AN INTELLIGENT SECURITY ARCHITECTURE FOR SDN-ASSISTED IOT NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMED DEMIRPOLAT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JANUARY 2021

Approval of the thesis:

AN INTELLIGENT SECURITY ARCHITECTURE FOR SDN-ASSISTED IOT NETWORKS

submitted by **AHMED DEMIRPOLAT** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oguztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ertan Onur
Computer Engineering, METU _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Sevil Şen
Computer Engineering, Hacettepe University _____

Assist. Prof. Dr. Hande Alemdar
Computer Engineering, METU _____

Assist. Prof. Dr. Mehmet Demirci
Computer Engineering, Gazi University _____

Date: January 26, 2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Ahmed Demirpolat

Signature :

ABSTRACT

AN INTELLIGENT SECURITY ARCHITECTURE FOR SDN-ASSISTED IOT NETWORKS

Demirpolat, Ahmed
Ph.D., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Pelin Angın

January 2021, 116 pages

The rise of the Internet of Things (IoT) paradigm in the past decade has had a significant impact on all aspects of our lives through the many use cases it has made possible, including smart farming, smart homes, and remote healthcare services, among many others. While the number of smart devices and utilization scenarios aimed at supporting them grow exponentially, the large attack surface created by the interconnectivity of millions of these devices is a concerning aspect that needs to be addressed with intelligent intrusion detection and prevention techniques. This dissertation proposes a highly available software-defined network-based intelligent security architecture for IoT networks. It utilizes a weighted average ensemble model, comprised of a *few-shot* learning classifier, namely Prototypical Networks, and Support Vector Machines (SVM), for highly accurate intrusion detection. Also, we propose to deploy the SDN controller and network function virtualization (NFV) solutions as micro-services into a Kubernetes cluster in a public cloud to provide high availability and uptime. We evaluate the attack detection performance of the proposed model with the recently released Bot-IoT dataset consisting of real-world IoT network flows, as well as an SDN dataset we generated and the UNSW-NB15 intrusion detection dataset, and show that

the proposed model achieves significantly better performance than state-of-the-art machine learning models for intrusion detection in the absence of large amounts of sample attacks in the training data. We also experimented with the attack mitigation module's performance in a Kubernetes cluster in the public cloud, with end-to-end tests. By building up different network topologies, we showed the efficacy of the proposed solution not only with the attack detection tests but also with the attack prevention scenarios. Besides the time measurements in preventing cyber-attacks, we observed the effects of the proposed security mechanism on normal traffic and proved that the proposed solution does not cause an additional burden on the SDN controller. The proposed architecture is promising to achieve intelligent security in the future's ubiquitous IoT networks with its low processing overhead and high intrusion detection accuracy.

Keywords: software-defined networking, prototypical networks, security, internet of things

ÖZ

YTA DESTEKLİ NESNELERİN INTERNETİ AĞLARI İÇİN AKILLI GÜVENLİK MİMARİSİ

Demirpolat, Ahmed
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Pelin Angın

Ocak 2021 , 116 sayfa

Son on yılda Nesnelerin İnterneti paradigmasının yükselmesi, akıllı tarım, akıllı evler ve diğerlerinin yanı sıra uzaktan sağlık hizmetleri de dahil olmak üzere mümkün kılınan birçok kullanım örneği aracılığıyla hayatımızın tüm yönleri üzerinde önemli bir etki yaratmıştır. Akıllı cihazların sayısı ve bunları desteklemeyi amaçlayan kullanım senaryoları katlanarak büyürken, bu cihazların milyonlarcasının birbirine bağlanabilirliği ile yaratılan geniş saldırı yüzeyi, akıllı saldırı tespit ve önleme teknikleriyle ele alınması gereken bir husustur. Bu tez, Nesnelerin İnterneti ağları için yazılım tanımlı ağ (YTA) tabanlı bir akıllı güvenlik mimarisi önermektedir. Saldırıların son derece hassas bir şekilde tespit edilmesi için *az vuruşlu* bir öğrenme sınıflandırıcısı olan Prototip Ağlar ve Destek Vektör Makineleri içeren ağırlıklı bir ortalama topluluk modeli kullanır. Ayrıca, yüksek kullanılabilirlik ve çalışma süresi sağlamak için YTA denetleyicisini ve ağ işlevi sanallaştırma çözümlerini birer mikro-servis olarak genel bulutta bir Kubernetes kümesine yüklemeyi öneriyoruz. Önerilen modelin saldırı algılama performansını, gerçek dünya IoT ağ akışlarından oluşan Bot-IoT veri kümesinin yanı sıra ürettiğimiz bir YTA veri kümesi ve UNSW-NB15 saldırı tes-

pit veri kümesiyle değerlendiriyoruz ve önerilen modelin eğitim kümesinde büyük miktarlarda örnek saldırı verisi olmadığında saldırı tespiti için son teknoloji makine öğrenme modellerinden çok daha iyi performans elde ettiğini gösteriyoruz. Ayrıca, uçtan uca testlerle, genel buluttaki bir Kubernetes kümesinde saldırı engelleme modülünün performansını da test ettik. Farklı ağ topolojileri oluşturarak, önerilen çözümün etkinliğini yalnızca saldırı tespit testleriyle değil, saldırı önleme senaryolarıyla da gösterdik. Siber saldırıları önlemeye yönelik zaman ölçümlerinin yanı sıra önerilen güvenlik mekanizmasının normal trafik üzerindeki etkilerini de gözlemledik ve önerilen çözümün YTA denetleyicisi üzerinde ek bir yük oluşturmadığını kanıtladık. Önerilen mimari, düşük işleme yükü ve yüksek saldırı tespit doğruluğu ile geleceğin her yerde bulunan Nesnelerin İnterneti ağlarında akıllı güvenlik elde etmeyi vaat etmektedir.

Anahtar Kelimeler: yazılım tanımlı ağlar, prototip ağlar, güvenlik, nesnelerin interneti

To my beloved sons Eren and Kerem .. hope they stick to the light of science for a bright future ..

ACKNOWLEDGMENTS

The thought of expressing my acknowledgments has made me feel closer to the end of a long run. It was a long bumpy ride, but it was quite enjoyable! The success of this thesis has been accomplished with the invaluable help of many people worth mentioning whom I feel indebted.

First of all, I sincerely thank my thesis advisor Assist.Prof.Dr. Pelin Angın for her outstanding support and assistance. Her guidance has always helped me through my research and writing this dissertation. I can't imagine how I would complete this thesis without her.

In addition to my advisor, I would especially like to express my great appreciation to Prof.Dr. Ertan Onur. It was his support and suggestions that encouraged me to expand my work in various ways.

I would also like to thank other jury members of my Ph.D. thesis, Assoc.Prof.Dr. Sevil Şen, Assist.Prof.Dr. Hande Alemdar and Assist.Prof.Dr. Mehmet Demirci for their precious discussions and suggestions about my thesis.

My colleague, Savaş Güven, deserves a special mention for his support, tolerance, and suggestions throughout my work. It is an honor and a rewarding experience for me to work with him during probably one of the most challenging times of my life.

I also want to thank my wife, Melek Demirpolat, for her inseparable support. She has always given me help and encouragement. Many thanks to her endless patience and love. I would like to thank everyone who contributed to this thesis's successful implementation and apologize whom I could not personally mention.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

PROTONET	Prototypical Networks
SDN	Software Defined Networking
NFV	Network Function Virtualization
ONF	Open Networking Foundation
IoT	Internet of Things
IIoT	Industrial Internet of Things
DoS	Denial of Service
DDoS	Distributed Denial of Service
SVM	Support Vector Machines
CNN	Convolutional Neural Networks
DAE	Deep Auto Encoders
NB	Naïve Nayes
FP	False Positives
FN	False Negatives
TP	True Positives
TN	True Negatives
ACCS	Australian Cyber Security Center
OF	Open Flow
KPI	Key Performance Indicators
CNCF	Cloud Native Computing Foundation
AKS	Azure Kubernetes Service
HPA	Horizontal Pod Auto-scaler

CHAPTER 1

INTRODUCTION

The exponential growth of the Internet of Things (IoT) in the past decade with the advances in computing and communications infrastructures has dramatically changed our lives with the many use cases IoT supports. The IoT paradigm has made applications like smart homes, remote healthcare monitoring devices, smart energy grids, smart cities, connected vehicles, smart farming, and surveillance/safety warning systems for emergency response, among many others possible and the number of use cases it will support in the future is increasing every day.

