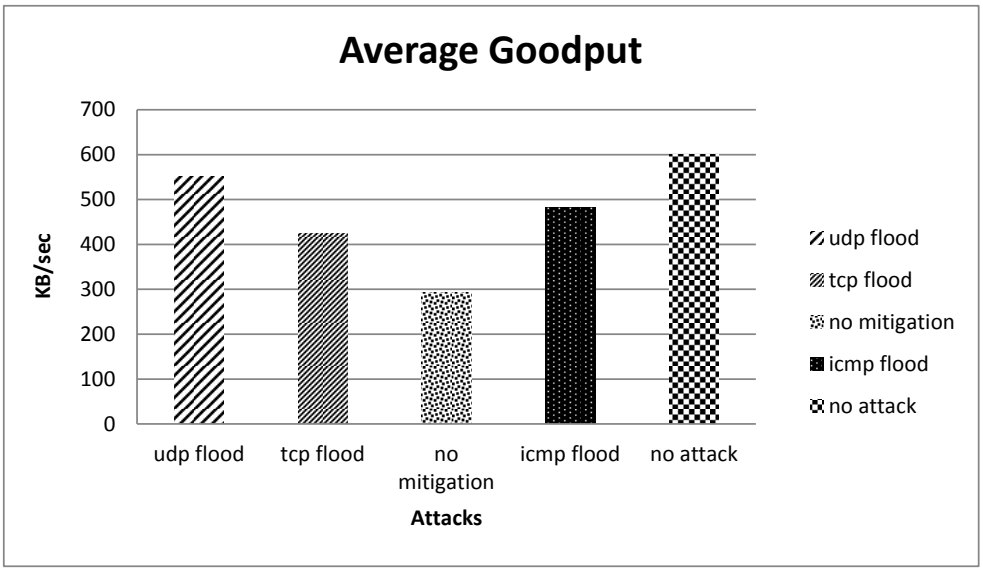

Figure 3.10 – Average goodput of legitimate and attack traffic

ArOMA design assumes that the customers of the ISP are behaving legitimately and they trust each other. It also ensures that every customer communicates to the ISP controller through a dedicated communication channel. In addition to pre-subscription and online authentication, other effective authentication schemes can be also deployed by the ISP, ensuring the integrity and confidentiality of alert messages sent from the customers. To cope with DDoS attacks, the ISP can restrict the amount of alert messages that are sent by a particular customer in a specific time interval. All attacks targeting at SDN controllers are not considered in this work, which have already attracted lots of attention from SDN and security communities [19, 37, 64, 111].

Another important concern of current *ArOMA* architecture is that it is implemented with a single SDN controller in the ISP and customer network. In practice, an ISP may have a large amount of customers, making it extremely challenging or even impossible to provide real-time mitigation service by relying on a single SDN controller. Extending the current *ArOMA* to have multiple SDN controllers is an non-trivial effort, which will be the focus of our future work. For example, multiple customers may have the conflicts in the installation of filtering rules. The scalability issue can be also caused by the latency resulting from controller-switch communication. To the best of our knowledge, new SDN controllers are able to handle millions of flows per second[65]. On a commodity machine with a single CPU core, state-of-the-art controllers are capable of responding to flow setup requests within milliseconds, and Open vSwitch is capable of installing

tens of thousands of flows per second with sub millisecond latency [123].

3.7 Related work

A comparative study on the existing DDoS mitigation techniques has been given in Section 3.1, this Section is devoted to the analysis of fundamental differences between our design *ArOMA* and the most related ones, regardless their categories specified in Section 3.1. For example, the Cooperative Detection and Reaction Architecture (CDR) [62] and CITRA framework [99] are related to our design *ArOMA* in terms of design purpose. However, one of the fundamental assumptions of CITRA and CDR is that the counter DDoS entities are deployed in trusted partners, so the successful operation relies on the collaboration of different networks. Such an assumption does not necessarily hold in *ArOMA*. Also, CITRA and CDR need to use multicast to communicate with different trusted domains, whereas *ArOMA* simply employs security API by leveraging the programmability of ISP. CoDef [72] is a collaborative defense mechanism against flooding attacks, enabling autonomous domains to collaborate with each other for redirecting the legitimate traffic of customers upon request. This mechanism depends on wide collaboration between different domains, and it does not block attack traffic in the source autonomous domain. Compared with CoDef, *ArOMA* does not require the collaboration between different ASes, and it can block attack traffic upon the request of customers.

The Overlay based architectures, such as SOS [60] and Mayday [9], consist of nodes which communicate with one another in the network. Traffic originating from the source node is redirected to an overlay, which is formed by the nodes for authentication, before it is finally forwarded to the target server. Clearly, these overlay nodes act as a buffer zone between the source of attack and target network. While more security can be possibly achieved, extra communication latency will be incurred. In *ArOMA*, the potential latency is only caused by the communication between SDN controllers, while the customers will not experience that.

One of the early DDoS mitigation schemes Pushback [59] relies on the upstream routers to block the identified suspicious flows close to the attack source. The difference between Pushback and *ArOMA* is that *ArOMA* shares the attack information via another communication channel between SDN controllers instead of data plane routers. Perimeter-based DDoS defense [30] maintains a multicast group at the ISP side and performs filtering when attack is originated from one of its customer network. However, our framework protects the customers of the ISP from the attack traffic originating outside the customer and ISP network. COSSACK [92] architecture also forms a multicast group at source and victim networks to mitigate the attacks. It requires wide collaboration between different networks to filter the attack traffic. Different from COSSACK, in *ArOMA*, ISP and their customers collaborate to mitigate the attack, where ISP provides security API to customers to request for the mitigation service.

Recent years have seen the efforts on developing novel Cyber defense techniques by leveraging the programmability of SDN controllers. For example, the authors of [73] proposes Drawbridge,

which allows the end hosts within an ISP to subscribe to the traffic engineering services provided by the ISP, so that an end host can specify its particular traffic engineering rules to the ISP's controller. ISP controller either installs these rules in its SDN switches directly or sends them to upstream ISP controllers. In doing so, it can avoid blind traffic engineering performed by the ISP without the knowledge of the end hosts. However, Drawbridge requires different ISPs to collaborate with each other to perform traffic engineering. Also, ISPs need to share their policy enforcement points (PEPs) with their customers, since the rules specified by the customers should be directly installed in the ISP's SDN switches. Different from Drawbridge, *ArOMA* allows ISP to enforce the mitigation policy according to the security alerts sent by the customer. More recently, an SDN-based DDoS defense mechanism called Bohatei has been proposed [41], which aims at helping ISP to provide DDoS defense as a service to its customers. Different from our framework *ArOMA*, Bohatei requires the ISP to perform the detection and mitigation, which brings huge computational burden to the ISP, as it has to analyze a huge amount of traffic for different customer networks, whereas *ArOMA* allows the customers run their local detection engines and only report alerts of concern to the ISP. In addition, Shin et al [101] proposes a framework FRESCO to enable security services to be customized and composed through the SDN controller, significantly facilitating the deployment of security services, including DDoS mitigation schemes. Our framework *ArOMA* can be treated as an extension to FRESCO, with an objective to provide collaborative DDoS mitigation between ISP and their customers.

3.8 Conclusion

It is widely recognized that without effective collaborations, it is extremely difficult, if not impossible, to mitigate DDoS attacks. However, the collaboration between different domains and parties in the Internet is challenging. In this chapter, we provide an on-demand DDoS mitigation framework called *ArOMA* by levering SDN, with an objective to facilitate the collaboration between the ISP and their customers. To demonstrate the feasibility and effectiveness of our proposed framework, we developed a proof-of-concept prototype and conducted both simulation based and testbed based experiments. The results indicated that our mitigation framework can ensure that the protected asset, a video streaming service and a server, was able to maintain satisfactory performance in the presence of DDoS flooding attacks, in terms of major QoE metrics.

Our experiments and analysis of the prototype have identified some key benefits of the framework: (1) the automated collaboration between customer and ISP would minimize the reaction time in an operational situation; (2) Without exposing their network topology to outside domain ISP can deploy policies to mitigate the attacks upon receiving the alert messages from its customers. Customers can also deploy their local polices in its network to filter the attack traffic; (3) Customer and ISP effectively communicate through the security API deployed at the ISP controller and exchange the alert in a standard IDMEF format; (4) ISP is able to provide different countermeasures to the suspicious flows by redirecting them to different paths, ensuring the best QoS to legitimate flows. As such, even the suspicious traffic are guaranteed to reach the

customer network but with degraded QoS, which potentially reduces the collateral damage in the ISP network. In particular, we demonstrated that the centralized SDN controller can significantly simplify the enforcement of DDoS mitigation policies in an automated way. The proposed architecture is not a detection system but rather a collaborative mitigation system to filter the attack traffic and provide good services to legitimate traffic. In the next, chapter 4, we present a mechanism to represent and translate the high-level security and QoS policy into low-level OpenFlow (OF) rules.

4 SDN-Oriented Policy Representation and Translation

In the large scale distributed network environment configurations of devices such as switches, routers, firewall, Intrusion Detection System (IDS) are frequently done to protect and adapt the network according to changing conditions. Manual configuration of these devices is tedious, complex and error prone. Moreover, recent IT trends further aggravate these problems. For instance, the growing demand of adaptive security and low performance overhead require massive and frequent changes in the configuration of security and network devices. Furthermore, manual configuration in such a diverse situation can negatively impact the performance of the services in the network. Even the policies are in low level format, they also don't adapt to continually varying network conditions. The lack of dynamic configuration leads to network downtime because of misconfiguration, as it requires manual work from the network operators.

Recent developments in SDN have led to a proliferation of studies that automate and simplify the network management tasks. The northbound API in the SDN enables us to define the high-level policies at the controller which can be enforced in the data plane devices through southbound interface. Policies can be modified without halting the operation of network devices. Moreover, network administrators do not have to configure all the switches, routers and middleboxes manually to adapt the network behavior according to varying conditions. Instead decisions are taken from a logically centralized controller.

In spite of the advantages, SDN alone does not have the ability to define policies in an expressive way. This impede the deployment of new services in the network. There are some works reported of using PBM (policy-based management) in SDN to govern the behavior of the network. Most of these mechanisms use low-level syntax for expressing the policies which are directly introduced in the controller. We argue that high-level policies should be written in the human readable format. It should free the network operator from learning the low-level device specific syntax for writing the policies. Policy translation techniques should be applied on the high-level policies to automatically translate them into low level rules for deployment. However, there are very few works reported on addressing the issue of translating the high-level policies into low-level rules for enforcement in SDN network [76]. If the translation of high-level policy to

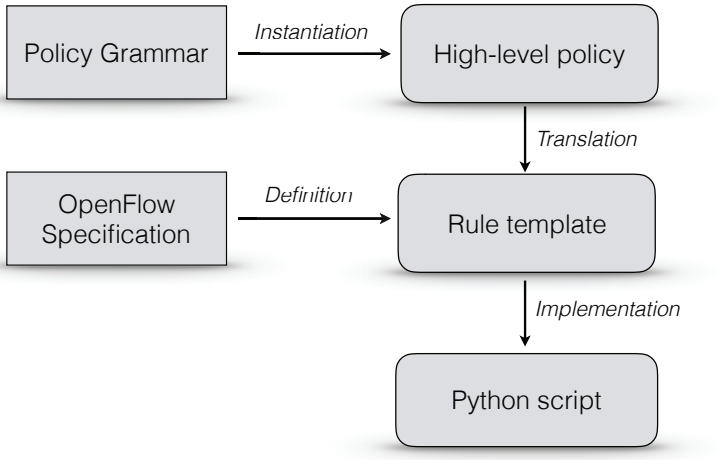

Figure 4.1 – High-level view of the policy translation process

low-level rule is not done properly then the administrator would not meet the requirement and it would lead to misconfiguration in the network. To address these issues, our aim is to develop policy representation and translation technique, where high-level policies can be translated into a low-level device specific rules, which are deployed into the network and security devices.

The aim of this chapter, is therefore, to address how the high-level policies can be represented in human readable format in SDN network and translated into low-level OpenFlow rules for the deployment in switches. The high-level view of our proposed mechanism as shown in Fig. 4.1 is supposed to: (1) define the high-level policies based-on the syntax specified in the policy grammar, (2) Specify the templates of the low-level rule corresponding to different types of actions, (3) translate and enforce the high-level policies into the OpenFlow switches using python scripts. Depending on the security alerts and QoS request message policies are activated and translated for the enforcement in the OpenFlow switches.

This chapter proposes a new methodology for translating the high-level policy into low level rules for QoS management and mitigating attack traffic in SDN. We built a framework with some requirements and considerations. The description of the framework is presented in Section 4.1. Moreover, high-level policies should also be readable by human. We express the policies using Controlled Natural Language (CNL) [68] so that network operators need not be familiar with the OpenFlow concepts for defining the policies. We propose a policy grammar to define the high-level policies in Section 4.2. Furthermore, OpenFlow rule templates are also provided as a guideline on what tuples the low-level rules should be expressed in the different network and security devices. It helps the infrastructure programmer to write the scripts which can translate the high-level abstract policies into low level rules. These OpenFlow rule templates are presented in Section 4.3. Then, we validate the policy translation process using two scenarios namely: (1)

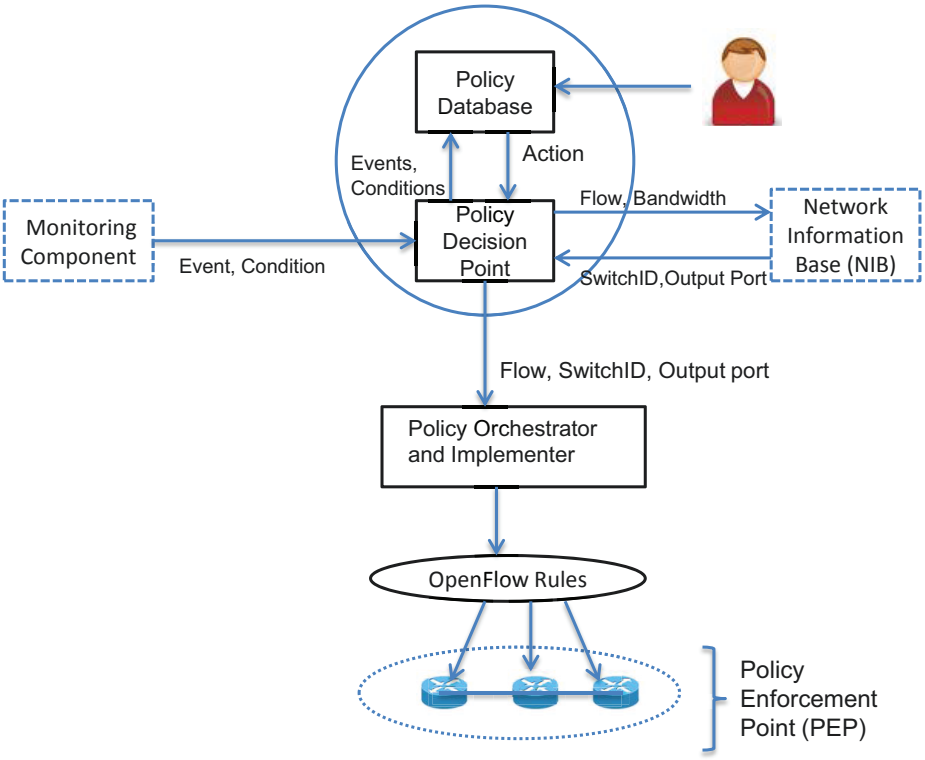

Figure 4.2 – Policy management and enforcement framework

on-demand QoS provision, (2) DDoS attack mitigation. The implementation of these scenarios are discussed in Section 4.4 and in Section 4.5. Then, we present the comparison of our policy language with the some existing high-level policy languages on SDN in Section 4.6. Section 4.7 discusses some existing works in the literature on policy translation and enforcement. Finally, we conclude in Section 4.8.

4.1 Policy Representation and Translation:Configuration

The network and security policies of a network should be specified at the highest level of abstraction, and then translation techniques should be used to map the higher-level policies to the low level. To achieve this gradual implementation, we relate events, conditions and actions described abstractly, to the low-level rules. High-level policies are defined by the network operators which are enforced depending on the conditions. In this chapter, as shown in Fig. 4.2, we mainly focus on specifying the high level policies in the policy database and translating these policies to low-level rules for deployment in data plane devices using Policy Decision Point (PDP)

and Policy Orchestrator and Implementer (POI) shown in thick black line. In detail description of other components will be given in chapter 5.

4.1.1 Major Components

Monitoring Component (MC) receives the security alert or QoS request message and forward it to the PDP.

Policy Decision Point (PDP) is in charge of the global policy decision. Based on the event and conditions it activates a policy in the policy database which provides a high-level action (forward, redirect, drop) to be enforced. Corresponding to the high-level action, flow information and bandwidth requested PDP gets the path details (switch ID, output port) from the NIB. PDP forwards the path details along with the flow informations as a match field to the POI for the low-level rule enforcement. Additionally, for the suspicious class of traffic PDP also maintains a table containing the list of middleboxes to traverse in the network. If the flow is suspicious in nature then PDP provides a list of middleboxes to NIB, which in return provides a path containing the middleboxes to traverse.

Policy Database (PDB) It is essentially a repository containing the high-level security and network policies specified by the network operator, without detailing the specific deployment strategy. Policies can be specified in XML or any other format defined by the network administrator. Based on the event and conditions as input from the PDP, it provides a high level action as an output. The high-level policies in the PDB are instantiated based-on the security alert, QoS messages forwarded from the PDP. A Policy ID is used to index the policies in the database, which helps in retrieving the policy when an event occurs.

Network Information Base (NIB) contains a list of middleboxes and switches in the network where they can be found. It contains the template to route the traffic through a different path in the network depending on the bandwidth. Details of this component is discussed in the chapter 5.

Policy Orchestrator and Implementer (POI) contains the OpenFlow rule templates of different actions for enforcement in the data plane devices. `OpenFlow rule templates` contain the guidelines to specify how the different activities can be executed in the network. It provides an additional level of modularity to define the format for the different types of tasks that can be performed. It is an important part of the translation process as it provides an abstract view for the concrete rules which are to be deployed. More details about the template module is given in the Section 4.3. Moreover, it contains python scripts which deploy the rules in the data plane devices. It takes the flow information as a match field along with path details (switchID, output port) to deploy the low-level OpenFlow rules depending on the high-level action.

Algorithm 3 shows all the process of receiving the event messages, extracting and matching of the information with the policies and enforcing the action in the network devices using FlowMod message. In the process of implementing the policy when an event message is received event

type and conditions are extracted. For example, if the QoS request message is received, then the event type(e.g. Gold, Bronze, Silver QoS) is extracted from the message. Moreover, the conditions from the QoS message such as bandwidth class, flow class are also extracted from the event message. Then, depending on the event type and the conditions a policy is instantiated in the policy database, which provides a high level action related to the policy. Based on the high level action and flow information a path is selected from the NIB. The path contains the list of switches in an order from ingress switch to egress switch.

If the rules are deployed in the order from ingress switch to egress switch then it would cause the huge packet loss in the case, when low-level policy rules need to be modified to redirect a flow through another path. So, our algorithm arranges the switches so that first it deploys the rules on the core switches then at last on the border routers, so that the in the meanwhile flow can traverse through the previous path. So Algorithm 3, copies the switch ID of the ingress switch in a stack and removes the ID of the ingress switch from the list of switches that it gets from NIB. Then, it deploys the low-level rules on the core switches and egress switch. After that, it pops switch of the ingress switch from the stack and deploys the rules. Then, the flow automatically traverses through the path. The effectiveness of this methodology is validated later in Sect. 4.4.3

Algorithm 3 Policy Implementation

for each Event *event* **do**
  $event\_type \leftarrow event.getEventType()$ // get the type of event from the message
  $event\_information \leftarrow event.getInfo()$ // Get condition from the event message
end for
for event_type and event_informations **do**
  $action \leftarrow event\_type.getAction(conditions)$ // Get the action from the policy repository based-on the conditions specified in the event
end for
for each actions and flowInfo **do**
  $(dpid, out\_port) \leftarrow topology.getPath()$ //Provides the list switchID and the output port to forward the flow from the NIB
  Copy the switch id of the ingress switch in a stack
  $FlowMod(match : S\_IP, D\_IP, actions : out\_port)$ // Deploy the rules on the list of switches
  $FlowMod(match : S\_IP, D\_IP, actions : out\_port)$ // Deploy the rules on the ingress switch at last
end for

4.1.2 Operational Workflow

Depending on the informations mentioned in the alert or QoS request message the ISP controller translates the high-level policy into low-level rules. The workflow of the policy enforcement process is described based-on Fig. 4.2.

1. In the framework, the Monitoring component (MC) receives the security alerts or QoS request message. MC extracts the events and conditions from the alert or QoS request message and forward these informations to the Policy Decision Point (PDP). Informations in the alert or QoS messages contain the flow informations (source IP, destination IP), flow

class, bandwidth class and attack type.

2. PDP triggers an appropriate policy in the Policy Database depending on the events and conditions. In return, it gets the high-level action for the enforcement from the Policy Database.

3. High-level action is further refined to get the path to route the traffic based on the flow informations and bandwidth. PDP gets the concrete path details from the Network Information Base (NIB) with specific switch ID and output port to steer the flow.

4. Then, PDP specifies the flow details as a match field along with Switch ID and output port informations to Policy Orchestrator and Implementer (POI) which generates the the low-level OpenFlow rules. The step by step process of policy translation process is described for QoS and mitigation scenario in Section. 4.4 and Section. 4.5.

4.2 Policy Grammar for High-level Policy

The high-level policy syntax provides the guidelines to define the abstract policy. It enables the network operator to express the network and security policies into an easy to understand language without getting into low level implementation details. These high-level network and security policies can be enforced into the data plane devices when the requirements arise. Our high-level policy syntax provides network administrators with a collection of constructs that allow them to define the intended behavior of the network at a high-level of abstraction.

Listing 4.1 – Grammar for the High-level policy language

```
1 Policy=PolicyID
2 Event=Message
3 Conditions=Condition Connective Conditions
4 Condition=<field_name><value>
5 Connective=And|Or
6 Action=Forward|Drop|Redirect
```

In our policy framework, high-level policies are defined as event, condition and action paradigm [63]. An event-condition-action (ECA) paradigm is the method in which actions are triggered, attributed to the presence of particular conditions. The ECA paradigm describes how the events trigger the desired response from the system. An event could be a message from a system or program, among varied number of other possibilities. Condition in the ECA paradigm is the logical part which, if evaluates to true then activates the action. Action is defined as an operation carried out on the available resources.

Our policy grammar provides the syntax to define the security and network policies. The Northbound APIs available in the SDN controller enables us to define the high-level policy language in a simple way. We use the policy grammar shown in Listing 4.1 to define the network

and security policies for the OpenFlow network. In the SDN network,OpenFlow switches can behave as middleboxes such as firewall, and some other middleboxes. With our policy language, we can define the policies for any equipment whether OpenFlow switches or switches behaving as firewall.

4.2.1 Event

Event represents the part specified for the policy to be triggered. (e.g. the arrival of security alert message, arrival of new packet as an event). In Listing 4.1, we show events as a "Message". Events are not explicitly mentioned in the grammar as they can be arbitrary strings. Flooding attack event can contain information related to flooding attacks such as UDP, TCP, ICMP, etc. Similarly, QoS_Request event can contain the request messages for providing differentiated QoS services. It may contains the QoS request message for Gold_QoS_Request, Bronze_QoS_Request, and Silver_QoS_Request. For instance, Gold_QoS_Request message is for providing the highest bandwidth path with no delay. Our policy language is not limited to the events described here, it can be extended to include more events based on the requirements of the network.

4.2.2 Condition

When an event is triggered, the associated parameters are checked against the condition specified in the policy. Condition is generally a boolean expression which can be evaluated to true, false or not applicable. Not applicable represents that no condition is specified for the event. In our grammar shown in Listing 4.1, Condition is specified with the field name and a value. Field name contains an arbitrary string and a value. For example, it can contain the security or network assessment information such as impact severity of a traffic on the network, flow class, bandwidth class, traffic type, service, etc. Impact severity, flow class, bandwidth class, etc. are field names.

Impact_Severity specifies the impact of attack traffic or traffic causing the congestion in the network. It can take either of three values such as as low, medium and high.

Flow class categorizes the flow into three different classes known as: suspicious, malicious and legitimate. Suspicious indicates that the flow is a mix of legitimate and malicious flow. The flow which is confirmed as an attack traffic is specified as malicious flow. The benign traffic is defined as legitimate flow. The flow informations include source and destination IP addresses. Traffic type specifies the different types of flows based on their protocols such as UDP traffic, HTTP traffic, TCP, etc.

A Service is defined as the IP address and port of a server that a flow may access. For example, we can define the service web server as: web_server = 10.0.0.3. It means that the server with the IP address 10.0.0.3 provides the web services. Moreover, if the destination address of the flow is (10.0.0.3), it means that flow is accessing the web server.

A Bandwidth class represents the different classes of paths in the network represented in terms of bandwidths. It contains three classes of bandwidth: Gold, Silver, and bronze class. Path in the OpenFlow network is defined as SwitchID and the output port. The classes of bandwidths are administrator defined based on the topology. For example, a network operator can define the gold class as a path which has a bandwidth higher than 400 Mbps.

4.2.3 Action

Action specifies the high-level decision which should be enforced when the conditions are met for the event. In Listing 4.1, we have listed a few high-level OpenFlow actions for our policy translation framework. OpenFlow specifies two generic high level actions:Forward, and Drop [89]. Apart from these actions, we have also specified the action `redirect` in our grammar.

The action `forward` represents that when a flow arrives in the network then it is routed through the network. `Forward` action signifies that there is no information about the flow in the data plane devices. `Redirect` action represents that there is information about the flow in the network path but it has to be redirected through another path in the network. Generally, redirection of the traffic occurs in the network because of link failure, congestion or when a flow has been identified as suspicious. `Drop` action specifies blocking of the flow in the network. Drop action is applied on the flows which are identified as malicious. Examples with the policy syntax are described later in the Section 4.4 and Section 4.5.

4.2.4 Mapping of High-level Policy to Low Level Rules:

As we specify the policies using ECA paradigm, therefore, a typical policy consists of three main parts:

ON (Event) IF (Condition) DO (Action)

Table 4.1 shows the list of Events, Conditions and Actions we use to specify the policies. Additionally, we provide the corresponding match field according to the OpenFlow specification for the events, conditions and actions. The QoS request and security alert events are translated into flow informations which include source and destination IP while specifying the OpenFlow rule. Condition in the high level policy is mapped to the protocol, VLAN_ID, destination IP and Input_Port informations. Furthermore, the high-level actions are mapped to switch ID (dpid) and setting a VLAN_ID field and providing an output port at the switch. Additionally, Drop action in OpenFlow is specified with empty action field.