As the number of connected instruments and use cases that they plan to serve are increasing at a tremendous speed, legacy mobile networks are falling short of achieving the bandwidth, delay, security requirements, and throughput of various IoT systems. As a major difference from existing networks, next-generation networks will significantly rely on newly introduced networking technologies such as software-defined networking (SDN) and network functions virtualization (NFV) for effective dynamic resource allocation and end-to-end operation.

While the IoT networks are designed for an infinite amount of devices and small data packets, SDN is the best IoT adaptability approach for rapidly resolving the shift around connecting an endless number of devices. One of SDN's key advantages is that not every single system requires individual configuration. This setup is done through a centralized controller. Also, SDN ensures accessibility, virtualization, and utilization of infrastructure [3]. While SDN-assisted IoT networks are highly promising to achieve the performance required by advanced autonomous systems of the future, their large-scale use faces security issues as described below.

1.1 Security Challenges in SDN-assisted IoT Networks

Although the many economic and societal benefits of SDN-assisted IoT continue to have wide impacts in our lives, the new security issues that have arisen due to the technology are concerning, standing in the way of wider adoption [4]. IoT-enabled devices have already created a large attack surface for hackers to exploit. Examples of notorious real-world IoT attacks include the massive distributed denial of service (DDoS) attack against a major DNS provider (Dyn), which hackers initiated through thousands of IP security cameras, and the destruction of nuclear centrifuges with the Stuxnet virus through causing them to spin out of control, among many other incidents.

Especially when we consider Industrial Internet of Things (IIoT), which involves organizations in a variety of sectors including government, energy, utilities, finance and healthcare, the possible serious consequences of attacks on IoT become more obvious. In the case of IIoT, vulnerabilities could affect safety and availability, just like in the example of attackers disrupting power supplies of a whole city or even country by hacking into their smart energy grids.

Achieving highly secure IoT networks is a difficult task due to the additional vulnerabilities introduced by the nature of IoT in addition to existing ones. Those vulnerabilities peculiar to IoT include things like the lack of even basic security primitives on many IoT devices, the lack of security standards for IoT protocols and the limited computation capacity and battery life of many IoT devices, rendering them incapable of performing complex cryptographic operations.

New communication protocols such as MQTT, CoAP, BLE, 6LoWPAN, Zigbee etc. at different network layers, introduced with IoT add to the vulnerabilities of these networks, as each protocol creates its own attack surface, unprotected by mechanisms designed for traditional network architectures. In order to mitigate the many undesirable consequences of attacks IoT networks face, we need techniques to detect intrusions in near real-time and quickly reconfigure the network to thwart the attacks.

The past decade has witnessed a overwhelming utilization of artificial intelligence techniques for intrusion detection on networking environments, as attacks have be-

come more complex, and legacy attack signature-based solutions fail to effectively detect previously unobserved attacks or those that dynamically adapt their behavior in response to the actions in the operation environment.

Both classical machine learning algorithms such as Support Vector Machines (SVM) [5] and deep learning models such as convolutional neural networks (CNNs) [6] and auto-encoders [7] have achieved very high (above 90% in most cases) attack detection rates on well-known intrusion detection datasets for legacy networks. Despite the promise of machine learning techniques to facilitate automated, high-performance detection of attacks in computer networks, an important issue with their usage is the requirement of abundant labeled data for different attack types, generated by taking into consideration the properties of the network in which they will operate.

Effective intrusion detection and prevention in IoT faces many challenges, even in the presence of fast stream data analytics tools and machine learning algorithms promising high accuracy. Successful machine learning-based approaches for detecting anomalies in legacy enterprise networks [8] and particular classes of IoT networks such as industrial IoT (IIoT) systems [9], and sensor data publish-subscribe systems [10] exist, however a solely anomaly-based approach that does not detect the classes of attacks makes it difficult to construct an effective automated mitigation strategy. Individual analysis of millions of network data records/system-generated alarms by expert analysts for identifying attack types and taking appropriate actions is not a feasible option, especially when the high volume and velocity of the data generated by 5G-based IoT networks are considered.

Existing approaches for intrusion detection in IoT networks, including the ones based on SDN, have mostly utilized datasets generated for legacy networks of 20 years ago, which may not reflect the true nature of the complex IoT networks today [11]. This is mostly because the generation of quality, large-scale datasets for use in the supervised machine learning process for intrusion detection can be a daunting task, especially when we consider the need to label each network flow with its correct class using expert opinion. As new types of attacks arise, we need to update our models with labeled data for these attacks as well, and this becomes a difficult process if using data-hungry learning algorithms such as different classes of deep neural networks.

1.2 Background

With the increasingly widespread deployment of IoT networks in the past decade, considerable research efforts have been dedicated to real-time detection and mitigation of cyber attacks on these systems. Recent research has mostly focused on the utilization of classical machine learning and deep learning techniques to create IoT intrusion detection systems that are capable of forming more generic representations of different classes of attacks compared to attack signature-based techniques.

Mehmood et al. [12] proposed the use of Naïve Bayesian classification in a multi-agent environment for securing IoT against DDoS attacks and achieved high accuracy on the NSL-KDD dataset [13].

Almiani et al. [14] developed a deep recurrent neural networks based approach for intrusion detection in fog-enabled IoT, achieving above 90% precision and recall for major attack classes in the NSL-KDD dataset.

Otoum et al. [15] proposed a deep learning-based intrusion detection framework, stacked-deep polynomial network, for IoT security, which achieved above 99% accuracy on the NSL-KDD dataset.

SDN has emerged as an important tool for fast detection and isolation of various types of attacks in IoT. The vast majority of the SDN-assisted IoT security approaches in the literature have focused on the detection and mitigation of denial of service (DoS) and distributed denial of service (DDoS) attacks.

Ravi and Shalinie [16] proposed an SDN-based architecture for detecting DDoS attacks in IoT networks using the semisupervised deep extreme learning machine. Their proposed approach was shown to achieve above 96% detection accuracy on two attack datasets.

Ahmed and Kim [17] proposed an inter-domain information exchange approach that utilizes statistics from various SDN domains to effectively mitigate DDoS attacks.

Yin et al. [18] proposed an approach for DDoS mitigation that measures the cosine similarity of the Packet_In rate at the SDN controller and drops packets when a spe-

cific threshold is exceeded.

Galeano-Brajones et al. [19] proposed an entropy-based approach for detecting DoS and DDoS attacks in SDN-based IoT networks, which achieved high detection rates on two datasets.

Sharma et al. [20] utilized deep belief networks to also detect DDoS attacks with a software-defined IoT architecture.

Bull et al. [21] proposed an SDN gateway approach for blocking anomalous flows in IoT networks and showed it is successfully able to mitigate TCP and ICMP flooding attacks. Although most of these approaches have achieved successful detection results, they only focus on detecting and mitigating DoS/DDoS attacks.

Li et al. [22] proposed a two-stage intrusion detection approach for software-defined IoT networks, where the first stage uses the BAT algorithm with swarm division and binary differential mutation for feature selection and the second stage uses the random forest algorithm to classify network flows. The model was shown to achieve higher detection accuracy than state-of-the-art models on the KDD Cup'99 dataset.

Dawoud et al. [23] proposed the utilization of Restricted Boltzmann Machines (RBM) based deep learning to detect attacks in software defined IoT networks and achieved higher detection performance than state-of-the-art machine learning models on the KDD Cup'99 dataset.