When an event occurs in the system policies can be triggered by the PDP. If the Event comply with the conditions in the policy then an action is returned from the Policy Database. We use the following high-level abstract policy to show how the logical reasoning works and how the high level policy is translated.

Table 4.1 – Syntax to Specify Policies

	Syntax	OpenFlow Match
Events	Gold QoS, Bronze QoS, Silver QoS, Security alerts	Source IP, Destination IP
Conditions	Flow Class, Bandwidth Class, Impact Severity, Traffic Type	Destination IP, ip_proto, In_Port,VLAN
Action	Forward, Redirect, Drop	dpid, action={VLAN,out_port},drop={}

"Bronze QoS request should receive a bandwidth higher than 200Mbps and less than 400Mbps".

The above high level policy means that when an Event occur which request to provide bronze QoS service for a flow then flow should be redirected through bronze path in the network. The steps to reach the conclusion are described below:

1. When the QoS request event occurs in the network, then the QoS class requested is checked in the request message, e.g., Gold QoS, Bronze, Silver. If there are occurrences of these QoS classes then the corresponding policy is triggered in the policy repository.

2. In the bronze QoS policy the condition part is evaluated to provide the bandwidth to the flow. In this case, the bandwidth is higher than 200Mbps and lower than 400Mbps. Depending on the evaluated condition the high level action is returned which is "Redirect".

3. This high-level action "Redirect" is further translated into the exact path depending on the bandwidth granted to the flow. This further processing can be generally applied to all processed actions for any flow. The low level rules in the OpenFlow switch is represented by the flow informations (source and destination IP address) as a match, and the output port. The translation of the high-level action redirect into the low level OpenFlow rule is described in detail in the Section. 4.4.2. In the OpenFlow switch, the packets can be represented as event and flow informations can be described as a match field and output port is the action which comply with the condition part of the policy.

4.3 OpenFlow Rule Templates

Rule templates provide a way which allows to instantiate the rules for different tasks in the network which can be enforced by different devices. Now, we discuss about the generic templates used in our framework for different tasks. OpenFlow rule templates are maintained at the PDP

except the template which specifies the path details.

4.3.1 Template for Specifying the Path

In our policy framework, a path is specified as a set of switch IDs and output ports. The Listing 4.2 shows a template to define a path with the switch ID and the output port to route the flow through a specific path. Each path is associated with a unique label which helps in fast forwarding the flow through that path. Template of the different paths are maintained in the NIB.

Listing 4.2 – A template to Specify the Paths in Network

```
1 Path={Switch:'SwitchID','Port':Output_Port}
2 Label={PathID:VLAN_ID}
```

4.3.2 Template for the forward Action

The high-level action forward is used for routing the traffic through a path. The Listing 4.3 shows the template for the action "Forward". The template shows that at the ingress switch, forwarding rules are specified with: source IP, destination IP address, setting a label in the Vlan ID field, and forward the flow through output port. It can be used for forwarding the flows originating at a specific source and going to a particular destination. In the core switches forwarding rules are specified only with matching a label and forwarding the flow through an output port. Moreover, at the egress switches rules are specified with: destination IP address, with deleting the label and forwarding the flow through an output port. To forward all the flows from a source, it requires to specify source IP address, providing a label in the flow and specifying the output port at the ingress switch. Furthermore, to forward all the flows traversing to a specific destination requires the destination IP address, setting a label in the flow and forwarding the flow through an output port.

Listing 4.3 – OpenFlow rules templates to forward the flow

```
1 Ingress Switch: ['Source IP': addr, 'Destination IP': addr, 'SetLabel':
    Vlan_ID,'Action':Out_Port]
2 Core Switch: ['Action':Out_Port,'Label':Vlan_ID]
3 Egress Switch:['Destination IP': addr, 'Label':pop,'Action':Out_Port]
4 All the flows originating at a a source: ['Source IP': addr, 'SetLabel':
    Vlan_ID,'Action':Out_Port]
5 All the flows traversing to a destination: ['Destination IP': addr, 'SetLabel
    ':Vlan_ID,'Action':Out_Port]
6 Specific flows originating at a source: ['Source IP': addr, 'Protocol': proto
    , 'SetLabel':Vlan_ID,'Action':Out_Port]
```

4.3.3 Template for the drop action

The drop action is used to filter the traffic. It can be specified at any location in the network block the traffic. Listing 4.4 shows an abstract template to specify the drop action depending on the condition in the network. For instance, to drop a specific flow, we specify its source IP, destination IP and protocol. Moreover, dropping all the flows originating from a particular source requires to specify only source IP and action drop. Similarly, to filter all the flows traversing to a particular destination, requires only destination IP address and action drop.

Listing 4.4 – OpenFlow rule template to Drop the Flows

```
1 Drop specific flows: ['Source IP': addr, 'Destination IP': addr, 'Protocol':
    proto,'Action':Drop]
2 All the flows originating at a particular source: ['Source IP': addr, 'Action
    ':Drop]
3 All the flows going to particular destination:['Destination IP': addr, '
    Action':Drop]
```

4.3.4 Template for the redirect action

The redirect action is used to divert the traffic from its previous path to a new path because of link failure or congestion in the previous path or a flow is required to be provided better QoS. Listing 4.5 shows a template to specify the action "Redirect" at the switches in the network for diverting flows depending on the conditions. For instance, to redirect specific flows OpenFlow rules are specified with: source IP, destination IP address, modifying a label and forwarding the flow through a output port of ingress switch. Moreover, to redirect all the flows originating from a particular source, we require only to specify the source IP address of the flow and setting a new label in the VLAN ID field at the ingress switch. In a similar way, to redirect all the flows traversing towards a particular destination, it only requires destination IP address and setting a new label on the flow and specifying the output port of the ingress switch to redirect the flow.

Listing 4.5 – OpenFlow rule template to redirect the flow

```
1 Redirect specific flows: ['Source IP': addr, 'Destination IP': addr, '
    SetLabel':Vlan_ID,'Action':Output_Port]
2 Redirect the flows originating at a specific source: ['Source IP': addr, '
    SetLabel':Vlan_ID,'Action':Output_Port]
3 All the flows going to particular destination:['Destination IP': addr, '
    SetLabel':Vlan_ID,'Action':Output_Port]
```

4.3.5 Implementation of the Redirect and Drop Action:

Python script shown in Listing 4.6 provides a template to redirect the traffic in the network through another path. Script is instantiated when the high-level action corresponding to a policy is "redirect". It takes as inputs the switch ID (dpid), output port (out_port), label (vlan_vid), and flow informations as a match to deploy the rules to redirect a flow. First, the OpenFlow rules are deployed in the core switches according to Algorithm 3. Then, the rules are modified at the ingress and egress switch respectively to redirect the flow through new path. As the script shows, at the ingress switch a label is inserted in the flow and the flow is redirected. Core switches only checks the label and forward the flow through an output port. At the egress switch, the label is popped and flow is forwarded depending on the destination IP address through an output port.

However, when the flow is specified as malicious then the script to drop the flow is instantiated. It takes the flow informations as a match field and ingress switch ID to drop the flow. Malicious flows are dropped at the ingress switch to reduce the collateral damage caused to legitimate flows. Empty action in the script specify to drop the flow.

Listing 4.6 – Script to enforce the redirection policy

```
1  def redirect(self,parser,dpid,out_port,match,dst,actions,vlan_vid)
2  {
3   if (dpid == core_switch and match):
4   actions = [parser.OFPActionPushVlan(0x8100), parser.OFPActionSetField(
      vlan_vid = (vid)),parser.OFPActionOutput(out_port)] // Redirect the flow
5   if (dpid == ingress_switch and vlan_vid == vid):
6   actions = [parser.OFPActionOutput(out_port)]
7   if (dpid == egress_switch and dst == 'dst_ip'):
8   actions = [parser.OFPActionPopVlan(0x8100),parser.OFPActionOutput(out_port)]
9   return actions
10 }
```

Listing 4.7 – Script to drop a malicious flow

```
1  def drop(self,parser,dpid,match,actions)
2  {
3    if (dpid == ingress_switch and match):
4   actions = []
5   return actions
6  }
```

4.4 Scenario 1: On-demand QoS Provisioning

One key feature of our policy translation mechanism is that it enables the ISP to automatically react on a QoS request message sent by its customers. We discuss this use case with the sample

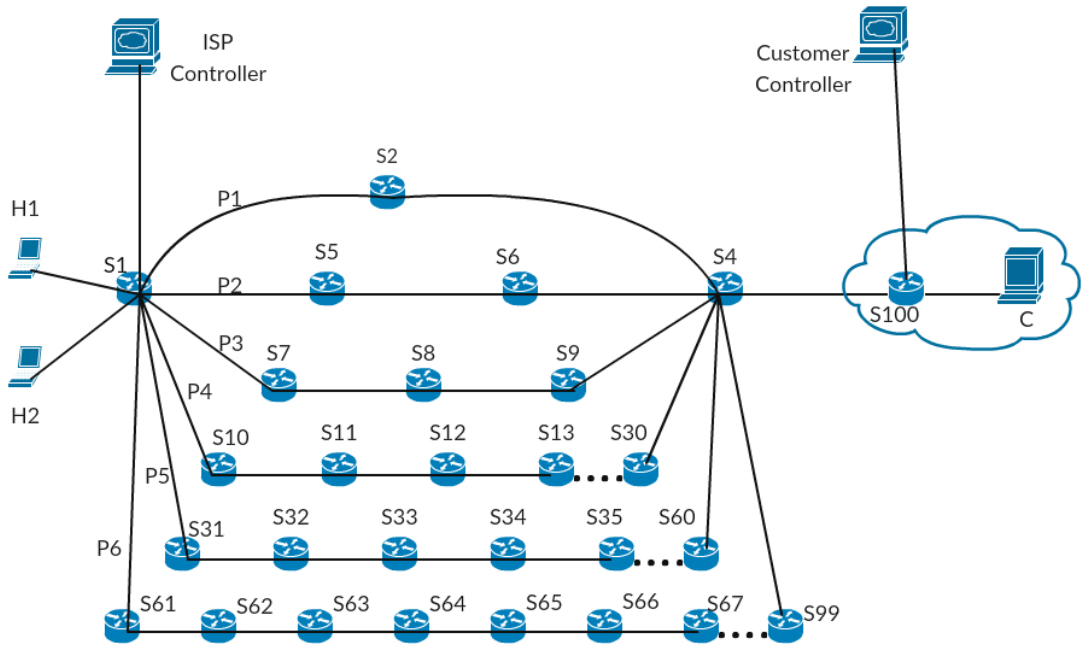

Figure 4.3 – Deployment of Policies in the Network

network, as shown in Fig. 4.3. The sample network consists of one ISP and a customer network represented as C. There are two external hosts shown as H_1 and H_2. These external hosts send the traffic to the customer network. The customer requests the ISP to activate the desired QoS policy for a specific flow by sending the flow class, bandwidth class request, source IP, and destination IP. In the following scenario, the customer requests for the *gold* class of bandwidth from its ISP. Step by step example of the gold QoS policy translation are given in the Sect. 4.4.2.

4.4.1 Configurations for on-demand QoS:

The translation of high-level QoS policy into low-level OpenFlow rules requires some configurations. In this regard, we discuss the required inputs for the enforcement of the QoS policy in the ISP network depending on the request from the customer. Moreover, we present the templates specific to the scenario of gold QoS policy enforcement.

QoS request message received from customer: On-demand QoS messages are the request from the customer to their service provider for differentiated QoS service. The request message contains the flow information and the type of QoS service requested. As shown in Listing 4.8 QoS request contains the flow informations as: source IP: 10.0.0.1, destination IP:10.0.0.3. Source IP 10.0.0.1 represents the external host H_1. Moreover, it contains the bandwidth class demanded as "Gold" and the flow class as "Legitimate". The event type which is "Gold QoS request" is mentioned in the classification event field of the request message.

Gold QoS Policy at the ISP Controller: The policy file shown in Listing 4.9 provides the gold bandwidth path in the ISP network. It contains the event type information which is "Gold_QoS". The condition part of the policy contains flow class as "Legitimate" and bandwidth class as "Gold". Based-on the event and conditions it provides the high-level action as "Redirect". The high-level action "Redirect" is further refined using Table 4.2 which is described in detail in Sect. 4.4.2. Bandwidth of 400 Mbps is provisioned for gold class.

Listing 4.8 – QoS Request Message Sent by Customer

```
<IDMEF-Message version="1.0">
<Alert>
    <Analyzer analyzerid="CUSTOMER_C"/>
    <Source>
       <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.1</address>
            </Address>
        </Node>
        <Service>
            <protocol>udp</protocol>
        </Service>
    </Source>
    <Target>
        <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.3</address>
            </Address>
        </Node>
    </Target>
   <Classification event="Gold QoS">
    </Classification>
    <AdditionalData> type ="string" meaning="flow class">
    <string>Legitimate</string>
    </AdditionalData>
    <AdditionalData> type ="string" meaning="bandwidth request">
    <string>Gold</string>
    </AdditionalData>
</Alert>
</IDMEF-Message>
```

Table 4.2 – A Mapping Table between High QoS Bandwidth and the Network Path

Path	Destination Network	Bandwidth Class
P1	10.0.0.3	Gold
P2	10.0.0.3	Silver
P3	10.0.0.3	Bronze

Listing 4.9 – A High-level Gold QoS Policy in the ISP Network

```
<Policy PolicyID="QoS">
    <Event Type = "Gold_QoS">
    </Event>
    <Condition>
       <flow class="Legitimate">
       <bandwidth class="Gold">
    </Condition>
    <Actions action="Redirect"/>
    </Actions>
</Policy>
```

Listing 4.10 – Instantiation of the template to steer the traffic through high QoS path

```
P1:{Switch:S1,output(5); Switch:S2, output(2); Switch:S4,output(2)}
P2:{Switch:S1,output(6); Switch:S5, output(2); Switch:S6, output(2); Switch:
  S4,output(2)}
P3:{Switch:S1, output(7); Switch:S7,output(2); Switch:S8,output(2); Switch:S9
  ,output(2); Switch:S4,output(2)}
Path={1:P1, 2:P2, 3:P3}
```

Instantiation of the template to route the flow through high QoS path:

The Table 4.2 provides the path to route the flow through gold, silver and bronze class of bandwidths. The bandwidth of 350 Mbps and 300 Mbps are provisioned for silver and bronze class respectively. Furthermore, as explained in the Sect. 4.3, paths are defined as switch ID and output port informations. Listing 4.10 provides the concrete path details along with the switch ID and the output port corresponding to the path shown in Table 4.2.

4.4.2 Step-by-Step Example

Listing 4.11 – OpenFlow rules in the Switch S1 before the deployment of QoS Policy

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:3->
    vlan_vid,output:7
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:3->
    vlan_vid,output:7
```

To summarize this section, we provide an example of how a high-level QoS policy is transformed into low-level rules for the enforcement. This use case is described using the network shown in Fig. 4.3. Customer C sends the request to the ISP to provision the gold QoS path to the flow coming to its network from H_1. As Listing 4.11 shows, rules in the switch S_1 to forward the flow through the path P_3 by inserting a label 3 in the packet through an output port 7.

1. Customer C sends the QoS request message to the ISP controller to provision for the gold path for the traffic coming to its network from host H_1. The QoS request message is shown in the Listing 4.8.

2. The Monitoring Component extracts the event type (Gold QoS) and conditions such as the bandwidth request ("Gold") and the Flow class ("Legitimate") from the QoS request message. Then, it forwards the event (Gold QoS) and conditions to the PDP. Using the event type (Gold QoS) and conditions, the PDP checks the policy database to get the high-level action. In this case, the high-level action is "Redirect". Listing 4.9 shows the high level QoS policy for the gold class.

3. The high-level action "Redirect" is further refined to get the concrete path details. PDP sends the high-level action redirect, flow informations (source and destination IP) and bandwidth class (Gold) to the NIB. Then, NIB uses the bandwidth class (Gold) and destination network informations to get the path to redirect the flow using Table 4.2. A concrete path details corresponding to the P_1 is is got from the template shown in the Listing 4.10. Note that in this case, the path consisting of switch ID and the output port is: (S_1:5, S_2:2,S_4:2). Moreover, the label for the path is 1.

4. The PDP uses the high-level action "Redirect" along with the bandwidth requested to instantiate the script in POI. The script to redirect the flow through the gold QoS path is shown in Listing 4.13. The concrete path details along with flow informations (source and destination IP) as a match field provided provided as an input to the script which deploys the OpenFlow rules in the switches. Listing 4.13 shows that the rules first deployed in the core switches, then at the egress and ingress switch respectively. Rules are deployed in the switches according to the template shown in Listing 4.5. Listing 4.12 shows the low-level rules in the switch S_1 to forward the flow containing source IP 10.0.0.1 and destination IP 10.0.0.3 through an output port 5 by inserting a label 1. So that flow can traverse through path P_1.

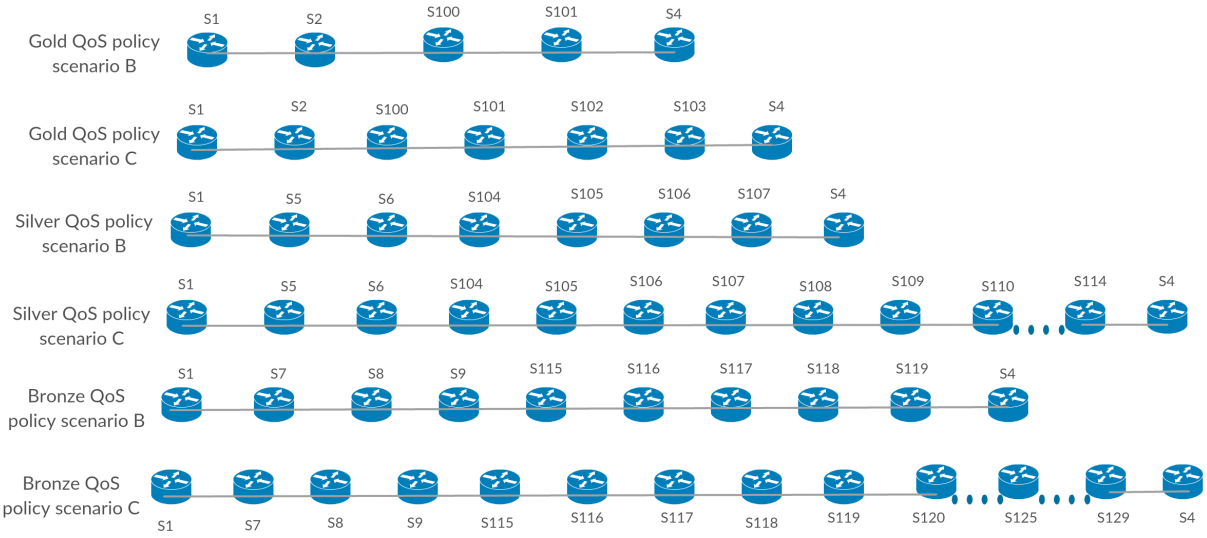

Figure 4.4 – Different QoS scenarios for policy deployment

Listing 4.12 – OpenFlow rules in the Switch S1 after the deployment of QoS Policy

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:8
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:5
```

Listing 4.13 – Implementation of the Redirection policy

```
1  def gold_qos(self,parser,dpid,out_port,match,actions,vlan_vid,dst_ip)
2  {
3   if (dpid == 2 and match):
4   actions = [parser.OFPActionPushVlan(0x8100), parser.OFPActionSetField(
      vlan_vid = (0x0001)),parser.OFPActionOutput(2)] // Redirect the flow
5   if (dpid == 4 and vlan_vid == 1):
6   actions = [parser.OFPActionOutput(2)]
7   if (dpid == 1 and dst_ip == '10.0.0.3'):
8   actions = [parser.OFPActionPopVlan(0x8100),parser.OFPActionOutput(5)]
9   return actions
10 }
```

4.4.3 Experimental Validation

Next, we evaluate the performance of our policy translation framework in implementing the different QoS policies. We measure the time needed to translate and deploy the high-level gold,

Table 4.3 – QoS Policies in different scenarios

Scenario	A	B	C
Gold QoS Policy	3	5	7
Silver QoS Policy	4	8	15
Bronze QoS Policy	5	10	20

silver and bronze QoS policies into low-level OpenFlow rules. The implementation time of these policies are evaluated according to the scenarios shown in Table 4.3. The implementation time of policies is the time needed to translate and deploy the high-level policies into low level OpenFlow rules. Implementation time is independent of the communication time between the customer controller and the ISP controller. However, the communication delay between the controller and the switch is included in the deployment time of low level OpenFlow rules from controller to the switch. Fig. 4.4 shows the list of switches during different QoS scenarios.

The QoS scenario is deployed in the mininet with the sample network topology shown in Fig. 4.3. We use IPERF [5] to generate the traffic in the scenario with varying rates.

Implementation time of QoS policies

It is worth noting that the implementation time of Gold QoS policy is less as compared to silver and bronze QoS policies. Since the implementation time considers the deployment of policy on the number of switches. As shown in Table 4.3 the path with the less number of hops is provisioned for the gold QoS policy. It is apparent from Fig. 4.5 that the increase in the number of elements in the topology is reflected as an increase in the time for calculation and deployment of the low-level OpenFlow rules. The implementation time for the gold QoS policy is high in the scenario *C* shown in Fig. 4.4 as it has the more number of switches as compared to scenario *A* and *B*. The implementation time reported in Fig. 4.6 for the silver QoS policy also follows the same increasing trend with the increase in the number of switches in the topology. As can be seen in Fig. 4.6 deployment time in scenario *C* is around 28 millisecond, which is reasonable considering the number of switches as shown in Fig. 4.4 for the scenario *C* of silver QoS policy deployment.

In the similar way, the deployment time of bronze QoS policy increases with the increase in the number of switches in the path. It takes around 36 milliseconds to deploy the low-level rules on the bronze path in scenario *C* as it has 20 switches shown in Fig. 4.4. Considering the number of switches in the path, the deployment time of policy in the bronze path is still not very high it is still in the order of few milliseconds.

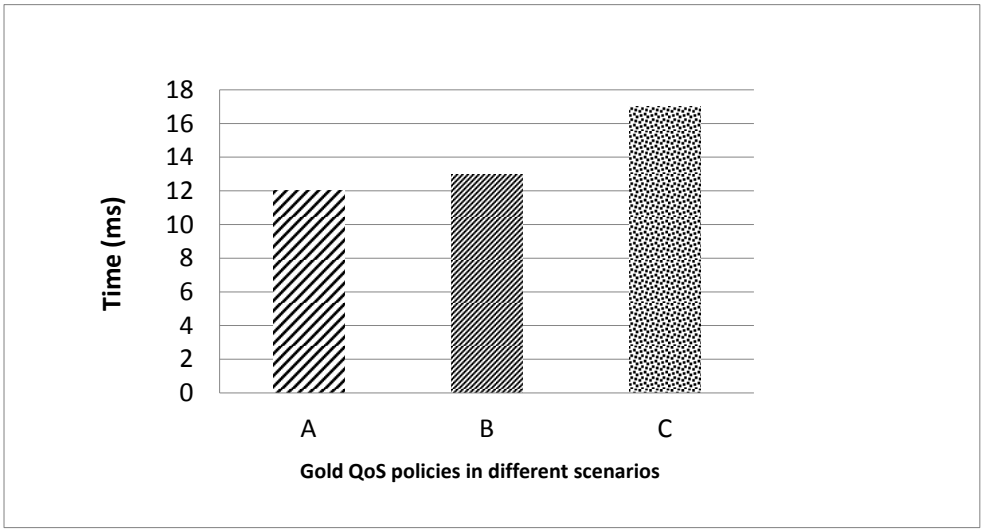

Figure 4.5 – Gold QoS policy implementation time with different scenarios

Packet loss during the deployment of OpenFlow rules in the switches

We measured the packet loss during the deployment of low-level rules according to our mechanism. As shown in Fig. 4.8, packet loss still follows the increasing trend. But, with our mechanism we minimize the packet loss to a large extent. It can be seen in Fig. 4.8 that there is no packet loss when the traffic rate is 1 Mbps. Even at the 15 Mbps of traffic rate packet loss is around 7.2 percent as compared to 37 percent in case of without our mechanism. It shows that our mechanism greatly reduces the packet loss percentage while deploying the low-level rules in the OpenFlow switches.

The main reason for the low packet loss is that first we deploy the low-level rules in the core switches. After that low level rules are modified at the ingress and egress switch respectively. So, when the rules are being deployed in the core switches flow traverses the previous path and it reduces the packet loss to a significant level. In this case, the packet loss occurs only because of the delay in deployment from the controller to the border switches.

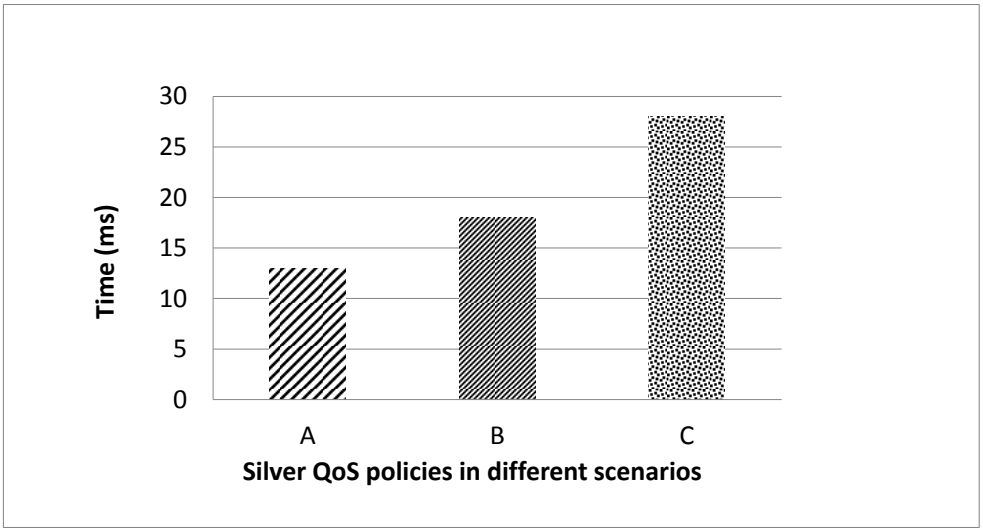

Figure 4.6 – Silver QoS policy implementation time with different scenarios

4.5 Scenario 2: On-demand DDoS attack mitigation

An important feature of our policy translation framework is that it provides flexibility to the ISP to automatically react on security events. In this section, we present in details how the DDoS attack mitigation is achieved in the network with out policy management.

4.5.1 Scenario description of attack traffic mitigation

We illustrate our use case with sample network shown in Fig. 4.3. Upon detecting the DDoS attacks, the customer sends the security alert to the ISP. We assume that, the customer uses some detection mechanism to detect the attacks [41]. Intrusion Detection Message Exchange Format (IDMEF) [44] is used by the customer to send the security alert. For instance, the customer sends the IDMEF alert message indicating that a tuplet source IP, destination IP, flow_class,impact severity requires mitigation. Details of all the steps involved in activating and deployment of the mitigation policy are discussed in the Sect. 4.5.2.

The input data contains the security alerts received from the customer and the policies at the ISP controller to deploy the rules in the network devices.

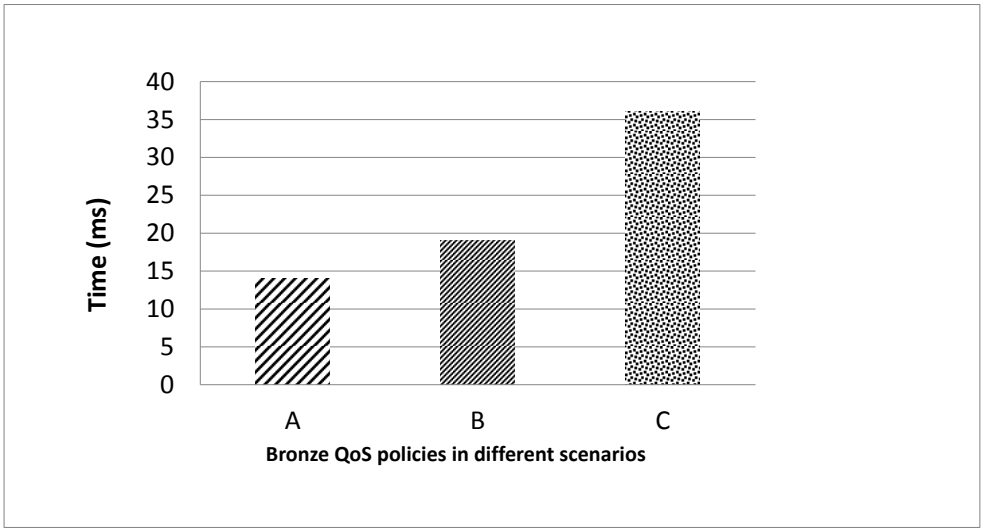

Figure 4.7 – Bronze QoS policy implementation time with different scenarios

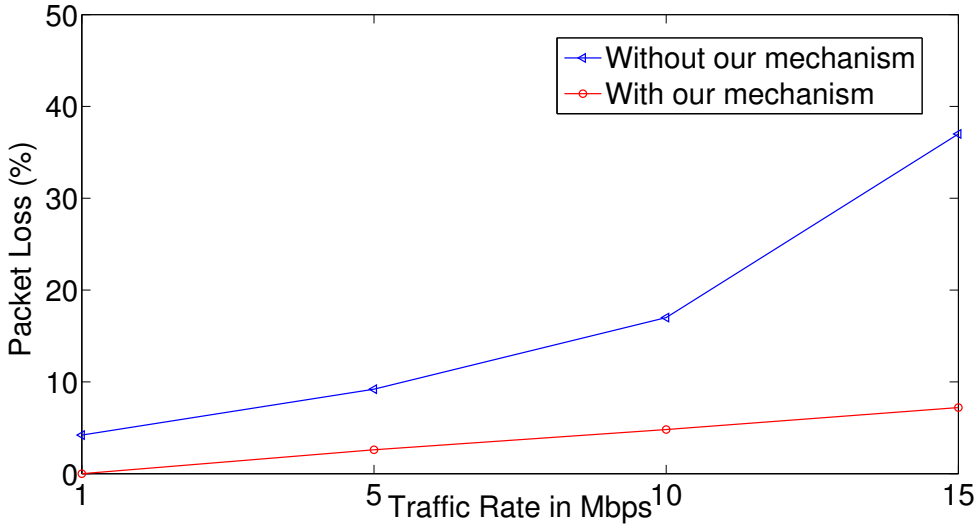

Figure 4.8 – Packet loss in the deployment of policy on the ingress and egress switch at last

Security alert received from customer: Security alerts are used to report about the attack traffic. Referring to the alert shown in Listing 4.14 contains the network and assessment attributes.

- The network attribute contains the IP addresses and attack type. The security alert contains network informations such as: source IP: 10.0.0.2, destination IP:10.0.0.3, attack type: ICMP Flood.
- The assessment attributes describe the effect of the attack on the network. For instance, a low impact severity, as shown in the alert represents that the congestion level in the customer network is 70 percent. Moreover, the security alert mentions the flow class as suspicious.

DDoS attack mitigation policy at the ISP controller: It contains the policy to process the received security alerts. The example policy shown in Listing 4.15 provides high level action "Redirect" for the ICMP flood traffic with a low impact severity.

Listing 4.14 – Security alert sent by the customer

```
<IDMEF-Message version="1.0">
<Alert>
    <Analyzer analyzerid="CUSTOMER_C"/>
    <Source>
       <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.2</address>
            </Address>
        </Node>
        <Service>
            <protocol>ICMP</protocol>
        </Service>
    </Source>
    <Target>
        <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.3</address>
            </Address>
        </Node>
    </Target>
   <Classification event="ICMP_FLOOD">
   </Classification>
   <Assessment>
       <Impact severity="Low"/>
    </Assessment>
    <AdditionalData> type ="string" meaning="flow class">
    <string>Suspicious</string>
    </AdditionalData>
</Alert>
</IDMEF-Message>
```

Table 4.4 – A Mapping Table between Bandwidths, Customer Network and the Path

Path	Bandwidth	Destination Network
P_4	Low	10.0.0.3
P_5	Medium	10.0.0.3
P_6	High	10.0.0.3

Listing 4.15 – A high-level Policy to Redirect Suspicious Traffic in the ISP Network

```
<Policy PolicyID="Mitigation">
    <Event Type = "ICMP_Flood">
    </Event>
    <Condition>
       <flow class="suspicious"/>
       <Impact severity="Low"/>
    </Condition>
    <Actions action="Redirect"/>
    </Actions>
</Policy>
```

Instantiation of the template to redirect the suspicious traffic: It contains the information about redirecting the suspicious traffic through the low bandwidth path or through the middleboxes. Table 4.4 provides the path to route the flow depending on the impact severity mentioned in the security alert by the customer. Furthermore, Listing 4.16 provides the concrete path details including the switch ID and output port to steer the flow.

Listing 4.16 – Instantiation of the template for redirecting the traffic through low bandwidth paths

```
P4:{Switch:S1,output(8);Switch:S10,output(2),Switch:S11,output(2);Switch:S12,
  output(2); Switch:S13,output(2);...;Switch:S30,output(2); Switch:S4,output
  (2)}
P5:{Switch:S1,output(9);Switch:S31,output(2);Switch:S32,output(2); Switch:S33
  ,output(2);Switch:S34,output(2);..;Switch:S60,output(2);Switch:S4,output(2)
  }
P6:{Switch:S1, output(10); Switch:S61,output(2); Switch:S62,output(2);Switch:
  S63,output(2); Switch:S64,output(2);Switch:S65,output(2);
Switch:S66,output(2);Switch:S67,output(2);..;Switch:S99,output(2);Switch:S4,
  output(2)}
Path={5:P5, 6:P6, 7:P7}
```

4.5.2 Step-by-Step Example

In this section, we describe a step-by-step example of how a high-level mitigation policy shown in Listing 4.15 is transformed into low-level rules for enforcement. This use case is described using

the sample network shown in Fig. 4.3. Specifically, we explain the scenario with the customer C sending the security alert to the ISP. In this scenario, host H_2 sends DDoS traffic towards the customer C. First, all the flows from H_1 and H_2 are traversing the network through the path P_1. As shown in Listing 4.17, the rule in the switch S_1, before the deployment of mitigation policy, forward the flow through an output port 5 by inserting a label 1 in the packet towards the path P_1.

Now, the steps are described below:

1. Customer C sends a security alert to the ISP for processing the ICMP flood traffic coming from the host H_2. The corresponding security alert is shown in Listing 4.14.
2. Extracted alert informations are forwarded by the Monitoring component to the PDP. Informations received by the PDP contains: event type (ICMP Flood), conditions: flow class (suspicious), and Impact severity (Low). PDP uses the policy ID which is "Mitigation" along with the event and conditions to instantiate the appropriate policy which is shown in Listing 4.15. The high-level action in this case is "Redirect".
3. The high-level action "Redirect" is further refined to get the path using Table 4.4. PDP sends the flow and bandwidth detail (Low) to the the NIB to get the concrete path detail. Then, NIB computes the path based on the inputs provided by the PDP. Note that, in this case the path is P_4.
4. Template shown in the Listing 4.16 is used to get the concrete path details depending on the high-level path information which is P_4. In this scenario, the path for the traffic from 10.0.0.2 to 10.0.0.3 is: (S_1:8, $S_1$0:2,$S_1$1:2,$S_1$2:2,$S_1$3:2,S_4:2). Earlier, flow from 10.0.0.2 to 10.0.0.3 was traversing through the path P_1.
5. The high-level action "Redirect" along with the bandwidth information is used by the PDP to instantiate the script in POI shown in Listing 4.19. The path details along with the flow informations as a match field are used by the script to deploy the rules to process the flow with low impact severity. First the rules are deployed in the core switches. Then, at the egress switch the rules are deployed. Finally, at the ingress switch flow informations are matched and a label is provided which is 4 in this case. At the egress switch S_4,label is popped and based on the destination IP address flow is forwarded. Listing 4.18 shows the low-level rule in the switch S_1 to forward the flow of source IP 10.0.0.2 and destination IP 10.0.0.3 by inserting a label 4 and through an output port 8. So that flow can traverse through path P_4.

Listing 4.17 – OpenFlow rules in the Switch S1 before the deployment of Mitigation Policy

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:5
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:5
```

Table 4.5 – Mitigation policies in different scenarios

Scenario	Number of switches in the path
Policy for low impact severity flows	20
Policy for medium impact severity flows	30
Policy for high impact severity flows	40

Listing 4.18 – OpenFlow rules in the Switch S1 after the deployment of Mitigation Policy

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:4->
    vlan_vid,output:8
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:5
```

Listing 4.19 – Script to redirect the Suspicious Traffic

```
1  def redirect_suspicious(self,actions,parser,dpid,dst_ip,datapath, match,
     vlan_vid):
2          if ((dpid == 10 and vlan_vid == '5')):
3              actions = [parser.OFPActionOutput(3-in_port)]
4          if ((dpid == 11 and vlan_vid == '5')):
5              actions = [parser.OFPActionOutput(3-in_port)]
6          if ((dpid == 12 and vlan_vid == '5')):
7              actions = [parser.OFPActionOutput(3-in_port)]
8          if ((dpid == 4 and dst == '10.0.0.3')):
9              actions =[parser.OFPActionPopVlan(0x8100),parser.OFPActionOutput
                 (2)]
10         if ((dpid == 4 and dst == '10.0.0.2' and src == '10.0.0.3)):
11             actions = [parser.OFPActionOutput(6)]
12             if ((dpid == 1 and match)):
13             actions = [parser.OFPActionPushVlan(0x8100), parser.
                 OFPActionSetField(vlan_vid = (0x0005)), parser.OFPActionOutput
                 (8)]
14         if ((dpid == 1 and dst == '10.0.0.2' and src == '10.0.0.3)):
15             actions =[parser.OFPActionPopVlan(0x8100),parser.OFPActionOutput
                 (2)]
16         return actions
```

4.5.3 Experimental validation of mitigation policies

We use mininet to deploy the attack mitigation scenario with the sample network shown in Fig. 4.3. Moreover, we use IPERF [5] to generate the traffic with different intensities. In this section, deployment time of different types of attack mitigation policies are discussed to process the suspicious flows. Table 4.5 shows the number of switches in the path provisioned for the different types of suspicious traffic. Suspicious flows based on their impact severities on the

destination network are routed through the low bandwidth path.

Implementation time of mitigation policy

We evaluated the time needed to translate the high-level mitigation policy into low level OpenFlow rules for the deployment. It also include the communication delay between the controller and the switch. We choose implementation time as a metric since it helps to evaluate how effective is our policy translation mechanism in translating the high-level policy into low level rules for the deployment. From the Fig. 4.9 we can see that the implementation time to deploy the policy to handle suspicious flow with high impact severity is close to 75 millisecond. It is significantly higher than the implementation time of other mitigation policies. Since, path with the highest number of hops is provisioned for the security policy processing the high impact severity flows. If we consider the number of switches in the path to process the high impact severity flows then it is still reasonable in between 75 to 80 millisecond. The implementation time to deploy the policy to handle low and medium impact severity flows are around 37 and 55 millisecond respectively.

Moreover, it is important to note that as shown in Fig. 4.9, the implementation time to deploy the low level rules to block the malicious traffic is very fast. Interestingly, the deployment time to block a flow is significantly lower than deployment time of other policies. It is because of the fact that the block action is only enforced at the ingress switch of the ISP network and no further enforcement is required. SO, it takes very less time for the deployment of the policy to drop the flow.

We conclude that the policies implementation time for our evaluation work well in reasonable time, but it can be further reduced if the number of devices for policy deployment can be reduced. This can be done with pre-installed rules in some devices in the network. For instance, as in the case of security policy with high impact severity, if the policies are pre-deployed in the core switches from S17 to S21 then only at the border switches $S1$ and $S4$ policies need to be deployed dynamically which can reduce the overall implementation time.

In one of the experimentation, we also evaluated the implementation time of the policy to handle suspicious flows with different traffic rates. Fig. 4.10, shows that the deployment time of the low level rules corresponding to the high-level policy to process the suspicious flows of high impact severity is same. Experimentation was run with varying traffic rates from 1 Mbps to 15 Mbps. In all the scenarios, we found that the time to translate and deploy the high-level policy into low level rules are same after the security alert is received by the ISP. In all the case, it is around 75 millisecond. It shows that the traffic rate does not affect our policy translation process and the deployment of corresponding low-level rules in the network.

Packets by-passed the Drop Action

Filtering of the malicious flows at the border router of the ISP network is better for their customers, as it reduces the impact on the incoming legitimate traffic to the customer network traversing

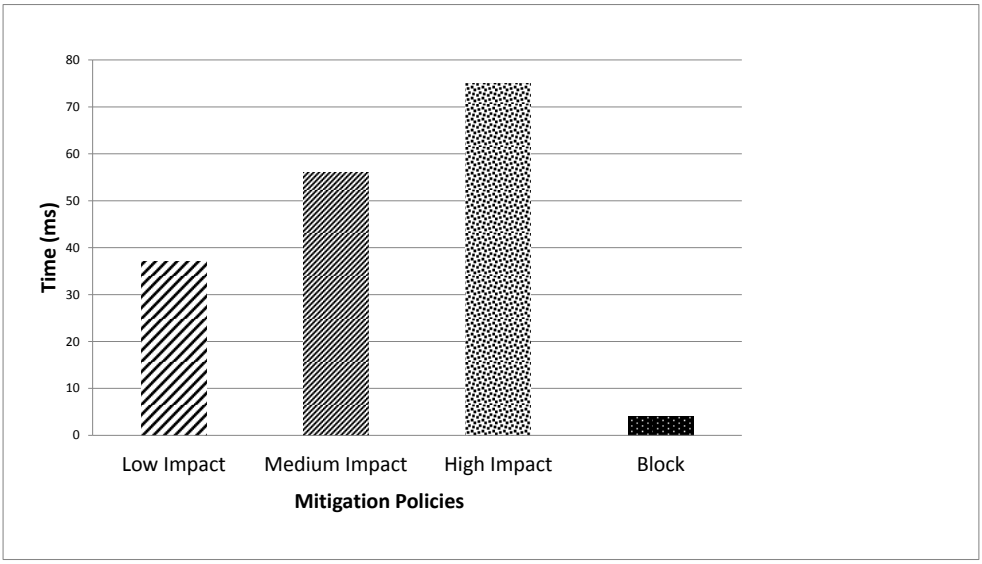

Figure 4.9 – Implementation time of mitigation policies

through the ISP network. Therefore, we measured the number of packets that by-passed the ingress switch while processing the alert to drop the malicious flows. Number of packets that cross the ingress switch with varied traffic rates were evaluated. This metric helps to analyze the effectiveness of our policy translation mechanism as it measures how many packets reach the destination while deploying the OpenFlow rule to block the malicious flow. As can be seen in Fig. 4.11, when the traffic rate is 1 Mbps then around 18 packets crossed the ingress switch and reached the customer network. It is worth noting that the number of packets that crossed the ingress switch and reached the customer network increases with the increase in the traffic rate. As shown in Fig. 4.11, when the traffic rate is 5 Mbps then close to 125 packets reached the customer network which was under attack. There is a sharp increase in the number of packets that reached the customer network when the traffic rate increased from 10 to 15 Mbps. It occurs because of the minimum delay in processing the security alert and the deployment low-level rules to drop the malicious flows.

4.6 Features and Comparison with Existing Policy Language

In this section, first we discuss the features of the policy language and then we compare our policy language with the existing policy languages.

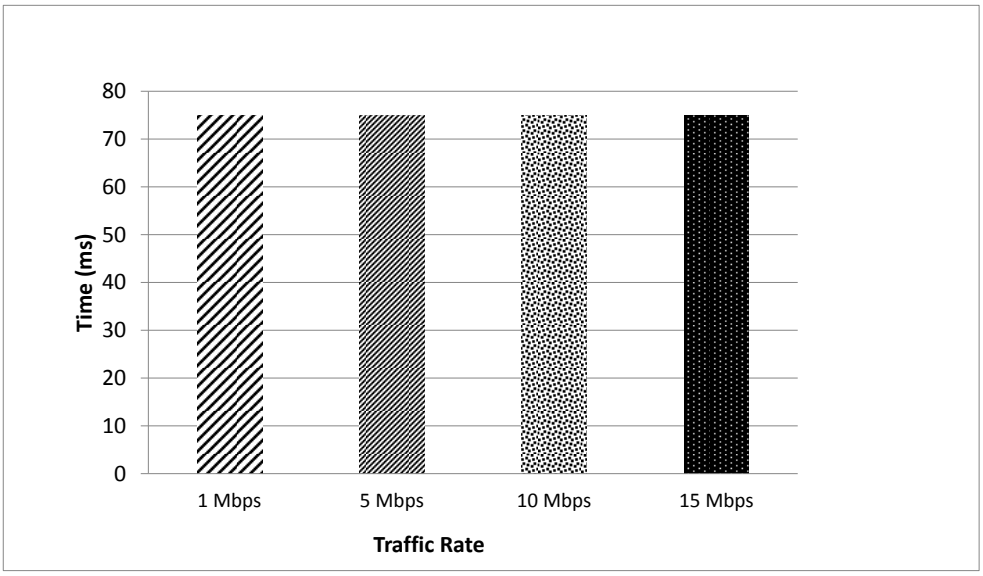

Figure 4.10 – Independent of the traffic rate time required to implement the mitigation policy for high impact severity flows

4.6.1 Features of the Policy Language

The main characteristics of our policy language is described below:

- Index: Index is a special item which helps the PDP in identifying and efficiently instantiate the policy in the policy database. PolicyID is used in our policy language to identify the specific policies such as QoS or mitigation policies.
- Rich Set of Actions: It allows the expression of a diverse set of actions to be enforced by the policies depending on the specified conditions.
- Controlled Natural Language (CNL): The advantage of using CNL is that it is more readable than typical computer language. Moreover, network operator do not need to be familiar with the low level syntax to express the policies.
- Policy Definition: Generally, a policy is defined as a set of rules. It is categorized in two groups static and dynamic. A fixed set of actions are applied according to the pre-defined parameters in static policies. However, dynamic policies are applied depending on the changing conditions such as congestion in the network.

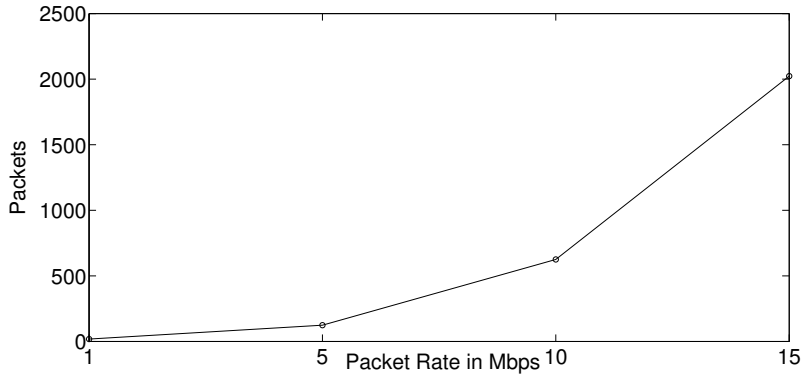

Figure 4.11 – Number of Packets that bypass during the implementation of block action at the ingress switch

- ECA: Our policy language follows the ECA paradigm to specify the network and security policies. As ECA paradigm can handle the composite events, so it is a good choice to design the policy language.

- Modular Composition: It enables the management of the complexity of the policies. Policies are decomposed into distinct modules with precisely specified interactions. Distinct modules are also necessary for maintainability.

- Security and QoS constructs: It allows the network operators to express the high level goals for traffic engineering and security policy enforcement if the specified conditions are met.

- XML: XML is a known international format to define the policies which is expressive and extensible. Since our policy language is encoded into the XML format, it provides usability and allows the interoperability of the policies on different system.

- Expressibility: Our policy language enables network operators to express different attack and congestion scenarios based on the impact severity of the traffic. It also allows network administrator to specify different levels of QoS services which can be triggered based on the requirement. Moreover, one can define new security and network goals with the existing terms of the grammar without the need of another grammar.

- Formalization: This feature in our policy language enables the network operator to leverage the grammar to define the network and security policies for different scenarios.

- Obligation: It specifies that once an Event occurs in the network, policy language should instantiate the task which must be executed. As explained in the QoS and mitigation policy scenario, our policy language enforces the Action once some Event is triggered in the network.

4.6.2 Comparison with Existing Policy Languages

Table 4.6 shows the comparison of our policy language with the Procera [115], OpenSec [70] and Merlin [105]. Procera and Merlin both provide a high-level language to express the policies in the network. While OpenSec proposed a policy specification language to divert the traffic to middleboxes for processing. Procera uses a complex syntax which makes it complicated and difficult to understand. It does not use simple plain language (CNL) to define the policies which makes it complicated and difficult to comprehend. Moreover, to define an event in Procera, it needs to be registered with its global data structure. Procera does not provide unique policy ID to retrieve the policy. It triggers the policy based on the events. It provides a rich set of actions such as allow, deny, rate limit and redirect host. "Allow" forwards the flow or provide access to a resource in the network to a user. "Deny" blocks the access of a network resource for a user or block the traffic. "Rate limit" action sets a threshold to throttling a flow. "Redirect host" specifies that a flow should be redirected to a particular host machine. Policy definition in Procera is dynamic i.e. policies are enforced depending on the varying conditions. Furthermore, it allows policies to be modular and event driven which provide dynamicity and flexibility to modify the high level policies. However, it does not provide constructs to define the security and QoS constraints in the network. The main focus of Procera is to program the network using high-level defined policies.

OpenSec [70] also introduce a policy specification language. Policies are directly defined on the flow which makes it scenario specific. It relies on the OpenFlow syntax to specify the policies, which require network operators to be familiar with the OpenFlow concepts. Moreover, it does not provide a rich construct of actions, only alert and block actions are specified. Alert action reroute a flow to middleboxes for processing. Block is enforced on the malicious flow. OpenFlow match field is used as conditions to specify the high-level action for enforcement. Furthermore, OpenSec does not provide an index to identify the policy for efficient retrieval. Policies in OpenSec are dynamic in nature and are triggered based on the events in the network. Moreover, it provides constructs to specify the security policies in the network. However, it does not provide modular composition while specifying the policies in the network. The main focus of OpenSec is to route the traffic through middleboxes for processing.

Merlin [105] enables the definition of high-level policies using programs in a declarative language. The language contains predicates to identify the packets and regular expression to encode the paths. It uses the packet header fields to classify and express the policies for the set of packets. An unique identifier is not specified for policy retrieval. Moreover, Merlin does not provide modular composition to express the high-level policies in the network. It allows the network operators to express the policies for guaranteed QoS in the network for the specific set of packets. Moreover, it allows the policies to be enforced based on the changing conditions in the network which makes the policies in Merlin dynamic in nature. However, Merlin policies are based on the condition-action paradigm. The main focus of Merlin is to provide the bandwidth guarantee to specific flows. Therefore, it does not provide diverse set of actions, mainly focus on the forward action.

Table 4.6 – Comparison with Procera, OpenSec and Merlin

	Procera	OpenSec	Merlin	Proposed Language
Syntax	Proc world ->do return A: policy req -> if flow_class=legitimate and bandwidth_class=gold and event already exist then allow	Flow: VLAN=vlan_id; Service: Gold_path; React: alert	(ip.src=10.0.0.1 and ip.dst=10.0.0.0.3) ->.*	Event=QoS_message Conditions: Flow class= legitimate Bandwidth class = Gold Action: Redirect
Index	No unique ID is used	Unique ID is not used	No unique ID is used	Unique ID is used to identify the policy in database
Action	allow; deny, redirect host, rateLimit rate	React, Block	forward	Forward, Redirect, Drop
Controlled Natural Language (CNL)	Do not use CNL to specify policy	Rely on OpenFlow syntax to define Policies	CNL is not used	CNL is used to define policy
Policy Definition	Dynamic	Dynamic	Dynamic	Dynamic
Event Driven	Yes	Yes	No	Yes
Modular Composition	Yes	No	No	Yes
Security and QoS	No	Security	QoS Guarantee	Both
Formalization	Yes	No	Yes	Yes
Obligation	Yes	Yes	Yes	Yes

However, our policy language uses CNL to define the high-level policy in the human readable format which makes it easy to understand. Network administrators do not need to be familiar with special programming syntax or OpenFlow concepts to define the high level policy language. It uses Event-Condition-Action format to define the high level policies. Moreover, it uses a unique ID to efficiently retrieve the policies from the policy repository. Furthermore, our policy language provides a formal grammar which can be leveraged by the network operator to define the policies for different scenarios. It provides syntax to define the network and security policies as well. Additionally, three high-level actions are specified by our policy language: Forward, Redirect and Drop.

4.6.3 Usability of the Policy Language

Our policy language allows to express the high-level policies in controlled natural language. Network administrator is not require to learn device specific syntax to specify the high-level policies. It makes the policies easy to write and understand. Moreover, event driven mechanism in our policy language enables to trigger the policies based on asynchronous notification. It is better in comparison to condition-action paradigm. Since in condition-action paradigm policy needs to check the condition all the time to enforce the action. Our policies are not hard coded in the data plane devices. When an event occurs it triggers the policy and then the high-level policy is translated into low-level rules for the deployment.