Amangele et al. [24] proposed a hierarchical decision tree-based framework with for IoT anomaly detection in SDN, where the first phase detects potentially malicious traffic based on flow data and suspicious packets are forwarded to the next stage for detailed analysis and attack classification.

Muthanna et al. [25] proposed a blockchain-based approach for achieving secure and reliable IoT networks with SDN and fog computing. Their model was shown to achieve high performance in terms of latency and resource utilization, however no intrusion detection analysis was performed.

Grigoryan et al. [26] proposed a cooperative IoT security model with SDN, where controller peers share information to mitigate detected attacks in IoT at a large scale.

The proposed model was shown to achieve fast failover, however no specific intrusion detection mechanisms were described.

Al Hayajneh et al. [27] proposed an SDN-based approach for mitigating man-in-the-middle attacks in IoT, which focused on IoT devices only communicating over HTTP.

A comparative summary of previous approaches in SDN-based IoT security is provided in Table 1.1. A major shortcoming of the previous machine learning based intrusion detection approaches in software-defined IoT is that they mostly assume the presence of large labeled datasets containing sufficiently many training samples for the attack classes they aim to detect.

Table 1.1: Summary of related work in SDN-based IoT security.

Author(s)	**Intrusion detection/ prevention method**	**Evaluation datasets**	**Attacks mitigated**
Ravi and Shalinie [16]	Semisupervised deep extreme learning machine	UNB-ISCX [28] and own SDN dataset	DDoS
Ahmed and Kim [17]	Inter-domain SDN statistics	–	DDoS
Yin et al. [18]	Cosine similarity threshold	–	DDoS
Galeano-Brajones et al. [19]	Entropy	Bot-IoT and own SDN dataset	DoS, DDoS
Sharma et al. [20]	Deep Belief Networks	UNB-ISCX	DDoS
Dawoud et al. [23]	Restricted Boltzmann Machines (RBM)	KDD Cup'99	Probe, DoS, U2R, R2L
Li et al. [22]	BAT algorithm and random forest	KDD Cup'99	Probe, DoS, U2R, R2L
Amangele et al. [24]	Decision tree (CART)	CICIDS2017 [29]	DDoS, Bot, Portscan

Also, as observed in Table 1.1, the majority of the proposed approaches have been evaluated with datasets that have not been prepared for IoT networks, which may not

truly reflect their performance in IoT environments. A similar argument is valid for SDN-based approaches whose performances have been evaluated with intrusion detection datasets, which contain features not easily extractable in an SDN environment.

The approach proposed in this dissertation overcomes these problems by introducing an ensemble learning technique capable of learning accurate models from fewer training samples, whose performance has been evaluated with realistic IoT environments with traffic features tailored for SDN-assisted operation.

1.3 Scope & Contributions

In this dissertation, we focus on the major shortcomings of SDN-based intrusion detection and mitigation approaches and introduce an intelligent security architecture for SDN-assisted IoT networks. The main contributions of the article can be summarized as follows:

- We propose an end-to-end SDN-based intrusion detection and prevention system architecture for SDN-assisted IoT networks, which monitors traffic flows at the controller and classifies the traffic using an intelligent attack detection module based on machine learning, according to which it makes flow rule update decisions.
- Also in our architecture, while we propose to deploy the SDN controller into a Kubernetes cluster on a public cloud, the intrusion detection system as an NFV service is connected to the SDN controller through its northbound API. The proposed architecture provides high availability for the controller microservice by auto-scaling on the increased loads.
- We propose the use of prototypical networks, which is a recently proposed class of few-shot learning, for effective classification of IoT network traffic to detect various classes of attacks in the absence of a large training dataset. To the best of our knowledge, this is the first approach utilizing prototypical networks for intrusion detection in IoT.
- We show through experiments with two publicly available datasets as well as an

SDN traffic dataset we have generated, that the proposed prototypical networks-based intrusion detection model achieves better performance than state-of-the-art machine learning models in detecting different types of attacks, even when a small amount of labeled data is used to train the model.

1.4 Organization

The remainder of this dissertation is organized as follows:

Chapter 2 first provides an overview of the main architectural components utilized in the scope of the thesis, especially software-defined networking, network function virtualization, and Kubernetes. Then, machine learning approaches used in the experiments, mainly prototypical networks, and other baseline models are explained.

In Chapter 3, problems and limitations of traditional networks are explained in detail. The general concepts presented by SDN against those problems are also discussed in the same section. Then, we focus on SDN security: first, "SDN for security" is the concept that provides solutions to secure the networks by using SDN's benefits, and secondly, "security for SDN" is the concept which aims at securing SDN assets, especially the controller. Besides, we clarify our discussion with an example network design considering SDN properties [1].

In Chapter 4, the proposed system architecture is presented to demonstrate the applicability of the developed approach in SDN-assisted IoT environments. The main elements of the proposed architecture, namely Statistics/Feature Extractor, Traffic Classification/Intelligence, and Attack Mitigation are explained in detail. Also, our Attack Detection & Mitigation algorithm is given as pseudocode.

Chapter 5 is separated into two main sections; section 5.1 gives a detailed information about three datasets, the BotIot dataset, the UNSW-NB15 dataset, and our self-developed SDN-dataset, which are used by the proposed intrusion detection model (Proto-S) and other baseline machine learning methods. Then, this section discusses the performance evaluations for proposed model and other baseline approaches. We

[1] This chapter is a slightly modified version of our published book chapter *Software-defined Network Security* [30].

clearly show that the proposed Proto-S model outperforms other machine learning approaches by means of the performance metrics. Section 5.2 shows the mitigation module's performance in a Kubernetes cluster in the public cloud using end-to-end tests. In this section, we introduced four different test scenarios on which experiments were carried out first, then gave the experiment results as the output of these scenarios. We showed the proposed solution's efficacy with the attack detection tests and the attack prevention scenarios by building up different network topologies. Besides the time measurements in preventing cyber-attacks, we also observed the effects of the proposed security mechanism on legitimate traffic. We proved that the proposed solution does not cause an additional burden on the SDN controller and does not negatively affect normal traffic. Finally, we provided high availability test results and explained how the proposed security solution could work in harmony with the auto-scaler mechanism in a Kubernetes cluster [2].

And, finally, Chapter 6 consists of conclusions, discussion, and foreseen future work directions which conclude the thesis.

[2] In chapters 4 and 5, we have benefited from some findings of our published article *ProtÉdge: A few-shot ensemble learning approach to software-defined networking-assisted edge security* [31] and some contents in these chapters are a slightly modified version of this article.

CHAPTER 2

PRELIMINARIES

In this chapter, network technologies and machine learning approaches addressed in the scope of the thesis are introduced. First, a brief overview of software-defined networking, network function virtualization, micro-services, containerization, and kubernetes clustering environment is given. Then, the approach of the prototypical networks, which is the main component of the proposed ensemble learning model, is explained in detail. The other baseline machine learning techniques, which are used for comparisons, are also presented.