Moreover, our policy language provides modular composition. It enables to clearly define the interaction among the different modules of the policies and reduces the complexity in managing the high-level policies. Furthermore, our policy language provides the flexibility to specify different security and QoS scenarios.

Additionally, our policy language uses an unique ID to instantiate the policy. It makes easier and faster to trigger a specific policy from the list of policies. Moreover, apart from three high-level actions forward, redirect and drop other different types of action can be added in our policy language.

4.7 Related Work

There have been some works on the policy refinement mechanisms. The work whose motivation is close to ours are Zhao *et al* [128] and Craven *et al* [31]. These mechanisms proposed policy languages to translate the high-level policies into low-level rules. However, they are limited by the characteristics of the legacy networks such as the lack of centralized control plane, programmability, and interface to deploy the rules to switches and routers. Bandra et.al. [15] presented an approach to policy refinement based on the event calculus, used in conjunction with the abductive technique to know the sequence of steps which will enable the system to achieve desired goal. First, high-level goals are mapped to concrete goals which are system requirements. Then, the system requirements are mapped to specific modules available in the system. It uses UML state charts to describe the system. The relationship between system description and the high level goals can be implemented either as a policy or system functionality. It can not be automated. Moreover, refinement process mainly focus on QoS management in differentiated service network. Verma [113] presents a mechanism of policy translation for QoS management, which is based on the set of tables to identify the relationship between users, applications, servers, routers and classes of service supported by the network. When a new Service Level Specification (SLS) is triggered, the system performs a table lookup to identify the configuration for the specified user, application and class of service. This mechanism is automated, however it relies on the correctness of the table which requires expert administrator. Moreover, this mechanism supports only specific type of SLA and low-level device policies.

Thanks to the evolution of SDN technology, our policy translation mechanism is based-on SDN [90] which provides a centralized control point and an interface between control plane and data plane to deploy the rules in the switches and routers.

However, programming the network in the SDN is also not an easy task. It requires the network operators to be familiar with the OpenFlow programming and syntax for specifying the network and security policies. Our mechanism provides a syntax to represent the high-level security and QoS policies which are deployed depending on the events in the network.

There have been some languages proposed to program the behavior of the network. P4 [23] is a high-level language for programming the network. Its aim is to provide the functionality of parsing the packets independently of the underlying hardware. P4 mainly aims at modeling the forwarding behaviors through programmable routers. These languages specify the routing policies close to data plane, while we provide the high level policies for mitigation and QoS services.

Recently, there have been some proposals on defining high-level policy enforcement. Tripathy et.al [110] proposed a policy management system to check the consistency and resolve the conflicts before policy enforcement. It supports functions to verify the trust, resolve the conflicts and check the consistency before policy enforcement. The main idea of this framework is to support security, correctness and adaptability for on-demand modification in the policies. A policy refinement based on Service Level Agreement (SLA) approach is proposed in [76]. Business level goals are specified as SLA and these SLAs are translated into service level objectives (SLOs) and these SLOs are used to enforce the policies in the network devices. Differently from these works, we provide a language to specify the high-level policies and contexts for the security and QoS policies. Moreover, we provide different high-level policy templates to deploy the low-level rules based on the high-level action.

4.8 Conclusion

In this chapter, we have proposed a policy translation mechanism to translate the high-level network and security policies into low-level rules. We used the Controlled Natural Language (CNL) to propose the grammar for our high-level policies. Moreover, we define high-level policy templates which helps in enforcing different actions in the network devices. The result is a top down translation of mitigation and QoS policies into the low-level OF rules. The system has been validated with the two use cases:(1) QoS policy,(2) mitigation policy with multiple customers scenarios. The use case are considered basic but generic-they can be easily extended to take into consideration of more sophisticated requirements and policies. In the next chapter 5, we discuss the policy management framework at the ISP controller and evaluate the framework which computes the path and process the requests of the multiple customers of the ISP.

5 A New SDN based Security Policy Management and Enforcement Framework

One of the major objectives of traffic engineering in the ISP network is to mitigate the impact of traffic congestion on its customers, which can be caused, among others, by the attacks. However, simply prioritizing the legitimate traffic or redirecting the suspicious traffic for one customer, without considering the network status of the ISP, may impact other customers of the same ISP. Because the same path in the ISP network can be shared between different customers of the ISP. Therefore, it is necessary to consider the network status of the ISP to perform the traffic engineering for the customers.

Moreover, for effective traffic engineering collaboration is required between the ISP and its customers. Without collaboration with their ISPs, customers do not have much control over the incoming traffic, apart from dropping the attack traffic at their border routers. However, complexity involved in the network management tasks such as configuring switches or routers for dynamic policy enforcement makes the real time collaboration difficult.

Furthermore, to mitigate the impact of attack traffic, ISPs statically deploy middleboxes inline with network devices at the choke points of the network to process the incoming traffic. These middleboxes are generally distributed in the network through a separate VLANs, while network policy is usually applied per VLAN. This leads to static service chaining with the deployment of static policies for steering traffic [85]. Moreover, all the traffic, whether it needs to be processed through the middleboxes or not, eventually traverse these devices, which causes processing overhead on these devices. It also causes extra latency to the flows which need not to be processed by the middleboxes. These static deployment of middleboxes and security policies further complicate the network management tasks. Therefore, a dynamic and automated policy management system is required to overcome these issues.

The aim of this chapter is to provide a dynamic and adaptive policy management framework for the ISP network. The framework enables the ISP to perform the traffic engineering considering its network status. We benefit from the SDN technology and use our policy specification and translation mechanism introduced in chapter 4 to specify the high-level policies which are enforced based on the conditions in the network. Moreover, customers can dynamically express

their requirements to the ISP for the instantiation of the policies. In summary, the framework takes into account multiple factors like the current network status of the ISP and policy agreements with its customers to perform traffic engineering. We implement our proposed framework for a specific use case, where the ISP and their customers collaborate with each other to mitigate the effect of congestion caused by the attack traffic. We experimentally demonstrate that the framework can help an ISP to provide good QoS to legitimate traffic, while reducing the impact on other customers' traffic.

The remainder of this chapter is organized as follows: Section 5.1 discuss the requirements for the policy management and enforcement framework. In Section 5.2 we present our policy framework, its workflow and functional components. Section 5.3 describes the mechanism to compute the path for the policy enforcement. Section 5.4 provides the workflow of the policy management framework. Experiments and results are reported in Section 5.5. Section 5.6 provides some discussion on the previous works and finally Section 5.7 concludes the paper.

5.1 Requirements for the Policy Management and Enforcement Framework

- **Dynamic and Automated:** The policy framework must be dynamic and highly automated to free the network operators from manual and error prone process of policy enforcement. Only at the initial stage network administrator should define the high level policies manually.

- **Well-formed:** The policy should be well formed so that a unique policy can be chosen for a given event and associated conditions. This would enable the framework to be deterministic.

- **Simple and Expressive:** Policy management framework should be simple and expressive enough so that network operators can define their diverse intents. It should allow the network operators to specify the policies in a simple to understand language without knowing how these policies will be implemented.

- **Service Chaining:** Specifying correct processing order for a packet depending on the policies is important: For instance, a packet coming from external network should be first processed through a firewall and then it should traverse the core network, or a suspicious traffic should traverse through a specific set of switches or middleboxes in the network.

- **Traffic Engineering:** Framework should enable to execute the traffic engineering policies in the ISP network for an end user without impacting the traffic of other users to reduce the collateral damage.

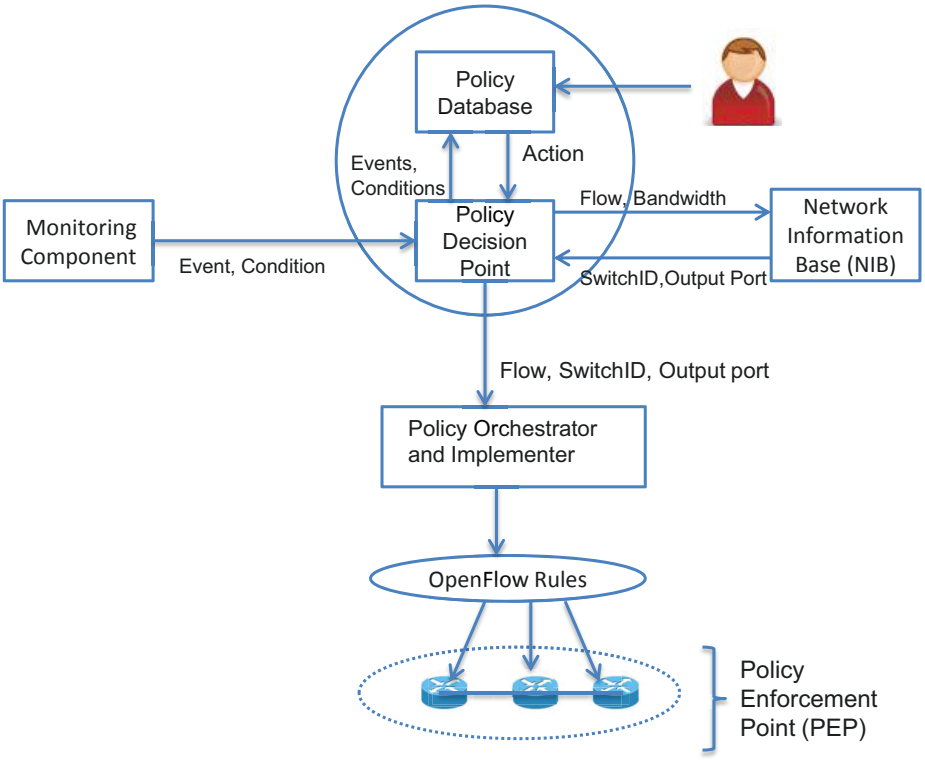

Figure 5.1 – Workflow of the Policy Management Framework

5.2 Design Components

In this section, we describe the components involved in the design of policy management and enforcement framework. Since policy database, policy decision point and POI have already been described in detail in chapter 4, i.e. how they enable to translate the high-level policies into low-level OpenFlow rules. Therefore, in this chapter we mainly describe about Monitoring Component (MC), Network Information Base (NIB) in more detail. The main focus of this chapter is to propose a policy-based management and enforcement system to manage the ISP network in an automated fashion. The functional components of the policy management framework are described as follows:

Monitoring Component (MC):

The MC is responsible for receiving alerts and notifications from the different customers. Particularly, it extracts the events and conditions which are used by the PDP to instantiate the policy in the policy database. The extracted events and conditions are: attack type (event), flow class

(suspicious, malicious, legitimate), impact severity (low, medium, high), and flow information (source and destination IP addresses).

Policy Database (Policy DB)

In the framework, policy database is used to store high-level policies defined by the network administrator in the ISP network. Policies are defined as event, condition and action paradigm.Please refer to chapter 4 for more detail on this component as it has already been described there.

Policy Decision Point (PDP)

This component works as an orchestrator between different modules in the framework. It is the centralized intelligence point in the framework. Details of this component has already been described in chapter 4.

Network Information Base (NIB)

NIB maintains a table containing the list of paths based on different bandwidths. It is responsible for computing the path depending on the network status of the ISP. It also considers the list of middleboxes to traverse, in order to steer suspicious flows in the network. It maintains a mapping table containing the list of middleboxes and their deployment location in the network. Depending on the inputs (flow informations, high-level action, and bandwidth requested) it computes the path. After computing the path, it provides the list of switches and output port along with Network Service Header (NSH) to the PDP. Paths are computed using the policy aware shortest path [27], which enables the traffic to be routed through a path based on the pre-defined policies. Details of the path computation algorithm is described in Section 5.3.

Moreover, this component is responsible to monitor the status of the switches and paths in the ISP network and provides the network status (congested, normal). Indeed, to monitor the ISP network, we can use a tool like OpenNetMon [112] that permits to maintain the traffic matrix for different paths and switches in the network.

Policy Orchestrator and Implementer (POI)

This module generates the OpenFlow rules based-on the match and action information provided by the PDP. It takes the flow informations (source and destination IP), path details (switchID:output port) and Network Service Header (NSH) from PDP to distribute the rules in the OpenFlow switches in the network. Description of POI module has been given in the chapter 4.

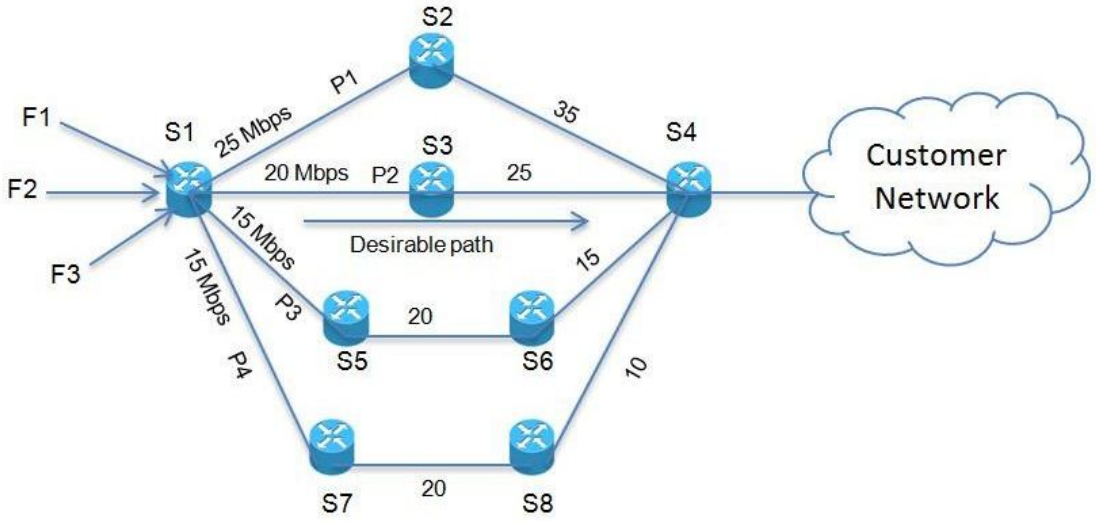

Figure 5.2 – An Example of Policy Aware Shortest Path

5.3 Path Computation and Network Service Header

This section presents the details of major functions in our policy enforcement framework such as path computation and Network Service Header (NSH).

5.3.1 Network Service Header (NSH)

Our policy management framework uses NSH to steer the traffic. In particular, we use VLAN ID (MPLS and VXLAN header can be used as well) to provide NSH in the packets, and it contains end-to-end path identifier (global segment) and middlebox or switch IDs (local segment). Both local and global segment are stacked in the NSH for forwarding, and these two labels are described as follows,

1. **Global Segment** helps to steer the flows from the ingress switch to the egress switch in the ISP network. Thanks to this label, core switches in the path can easily identify the output port through which a packet is to be forwarded. Flow states containing NSH is maintained only at ingress switch. It significantly reduces the flow table entries in the core switches [56].

2. **Local Segment** is used to identify the middlebox through which the traffic of interest needs to be processed. It avoids broadcasting the traffic towards the middleboxes for processing [78]. Moreover, middleboxes are not required to be deployed outside the main datapath in the network. Furthermore, it reduces the processing overhead on the middleboxes by avoiding the unnecessary processing of all the traffic in the path through the middleboxes, if it is not required.

It is worth noting that although inserting extra header may increase the size of packets, but the processing time for switches and middleboxes can be significantly saved. Moreover, NSH can reduce the number of flow table entries in the core switches, because its operations only happen at ingress switch of ISP. Moreover, ISPs can use software switches at its ingress point to overcome the limitations of flow table entries.

5.3.2 Path Computation

Algorithm 4 *Path_Computation*

```
1: procedure PATH_COMPUTE(flow,bw_req,action,step)
2:     hop ← 0
3:     F ← flow
4:     path[] ← get_paths(destinationIP) // Provide the list of paths to forward the traffic to destination
5:     Total_bandwidth ← (bw_req + bandwidth_occupied)
6:     if action is fwd or redirect and bw_req then
7:         for Flow F and each path do
8:             b ← compute_Bandwidth(min(all_links)) //Takes the minimum bandwidth in the path
9:             d ← Hop_Count(step[i] + step[i+1])
10:            hop_count ← hop + d // Computes the hop count in the path
11:            path.addList(p)
12:            if (b − Total_bandwidth) ≥ 0 then //New flow does not impact other flows traversing the
    link.
13:                return hop_count, path(p)
14:            else
15:                return No Paths are available
16:            end if
17:        end for
18:    end if
19:    if action is Fwd_Middlebox and flow F then
20:        middleboxes ← Fwd_Middlebox //Get the list of middleboxes to traverse
21:        path[] ← list_paths(destinationIP, middleboxes)
22:        for each path do
23:            b ← compute_Bandwidth(min(all_links)) //Takes the minimum bandwidth in the path
24:            hop_count ← hop + d // Computes the hop count in the path
25:            path.addList(p)
26:            if (b − Total_bandwidth) ≥ 0 then //New flow does not impact other flows traversing the
    link.
27:                return hop_count, path(p)
28:            else
29:                return No Paths are available
30:            end if
31:        end for
32:    end if
33: end procedure
```

Algorithm 4 takes into account the flow, bandwidth requested and action provided by the PDP for computing the path. Flow informations specially destination IP helps in identifying the egress points for computing the path. Action specifies whether to forward, redirect or route the flow towards middleboxes. The bandwidth to be provided to the flow is specified in the variable

`bw_req`.

For instance, PDP may specify that the flow needs to be steered through firewall and NAT devices, then NIB module identifies the paths which have firewall and NAT devices and whether they are connected to the destination network mentioned in the flow or not. After computing the path, NIB returns the path details to the PDP.

For better illustration, an example is shown in Fig. 5.2, showing how the paths are computed. For a flow F, in order to obtain a bandwidth b, one constraint that must be satisfied is that the maximum rate of the traffic from other flows in the shared link should be under $(b - Total\_bandwidth)$, where b is the minimum link capacity in a path. In other words, bandwidth requirement of a flow should not impact other flows which are traversing through the network.

As shown in Fig. 5.2, paths (P_1, P_2) are provisioned for high QoS, while the paths P_3 and P_4 are of low bandwidth. Moreover, the path P_1 is shared by the flow F_1 and F_2. While the flow F_3 traverses through the path P_2. Traffic rates of F_1, F_2 and F_3 are 10, 15 and 20 Mbps respectively. Flow F_2 is legitimate and needs to be redirected to another path for better QoS.

In this example, NIB gets the status of the path P_2 as congested since flow F_3 of 20 Mbps is traversing through P_2. Moreover, the capacity of the link between S_1 and S_3 is 20 Mbps. So, the path P_2 can not be used for redirecting the flow F_2 as it will negatively impact both the traffic F_2 and F_3. Therefore, the available paths for computation are P_3 and P_4. The result after running Algorithm 4, indicates that the shortest path with the lowest hop count is path P_3. Thus, flow F_2 is redirected to path P_3 rather than P_4 (which has the same hop count as that of P_3 but has lower bandwidth capacity between switch S_8 and S_4).

Moreover, the Algorithm 4, also computes the path when a flow needs to be steered through the sequence of middleboxes. In this scenario, the action variable contains the list of middleboxes through which flow is to be routed. Algorithm 4 checks if the action is `Fwd_Middlebox` then it computes the path for the flow considering the list of middleboxes which needs to be traversed. Variable `Fwd_Middlebox` provides the list of middleboxes through which flow is to be routed. Moreover, algorithm ensures that the flow F which needs to be forwarded through the path containing middleboxes should not impact other flows traversing the path negatively. In other words, bandwidth should be under (b - Total_bandwidth).

5.4 Workflow of the Framework

The design overview of our framework is shown in Fig. 5.1, consisting of several functional components as described in Section 5.2. As most of the operations are carried out within the ISP domain, the customer network is not shown here. To better illustrate the specific functions of the different components, the operational workflow is given as follows,

1. An event is triggered at the ISP controller when a security alert is received by the

`Monitoring component` from the customer controller. Flow informations (source IP, destination IP), impact severity of the traffic on the customer network (low, medium, high), flow class (legitimate, suspicious, and malicious) and attack type details are extracted by the MC module [54]. The extracted informations are forwarded to the PDP [55].

2. PDP selects the high-level action from the policy database to be applied on the flow based on the event and its corresponding conditions received from MC. Then, it forwards the high-level action, bandwidth requested, and flow informations to the NIB.

3. The NIB computes one or multiple best-fitted paths (more details will be given in Section 5.3.) given flow informations (i.e., it allows to identify the ingress and egress nodes within the ISP networks), bandwidth requested, and the high-level action.

4. NIB forwards the path details along with switch ID, output port and Network Service Header (NSH) (e.g., VLAN ID) [95] to the PDP. PDP forwards these details to the POI module for the deployment of the rules in the OpenFlow switches. If all the paths in the ISP network are congested then NIB returns message stating "no paths are available" to the PDP. In this case, PDP forwards *drop* action for the flow of concern to the POI to reduce the congestion level in the network.

5. Moreover, PDP instantiates the script in the POI based on the bandwidth requested by the customer. Then, script in the POI module deploys the OpenFlow rules in the switches.

5.5 Experiments

We consider the ISP network which provides the traffic engineering to its customers to mitigate the impact of network congestion. The aim of our experiments is to show, in a multiple-customer scenario, the effectiveness of our proposed policy management and enforcement system on mitigating the impact of traffic congestion and collateral damage in the presence of DDoS attacks.

5.5.1 Experiment Settings

The prototype our policy enforcement framework is implemented in Python and run as an OpenFlow application on Ryu SDN controller. The experiments were carried out in Mininet [69], which provides prototyping environment for the OpenFlow switches. The experimental scenario is shown in Fig. 5.3, in which the ISP network contains 14 OpenFlow switches and 6 paths. In particular, the paths noted with P_1, P_2, and P_3 are configured with higher bandwidth as compared to the paths P_4, P_5 and P_6. The bandwidth and link loss probability of different paths are shown in Table 5.2. In addition, each customer network has one OpenFlow switch attached to its SDN controller, which communicates with the SDN controller of the ISP network using the REST API. Four hosts are configured to send the traffic to the customers C_1, C_2 and C_3.

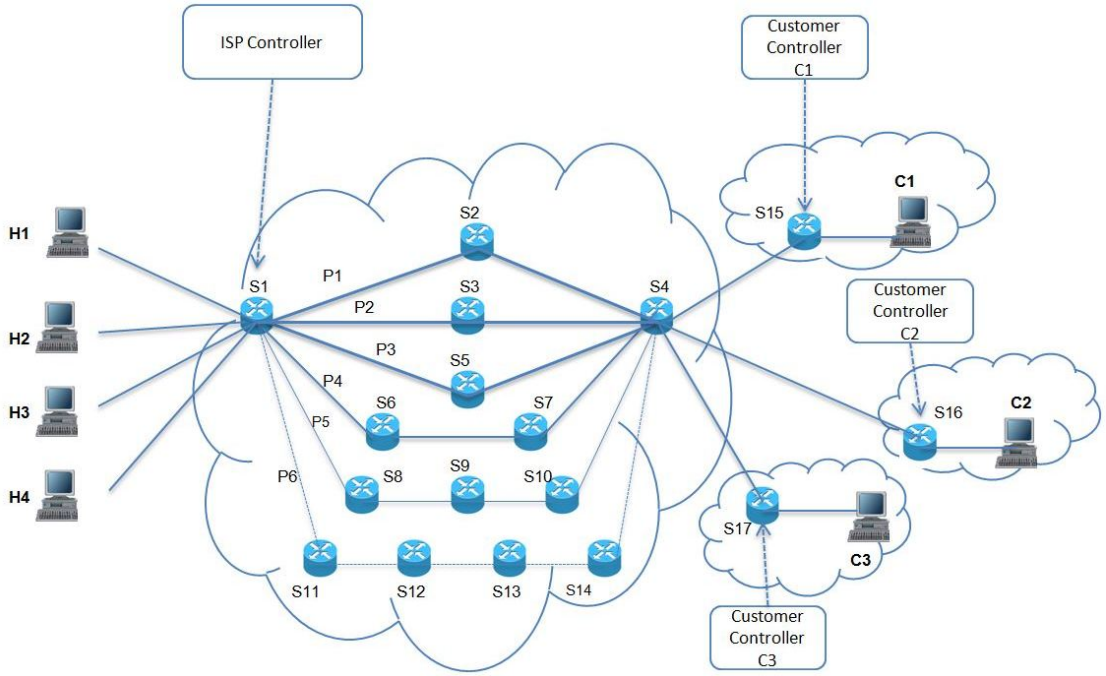

Figure 5.3 – Experimental scenario for three customer networks with one ISP network.

Table 5.1 – Traffic its flow class and Destination Network

Flow	Flow Class	Destination
H_1	Legitimate	C_1
H_2	Suspicious	C_1
H_3	Legitimate	C_2
H_4	Legitimate	C_3

5.5.2 Example Scenario

Host H_2 sends UDP flood traffic at the rate of 400 Mbps towards customer C_1 with an aim to congest the network. Hosts H_1, H_3 and H_4 sends traffic at low rate to the customer networks C_1, C_2 and C_3 respectively. Table 5.1 shows the traffic from its source to the destination network with its flow class. When network congestion occurs, C_1, C_2 and C_3 send the alerts to the ISP requesting for better QoS for their legitimate traffic. In the scenario, ISP receives a security alert in IDMEF format [44], from customer C_3 which requests to redirect the legitimate traffic originating from H_4.

The alert in the IDMEF format provides a common exchange format between the ISP and its customers. Specifically, `Analyzer ID` specifies the ID of the sender, which is the customer network in our case. `Source` and `Target addresses` specify the IP addresses of the source and destination networks respectively. The field `Service` contains the specific information about

Table 5.2 – Traffic paths in terms of bandwidth and link loss probability

Paths	Bandwidth	Link Loss Percentage
P_1	400 Mbps	0
P_2	400 Mbps	0
P_3	400 Mbps	0
P_4	200 Mbps	3
P_5	100 Mbps	5
P_6	50 Mbps	7

the traffic, e.g., protocols. `Assessment` field specifies the impact severity of the traffic on the customer network, which has either of the following values: low, medium and high. In our case, low impact severity means that the customer network is stable and congestion level is 60%, medium and high impact severity means congestion level is 70% and more than 80% respectively. Field `AdditionalData` provides flexibility to extend the standard IDMEF alert with more information. For example, we use this field to accommodate *flow class* to denote the class of traffic.

Upon receiving an alert, MC module extracts the information of interest and forward it to PDP as explained in Section 5.2. In the scenario, alert from C_3 contains flow class (legitimate), Flow (source IP, destination IP), impact severity (low) and traffic type (UDP). With these inputs, PDP gets high-level action (redirect) from policy database as shown in Listing 5.2. The Listing 5.2 shows a sample policy used in our experimentation to address UDP flood attacks. Flows identified as suspicious are diverted to other paths in the network if they have a low impact on the network. Likewise, in Listing 5.3, a sample policy to provide better QoS to legitimate traffic is shown.

Then, PDP forwards the high-level action, flow information and flow class to NIB for computing the path. In this scenario, NIB provides the path P_3 for redirecting the flows for C_3. Then *POI* inserts a global segment in the packets from H_4 to C_3 for redirection. *POI* also distributes the rules in the switches (S_1, S_5, S_4) to forward the flow from H_4.

Similarly, traffic from H_3 to C_2 is redirected through path P_2 based on the request from customer C_2. In the same way, if customer C_1 requests the ISP to redirect the legitimate traffic originating from host H_1, one of the available paths will be chosen. If customer C_1 requests to redirect the traffic of H_2 with alert containing flow class *suspicious* and impact severity *medium*, then flow is redirected to path P_5, which is provisioned for suspicious traffic of medium impact severity. In doing so, ISP can guarantee the good QoS for legitimate traffic.

Listing 5.1 – Alert Message Exchange Format

```
<IDMEF-Message version="1.0">
<Alert>
    <Analyzer analyzerid="CUSTOMER_C3"/>
    <Source>
       <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.4</address>
            </Address>
        </Node>
        <Service>
            <protocol>udp</protocol>
        </Service>
    </Source>
    <Target>
        <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.7</address>
            </Address>
        </Node>
        <Service>
            <protocol>udp</protocol>
        </Service>
    </Target>
    <Classification event="UDP-Flood Traffic">
   </Classification>
    <Assessment>
        <Impact severity="Low"/>
    </Assessment>
    <AdditionalData> type ="string" meaning="flow class">
    <string>Legitimate</string>
    </AdditionalData>
</Alert>
</IDMEF-Message>
```

Listing 5.2 – A Sample Policy File to redirect suspicious traffic

```
<Policy PolicyID="Security_policy">
    <Event event="UDP-Flood">
    </Event>
    <Condition>
        <flow class="suspicious"/>
        <Impact severity="medium"/>
    </Condition>
    <Actions action="redirect"/>
    </Actions>
</Policy>
```

Listing 5.3 – A Policy File to provide QoS to Legitimate flows

```
<Policy PolicyID="QoS_policy">
    <Event event="QoS_request">
    </Event>
    <Condition>
        <flow class="legitimate"/>
    </Condition>
    <Actions action="redirect"/>
    </Actions>
</Policy>
```

5.5.3 Results and Analysis

We use *throughput* and *network jitter*, two well accepted QoS metrics, to evaluate the effectivenss. Some results are reported in the following.

Throughput of legitimate traffic

Throughput is measured in the presence of DDoS attacks. As we can see in Figure 5.4, the throughput of all the legitimate traffic dropped sharply as soon as H_2 started to attack. As a result, the SDN controller of customer C_3 sends an alert, which contains the Flow informations (source IP, destination IP) and flow class (legitimate), to *MC* at the ISP controller, making *PDP* decide on redirecting the flow. Subsequently, *NIB* computes the best path, which is, P_3 in this case, and provides the path details and NSH to the PDP. Finally, the corresponding OpenFlow rules are loaded to *PEPs*, namely the OpenFlow switches by POI based on the inputs from PDP. As shown in Figure 5.4, the legitimate traffic traversing to C_3 was thus able to quickly return to its normal level.

Similarly, the traffic flow going to C_2 was redirected through path P_2 upon the request of customer C_2, in order to restore its throughput to the normal level. Afterwards, the alert of customer C_1 reached the ISP controller, which interestingly redirected the traffic originating from host H_1 (which has the higher throughput) to path P_2 as well. Therefore, pushing the throughput of the traffic (originating from host H_3) to customer C_2 down to zero, as shown in Figure 5.4. This indicates that, due to the limited availability of high QoS paths in the ISP network, ensuring the QoS for one customer may incur negative impact on other customers.

QoS provisioning for legitimate traffic.

Following the previous experiment, we examine how the QoS of legitimate traffic can be provisioned if all the paths with high bandwidth are congested. In this experiment, when the customer C_1 requests for getting better QoS. The traffic from H_1 to C_1 is redirected to lower bandwidth path P_4 instead of P_2. As shown in Figure 5.5, it avoided the collateral damage caused to the

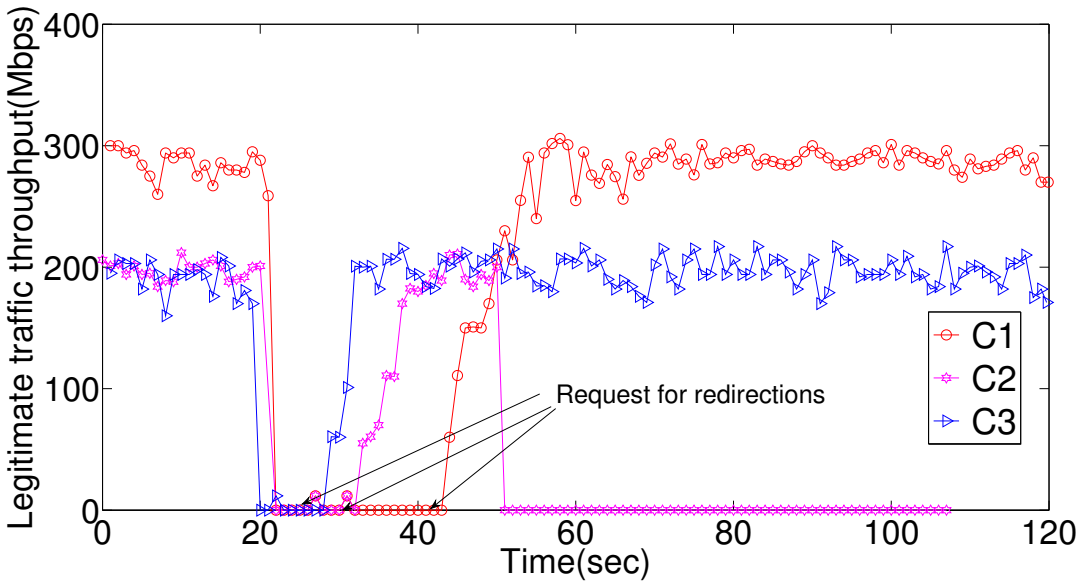

Figure 5.4 – Throughput of legitimate traffic going towards customer network after redirection

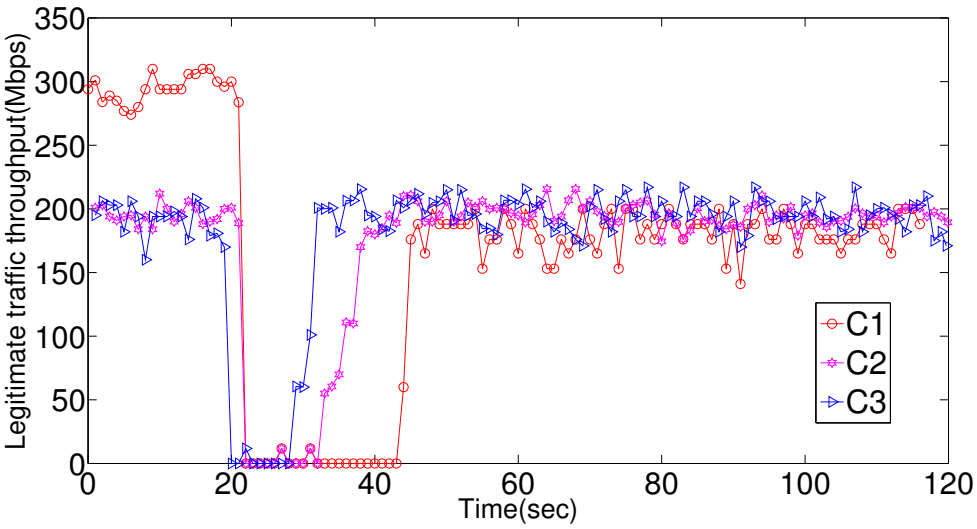

Figure 5.5 – Throughput of legitimate traffic in the case traffic going towards C_1 is redirected through low suspicious path

traffic going towards C_2 and C_3. Moreover, traffic from H_1 to C_1 is not heavily impacted in spite of the congestion in the legitimate path LP_1.

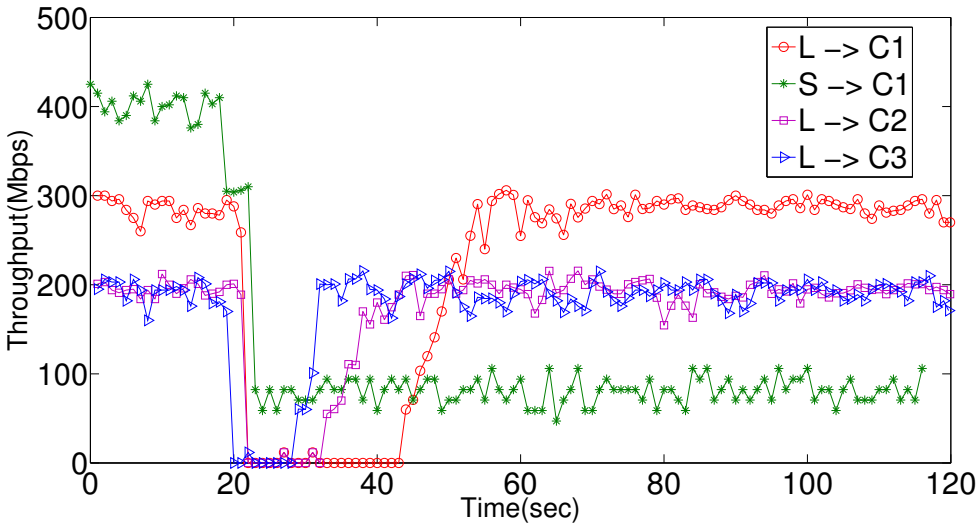

Figure 5.6 – Throughput of legitimate and suspicious traffic going towards customer network after redirection.

Aggressive traffic redirected through suspicious path

In this experiment, we examined the effect of redirecting the suspicious traffic through suspicious path i.e. path provisioned with low bandwidths. Customer C_1 sends an alert for the traffic coming from H_2 by specifying the flow class as *suspicious* and impact severity as *medium*. As a result, traffic from H_2 is redirected through suspicious path SP_2 with medium impact severity. As Fig. 5.6 shows, legitimate flows from H_1 obtained its fair share of bandwidth close to 300 Mbps. Also, flows from H_3 and H_4 heading to C_2 and C_3 got appropriate bandwidth around 200 Mbps. However, suspicious traffic from H_2 to C_1 receives throughput of 100 Mbps which is provisioned for the suspicious traffic with medium impact severity.

Network Jitter of legitimate traffic

Finally, we test how the network jitter of legitimate traffic varies because of congestion in the network. As Fig. 5.7 shows, the network jitter of legitimate traffic going towards customers C_1, C_2, and C_3 started to increase when the attack traffic from H_2 congested the network. However, all of them immediately decreased when the ISP controller redirected the traffic flows upon receiving the mitigation requests from the customers. Despite the similar changing pattern, the network jitter of the traffic going to C_3 decreased earlier compared to those of C_2 and C_1. This is simply because customer C_3 sent alert earlier than customers C_2 and C_1 did.

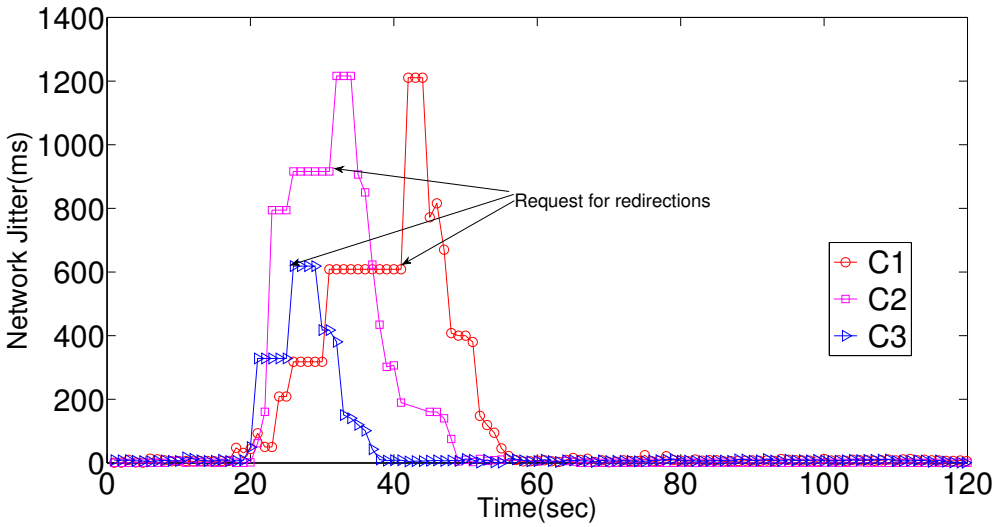

Figure 5.7 – Network jitter of legitimate traffic

5.6 Discussions

To the best of our knowledge, there is a number of works dealing with policy-based network management leveraging the SDN paradigm [16, 18, 70, 77]. They usually exploit key features such as data plane programmability and network visibility of SDN to ease the network management tasks. In this section, we review existing policy-based systems, that inspired them, and earlier proposals in traffic steering.

Traffic steering is an exemplary instance of how to take advantage of SDN switches to enforce routing and security policies in an efficient manner for middlebox-specific networks. One such effort, SIMPLE [93], alleviates manual operation from administrators by allowing them to specify a logical routing policy and translates it into forwarding rules, in compliance with physical resource constraints. Additionally, FlowTags [43] provide a technique to enforce network-wide policies in spite of packet modifications imposed by middleboxes.

Policy management frameworks would often rely on the above-mentioned technological building blocks to satisfy user-centric requirements. EnforSDN [18] proposes to simplify network service management by decoupling policy resolution (computing concrete rules) from policy enforcement (pushing low-level rules) by remarking that the former deals with flows, e.g., security policies, while the latter forwards packets at the data plane. It tackles middlebox-induced problems but fails to accommodate other contexts than the network. PolicyCop [16] is such a QoS policy enforcement framework that provides an autonomic management of user-centric policies by monitoring and enforcing the users' Service-Level Agreements (SLAs). It relies on common data plane interfaces exposed by OpenFlow forwarding devices for statistics collection and flow information retrieval, and includes a number of controller applications to support policy

monitoring and enforcement. Our work is interested in extending the inputs to the policy engine with security events. Additionally, the computation of policy routes do not take into account the availability of security services. Business-level goals are also taken into account in a policy authoring framework proposed by Machado et al. [77]. Their framework matches these requirements with the capacities provided by the network infrastructure in order to decide on an appropriate policy, through abductive reasoning. It is however unclear how network elements are requested and configured beyond the policy path computation (referred to as *Analysis Phase* in their work). OpenSec [70] is close to our proposal in that this framework provides a language that blends security services within the autonomic reaction process. However, it seems to focus the deployment on edge switches, while we are interested in distributing the policies across the controller's network domain.

Our work aims at accommodating multiple customer services sharing a network service provider, who doubles as a security service provider. Going beyond routing and QoS requirements, we aim at reacting to security events, with the collaboration of the customers, and offer network-status-aware security reaction policies. The presence of multiple competing customers raises a supplementary challenge in that the reactive policy targeted at a given customer should cause little to no impact on other customer services. It is important to consider the whole network then, and not only the edge switches, in order to distribute the rules along the policy path. This path traverses a number of forwarding switches and security services (either static middleboxes, or virtualized network functions) in a fashion similar to OpenSec [70] and SIMPLE [93].

There are some limitations of our policy management and enforcement framework. We implemented the framework with one ISP and three customers scenario. However, ISPs generally have hundreds and thousands of customers. So, scalability is a major concern. However, to manage large customer base ISPs can deploy multiple SDN controllers. Different SDN controllers can manage different groups of customers. Therefore, with multiple SDN controllers scalability can be increase to some extent. Moreover, while performing traffic engineering our framework considers the total bandwidth of the path and the current load in the path. Additionally, it considers the QoS requirement and middleboxes need to be traversed in case the traffic is suspicious. However, it can be modeled as an optimization problem by considering different constraints such as load on switches and middleboxes. Load on SDN controllers should also be considered while deploying the rules. So, there are some factors which needs to be considered for the deployment of our policy management and enforcement framework.

5.7 Conclusion

In this chapter, we proposed an automated and dynamic policy enforcement framework for mitigating the impact of attack traffic between the ISP and its customers. Framework allows the administrators in the ISP network to define high-level security policies in human readable format. Security policies at the ISP controller can be dynamically specified according to the security alerts sent from the customer controller to address the different circumstances. The policies are

specifically enforced at OpenFlow switches via APIs of SDN controller for achieving dynamic policy deployment. Specifically, our policy management framework provides collaborative and user centric automated response for mitigating the impact of attack traffic and providing the QoS service to the customers of the ISP.

6 Conclusion and Future Work

The main objective of this thesis was to propose automated mechanism to handle the cyber attacks. Imagine, for instance service provider and enterprise infrastructures, with attackers targeting both. Cyber attacks, targeting both may affect the wide diversity users and services over the Internet. Unavailability of the network administrators to mitigate the attacks further increase the impact of attack traffic.

In terms of contributions, we have started this dissertation by surveying existing mechanisms to mitigate the cyber attacks. Firstly, we presented the state of autonomic cyberdefense, as well as the existing mechanisms. The, we introduced the SDN and discussed its features. More specifically, we surveyed SDN based security mechanisms and services. We described the attack mitigation, detection, traffic monitoring and traffic engineering mechanisms based on SDN, as well as their characteristics. Our focus has been on how security mechanisms can be improved using the SDN technology. Specifically, we presented the techniques leveraging the characteristics of SDN such as decoupling of control and data plane, programmability and global visibility from a security standpoint.

The state-of-the art and related work has been complemented by three main contributions. First of all, we proposed an autonomic mitigation framework which provides a collaborative and automated response. The framework is distributed between ISP and its customer. The framework allows the customer to request for mitigation upon attack detection. The idea is validated in the mininet as well as in the testbed. Second, we presented a policy grammar to represent the high-level security and network policy in the ISP network. We introduced policy templates and tables to map the high-level policies into low-level OpenFlow rules for deployment in data plane devices. The idea is to provide a way to represent the high-level policies in the ISP network and provide an automated way to translate these policies in to low level rules. Third, we proposed a policy management and enforcement framework to manage the ISP network and provide security and QoS services to its customers considering the network status of the ISP. The aim is to reduce the impact of collateral damage caused by the attack traffic to different customers. The idea is to compute the path dynamically considering the network status of the ISP and provide the path to

redirect the traffic. A prototype of policy management and enforcement framework is designed and validated in the mininet considering a ISP network and multiple customers scenario.

In terms of perspective for future research, several actions remain to be done. It is worth noting that security mechanisms using SDN have a vast number of challenges to be addressed. This dissertation has handled, with a limited scope, some of the challenges in the topic, paying special attention to mitigation of attack traffic in an automated way. Automated response system encompass many other fields that have to be handled together in order to improve their resilience against attack and misuse.

With this in mind, a first perspective would include a more thorough analysis of the performance impact of using SDN for mitigating attack traffic. Indeed, the performance of our autonomic framework is an important issue that is necessary to handle. In this dissertation, we expanded the single ISP and single customer, into three customer scenario, while reducing the impact of attack. However, impact of scalability on ISP network needs to be considered for good performance.

In this dissertation, we assumed that customer uses some detection mechanism to detect the attacks. Accurate and quick detection of attack is required for timely and effective mitigation. Moreover, integration of detection tool at the customer network with the mitigation components in the ISP network is also a concern. Moreover, in the framework, monitoring component needs to be integrated at the ISP side to detect the congestion in the ISP network, and provide the network status of the ISP network for effective traffic engineering in the ISP network. Likewise, the performance of the framework with all the components integrated together, with more customers needs to be analysed.

A French Summary

A.1 Introduction

Les cyber-attaques sont défendues par deux mécanismes: la détection et l'atténuation. Le diagnostic d'attaque repose sur le mécanisme de détection. L'atténuation est le mécanisme de réponse qui tente de réduire l'impact de l'attaque et sa sévérité sur le réseau. Contrairement aux mécanismes de détection, la recherche sur les approches d'atténuation a reçu moins d'attention en raison de la complexité inhérente au déploiement de mécanismes automatisés d'atténuation. En outre, les mécanismes d'atténuation existants nécessitent le déploiement de périphériques matériels ou logiciels spéciaux qui les rendent complexes et coûteux. Pour ces raisons, les petites entreprises et les fournisseurs de services sont réticents à déployer ces mécanismes complexes dans leurs réseaux.

Par ailleurs, les cyber-attaques ne causent pas seulement des problèmes aux réseaux d'entreprise ou aux utilisateurs finaux, mais aussi aux fournisseurs de services Internet (FAI). Ce sont en effet les FAI qui fournissent aux entreprises et aux utilisateurs finaux un accès à Internet qui y souscrivent par un abonnement. Lorsque les attaquants ciblent le réseau des utilisateurs finaux, ils provoquent également des dommages collatéraux aux réseaux des FAI lorsque le trafic malveillant traverse le réseau du FAI avant d'atteindre les utilisateurs finaux. Par conséquent, les FAI sont devenus des acteurs importants de la cybersécurité et leur intervention est essentielle pour atténuer les cyber-attaques et leurs impacts. Cependant, il est difficile pour le FAI de détecter les attaques vers les utilisateurs finaux ou les réseaux d'entreprise. Cela entraîne un traitement additionnel pour les dispositifs de détection déployés dans le réseau du FAI ainsi qu'une violation de la politique pour les utilisateurs finaux. Généralement, les FAI disposent d'une base de clientèle conséquente et, s'ils devaient réaliser le processus de détection pour chacun de leurs clients, ils subiraient un coût de traitement prohibitif. En outre, si la détection et l'analyse sont effectuées par le FAI et si, pendant cette durée, le flux est modifié, cela pourrait causer une violation de la politique des clients. Par conséquent, il est crucial de fournir une technique d'atténuation qui peut coexister entre les FAI et les utilisateurs finaux, et qui respecte la politique du réseau et celles des utilisateurs. Il est donc crucial pour les utilisateurs finaux de fournir l'information adéquate au

FAI pour atténuer les cyber-attaques.

De nos jours, les cyber-attaques ont gagné en ampleur et leur impact est s'est étendu, pour ne plus se limiter au FAI et à l'utilisateur final cible des attaques, mais affecter aussi les autres utilisateurs finaux et FAI qui dépendent du FAI fournisseur de l'utilisateur victime. Cela inclut également les ressources des différents fournisseurs de services. Par conséquent, il existe un besoin pour un mécanisme de défense coopératif et à grande échelle impliquant les différents acteurs victimes des attaques [51]. Ces mécanismes coopératifs visent à être évolutifs au niveau technique et commercial. Ils nécessitent bien souvent l'usage ou l'installation d'un logiciel ou d'un matériel spécifique. Cela rend ces systèmes complexes à déployer

La gestion de ce mécanisme de défense coopérative est une tâche difficile, car elle présente différents défis: (1) le partage des informations d'alerte entre différents domaines réseau doit être rapide et automatisé, (2) le déploiement de politiques réseau et de sécurité dépendant des accords de niveau de service (SLA) avec un domaine de réseau différent doit être réalise sans aucune violation de la politique, (3) la configuration des règles spécifiques au périphérique de bas niveau sur les périphériques réseau hétérogènes ne doit pas dépendre du travail manuel des administrateurs réseau, (4) le déploiement d'un tel mécanisme de défense ne doit pas nécessiter l'installation d'un matériel ou d'un logiciel spécifique. Ceci afin d'obtenir un mécanisme de défense automatisé, qui répond aux changements du réseau et aux alertes de sécurité, sans intervention des administrateurs réseau.

Cependant, il existe des difficultés à fournir l'automatisation nécessaire à la cyber-défense. Il existe un écart clair entre les différents composants, de la surveillance et de l'analyse à la réponse. L'automatisation ne peut être réalisée sans assurer le contrôle du réseau, en fonction des variations de son état. Les différents composants d'une défense automatisée doivent égalelement communiquer efficacement les uns avec les autres pour assurer la défense globale du système. Et cela, sans augmenter le coût de déploiement pour le réseau d'entreprise en matière de logiciels et de matériels spécifiques. En effet, s'il est nécessaire d'implémenter des dispositifs spécifiques ou d'intégrer des logiciels requérant beaucoup de modifications dans le réseau, les entreprises seraient alors réticentes à déployer ce type de mécanismes automatisés. À l'heure actuelle, les mécanismes de défense automatisés sont fortement couplés aux périphériques matériels faisant ainsi du déploiement et de la gestion du réseau une tâche complexe et fastidieuse. En outre, le système de défense automatisé doit être robuste et à l'abri de nouvelles formes d'attaques. Ainsi, ces défis doivent encore être abordés pour rendre le système de défense du réseau automatisé et opérationnel.

L'objectif de cette thèse est d'étudier la faisabilité et l'efficacité de la construction d'un mécanisme de cyber-défense autonome en explorant de manière exhaustive le paradigm Software-Defined Networking (SDN) et en tirant parti de ses caractéristiques uniques, principalement la programmabilité du plan de données, la visibilité globale sur l'ensemble du réseau et le contrôle centralisé. En d'autres termes, la thèse vise à étudier comment et dans quelle mesure la cyber-défense autonome peut être réalisé en matière d'auto-configuration, d'auto-optimisation, d'auto-protection

et d'auto-réparation.

Le rapport d'enquête d'AlgoSec montre qu'en raison du manque d'automatisation:

- 20% des organisations ont subi des menaces contre leur sécurité;
- 42% des organisations ont eu une panne de réseau en raison de mauvaises configurations du réseau causées par des tâches liées à la sécurité. Selon l'enquête, de nombreuses organisations ont éprouvé des violations de sécurité en raison de configurations manuelles de sécurité;
- pour 19% des organisations, un jour ouvrable complet ou plus est nécessaire pour résoudre la violation de sécurité en raison du travail manuel réalisé par les administrateurs réseau.

Cependant, avec les menaces cybernétiques accrues, il est nécessaire d'automatiser les mécanismes de défense. Les attaquants utilisent des outils automatisés pour cibler les ressources du réseau, rendant la défense automatisée nécessaire pour contrer ces attaques.

AUTONOMIA [29] est un système reposant sur la configuration automatique pour contrôler et gérer les systèmes de sécurité du réseau. Il configure le système et modifie dynamiquement les stratégies pour protéger le réseau. Le système comporte deux modules: une interface de gestion de composants (CMI) et un gestionnaire d'exécution des composants (CRM). Le CMI permet de définir les politiques de haut niveau pour chaque composant du réseau, qu'il s'agisse du logiciel ou du matériel. Le CRM gère les composants en utilisant les politiques définies dans le CMI. Le travail rapporté dans [94] classifie le trafic comme normal, anormal ou incertain, pour prioriser les différentes classes de trafic afin de fournir de la qualité de service. Un outil décrit dans [116] met à jour les règles de pare-feu en fonction des informations de trafic destiné à un réseau de pots de miel (honeynet). Il contient un module qui analyse les journaux de trafic générés par le honeynet et utilise des mécanismes d'exploration de données pour générer les nouvelles règles de pare-feu pour le déploiement.