2.1 Networking Technologies

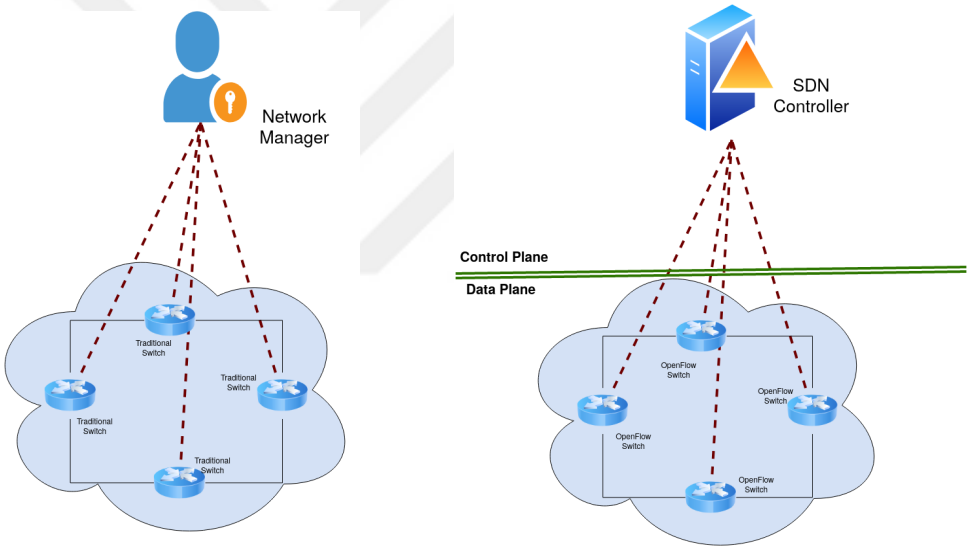
2.1.1 Software-Defined Networking

Routers and switching elements are of great importance in IP-based computer networks that are widely used today. Network operators configure each network device using low-level or manufacturer-specific commands to manage this complex structure and enforce high-level network policies. In addition to this complexity in network configuration, computer networks must be able to withstand errors occurring and to adapt to changes in the load that occurs. In addition, the implementation of the necessary policies poses major difficulties, since automatic configuration mechanisms are practically absent in traditional IP-based computer networks.

In addition to all this complex structure, today's networks operate in an integrated structure (Figure 2.1a). The control plane that decides how the network traffic is handled and the data plane that directs the traffic according to the decisions reported

by the control plane are intertwined in the network devices. As a result, flexibility is weakened and the development of network infrastructure is hampered.

There is an increasing demand for software-based solutions for the solution of these problems. Software-based solutions are preferred more than hardware-based solutions due to factors such as eliminating errors and applying new versions easily. Software-defined networks are emerging as a new hope for the solution of problems encountered in traditional computer networks. In the SDN infrastructure, the network's control logic (control plane) decouples from routers and distributors (data plane) that transmit traffic (Figure 2.1b). Then, with the separation of control and data planes, network switches become simple transmission devices. Control logic turns into a central controller that simplifies the implementation of network policies, network reconstruction, and network development.



(a) Traditional Networks

(b) SDN

Figure 2.1: Traditional networks vs. Software-defined networking.

In Figure 2.2, the SDN architecture which is proposed by the Open Networking Foundation (ONF) [32] is depicted. An additional application layer provides services such as network management, security, analytics, business applications, etc. through communication with the control plane. Applications communicate with the controller using the north-bound interface, for which there is no current standard. The controller

communicates with the switches using the south-bound interface, for which the most commonly used protocol is OpenFlow [33].

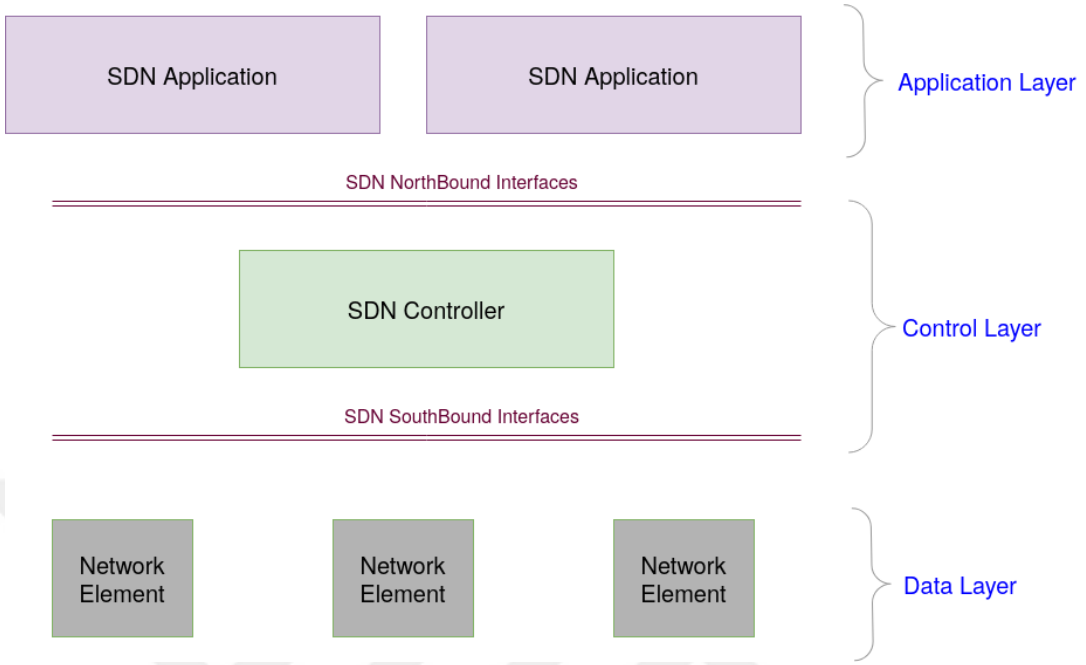


Figure 2.2: Basic SDN components that construct the architecture.

SDN centralizes network packet routing decisions at the controller, which has a global view of the network, instead of allowing routers/switches to make these decisions in a decentralized manner. Each element of the data plane maintains flow tables containing flow rules for acting upon packets they receive, but the contents of these tables are now dictated by the controller instead of the router/switch itself. The controller installs flow rules called *flow entry* into the flow tables of the switches. Figure 2.3 shows the structure of a flow entry.

When a packet arrives, the switch checks whether it matches any flow entry in the flow tables as illustrated in Figure 2.4. A table miss occurs if an incoming packet does not meet some flow rule in flow tables. The switch stores the packet and sends a Packet_in message containing the header of the packet to the controller. The controller determines the action (i.e. forward, drop, broadcast etc.) and sends a Packet_out message, containing instructions for the packet which caused the table miss, to the switch. The packet will be transmitted or blocked according to the message Packet_out. By sending a Flow_mod message, after sending the Packet_out message to the switches that

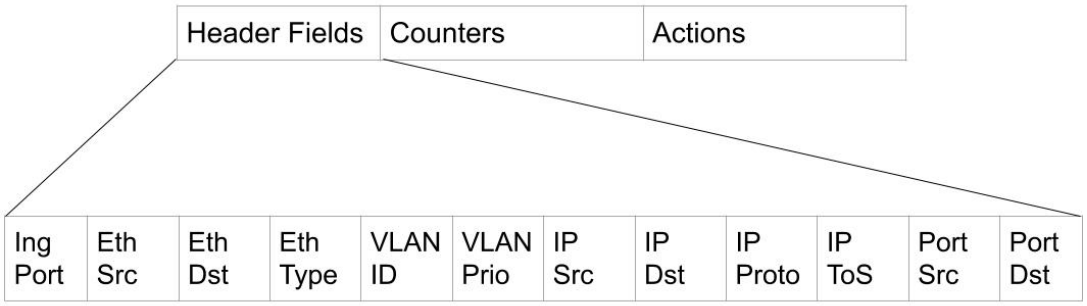


Figure 2.3: OpenFlow entry structure.

are along the path, the controller may also deploy a flow rule into the corresponding switches [34].