ISDS [117] a été développé comme un système de sécurité basée sur le paradigme d'informatique autonome (*Autonomic Computing*). Le but de ce logiciel est de fournir de l'intelligence à ses composants pour s'adapter dynamiquement en fonction des conditions de sécurité du réseau.

Ces systèmees nécessitent souvent un logiciel ou un matériel spécifique pour leur déploiement. Cela rend ces systèmes difficiles à déployer. Grâce à l'émergence de la mise en réseau définie par logiciel (SDN) [88], il nous est permis de réviser les mécanismes de défense pour les automatiser. SDN est un nouveau paradigme réseau qui sépare les plans de contrôle et de données. Cette séparation permet d'ajouter facilement de nouvelles fonctions réseau en fonction des besoins actuels du réseau. Le terme SDN a été inventé à l'Université de Stanford pour identifier le travail réalisé sur OpenFlow [80]. Depuis, il a attiré beaucoup d'attention tant du milieu universitaire que de l'industrie. De nombreuses idées nouvelles basées sur SDN ont été proposées [41, 66, 101].

Dans l'industrie, SDN est considérée comme une technologie future qui peut réduire le coût opérationnel et renforcer le réseau.

Une architecture SDN se caractérise par les caractéristiques suivantes:

- *un contrôleur logiquement centralisé et une visibilité globale du réseau :* dans un réseau SDN, tous les appareils du plan de données sont connectés à un contrôleur centralisé. Le contrôleur peut alors envoyer des messages de contrôle à ces appareils. Le contrôleur peut également leur envoyer des requêtes pour obtenir des statistiques qui permettent de déduire l'état du réseau. Cette vision centralisée de l'état global du réseau facilite la prise de décision, par opposition aux réseaux existants qui sont construits sur une vision autonome du système où les noeuds ignorent l'état général du réseau.
- *la programmabilité du plan de contrôle :* les réseaux SDN sont contrôlés par un logiciel, qui peut être fourni par les FAI ou les opérateurs de réseau eux-mêmes. Les éléments du plan de données peuvent être contrôlés par les applications déployées sur le contrôleur.
- *un niveau d'abstraction :* les applications métier utilisent des services SDN qui abstraient les technologies de réseau sous-jacentes. Les périphériques réseau sont également abstraits par la couche de contrôle SDN pour assurer la portabilité.
- *une gestion basée sur les flux :* une règle de base des périphériques réseau (éléments du plan de données) est de transmettre les paquets d'un flux au contrôleur lorsque ceux-ci ne possèdent pas de règles pour gérer ce flux. Cette fonctionnalité de SDN nous permet de gérer le flux de manière dynamique en fonction des conditions variables du réseau [6].

Dans la section suivante (A.2), nous proposons un cadre d'atténuation autonome pour atténuer les attaques de manière automatisée. Nous évaluons le cadre sur une plateforme de test ainsi par simulation. Nous démontrons les performances du cadre par une atténuation d'attaques DDoS (déni de service distribué).

A.2 Un Cadre de Mitigation de DDoS Autonome basé sur SDN

Une grande variété de mécanismes d'atténuation des attaques DDoS ont été proposés, qui couvrent la détection ainsi que les phases d'atténuation. Cependant, aucun des mécanismes existants n'a été envisagé pour un déploiement à grande échelle en raison de la complexité inhérente aux tâches de gestion du réseau.

Dans [78], les auteurs ont proposé un système appelé *dFence* capable de fournir une atténuation DDoS à la demande en redirigeant le trafic vers un ensemble de dispositifs réseaux (aussi appelées *middleboxes*) déployés statiquement. Étant donné qu'un grand nombre de middleboxes doivent être déployées statiquement dans le cœur du réseau du FAI afin de traiter le trafic, cela provoque une congestion lorsque le FAI doit rediriger le trafic. En outre, le manque de mécanisme

efficace d'échange d'informations entre différents domaines rend difficile le déploiement et l'exploitation de ces middleboxes. Par exemple, le processus actuel d'échange d'informations entre le fournisseur de services Internet et ses clients dépend encore de la disponibilité de l'administrateur réseau.

Une leçon importante tirée des dernières attaques DDoS est que les clients et les FAI doivent collaborer étroitement les uns avec les autres. Idéalement, lorsque le client détecte l'attaque, les informations connexes doivent être partagées avec le FAI à temps pour une atténuation rapide. Cependant, à notre connaissance, seuls quelques programmes collaboratifs d'atténuation des DDoS sont actuellement disponibles [62, 99]. Ces mécanismes sont basés sur une communauté de partenaires fiables, ce qui rend nécessaire pour tous les routeurs réseau d'en maintenir une liste, entraînant un traitement additionnel occasioné par l'échange d'informations de sécurité. En outre, les dispositifs de réseau dédiés, ainsi que les routeurs modifiés, doivent être déployés au préalable, entraînant une complexité opérationnelle, de déploiement et de gestion, qui rendent les interventions humaines non triviales inévitables. Par conséquent, il est nécessaire d'automatiser de manière significative l'ensemble de la gestion du cycle de vie des approches d'atténuation des attaques DDoS.

Dans l'ensemble, les leçons tirées des conceptions passées de l'atténuation des attaques DDoS indiquent clairement que les exigences suivantes méritent une attention particulière:

1. Déploiement et fonctionnement faciles, évitant de recourir à un logiciel ou des périphériques matériels spécifiques;
2. Echange efficace d'informations entre les différentes parties impliquées (c'est-à-dire les FAI et leurs clients), afin qu'un client puisse envoyer ses informations sur les menaces au FAI à un stade très précoce de l'attaque DDoS;
3. Le FAI doit traiter en temps opportun les demandes de ses clients et appliquer les politiques d'atténuation d'attaques DDoS appropriées;
4. Tout le cycle de vie du programme d'atténuation d'attaques DDoS doit être géré de manière adaptative et automatisée.

Malheureusement, les caractéristiques intrinsèques (par exemple, le fort couplage entre le plan de contrôle et le plan de données) de l'infrastructure de réseau existante font que la majorité des systèmes d'atténuation d'attaques DDoS ne satisfont pas aux exigences requises. Grâce au développement récent de la technologie SDN, qui offre une séparation claire du plan de contrôle du réseau et du plan de données, un certain nombre de fonctionnalités nous sont ainsi promises telles que la programmabilité des périphériques réseau par le contrôleur SDN. Dans ce chapitre, nous proposons un cadre d'atténuation de DDoS autonome, *ArOMA*, basé sur SDN, qui a le potentiel de répondre parfaitement aux exigences de conception en intégrant différents modules d'atténuation d'attaques DDoS, allant de la surveillance du trafic à l'atténuation, en passant par la détection d'anomalies.

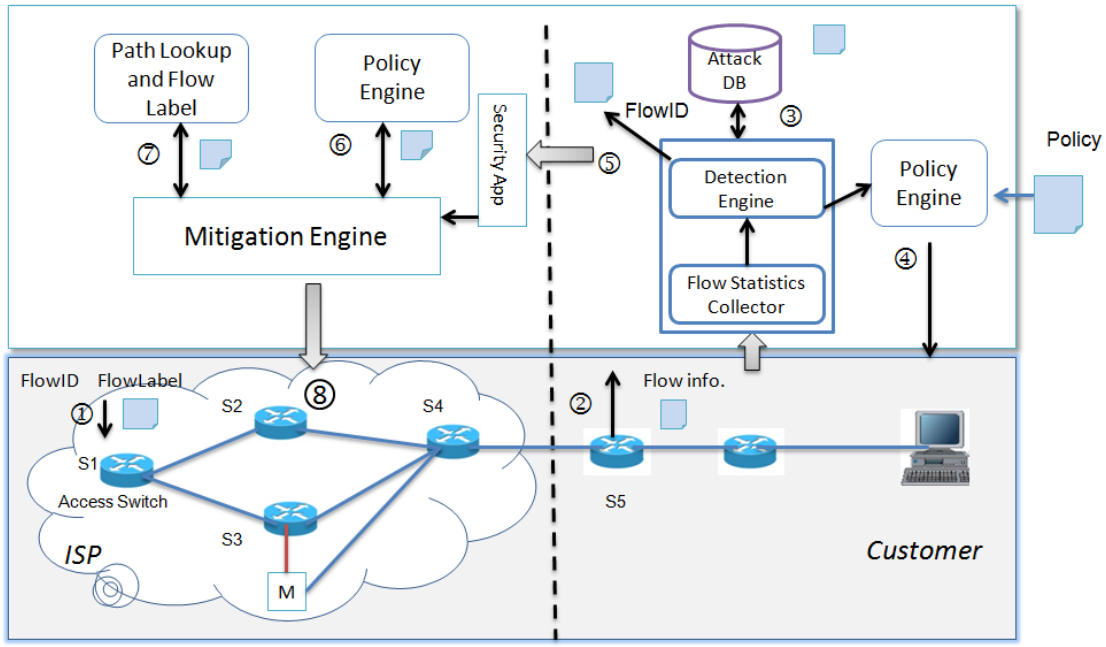

Figure A.1 – Cadre d'atténuation d'attaques DDoS basé sur SDN

A.2.1 Architecture et Flux Opérationnel

Dans le cadre *ArOMA*, les FAI et les clients disposent de leur propre contrôleur SDN dans leur réseau respectif et communiquent de manière sécurisée. Comme le montre la figure A.1, le cadre est réparti entre le FAI et les réseaux clients, et le flux opérationnel (étiqueté avec des numéros d'étape) peut être décrit comme suit:

1. Lorsque le trafic entre dans le réseau du FAI, le contrôleur attribue un identificateur de flux unique, `FlowID`, pour chaque nouveau flux, généralement en utilisant le message `PACKET-IN`. `FlowID` est utilisé par le contrôleur du client pour demander l'atténuation d'une attaque au FAI. Les étiquettes sont également attribuées aux flux par les commutateurs d'entrée du FAI pour accélérer la transmission de paquets (*forwarding*). L'étiquette aide enfin à préserver la politique appliquée au flux afin d'assurer un routage inaltéré.

2. Du côté du client, des statistiques de flux sont collectées périodiquement via les agents OpenFlow et transmises au moteur de détection pour être traitées.

3. Le moteur de détection repose sur une base de données d'attaque, qui peut interagir avec des bases de données externes via certaines méthodes d'échange d'informations de menaces telles que l'interface de programmation n6 [61].

4. Le moteur de détection génère des alertes de sécurité en présence de flux de trafic malveillants ou suspects et déclenche une réaction à partir d'un moteur de politiques local, chargé de générer et d'installer des règles de politique lors de l'entrée du flux dans le réseau client.

5. Les `FlowID` s de flux suspects et malveillants sont encapsulés dans les requêtes de sécurité envoyées au moteur d'atténuation qui réside sur le contrôleur SDN du FAI. À cette fin, une interface de programmation (API) de sécurité est exposée par le contrôleur SDN du FAI. Cette API de sécurité est le principal composant de la communication entre contrôleurs.

6. Le moteur d'atténuation demande au moteur de politique quelles sont les actions d'atténuation à implémenter afin de traiter les flux suspects ou malveillants détectés.

7. De plus, le moteur d'atténuation vérifie avec le module *de recherche de chemin* quel est le meilleur chemin pour acheminer les flux détectés en fonction des actions d'atténuation.

8. Les actions d'atténuation, par exemple la redirection de trafic vers des middleboxes de sécurité, sont finalement appliquées en mettant à jour les étiquettes des flux concernés. Ce processus d'atténuation implique un déploiement automatisé et dynamique des politiques au sein du réseau du FAI.

A.2.2 Communication entre Contrôleurs

Une autre caractéristique importante d'*ArOMA* est la communication entre le contrôleur du réseau du FAI et celui du réseau client. La communication entre contrôleurs permet au client de partager les informations d'alerte avec le FAI de manière automatisée. Ce canal de communication a effectivement deux objectifs: partager le `FlowID` du flux malveillant détecté et alerter le FAI sur la menace de ces trafics malveillants destinés au réseau client. Plus précisément, la façon de partager le `FlowID` est décrite comme suit:

1. Initialement, lorsque le trafic atteint le réseau client, le contrôleur client demande le `FlowID` du flux correspondant en envoyant les adresses IP (source et destination) de ce flux à l'API de sécurité du contrôleur du FAI.

2. Le `FlowID` est maintenu et mis à jour périodiquement au niveau du commutateur d'entrée du FAI, dans le champ *Cookie*.

3. Lors de la réception de la demande du contrôleur client, le contrôleur du FAI vérifie la table du commutateur d'entrée correspondant pour obtenir le `FlowID` correspondant et le renvoie au contrôleur client.

Pour illustrer, la communication entre contrôleurs dans le cadre de l'atténuation d'une attaque DDoS contient les opérations suivantes:

1. Lors de la détection du trafic DDoS côté client, le moteur de détection envoie l'alerte pour les flux détectés;

2. L'alerte de sécurité est envoyée au format IDMEF (mais d'autres formats peuvent également être utilisés).

3. L'interface de sécurité (API de sécurité) exposée par le contrôleur du FAI est atteinte via une requête HTTP encapsulant le message d'alerte de sécurité généré par le moteur de détection du contrôleur client.
4. L'API de sécurité extrait de l'alerte des informations telles que la classe de sécurité, les informations du flux (IP source, IP destination), la sévérité de l'attaque (en matière d'impact sur le réseau) et le type d'attaque.
5. Un accusé de réception contenant les informations de flux et les actions appliquées est renvoyé au contrôleur du client après l'application de l'action sur le flux.

A.2.3 Expérimentations sur la Plateforme de Test

Nous avons réalisé nos expérimentations sur une plateforme physique pour évaluer notre prototype du cadre *ArOMA*. Dans cette plateforme, nous illustrons la performance de protection du cadre par sa capacité à maintenir la qualité d'un flux vidéo, face aux attaques DDoS. Nous mesurons cette performance en matière de qualité de l'expérience utilisateur (QoE).

Plateforme. La topologie de notre plateforme expérimentale est similaire à la configuration du plan de données montrée dans la Fig. A.1, où le FAI et les réseaux clients ont leur propre contrôleur SDN. Pour simplifier, deux chemins de routage sont configurés dans la plateforme pour le réseau du FAI: l'un est garanti en matière de qualité de service et utilisé exclusivement pour le trafic légitime (chemin passant par les commutateurs S_1, S_2 et S_4), tandis que l'autre est dédié au trafic suspect ou malveillant (chemin passant par S_1, S_3 et S_4). Dans le réseau client, le commutateur OpenFlow S_5 est rattaché au contrôleur client. En outre, dix machines hôtes sont virtualisées dans la plateforme, pour représenter les utilisateurs légitimes, les attaquants et le serveur de streaming vidéo.

Génération de trafic. La raison, pour laquelle nous avons choisi d'illustrer les performance du cadre *ArOMA* en utilisant la vidéo comme source de trafic légitime, tient au fait que le trafic vidéo actuel représente plus de 70% du trafic Internet des consommateursc̃iteseventy. Nous utilisons ici *TAPAS* [34], un outil de streaming vidéo pour la génération de trafic vidéo entre le serveur HTTP dans le réseau client *C* et les hôtes légitime *L*. 10 clients *TAPAS* permettent de générer un trafic vidéo légitime en provenance du serveur HTTP. De plus, l'outil BotNet Simulator (BoNeSi) est utilisé pour générer des attaques DDoS volumétriques avec des taux de trafic élevés. Nous générons un trafic d'attaque de près de 250.000 paquets par seconde.

A.2.4 Résultats et Analyse

Nous avons mené plusieurs séries de tests pour étudier comparativement les paramètres donnés dans trois cas,

- Services de diffusion vidéo en condition normale (sans trafic d'attaque);

- Face à différentes attaques et sans atténuation;

- Face aux attaques et en présence de notre prototype *ArOMA*.

Les métriques sont mesurées par les clients de streaming vidéo en collectant et en analysant les fichiers journaux générés par *TAPAS*.

Taille Moyenne de la Mémoire Tampon

Nous avons mesuré la taille moyenne de la mémoire tampon (ou taille du *buffer*, exprimée en secondes), et nous avons observé que la taille du buffer était maintenue au-dessus de 16 secondes en l'absence d'attaque. Comme le montre la Fig. A.2, en présence d'une attaque par inondation UDP, la taille du buffer a été maintenue à près de 14 secondes par notre cadre *ArOMA*. De même, la taille du buffer a été maintenue respectivement à 12 et 10 secondes, face aux attaques ICMP et TCP. En revanche, la taille moyenne du buffer était d'environ 4 secondes lorsque le module d'atténuation n'était pas activé pendant les attaques.

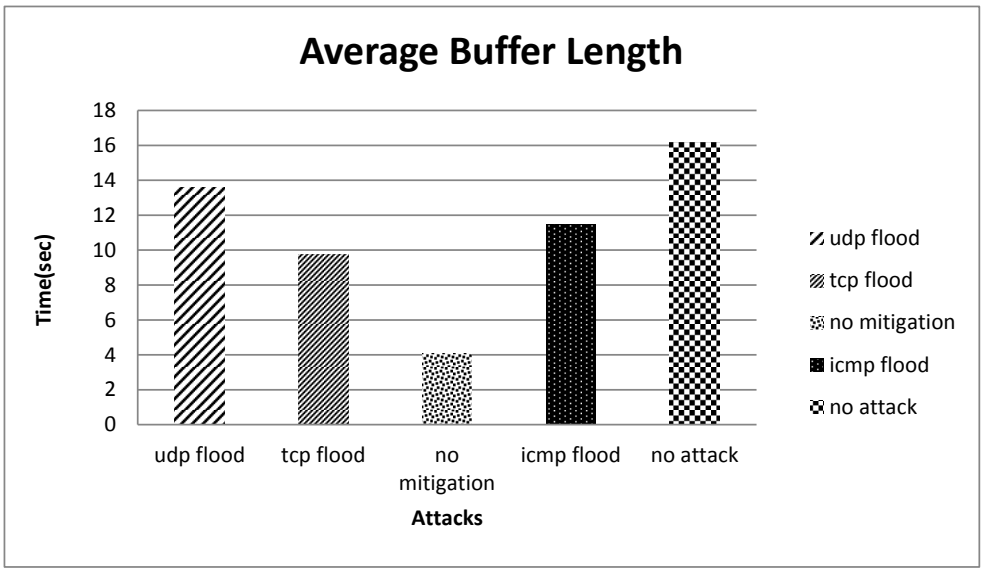

Figure A.2 – Taille moyenne du buffer en fonction des scénarios d'attaque

Débit applicatif moyen

Nous avons également examiné le *débit applicatif moyen*, et avons observé qu'il était maintenu à près de 700 Ko/s en l'absence d'attaque. Lorsque une attaque DDoS par inondation UDP est lancée et que *ArOMA* a été activé, le débit moyen peut être maintenu à 550 Ko/s. En ce qui concerne les attaques par inondation ICMP et TCP SYN, le débit applicatif moyen peut être maintenue à près de 500 et 450 Ko/s, respectivement (voir Fig. A.3). En revanche, lorsque *ArOMA* n'est pas activé, le débit moyen tombe en dessous de 300 Ko/s et ne peut pas revenir à un niveau normal.

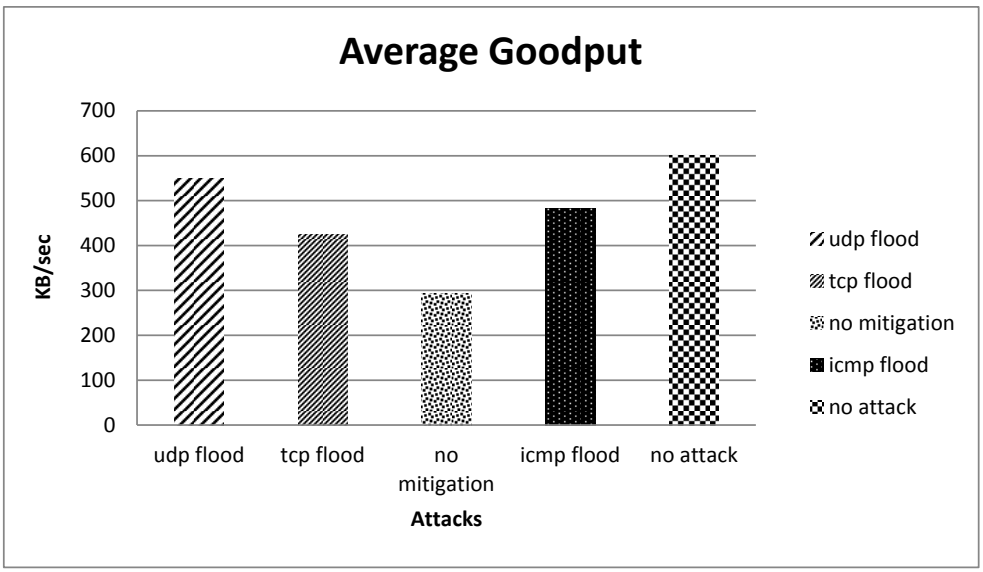

Figure A.3 – Débit applicatif moyen en fonction des scénarios d'attaque

A.2.5 Discussion

Il est largement reconnu que sans collaboration efficace, il est extrêmement difficile, sinon impossible, d'atténuer les attaques DDoS. Cependant, la collaboration entre les différents domaines et acteurs d'Internet est difficile. Dans ce chapitre, nous fournissons un cadre d'atténuation DDoS à la demande appelé *ArOMA* qui tire parti des technologies SDN, dans le but de faciliter la collaboration entre le FAI et leurs clients. Pour démontrer la faisabilité et l'efficacité de notre proposition, nous avons développé un prototype et effectué à la fois des expérimentations en simulation et sur une plateforme physique. Les résultats indiquent que notre cadre d'atténuation peut

garantir que l'actif à protéger – dans notre scénario, il s'agissait d'un service de diffusion vidéo – a pu maintenir des performances satisfaisantes en présence d'attaques DDoS par inondation. Ces résultats ont été mesurés en matière de QoE, qui indique la satisfaction de l'utilisateur, une métrique importante pour les diffuseurs de vidéo. Dans la section suivante (A.3), nous présentons un mécanisme pour représenter et traduire la politique de sécurité d'un client du FAI ou une politique de qualité de service (QoS) abstraite en règles OpenFlow de bas niveau.

A.3 Représentation et Traduction des Politiques de Haut Niveau

Dans les architectures réseau distribuées de grande échelle, des périphériques tels que les commutateurs, les routeurs, les pare-feux, les systèmes de détection d'intrusion (IDS) sont souvent utilisés pour protéger et adapter le réseau en fonction des conditions variables. La configuration manuelle de ces nombreux appareils demeure néanmoins fastidieuse, complexe et susceptible d'erreurs.

En outre, les récentes tendances de l'informatique aggravent davantage ces problèmes. Par exemple, la demande croissante de sécurité adaptative et d'optimisation des performances nécessitent des changements massifs et fréquents dans la configuration des périphériques de sécurité et de réseau. En outre, la configuration manuelle dans une situation aussi complexe peut avoir un impact négatif sur la performance des services dans le réseau. Quand bien même les politiques sont représentés dans un format proche de la configuration des périphériques, elles ne s'adaptent pas aux conditions réseaux en évolution constante. Le manque de dynamicité de la configuration entraîne donc des temps d'arrêt du réseau, en raison d'une configuration non adaptée qui requiert un intervention manuelle des opérateurs de réseau.

Les développements récents dans les réseaux SDN ont conduit à une prolifération de travaux qui tentent d'automatiser et de simplifier les tâches de gestion du réseau. L'API Nord du modèle SDN nous permet de définir des politiques de haut niveau que le contrôleur peut appliquer aux périphériques réseau du plan de données via son interface Sud. Les politiques peuvent être modifiées sans interrompre l'exploitation des périphériques réseau. Malgré ces avantages, SDN seul n'a pas la capacité de définir des politiques de manière expressive. Si la traduction de la règle de haut niveau en une règle de bas niveau n'est pas effectuée correctement, elle risque de ne pas satisfaire aux exigences et entraîner une mauvaise configuration des périphériques réseau. L'objectif de ce chapitre est donc d'aborder comment les politiques de haut niveau peuvent être représentées dans un format lisible par un humain et traduites en règles OpenFlow de bas niveau pour le déploiement dans les commutateurs.

A.3.1 Cadre de Gestion et d'Application des Politiques

Les politiques de gestion et de sécurité d'un réseau doivent être spécifiées au plus haut niveau d'abstraction. Par ailleurs, des techniques de traduction peuvent être utilisées pour faire corre-

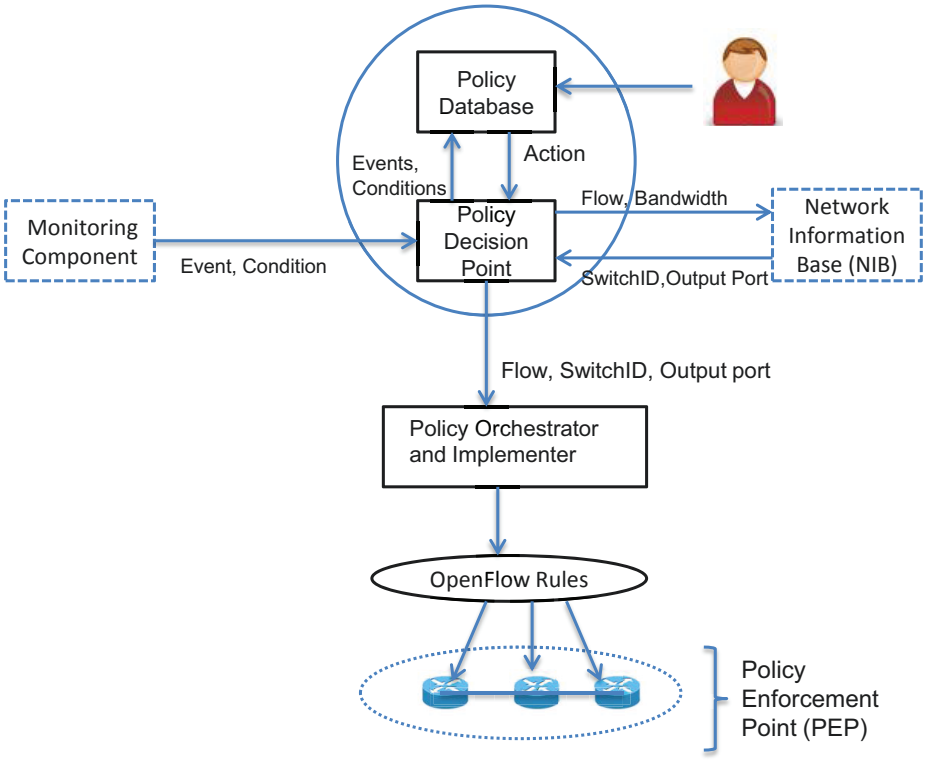

Figure A.4 – Cadre de gestion et d'application des politiques

spondre les politiques de niveau abstrait avec les instructions de bas niveau. Pour réaliser cette mise en correspondance, nous rapportons les événements, les conditions et les actions décrites abstraitement, aux règles OpenFlow de bas niveau. Dans ce chapitre, comme indiqué dans la Fig. A.4, nous nous concentrons principalement sur la spécification des politiques de haut niveau dans la base de données des politiques et la traduction de ces politiques en règles de bas niveau pour le déploiement dans les appareils du plan de données grâce au Policy Decision Point (PDP) d'une part et au Policy Orchestrator and Implementer (POI), d'autre part. La description détaillée des autres composants sera donnée dans la Section A.4.

A.3.2 Composants Principaux

Monitoring Component (MC). Le composant de surveillance (MC) reçoit l'alerte de sécurité ou le message de demande de QoS et le renvoie au moteur de politique (PDP).

Policy Decision Point (PDP). Le point de décision de la politique (PDP) est responsable de l'application de politiques réseau au niveau global. En fonction de l'événement et des conditions

du réseau et de sa sécurité, le PDP active une politique dans la base de données des politiques. Cette politique décide d'une action de haut niveau, p.ex., transférer (*forward*), rediriger (*redirect*), ou jeter (*drop*), à appliquer. En se basant sur l'action de haut niveau, l'information de flux et la bande passante demandée, il obtient les détails du chemin d'accès (identifiant du commutateur, port de sortie) de la base d'information réseau (NIB). Le PDP transmet les détails du chemin avec les informations de flux pour servir de filtre à l'orchestrateur de politique (POI) pour l'application des règles de bas niveau.

Policy Database (PDB). La base de politiques (PDB) est essentiellement un référentiel contenant les politiques de sécurité et de réseau de haut niveau spécifiées par l'opérateur réseau, sans préciser de stratégie de déploiement spécifique. Les stratégies peuvent être spécifiées en XML ou dans tout autre format défini par l'administrateur réseau. Ces politiques de haut niveau sont instanciées sur la base de l'alerte de sécurité ou des messages QoS transmis par le PDP. Un identifiant de stratégie est utilisé pour indexer les stratégies dans la base de politiques, ce qui aide à récupérer la stratégie liée à un événement donné. Il permet à l'administrateur du réseau d'exprimer des politiques de sécurité et de réseau de haut niveau sans avoir à préciser comment les mettre en application.

Network Information Base (NIB). La base d'informations réseau (NIB) Contient une liste de middleboxes et de commutateurs réseau auxquels les middleboxes sont attachées. Il offre des configurations-types qui permettent d'instancier les configuration pour acheminer le trafic par un chemin réseau différent en fonction de la bande passante. Les détails de ce composant sont décrits dans la Section A.4.

Policy Orchestrator and Implementer (POI). L'orchestrateur de politiques (POI) contient les modèles de règles OpenFlow pour différentes actions de haut niveau qui permettent l'implémentation de celles-ci dans les composants réseau du plan de données. Ces `modèles de règle OpenFlow` contiennent les directives pour spécifier comment les différentes activités peuvent être exécutées dans le réseau. En outre, il contient des scripts python qui déploient les règles dans les périphériques réseau. Sur la base de l'information de flux, il retourne les détails du chemin (switchID, port de sortie) pour déployer les règles OpenFlow de bas niveau en fonction de l'action de haut niveau.

Dans la Section A.3.3, nous présentons la grammaire pour définir les politiques de haut niveau dans la base PDB du cadre.

A.3.3 Grammaire de Politique de Haut Niveau

La syntaxe de notre politique de haut niveau nous permet définir un format unique pour exprimer de manière abstraite la politique réseau au sein d FAI. Il permet à l'opérateur de réseau d'exprimer également des politiques de sécurité dans un langage facile à comprendre sans entrer dans les détails de mise en œuvre de bas niveau. Ces politiques de sécurité et de réseau de haut niveau peuvent être appliquées dans les périphériques du plan de données lorsque les exigences sont

satisfaites.

Listing A.1 – Grammaire pour les une politiques de haut niveau

```
1 Policy=PolicyID
2 Event=Message
3 Conditions=Condition Connective Conditions
4 Condition=<field_name><value>
5 Connective=And|Or
6 Action=Forward|Drop|Redirect
```

Dans notre cadre de politique, les politiques de haut niveau sont définies selon le paradigme événement-condition-action (ECA) [63]. Notre approche permet de spécifier les actions qui seront déclenchées, face à un événement particulier en fonction d'un certain nombre de conditions. Le paradigme ECA décrit ainsi comment les événements déclenchent la réponse souhaitée du système. Un événement pourrait être un message d'un système ou d'un programme, entre autres. Les conditions décrivent la logique d'activation de l'action. L'action est définie comme une opération effectuée sur les ressources disponibles.

Les API Northbound exposées par le contrôleur SDN nous permettent de définir le langage de politique de haut niveau de manière simple. Nous utilisons la grammaire de politique présentée dans le Listing A.1 pour définir le réseau et ses stratégies de sécurité. Au niveau du plan de données, les commutateurs OpenFlow peuvent se comporter comme des middleboxes telles que un pare-feu, par exemple. Avec notre langage de politique, nous pouvons définir les politiques de tout tye d'équipement de sécurité, y compris les commutateurs OpenFlow ou les middleboxes.

Dans la Section A.3.4, nous décrivons le flux opérationnel du cadre.

A.3.4 Flux Opérationnel

En fonction des informations mentionnées dans l'alerte ou la requête de QoS, le contrôleur du FAI traduit la politique de haut niveau en règles de bas niveau. Le flux opérationnel du processus d'instantiation et de traduction de la politique est décrit selon la Figure A.4.

1. Dans le cadre, le composant de supervision (MC) reçoit les alertes de sécurité ou la requête de QoS. Le MC extrait les événements et les conditions du message reçu et les transmet au point de décision de politique (PDP). Les informations contenues dans les alertes ou les requpetes de QoS incluent les identifiants de flux (IP source, IP de destination), la classe du flux, la classe de bande passante et le type d'attaque.

2. Le PDP sélectionne une politique appropriée dans la base de règles en fonction des événements et des conditions. En retour, il obtient l'action de haut niveau à appliquer au plan de données.

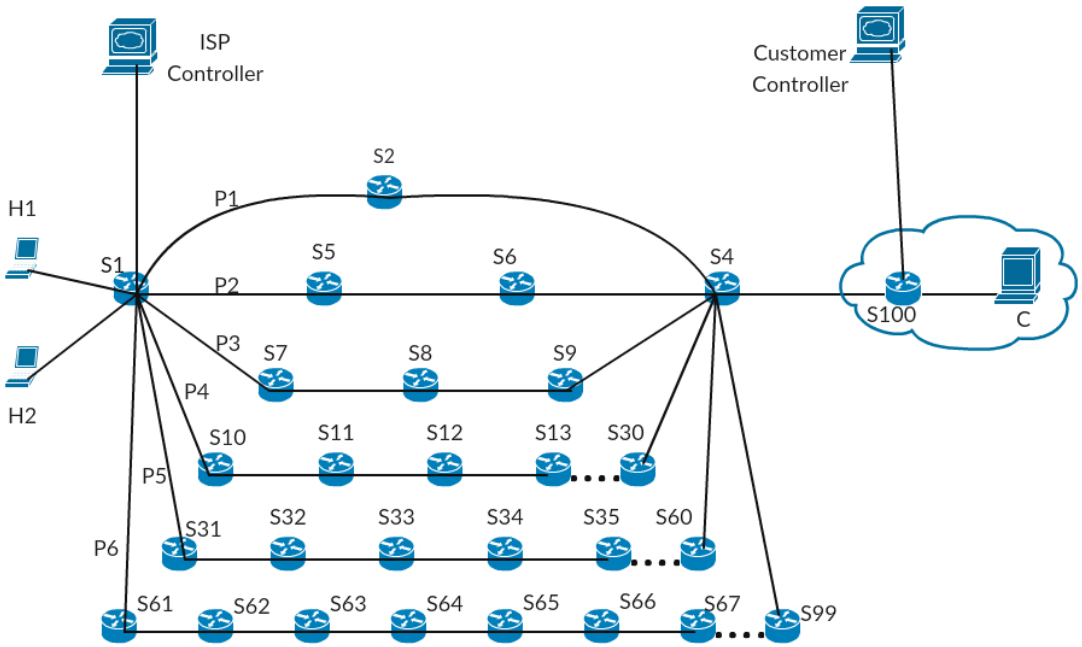

Figure A.5 – Déploiement de politiques dans le réseau

3. L'action de haut niveau est affinée pour obtenir le chemin d'accès au trafic en fonction des informations du flux et de la bande passante. Le PDP obtient les détails du chemin concret à partir de la base d'informations réseau (NIB) avec l'identifiant de commutation spécifique et le port de sortie pour orienter le flux.

4. Ensuite, le PDP spécifie au POI les détails du filtre pour capturer le flux d'interêt en utilisant les informations de ce flux, afin que celui-ci y associe les instrutions de bas niveau OpenFlow comprenant l'identifiant du commutateur (Switch ID) et les informations du port de sortie.

A.3.5 Cas d'Utilisation: Provision de QoS à la Demande

L'un des principaux éléments de notre mécanisme de traduction des politiques est qu'il permet au FAI de réagir automatiquement sur une requête de QoS envoyé par ses clients. Nous discutons ce cas d'utilisation sur la base du réseau présenté en Figure A.5. Ce réseau se compose de deux sous-réseaux distincts: un FAI et un sous-réseau client représenté par C. Deux hôtes externes sont dénotés H_1 et H_2. Ces hôtes externes génèrent du trafic à destination du réseau client. Le client demande au FAI d'activer la politique QoS souhaitée pour un flux spécifique en lui envoyant la classe du flux, la requête de QoS (largeur de bande passante), les informations de flux (adresses IP source et destination). Dans le scénario suivant, le client fait une requête de QoS à son FAI.

Requête QoS reçue depuis un client : Les requêtes de QoS à la demande émanent du client et instruisent le fournisseur de services de service une QoS différenciée. Cette requête contient les informations de flux et le type de service QoS demandé. Comme le montre le Listing A.3, la requête de QoS contient les informations suivantes: {IP source: 10.0.0.1, IP destination: 10.0.0.3}. L'adresse IP 10.0.0.1 représente l'hôte externe H_1. En outre, il contient la classe de QoS demandée ("Gold") et la classe de flux ("Legitimate"). Pour le type d'événement, "Gold_QoS" est mentionné dans le champ *Event* de la rubrique *Classification* de l'alerte.

Listing A.2 – Une politique QoS de haut niveau de classe Gold

```
<Policy PolicyID="QoS">
    <Event Type = "Gold_QoS">
    </Event>
    <Condition>
       <flow class="Legitimate">
       <bandwidth class="Gold">
    </Condition>
    <Actions action="Redirect"/>
    </Actions>
</Policy>
```

Politique de QoS Gold appliquée par le FAI : Le fichier de politique affiché dans le Listing A.2 fournit le chemin d'accès de classe Gold dans le réseau du FAI. Il contient les informations de type d'événement, p. ex., "Gold_QoS". Les conditions constraignent la stratégie pour une classe de flux légitimes ("Legitimate") et une classe de bande passante "Gold". La politique sélectionnée selon ces paramètres retourne l'action abstraite "Redirection". Cette action est ensuite affinée à l'aide de la Table A.1, décrite en détail dans la Section A.3.6. Une bande passante de 400 Mbps est provisionnée pour la classe "Gold".

Table A.1 – Table de correspondance entre largeurs de bande passante et chemins réseau

Path	Destination Network	Bandwidth Class
P1	10.0.0.3	Gold
P2	10.0.0.3	Silver
P3	10.0.0.3	Bronze

Listing A.3 – Requête de QoS envoyée par le client

```
<IDMEF-Message version="1.0">
<Alert>
    <Analyzer analyzerid="CUSTOMER_C"/>
    <Source>
       <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.1</address>
            </Address>
        </Node>
        <Service>
            <protocol>udp</protocol>
        </Service>
    </Source>
    <Target>
        <Node>
            <Address category="ipv4-addr">
                <address>10.0.0.3</address>
            </Address>
        </Node>
    </Target>
   <Classification event="Gold QoS">
   </Classification>
    <AdditionalData> type ="string" meaning="flow class">
    <string>Legitimate</string>
    </AdditionalData>
    <AdditionalData> type ="string" meaning="bandwidth request">
    <string>Gold</string>
    </AdditionalData>
</Alert>
</IDMEF-Message>
```

Listing A.4 – Instantiation du modèle pour diriger le trafic via un chemin QoS élevé

```
1 P1:{Switch:S1,output(5); Switch:S2, output(2); Switch:S4,output(2)}
2 P2:{Switch:S1,output(6); Switch:S5, output(2); Switch:S6, output(2); Switch:
    S4,output(2)}
3 P3:{Switch:S1, output(7); Switch:S7,output(2); Switch:S8,output(2); Switch:S9
    ,output(2); Switch:S4,output(2)}
4 Path={1:P1, 2:P2, 3:P3}
```

Instanciation du modèle pour acheminer le flux via un chemin de QoS élevée : La Tableau A.1 indique le chemin pour acheminer le flux à travers la classe de bande passante “Gold”, “Silver” ou “Bronze”. Des largeurs de 350 Mbps et 300 Mbps sont provisionnées pour les classes “Silver” et “Bronze”, respectivement. Les chemins sont définis comme des ensembles de couples (identifiant du commutateur:port de sortie). Le Listing A.4 fournit les détails du chemin concret.

A.3.6 Exemple étape par étape

Pour résumer cette section, nous fournissons un exemple de la façon dont une politique QoS de haut niveau est transformée en règles de bas niveau pour l’application. Ce cas d’utilisation est décrit en utilisant le réseau représenté sur la Fig. A.5. Le client C envoie la requête au FAI de fournir un chemin de QoS “Gold” aux flux qui arrivent à son réseau depuis l’hôte H_1. Comme la Liste A.5 le montre, les règles dans le commutateur S_1 pour transférer le flux à travers le chemin P_3 insérent une étiquette 3 dans le paquet via un port de sortie 7.

Listing A.5 – Règles OpenFlow dans le Switch S1 avant le déploiement de la Politique QoS

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:3->
    vlan_vid,output:7
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:3->
    vlan_vid,output:7
```

1. Le client C envoie le message de requête QoS au contrôleur du FAI pour provisionner le chemin de QoS “Gold” pour le trafic arrivant sur son réseau à partir de l’hôte H_1. La requête de QoS s’affiche dans le Listing A.3.

2. Le MC extrait le type d’événement (Gold QoS) et les conditions telles que la demande de bande passante (“Gold”) ainsi que la classe de flux ("Legitimate") et les transmet au PDP. Selon le type d’événement et les conditions, le PDP vérifie la base de règles pour obtenir les actions de haut niveau. Dans ce cas, l’action de haut niveau est “Redirect”. Le Listing A.2 montre la politique QoS de haut niveau pour la classe Gold.

3. Le PDP envoie l’action de haut niveau, les informations de flux (adresses IP source et destiation) et la classe de bande passante (Gold) à la NIB. Basé sur l’information sur le niveau de bande passante (Gold) et le réseau de destination, la NIB fournit le chemin vers

lequel le flux doit être redirigé, en se basant sur la Table A.1. Notez que, dans ce cas, le chemin est alors: (S_1:5, S_2:2, S_4:2). De plus, l'étiquette pour le chemin est 1.

4. Le PDP utilise l'action de haut niveau "Redirect" avec la bande passante demandée pour instancier le script dans le POI qui déploie les règles OpenFlow dans les commutateurs. Le Listing A.6 montre les règles de bas niveau dans le commutateur S_1 pour transférer le flux contenant identifié par {IP source: 10.0.0.1, IP destination: 10.0.0.3}, via le port de sortie 5 en insérant une étiquette 1. Donc, ce flux est autorisé à traverser le chemin P_1.

Listing A.6 – Règles OpenFlow dans le Switch S1 après le déploiement de la politique QoS

```
1 nw_src=10.0.0.2,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:8
2 nw_src=10.0.0.1,nw_dst=10.0.0.3,ip,actions=push_vlan:0x8100,set_field:1->
    vlan_vid,output:5
```

A.3.7 Conclusion

Dans cette section, nous avons décrit le langage de politique proposé. En outre, nous avons fourni un mécanisme pour traduire la politique de haut niveau pour le déploiement de règles OpenFlow de bas niveau dans les commutateurs. Le mécanisme est décrit dans le contexte d'un scénario de provisionnement de QoS. Dans la section suivante, nous présentons le cadre de gestion basé sur les politiques pour gérer le réseau du FAI de manière automatisée.

A.4 Cadre de Gestion et de Mise en Œuvre des Politiques de Sécurité SDN

Notre objectif est de fournir un système de gestion de politique dynamique et adaptable pour le réseau des FAI. Notre système de gestion de politiques spécifie les politiques de sécurité et de réseau de haut niveau et fournit une collaboration en temps réel avec les clients pour atténuer l'effet de la congestion dûe à une augmentation du trafic, des incidents réseau ou des attaques. Nous tirons parti du paradigme SDN et utilisons notre spécification de politiques et le mécanisme de traduction introduit dans la Section A.3 pour concevoir un système de gestion basé sur les politiques pour le réseau du FAI. Ce système permet au FAI de spécifier les politiques de sécurité et de réseau de haut niveau. En outre, les clients peuvent exprimer dynamiquement leurs exigences au FAI pour l'instanciation des politiques. Ces politiques peuvent être déployées dynamiquement dans les périphériques du plan de données. En particulier, le système de gestion des politiques dans le FAI tient compte de multiples facteurs tels que l'état actuel du réseau du FAI et les accords stratégiques avec ses clients. En outre, il doit s'adapter aux demandes de certains clients, de sorte que l'ingénierie du trafic effectuée pour un client n'aura pas d'impact sur le trafic và destination du réseau d'autres clients. Plus précisément, notre système de gestion des

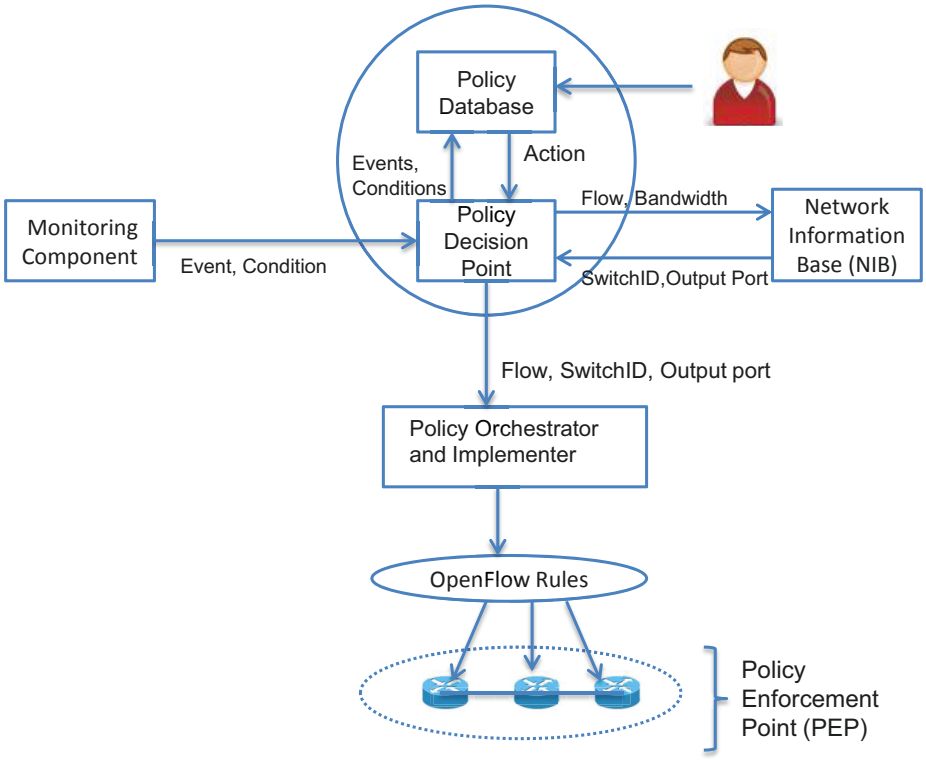

Figure A.6 – Flux opérationnel du cadre de gestion de politiques

politiques fournit une réponse automatisée collaborative et centrée sur l'utilisateur pour atténuer l'impact du trafic d'attaque et fournir un service garanti en QoS aux clients du FAI.

A.4.1 Composants Principaux

Les composants fonctionnels du système de gestion des politiques sont les suivants:

Composant de supervision (MC):

Le MC est responsable de la réception des alertes et des notifications provenant des différents clients. En particulier, il extrait les événements et les conditions utilisés par le PDP pour instancier la politique depuis la base de politiques.

Base de politiques (Policy DB)

Dans le cadre de gestion et d'application de la politique, la base de politiques est utilisée pour stocker les stratégies de haut niveau définies par l'administrateur réseau dans le réseau du FAI. Les politiques sont décrites selon le paradigme ECA (voir Section A.3 pour plus de détails).

Point de décision de la politique (PDP)

Ce composant fonctionne comme un orchestrateur des différents modules composant le cadre. C'est le point central de la logique du cadre.

Base d'informations réseau (NIB)

La NIB maintient une table contenant la liste des chemins en fonction de différentes largeurs de bande. La NIB est responsable du calcul du chemin en fonction de l'état du réseau du FAI. Selon les entrées (informations de flux, action de haut niveau et bande passante demandée), la NIB calcule le chemin. Les chemins sont calculés en utilisant le chemin le plus court en adéquation avec la politique [27], permettant de faire transiter le trafic via un chemin basé sur des stratégies prédéfinies. Les détails de l'algorithme de calcul du chemin sont décrits dans l'Algorithme 5. Une fois le chemin calculé, la NIB fournit la liste des couples commutateurs-ports de sortie ainsi qu'une en-tête spécifique, l'en-tête de service réseau (NSH) au PDP.

Par ailleurs, ce composant est responsable de surveiller l'état des commutateurs et des chemins d'accès dans le réseau du FAI et de fournir l'état du réseau (congestionné, normal) aux composants qui en ont besoin. Pour surveiller le réseau du FAI, nous pouvons exploiter un outil comme OpenNetMon [112] qui permet de maintenir une matrice de trafic des différents chemins et commutateurs dans le réseau.

Algorithm 5 *Calcul de chemin réseau*