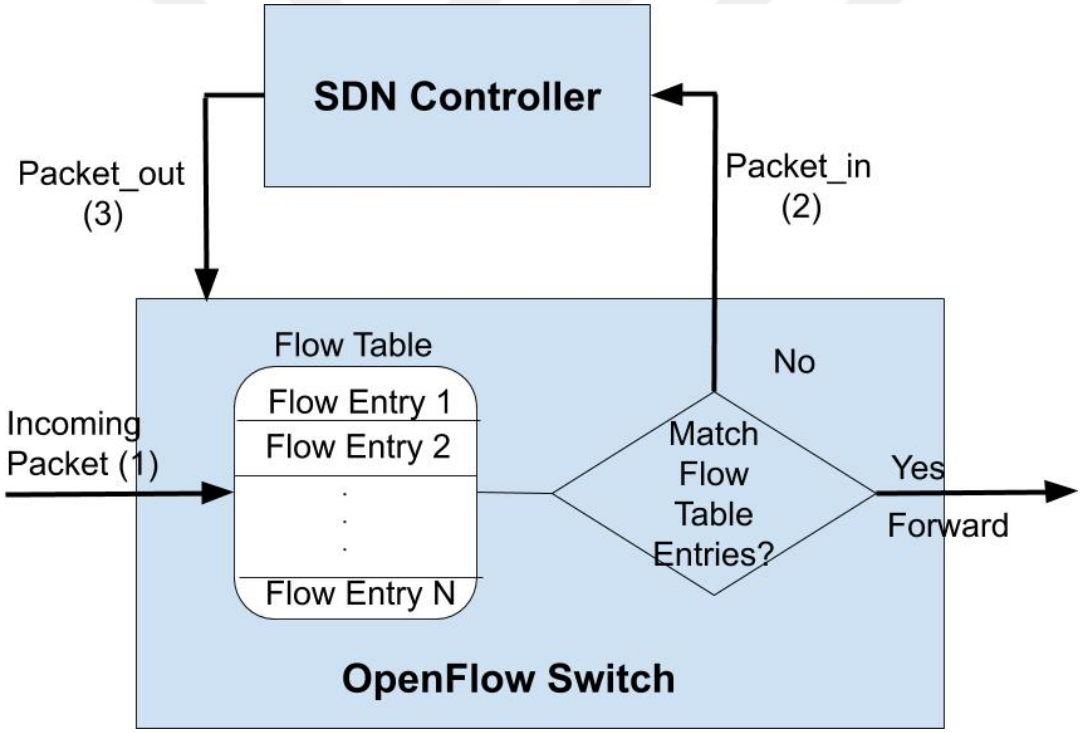


Figure 2.4: Network packet processing in SDN.

Separating the network's control plane from the data plane and using a centralized controller that has the global view makes the networks programmable [35]. The centralized controller provides dynamic management and configuration of the network [36]. SDN has already been deployed in Google's WAN [37] and is expected

to meet the requirements of next generation networks (5G and beyond) and IoT networks. According to a recent survey from Verizon, 15% of the respondent companies were already using SDN and it was expected to rise to 57% within next two years [38].

By centralizing control with a global view of the managed network, SDN creates an immense opportunity for swift reconfiguration of the networks, which will play an important role in supporting the requirements of the future's ubiquitous IoT. In addition to providing effective resource management and fast response to performance bottlenecks in wired and wireless networks, SDN will also be an indispensable tool for security by enabling effective monitoring of the whole network to detect anomalies, finer-grain enforcement of security policies, and fast network reconfiguration to swiftly mitigate attacks [30].

SDN has already demonstrated significant benefits for various types of networks including vehicular networks [39], intelligent transportation systems [40], industrial networks [41] and common enterprise networks among many others. The role of SDN in managing IoT networks will definitely be at least as important due to the resource-constrained and very dynamic nature of these networks. Devices in SDN-assisted IoT environments are vulnerable to various attacks not only due to the resource constraints making it difficult to implement security measures, but also the heterogeneous nature of these networks.

The management of SDN-assisted IoT networks provides important advantages: malicious devices that have entered the network can be detected and isolated quickly, and load balancing on the servers can be performed more effectively to prevent service disruptions due to DoS/DDoS attacks or non-malicious overloading. The IoT devices themselves can also be protected from malicious traffic aiming to cause harm by intelligent detection using intrusion detection techniques at the application layer. Through intrusion detection followed by swift reconfiguration of flow rules in the switches that IoT devices are connected to, SDN will enable IoT networks to achieve fast recovery from attacks.

2.1.2 Network Function Virtualization

In today's traditional network architecture, different types of special hardware tools are used to perform basic network operations. More specifically, each of the network functions is operated on separate hardware belonging to the company from which it was developed. In terms of adding a new service to such an architecture, finding a new location for the device that will run that service, ensuring the interoperability of devices from different companies, and re-designing the entire architecture, it is quite difficult.

In addition, the need to use a separate special device for each function has negative consequences in terms of cost. NFV is a new network technology that provides solutions to the problems mentioned above by separating the network functions from special hardware into software and enabling them to run on general-purpose servers [42].

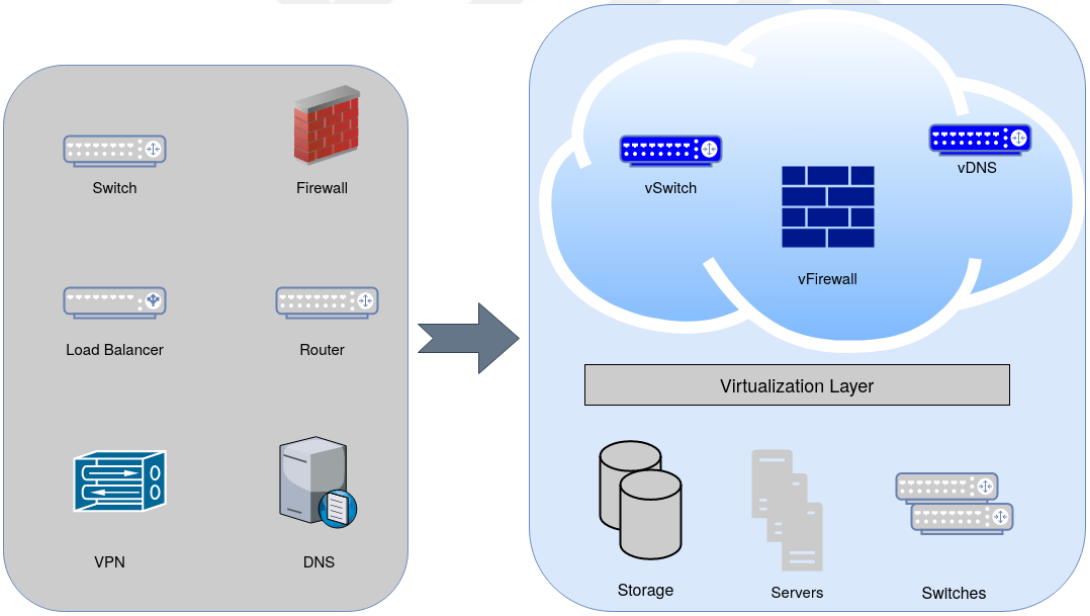


Figure 2.5: Traditional network appliances vs. NFV.

As seen in Figure 2.5, network functions are virtualized with NFV and can be placed in appropriate locations on general-purpose servers without the need for using special hardware for each service. NFV eliminates hardware dependency as in SDN, thereby reducing hardware cost and energy consumption. Besides, resources between func-

tions can be shared with NFV, the need for manpower and expertise is reduced, and functions can be re-positioned on the network without the need for hardware installations.

2.1.3 Micro-Services, Containerization and Kubernetes

Micro-services are a realization of the service-oriented architecture model for designing applications composed of small services that can be distributed and individually scaled by completely integrated delivery machines, with limited centralized administration. Micro-services are designed around different functionalities of the organization. Each micro-service operates in its own process and communicates through mechanisms of lightweightness, often using application programming interfaces (API).

Micro-services tackle the drawbacks of monolithic applications. They are small and can reboot faster when upgrading or recovering from failure. Micro-services are loosely coupled, so the failure of one micro-service would have little effect on other micro-services. This architectural style's fine granularity makes scaling more flexible and more efficient, as each micro-service can evolve at its own pace.

To make use of such advantages, one will use the technology compatible with the features of this architectural form. Containerization is the infrastructure that allows operating system-level virtualization. Containers are compact and light-weight. Hence they are suitable for micro-services development. Docker is the leading container platform where code and dependencies are packaged together and shipped as a container image.

Since containers are isolated they do not know each other. Therefore, an orchestration framework is required to handle container deployment. Kubernetes is an open-source platform that enables containerized applications to be deployed, scaled, and managed automatically. Kubernetes relieves developers of technology from the difficulty of integrating the reliability of their code. It has thus become a popular platform for deploying applications based on micro-services.

Figure 2.6 illustrates an architecture for running applications in a Kubernetes clus-

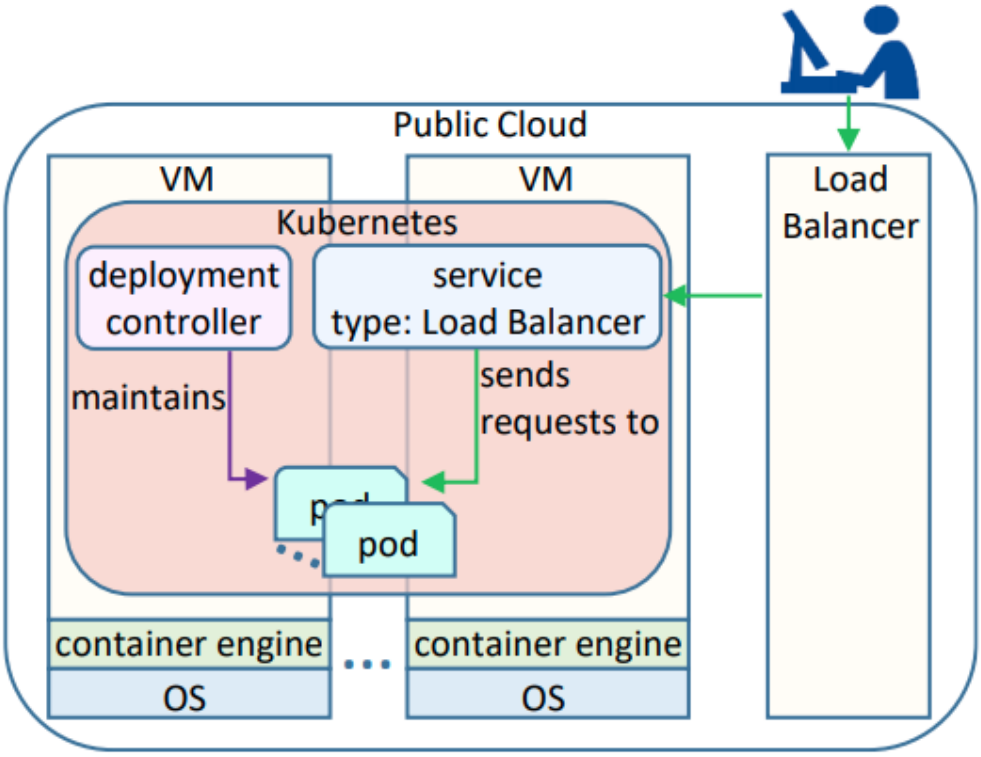


Figure 2.6: Architecture for deploying applications to Kubernetes clusters running in a public cloud.

ter utilizing a Load Balancer type micro-service in a public cloud. In addition to an in-cluster IP, Load Balancer services have an external IP address which is set automatically to the load-balancer IP address of the cloud provider. It is possible to reach the pods (containers) from outside of the cluster using this external IP address, which is available from the Internet.

When a micro-service faces an increased load while running in routine, the Kubernetes auto-scaler reacts to load in order for the micro-service to continue to work in the same way. The number of container replicas of the micro-service is boosted and thus the high availability is provided.

2.2 Machine Learning Approaches

2.2.1 Convolutional Neural Networks

A convolutional neural network (CNN, or ConvNet) in deep learning is a category of deep neural networks, most widely applied in computer vision research. One or more convolutional layers (CONV), and following one or more fully connected layers (FC) comprise the convolutional neural networks (CNN). CNN architecture takes advantage of two-dimensional input data. This situation is achieved with local connections and assigned weights followed by some pooling layer forms that result in translation-invariant features. The second advantage of CNN architecture is that it is easier to train and has many fewer parameters than FC networks that have the same number of hidden units [43].

CNNs differ significantly from other ML algorithms in that they integrate both feature extraction and classification phases. The description of a simple graphical representation of a basic CNN is seen in Figure 2.7. There are five different layers in this basic network: an input layer, a CONV layer, a pooling layer, an FC layer, and an output layer. Such layers are divided into two parts: feature extraction and classification. The feature extraction phase consists of an input layer, a CONV layer, and a pooling layer, whereas the classification phase consists of an FC layer and an output layer. Fixed-size input data is defined in the input layer. Then the input is transformed by the convolution layer to several learned kernels, using shared weights. The pooling layer then decreases the size of the data when attempting to preserve the information stored therein. The feature extraction outputs are known as the feature maps. The classification incorporates the extracted features in the FC layers. Ultimately, one output neuron occurs for each object category in the output layer. Though originally created for multi-dimensional data like images, we can benefit from CNNs in intrusion detection tasks, thanks to the one-dimensional versions of convolutional layers.

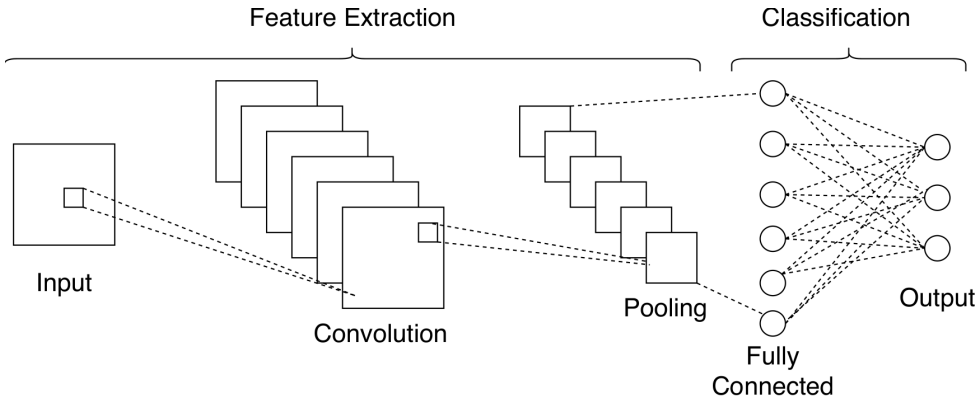


Figure 2.7: Diagram of a basic convolutional neural network architecture.

2.2.2 Prototypical Networks

Recent advances in machine learning, particularly in the computer vision domain, have resulted in classification models that are capable of learning accurate representations of object classes from very few training data samples. The ability to construct accurate classification models from small amounts of training data is an important capability for many machine learning-based systems, as collection of quality labeled data is quite difficult in many domains.

Initially proposed for object recognition in computer vision, one-shot learning aims to construct an accurate classification model from a single training instance of each class [44]. When we include N samples of each of the k possible classes in the training data for learning a classification model, we are performing N-shot learning. If N is small enough, the learning task is called few-shot learning.

Few-shot learning scenarios try to address the problem of training machine learning models with limited data. In order to achieve high accuracy in few-shot learning tasks, many meta-learning algorithms paired with traditional deep learning approaches have been proposed [45], and prototypical networks is one of the most successful examples.

Prototypical networks [46] are different from classical deep learning models in that they do not explicitly classify the given inputs, but learn to map their input into a metric space instead. The encoding function $f_\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ embeds each input to an

M-dimensional prototype $c_k \in \mathbb{R}^M$ where ϕ corresponds to the trainable parameters.

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i)} f_\phi(x_i) \tag{2.1}$$

Each prototype (2.1) is the mean vector belonging to its class of embedded support points S_k, where $S = (x_1, y_1), ..., (x_N, y_N)$, $x_i \in \mathbb{R}^D$ and $y_i \in 1, ..., K$. The D-dimensional feature vector x_i owns the label y_i out of K total classes. Using a distance function $d : \mathbb{R}^M X \mathbb{R}^M \longrightarrow [0, +\infty)$, prototypical networks compute the distances for a query point x to the prototypes in the metric space:

$$p_\phi(y = k|x) = \frac{exp(-d(f_\phi(x), c_k))}{\Sigma_{k'} exp(-d(f_\phi(x), c_{k'})} \tag{2.2}$$

In each training step, a sub-set of classes from the training set is randomly selected, then a sub-set of samples within each class are chosen to behave as the support set and a sub-set of the rest to function as query points. After the prototypes are calculated utilizing the support set, distances from the query set to those prototypes then computed. The training process continues by reducing the negative log probability $J(\phi) = -logp_\phi(y = k|x)$ via the optimizer for the true class k.