```
1: procedure PATH_COMPUTE(flow,bw_req,action,step,Max_rate)
2:     hop ← 0
3:     path ← []
4:     for Flow F and each path p do
5:         p ← compute_Bandwidth(max(all_links)) //Takes the maximum bandwidth in the path
6:         d ← Hop_Count(step[i] + step[i+1])
7:         hop_count ← hop + d // Computes the hop count in the path
8:         path.addList(p)
9:         if (C − bw_req) ≥ Max_rate then //New flow should not impact other flows traversing the
   link.
10:            return hop_count, path
11:        else
12:            return No Paths are available
13:        end if
14:    end for
15: end procedure
```

Orchestrateur de politiques (POI)

Ce module génère les règles OpenFlow basées sur les informations fournies par le PDP. La description du module POI a été donnée dans la Section A.3.

Ci-dessous, nous décrivons le flux opérationnel de la gestion et l'application de politiques:

1. Un événement est déclenché au niveau du contrôleur du FAI lorsqu'une alerte de sécurité est reçue par le MC, provenant du contrôleur client. Les informations de flux (IP source, IP de destination), la gravité de l'impact du trafic sur le réseau client (faible, moyen, élevé), la classe de flux (légitime, suspecte ou malveillante) et les détails du type d'attaque sont extraites par le module MC [54];

2. Les informations extraites sont transmises au PDP [55]. Le PDP sélectionne l'action de haut niveau à partir de la base de politiques à appliquer sur le flux en fonction de l'événement et de certaines conditions. Ensuite, il transmet l'action de haut niveau, la bande passante demandée et les informations de flux à la NIB (qui permettent d'identifier les nœuds d'entrée et de sortie dans le réseau du FAI).

3. La NIB calcule un ou plusieurs chemins adaptés, à partir des informations reçues précédemment.

4. La NIB envoie au PDP les détails du chemin avec l'identifiant des commutateurs, les ports de sortie et l'en-tête de service réseau (NSH) (qui peut etre stocké dans le champ VLAN de la couche Ethernet) [95]. Le PDP transmet ces détails au module POI pour le déploiement des règles dans les commutateurs OpenFlow. Si tous les chemins du réseau du FAI sont encombrés, la NIB renvoie le message indiquant "aucun chemin d'accès n'est disponible". Dans ce cas, le PDP renvoie une action *drop* pour le flux concerné au POI afin de réduire le niveau de congestion dans le réseau.

5. Puis, le PDP instancie le script dans le POI en fonction de la bande passante demandée par le client. Le PDP transmet également l'identifiant du commutateur, le port de sortie et l'en-tête NSH au POI qui déploie les règles dans les commutateurs pour traiter le trafic.

A.4.2 Paramètres Expérimentaux

Les expériences ont été menées dans Mininet, un environnement de simulation d'un réseau de commutateurs OpenFlow. Le scénario expérimental est illustré dans la Figure A.7, dans laquelle nous supposons que le réseau du FAI contient 14 commutateurs et 6 chemins. La probabilité de perte liaison et la bande passante des différents chemins est supposée être comme dans le Tableau A.2. Par ailleurs, les alertes de sécurité reçues par le FAI sont représentées au format IDMEF [44].

Gigue du trafic légitime.

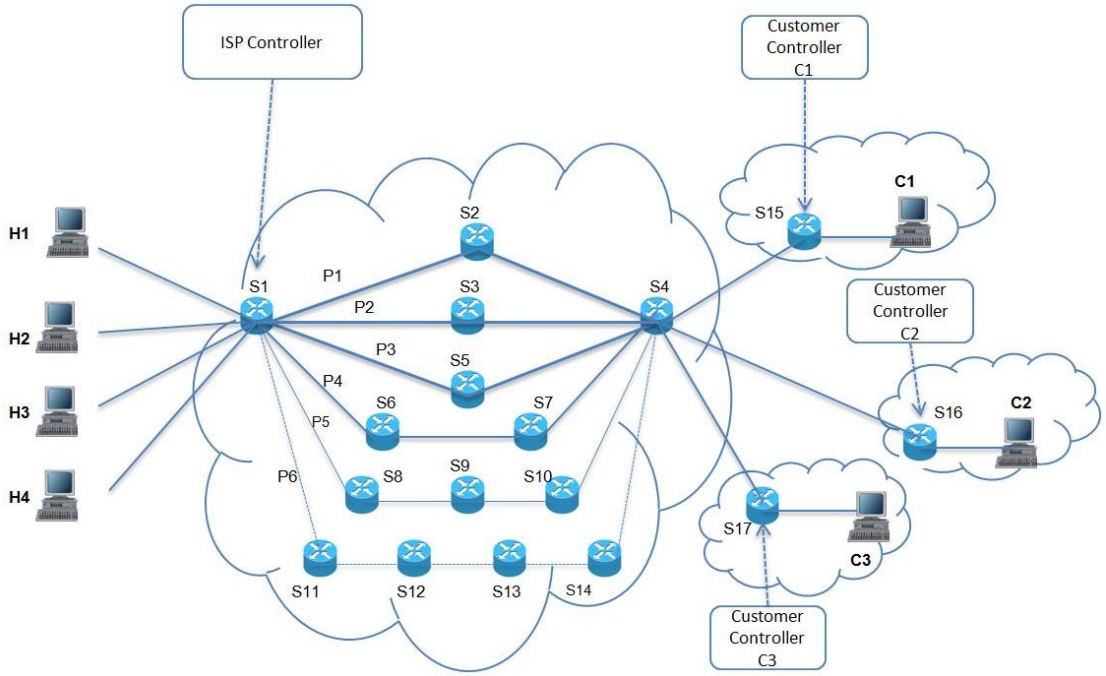

Figure A.7 – Scénario expérimental pour trois réseaux clients avec un réseau FAI.

Table A.2 – Bande passante et probabilité de perte de liaison pour les chemins réseau.

Paths	Bandwidth	Link Loss Percentage
P_1	400 Mbps	0
P_2	400 Mbps	0
P_3	400 Mbps	0
P_4	200 Mbps	3
P_5	100 Mbps	5
P_6	50 Mbps	7

Dans cette expérimentation, nous vérifions la fluctuation du signal du trafic légitime en présence de congestion. Comme le montre la Figure. A.7, nous avons utilisé l'hôte H_2 pour générer un trafic d'attaque DDoS et avons observé l'impact sur le trafic légitime destiné aux clients C_1, C_2 et C_3. Comme visible dans la Figure A.8, la gigue du réseau de tout le trafic légitime a fortement augmenté dès que H_2 a débuté son attaque. En conséquence, le contrôleur SDN du client C_3 envoie une alerteau *MC* contrôleur du FAI, ce qui déclenche la réaction du *PDP* qui décide de rediriger le flux. Par la suite, la *NIB* calcule le meilleur chemin, c'est-à-dire P_3 dans ce cas, et fournit les détails du chemin d'accès et le NSH au PDP.

Enfin, les règles OpenFlow correspondantes sont poussées sur les *PEPs*, à savoir les commutateurs OF, par le POI. Ainsi, le transport du trafic légitime vers C_3 a été rapidement rétabli à son niveau normal. De même, à la demande de C_2, le flux destiné C_2 est redirigé sur le chemin P_2, ce qui

Table A.3 – Classe de flux et réseau de destination de chaque flux client

Flow	Flow Class	Destination
H_1	Legitimate	C_1
H_2	Suspicious	C_1
H_3	Legitimate	C_2
H_4	Legitimate	C_3

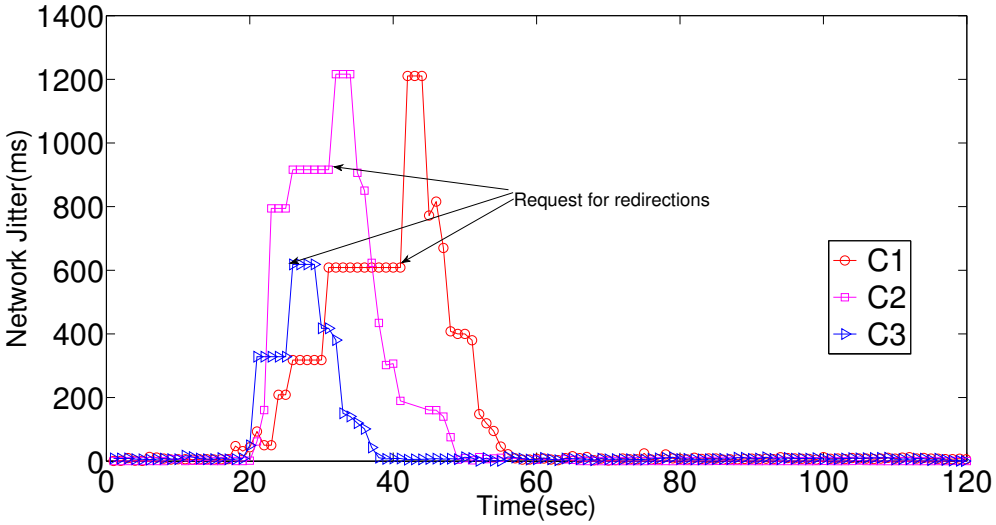

Figure A.8 – Gigue du trafic légitime

permet de restaurer rapidement la gigue de ce flux à un niveau normal. Enfin, le trafic de H_2 est redirigé sur le chemin dédié au trafic suspect, P_5, restaurant la gigue du trafic allant de H_1 à C_1 à un niveau normal. En ce qui concerne la gigue du trafic destiné à C_3, celle-ci a diminué plus tôt par rapport à celles de C_2 et C_1. Cela s'explique simplement par le fait que le client C_3 a envoyé son alerte plus tôt que les clients C_2 et C_1.

A.4.3 Conclusion

Dans cette section, nous avons décrit notre cadre de gestion des politiques SDN pour fournir des services automatisés de QoS aux clients du FAI. Le cadre fournit un algorithme de calcul de chemin qui prend en compte l'état du réseau du FAI pour réaliser des opérations d'ingénierie de trafic. Il réduit les dommages directs causés par le trafic d'attaque sur le trafic légitime. En outre, cela réduit également les dommages collatéraux causés par l'ingénierie de trafic effectuée pour un client sur le trafic à destination d'autres clients du FAI. De plus, l'en-tête NSH inséré par notre cadre permet aux middleboxes de ne pas être déployées en dehors du chemin des flux clients.

A.5 Conclusion

L'objectif principal de cette thèse était de proposer un mécanisme automatisé pour gérer les cyber-attaques. En particulier, certaines attaques ne ciblent pas seulement les fournisseurs d'infrastructures mais aussi les fournisseurs de services. Les cyber-attaques ciblant les deux peuvent ont le potentiel de gravement affecter l'expérience des utilisateurs, en particulier par rapport aux services exposées sur Internet. La disponibilité limitée des administrateurs réseau pour atténuer les attaques accroît encore l'impact du trafic d'attaque.

En termes de contributions, nous avons commencé cette thèse en examinant les mécanismes existants d'atténuation des cyber-attaques. Tout d'abord, nous avons présenté l'état de la cyber-défense autonome, ainsi que les mécanismes existants. Nous avons ensuite introduit le paradigm SDN et discuté de ses fonctionnalités. Plus précisément, nous avons examiné les mécanismes et les services de sécurité basés sur SDN. Nous avons décrit les mécanismes d'atténuation des attaques, de détection, de surveillance du trafic et d'ingénierie du trafic basés sur SDN, ainsi que leurs caractéristiques. Notre attention a porté sur la façon dont les mécanismes de sécurité peuvent être améliorés et étendus en utilisant la technologie SDN. Ainsi, nous avons présenté des techniques qui tirent avantage des caractéristiques du paradigme SDN telles que le découplage du plan de contrôle et du plan de données, la programmation et la visibilité globale du point de vue de la sécurité.

L'état de l'art du domaine de recherche a ainsi été complété par trois contributions principales. Dans un premier temps, nous avons proposé un cadre d'atténuation autonome qui fournit une réponse collaborative et automatisée. Le cadre est distribué entre le FAI et son client. Le cadre permet au client de demander des mesures d'atténuation lors de la détection d'une attaque. L'approche a été validée à l'aide de mininet, un logiciel de simulation d'un réseau SDN, ainsi que sur une plateforme physique constituée de commutateurs OpenFlow. Deuxièmement, nous avons présenté une grammaire permettant de représenter la politique de sécurité et la politique réseau de haut niveau dans le réseau du FAI. Nous avons introduit des modèles et des tableaux de stratégie pour faire mettre en correspondance les politiques de haut niveau d'un côté, et les règles OpenFlow de bas niveau de l'autre. Cette "traduction" facilite le déploiement automatisé d'une politique vers les périphériques du plan de données. En dernier lieu, nous avons proposé un cadre de gestion et d'application de la politique pour gérer le réseau du FAI et fournir des services de sécurité et de QoS à ses clients en tenant compte de l'état du réseau du FAI. L'objectif est de réduire l'impact des dommages directs causés par le trafic d'attaque à différents clients ainsi que les dommages collatéraux causés par la mesure d'atténuation du trafic d'attaque, destiné à un client, sur les autres clients du FAI. L'idée est de calculer les chemins des flux clients de manière dynamique en tenant compte de l'état du réseau du FAI, et de rediriger potentiellement ces trafics sur des chemins moins congestionnés. Un prototype de la gestion des politiques et du cadre d'application a été réalisé et validé dans mininet en tenant compte d'un scénario comprenant un réseau FAI et plusieurs clients.

Concernant les perspectives de recherche future, plusieurs travaux complémentaires sont envis-

agés. En effet, il convient de noter que les mécanismes de sécurité utilisant SDN ont un grand nombre de défis à relever. Cette thèse a traité, avec une portée limitée, certains des défis du sujet, en accordant une attention particulière à l'atténuation du trafic d'attaque d'une manière automatisée. Le système de réponse automatisé englobe de nombreux autres domaines qui doivent être gérés ensemble afin d'améliorer leur résilience contre les attaques et les abus.

Dans cette optique, une première perspective comprendrait une analyse plus approfondie de l'impact en matière de performance de l'utilisation de SDN dans le processus d'atténuation du trafic d'attaque. En effet, la performance de notre cadre autonome est une question importante qu'il est nécessaire d'aborder. Dans cette dissertation, nous avons élargi nos cas d'utilisations d'un seul FAI et un client unique à un scénario à plusieurs clients, tout en réduisant les dommages collatéraux de l'attaque. Cependant, l'impact du passage à l'échelle reste à considérer pour des raisons de performance.

Dans cette dissertation, nous avons supposé que le client utilise un mécanisme de détection pour détecter les attaques. Une détection précise et rapide de l'attaque est nécessaire pour une atténuation efficace. De plus, l'intégration d'un outil de détection sur le réseau client avec les composants d'atténuation dans le réseau du FAI est également une préoccupation. En outre, dans notre cadre, le composant de supervision doit être intégré du côté du FAI pour détecter la congestion dans le réseau du FAI et fournir cet état aux autres composants d'ingénierie du trafic. De même, la performance du cadre avec l'ensemble de ces composants intégrés doit être analysée, en particulier dans des scénarios avec plusieurs clients.

Bibliography

[1] Autonomic cyber security (acs):a paradigm shift in cyber security.

[2] Openflow in europe : Linking infrastructure and applications.

[3] S. Agarwal, M. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *2013 Proceedings IEEE INFOCOM*, pages 2211–2219, April 2013.

[4] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, pages 294–311, Berlin, Heidelberg, 2003. Springer-Verlag.

[5] F. Q. Ajay Tirumala, J. Jon, and Kevin. IPERF, 2003.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[7] AlgoSec. The State of Automation in Security. Technical report, AlgoSec, 2016.

[8] H. Aljifri, M. Smets, and A. Pons. IP Traceback using header compression. *Computers and Security*, 22(2):136 – 151, 2003.

[9] D. G. Andersen. Mayday: Distributed filtering for internet services. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 3–3, Berkeley, CA, USA, 2003. USENIX Association.

[10] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford. A slick control plane for network middleboxes. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 147–148, New York, NY, USA, 2013. ACM.

[11] Arbor Networks. Worldwide Infrastructure Security Report. Technical report, Arbor Networks, 2016.

[12] C. Argyropoulos, D. Kalogeras, G. Androulidakis, and V. Maglaris. Paflomon – a slice aware passive flow monitoring framework for openflow enabled experimental facilities. In *2012 European Workshop on Software Defined Networking*, pages 97–102, Oct 2012.

[13] C. Bai, G. Feng, and G. Wang. Algebraic geometric code based IP traceback. In *23rd IEEE International Conference on Performance, Computing, and Communications (IPCCC)*, pages 49–56, 2004.

[14] J. R. Ballard, I. Rae, and A. Akella. Extensible and scalable network monitoring using opensafe. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.

[15] A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou. Policy refinement for diffserv quality of service management. In *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.*, pages 469–482, May 2005.

[16] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. Policycop: An autonomic qos policy enforcement framework for software defined networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, Nov 2013.

[17] A. Belenky and N. Ansari. On Deterministic Packet Marking. *Comput. Netw.*, 51(10):2677–2700, July 2007.

[18] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein. Enforsdn: Network policies enforcement with sdn. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 80–88, May 2015.

[19] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 151–152, New York, NY, USA, 2013. ACM.

[20] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM.

[21] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer. Position paper: Software-defined network service chaining. In *2014 Third European Workshop on Software Defined Networks*, Sept 2014.

[22] V. Bollapragada, R. White, and C. Murphy. *Inside Cisco IOS Software Architecture*. Cisco Press, 1 edition, 2008.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[24] R. Braga, E. Mota, and A. Passito. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *35th IEEE Conference on Local Computer Networks (LCN)*, pages 408–415, Oct 2010.

[25] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 15–28, Berkeley, CA, USA, 2005. USENIX Association.

[26] Z. Cao, M. Kodialam, and T. V. Lakshman. Traffic steering in software defined networks: Planning and online routing. *SIGCOMM Comput. Commun. Rev.*, 44(4):–, Aug. 2014.

[27] Z. Cao, M. Kodialam, and T. V. Lakshman. Traffic Steering in Software Defined Networks: Planning and Online Routing. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, DCC '14, pages 65–70, New York, NY, USA, 2014. ACM.

[28] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, Aug. 2007.

[29] H. Chen, Y. B. Al-Nashif, G. Qu, and S. Hariri. Self-configuration of network security. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 97–97, Oct 2007.

[30] S. Chen and Q. Song. Perimeter-based defense against high bandwidth ddos attacks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):526–537, June 2005.

[31] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman. Policy refinement: Decomposition and operationalization for dynamic domains. In *2011 7th International Conference on Network and Service Management*, pages 1–9, Oct 2011.

[32] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[33] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, UK, 2001. Springer-Verlag.

[34] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. Tapas: A tool for rapid prototyping of adaptive streaming algorithms. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, VideoNext '14, pages 1–6, New York, NY, USA, 2014. ACM.

[35] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental), Mar. 2007.

[36] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. In *35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 159–166, 2003.

[37] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. SPHINX: detecting security attacks in software-defined networks. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[38] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. *SIGCOMM Comput. Commun. Rev.*, 41(4):362–373, Aug. 2011.

[39] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 362–373, New York, NY, USA, 2011. ACM.

[40] L. Dunbar, M. Zarny, C. Jacquenet, and S. Chakrabarty. Interface to Network Security Functions Problem Statement. Internet-Draft dunbar-i2nsf-problem-statement-01, IETF, September 2014.

[41] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic ddos defense. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 817–832. USENIX Association, Aug. 2015.

[42] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546, Berkeley, CA, USA, 2014. USENIX Association.

[43] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 19–24, New York, NY, USA, 2013. ACM.

[44] B. Feinstein, D. Curry, and H. Debar. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, Oct. 2015.

[45] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Hierarchical policies for software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 37–42, New York, NY, USA, 2012. ACM.

[46] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker. Frenetic: A high-level language for openflow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 6:1–6:6, New York, NY, USA, 2010. ACM.

[47] J. Gao, C. Xia, S. Wang, and H. Zhang. A sdn-based deployment framework for computer network defense policy. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 01, pages 1253–1258, Dec 2015.

[48] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 7–12, New York, NY, USA, 2012. ACM.

[49] K. Giotis, G. Androulidakis, and V. Maglaris. Leveraging sdn for efficient anomaly detection and mitigation on legacy networks. In *Proceedings of the 2014 Third European Workshop on Software Defined Networks*, EWSDN '14, pages 85–90, Washington, DC, USA, 2014. IEEE Computer Society.

[50] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62(0):122 – 136, 2014.

[51] GSMA09. Inter-Operator IP Backbone Security Requirements For Service Providers and Inter-operator IP backbone Providers 2.1. Technical report, GSMA Association, 2009.

[52] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[53] C. W. Haas, D. C. Salerno, and D. Sheinbein. Stored program controlled network: 800 service using spc network capability x2014; network implementation and administrative functions. *The Bell System Technical Journal*, 61(7):1745–1757, Sept 1982.

[54] N. Hachem, H. Debar, and J. Garcia-Alfaro. HADEGA: A novel MPLS-based mitigation solution to handle network attacks. In *31st IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 171–180, Dec 2012.

[55] S. Herzog. The COPS (Common Open Policy Service) Protocol. RFC 2748, Jan. 2000.

[56] S. Homma, D. Lopez, D. Dolson, A. Gorbunov, N. Leymann, K. Naito, M. Stiemerling, P. Bottorff, and don.fedyk@hpe.com. Analysis on Forwarding Methods for Service Chaining. Internet-Draft draft-homma-sfc-forwarding-methods-analysis-05, Internet Engineering Task Force, Aug. 2016. Work in Progress.

[57] H.-C. Hsiao, T. H.-J. Kim, S. B. Lee, X. Zhang, S. Yoo, V. Gligor, and A. Perrig. STRIDE: Sanctuary Trail – Refuge from Internet DDoS Entrapment. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, May 2013.

[58] T. Iimura, D. Miyamoto, H. Tazaki, and Y. Kadobayashi. Necomatter: curating approach for sharing cyber threat information. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, page 19. ACM, 2014.

[59] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2002.

[60] A. Keromytis, V. Misra, and D. Rubenstein. Sos: an architecture for mitigating ddos attacks. *Selected Areas in Communications, IEEE Journal on*, 22(1):176–188, Jan 2004.

[61] P. Kijewski and P. Pawliński. Proactive detection and automated exchange of network security incidents. *Abgerufen am*, 20, 2014.

[62] G. Koutepas, F. Stamatelopoulos, and B. Maglaris. Distributed management architecture for cooperative detection and reaction to ddos attacks. *J. Netw. Syst. Manage.*, 12(1):73–94, Mar. 2004.

[63] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, Jan. 1986.

[64] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[65] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[66] R. Krishnan and M. Durrani. Real-time SDN Analytics for DDoS mitigation, 2014.

[67] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. *IEEE/ACM Trans. Netw.*, 21(6):2001–2014, Dec. 2013.

[68] T. Kuhn. A survey and classification of controlled natural languages. *Comput. Linguist.*, 40(1):121–170, Mar. 2014.

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

[70] A. Lara and B. Ramamurthy. Opensec: Policy-based security using software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):30–42, March 2016.

[71] A. A. Lazar, K.-S. Lim, and F. Marconcini. Realizing a foundation for programmability of atm networks with the binding architecture. *IEEE Journal on Selected Areas in Communications*, 14(7):1214–1227, Sep 1996.

[72] S. B. Lee, M. S. Kang, and V. D. Gligor. Codef: Collaborative defense against large-scale link-flooding attacks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 417–428, New York, NY, USA, 2013. ACM.

[73] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei. Drawbridge: Software-defined DDoS-resistant Traffic Engineering. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 591–592, New York, NY, USA, 2014. ACM.

[74] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang. A sdn-oriented ddos blocking scheme for botnet-based attacks. In *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 63–68, July 2014.

[75] X. Liu, X. Yang, and Y. Xia. NetFence: preventing internet denial of service from inside out. *SIGCOMM Comput. Commun. Rev.*, 41(4), Aug. 2010.

[76] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt. Towards sla policy refinement for qos management in software-defined networking. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 397–404, May 2014.

[77] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. Policy authoring for software-defined networking management. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 216–224, May 2015.

[78] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *Proceedings of the 4th USENIX Conference on Networked Systems Design Implementation (NSDI)*, pages 24–24, Berkeley, CA, USA, 2007. USENIX Association.

[79] J. Mazel, R. Fontugne, and K. Fukuda. A taxonomy of anomalies in backbone network traffic. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, pages 30–36, Aug 2014.

[80] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[81] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[82] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting Traffic Anomaly Detection Using Software Defined Networking. In R. Sommer, D. Balzarotti, and G. Maier, editors, *Recent Advances in Intrusion Detection*, volume 6961 of *Lecture Notes in Computer Science*, pages 161–180. Springer Berlin Heidelberg, 2011.

[83] S. A. Mehdi, J. Khalid, and S. A. Khayam. *Revisiting Traffic Anomaly Detection Using Software Defined Networking*, pages 161–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[84] S. Morgan. Cybercrime costs projected to reach 2 trillion dollar by 2019, 2016.

[85] T. Nadeau and P. Quinn. Problem Statement for Service Function Chaining. RFC 7498, Nov. 2015.

[86] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. Netfpga: Reusable router architecture for experimental research. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '08, pages 1–7, New York, NY, USA, 2008. ACM.

[87] Open Networking Foundation. SDN Security Considerations in the Data Center. Technical report, ONF, 2012.

[88] Open Networking Foundation. Software-Defined Networking:The New Norm for Networks. Technical report, ONF, 2012.

[89] Open Networking Foundation. OpenFlow Switch Specification Version 1.4.0. Technical report, ONF, 2013.

[90] Open Networking Foundation. SDN Security Considerations in the Data Center. Technical report, ONF, 2013.

[91] Z. Pan, S. Hariri, and Y. Al-Nashif. Anomaly based intrusion detection for building automation and control networks. In *2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA)*, pages 72–77, Nov 2014.

[92] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. Cossack: Coordinated suppression of simultaneous attacks. In *DISCEX*, 2003.

[93] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. *SIGCOMM Comput. Commun. Rev.*, 43(4):27–38, Aug. 2013.

[94] G. Qu, S. Hariri, S. Jangiti, J. Rudraraju, S. Oh, S. Fayssal, G. Zhang, and M. Parashar. Online monitoring and analysis for self-protection against network attacks. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 324–325, May 2004.

[95] P. Quinn and U. Elzur. Network Service Header. Internet-Draft draft-ietf-sfc-nsh-05, Internet Engineering Task Force, May 2016. Work in Progress.

[96] A. Research. Security policy orchestration and automation to next-generation cybersecurity for enterprises. Technical report, ABI Research, 2016.

[97] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. *SIGCOMM Comput. Commun. Rev.*, 30(4):295–306, Aug. 2000.

[98] E. J. Scheid, C. C. Machado, R. L. dos Santos, A. E. Schaeffer-Filho, and L. Z. Granville. Policy-based dynamic service chaining in network functions virtualization. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 340–345, June 2016.

[99] D. Schnakengerg, H. Holliday, R. Smith, K. Djahandari, and D. Sterne. Cooperative Intrusion Traceback and Response Architecture (CITRA). In *Proceedings of the DARPA Information Survivability Conference & Exposition II*, DISCEX, pages 56–68. IEEE, June 2001.

[100] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys Tutorials*, 17(1):469–492, Firstquarter 2015.

[101] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, 2013.

[102] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang. Traffic engineering in software-defined networking: Measurement and management. *IEEE Access*, 4:3246–3256, 2016.

[103] S.McGillicuddy. Radware Adds Open Source DDoS Protection to OpenDaylight Project. Technical report, Radware, 2013.

[104] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based ip traceback. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 3–14, New York, NY, USA, 2001. ACM.

[105] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster. Managing the network with merlin. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pages 24:1–24:7, New York, NY, USA, 2013. ACM.

[106] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, Feb. 2004.

[107] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 15–15, 2000.

[108] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan 1997.

[109] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. *OpenTM: Traffic Matrix Estimator for OpenFlow Networks*, pages 201–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[110] B. K. Tripathy, A. G. Sethy, P. Bera, and M. A. Rahman. A novel secure and efficient policy management framework for software defined network. In *2016 IEEE 40th Annual Computer Software and Applications Conference*, volume 2, pages 423–430, June 2016.

[111] Y. Tseng, Z. Zhang, and F. Naït-Abdesselam. Srv: Switch-based rules verification in software defined networking. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 477–482, June 2016.

[112] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, May 2014.

[113] D. C. Verma. *Policy-Based Networking: Architecture and Algorithms*. New Riders Publishing, Thousand Oaks, CA, USA, 2000.

[114] A. Voellmy and P. Hudak. Nettle: Taking the sting out of programming network routers. In *Proceedings of the 13th International Conference on Practical Aspects of Declarative Languages*, PADL'11, pages 235–249, Berlin, Heidelberg, 2011. Springer-Verlag.

[115] A. Voellmy, H. Kim, and N. Feamster. Procera: A language for high-level reactive network control. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[116] B. Wang, P. Zhu, Q. Wen, and X. Yu. A honeynet-based firewall scheme with initiative security strategies. In *2009 International Symposium on Computer Network and Multimedia Technology*, pages 1–4, Jan 2009.

[117] K. Wang, J. d. Wang, L. q. Shen, and Z. h. Han. An intelligent security defensive software scheme and realization. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 793–796, April 2010.

[118] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang. Netfuse: Short-circuiting traffic surges in the cloud. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3514–3518, June 2013.

[119] M. Wichtlhuber, R. Reinecke, and D. Hausheer. An sdn-based cdn/isp collaboration architecture for managing high-volume flows. *IEEE Transactions on Network and Service Management*, 12(1):48–60, March 2015.

[120] A. Yaar, A. Perrig, and D. Song. Pi: a path identification mechanism to defend against DDoS attacks. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 93–107, May 2003.

[121] A. Yaar, A. Perrig, and D. Song. SIFF: a stateless Internet flow filter to mitigate DDoS flooding attacks. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 130–143, May 2004.

[122] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 35(4):241–252, Aug. 2005.

[123] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2):136–141, February 2013.

[124] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, Lombard, IL, 2013. USENIX.

[125] M. Yu, Y. Zhang, J. Mirkovic, and A. Alwabel. SENSS: Software Defined Security Service. In *Presented as part of the Open Networking Summit 2014 (ONS 2014)*, Santa Clara, CA, 2014. USENIX.

[126] C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, and C. YanRen. A novel design for future on-demand service and security. In *2010 IEEE 12th International Conference on Communication Technology*, pages 385–388, Nov 2010.

[127] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula. Steering: A software-defined networking for inline service chaining. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, Oct 2013.

[128] H. Zhao, J. Lobo, A. Roy, and S. M. Bellovin. Policy refinement of network services for manets. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 113–120, May 2011.