In Figure 2.8 the embedding architecture is shown. To provide more appropriate data for the embedding network, data is pre-processed first. Label-encoding and data-normalization steps are performed therein. Label-encoding consists of encoding numerical values to the categorical attributes. This data is then normalized to generate more homogeneous values between zero and one and given to the embedding network.

Afterwards, the first CNN block takes the pre-processed data, applies operations on it, and sends it to the next one. Every block contains a convolution (64-filter, 3-kernel), a batch-normalization, a non-linearity of the Rectified Linear Unit (ReLU) which directly outputs the input if it is positive (otherwise, it outputs zero), and a max-pooling (pool size= 2) layer. This architecture converts input data into a 64-dimensional point in the metric space. Finally, distances to class centroids are calculated by using the Euclidean distance formula. The class likelihoods are maximum for the shortest dis-

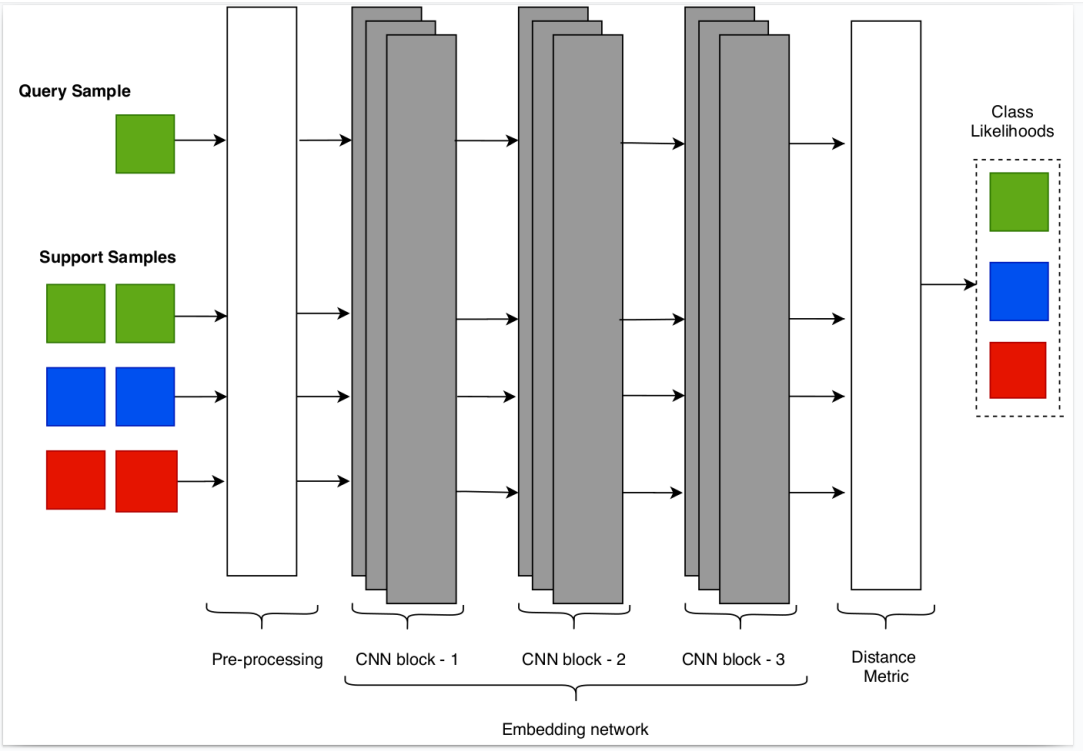


Figure 2.8: Prototypical network embedding architecture.

tances.

Figure 2.9 shows the functioning of prototypical networks for the sample task of building a classifier for the three attack classes DoS, DDoS and service scan. As seen in the figure, the encoder maps samples of the same class close to each other, whereas different classes are spread out at a large distance. This means that whenever a new sample comes in, the model simply finds the closest cluster and assigns the instance to that cluster. In order to do this, by using a Convolutional Neural Network (CNN) [47], we encode the inputs (support set and query set) into an embedding space and take the prototype of each class as the mean of its support set.

Classification for an embedded query point is then performed, by calculating the closest prototype. The N labeled samples for each class form the support set for the classifier and the set of samples we try to classify form the query set.

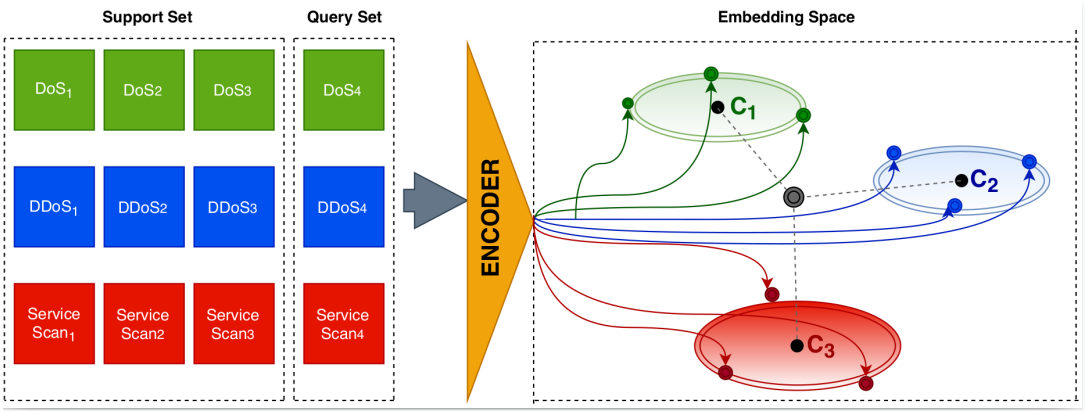


Figure 2.9: A prototype c_k is calculated as the mean of a class, by using support set embeddings.

2.2.3 Support Vector Machines

As a binary classification technique, support vector machine (SVM), uses maximum-margin separator hyper-plane between the two target classes to linearly separate those classes [48]. If the data is nonlinearly separable, SVM gets a big advantage and it can linearly separate the data by using kernel functions.

A kernel function maps the data patterns of the input to some high dimensional space in order to make the data points linearly separable. Practically, the mapping of the data points are explicitly defined as the inner product between them in order to be distinguished in high dimensional space [49] and there isn't any implicit definition. SVM loss function of the i-th example L_i is represented as

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \tag{2.3}$$

where s_j and Δ represents incorrect class scores and delta value, respectively. In this work, we add to the loss function a regularization penalty

$$R(W) = \sum_k \sum_l W_{k,l}^2 \tag{2.4}$$

that is the L2 norm using an element-wise quadratic penalty on all weights $W_{k,l}$ to

discourage the large parameters. This is because squaring a number punishes large values more than it punishes small values. Then, the loss function used by SVM classifier becomes

$$L = \frac{1}{N} \sum_{i} L_i + \lambda R(W) \tag{2.5}$$

where N is the number of training examples and λ is a weighted hyper-parameter. The value of this hyper-parameter is usually determined by cross-validation technique since there is no a simple way of setting it manually.

Multi-class SVM uses support-vector machines to allocate labels to instances where the labels are taken from a finite set of many elements. The dominant solution to this is to reduce the multi-class problem to multiple binary classification problems.

2.2.4 Naïve Bayes

It is a technique of classification based on Bayes' theorem, with an assumption of independence between predictors. A Naïve Bayes classifier presumes in simple terms that the existence of a particular feature in a class is not related to the presence of any other feature [50].

For instance, if a fruit is red, circular, and around 3 inches in diameter, it can be called an apple. Particularly if these characteristics depend on each other or on the presence of the other characteristics, all these characteristics contribute individually to the probability that this fruit is an apple, which is why it is called 'Naïve.'

The Naïve Bayes model is simple to create and particularly useful for very large data sets. Naïve Bayes is considered to outperform even extremely advanced methods of classification alongside simplicity.

Bayes theorem provides a means of estimating the posterior probability $P(A|B)$ by

using $P(A)$, $P(B)$, and $P(B|A)$.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{2.6}$$

Above,

- $P(A|B)$ is the posterior probability of class (B, target) given predictor (A, attributes).
- $P(A)$ is the prior probability of class.
- $P(B|A)$ is the likelihood which is the probability of predictor given class.
- $P(B)$ is the prior probability of predictor.

2.2.5 Deep Auto-encoders

An auto-encoder [51] is a neural network equipped to seek to copy its input to the output. Internally, it has a hidden layer h representing a language that describes how to use the input data. The network contains two parts: an encoder function $h = f(x)$ and a decoder function $r = g(h)$ for reconstruction.

An auto-encoder is unuseful if it basically succeeds in learning to set $g(f(x)) = x$ everywhere. Instead, auto-encoders are designed so that they can not learn how to copy perfectly. Usually, they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. As the model is required to prioritize which elements of the input will be replicated, it often discovers valuable data properties.

In the deep auto-encoders, the term deep comes from multiple encoding and decoding of sequences. The encoder and decoder parts of a deep auto-encoder are two separate Deep Belief Networks (DBN) where each has shallow layers. Figure 2.10 shows the architecture of deep auto-encoders.

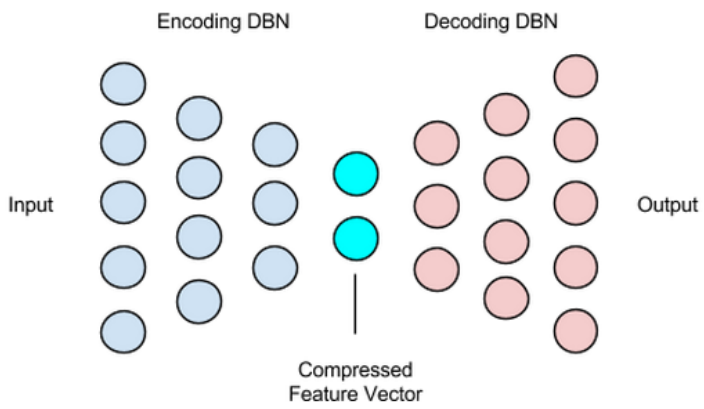


Figure 2.10: Deep auto-encoders.

2.2.6 Adaptive Boosting

Adaptive Boosting (AdaBoost) [52] is a boosting model as its name refers to. Boosting algorithms are the set of algorithms that convert weak learners to strong learners. Hence, in AdaBoost, we fit a sequence of vulnerable learners on different weighted training data. It starts by forecasting the original data-set and results in equal weight for each representation.

If the forecast is false for the first learner, it adapts by offering higher weight to representations that predicted incorrectly. Being an iterative process, it continues to add learner(s) until it reaches the limit in the number of models or accuracy. Figure 2.11 illustrates the iterative process which takes place in the AdaBoost algorithm.

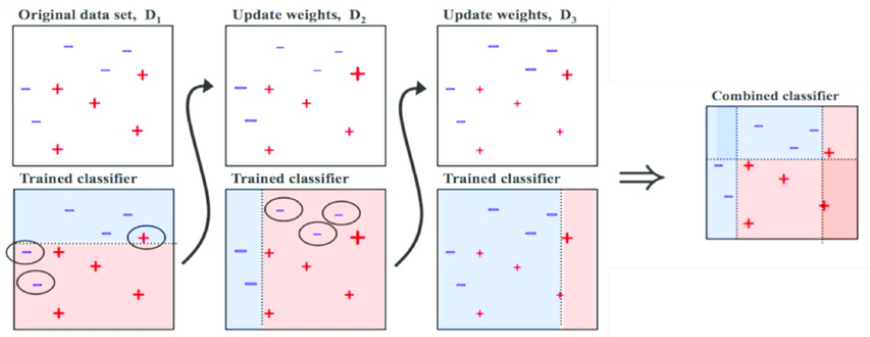


Figure 2.11: AdaBoost iterative process.

2.2.7 Weighted Average Ensemble

An average model ensemble integrates the prediction of each algorithm equally and, on average, often leads to better performance than a given model. Sometimes there are some excellent models that we want to contribute more to an ensemble prediction and perhaps less capable models that may be useful but needless to contribute to an ensemble prediction. A weighted average ensemble is an approach that allows multiple models to contribute to an estimate in proportion to their confidence or estimated performance.

In regression problems, the ensemble prediction is determined as the participant predictions average. If a class mark is predicted, the prediction is determined as the mode of the member predictions. If a class probability is estimated, the prediction may be measured as the argmax of each class tag's cumulative probabilities.

A weighted ensemble [53] is an extension of a model averaging ensemble, where each model's performance weights each member's contribution to the final prediction. Model weights are small positive values, and the total of all weights is equal to one, which allows weights to indicate the expected out or level of trust from each member.

Uniform weight values (e.g., 1/k such that k is the number of members of the ensemble) mean the weighted ensemble works as a simple, average ensemble. There is no empirical approach to calculating the weights; preferably, either the training dataset or a holdout test dataset should be used to approximate the weights' value.

Determining the weights with the same dataset used to train the member models would possibly lead to an overfit. A more reliable solution is to use a holdout testing dataset that is not used by the ensemble members during training.

CHAPTER 3

SDN SECURITY

3.1 Overview

Changing business and consumer demands in technology have transformed the networking perspective. Rather than the former person-to-person or person-to-computer interactions, with the rapidly spreading Internet of Things (IoT) and smart devices, the world has become a complete "anything-to-anything" connected network. From this perspective, connection to anything anytime and anywhere is perceived as the most basic requirement for very near future.

All these paradigm changes entail quite significant improvements to adopt requirements of the future networks. For example, the key performance indicators (KPI) of 5G networks directly infer the technological exigence of the near future. These indicators include more than 1 Gbps data rate with ultimately 1ms end-to-end latency. 99% availability and reliability are also expected even in high-device density which supports nearly 10,000 per km2 [54]. However, it is not possible for present mobile networks to address the 1000x key performance requirements of 5G because of their limitations.

Today, the networking infrastructures are very complex and hard-to-be-operated as they reflect outdated technological requirements which belong a decade ago. They are lack of common control functions and interfaces and network operators have to separately configure network devices using vendor-defined commands to apply the high-level management policies [55]. It obliges the existing mobile networks to be structured with vendor locked-in and the dedicated hardware and software. These components implement such network functions which are not only prone to miscon-

figuration but also costly and require trained/expert administrators [56].

Besides, from a security point of view, since network policies are tightly bundled to physical resources instead of services and applications, today's solutions are strained to deploy, manage, program and scale the security throughout heterogeneous equipment from multiple vendors.

To enforce consistent security policies through computing, storage, and network domains and especially across multiple data centers is very difficult. It should be noted that there is no end-to-end solution for security orchestration across data center networks today [57]. Eventually, all those limitations significantly reduce flexibility and slow down the evolution of the infrastructure for the future networks [58].

Software-defined networking is a novel technique that may overcome the limitations of traditional networks. It is, for instance, expected to be used in the implementation of 5G networks since SDN perfectly compromises the logical and functional architecture of 5G which targets modularity, flexibility, and scalability in the first place.

Besides, considering the significantly complex architecture of the future networks, it is very natural to take advantage of "softwarization" for more manageable, reusable, high-performance and optimized systems. To provide those conditions, SDN architecture fundamentally abstracts lower-level functionality by detaching the control plane from the data plane.

As shown in Figure 3.1, a software-defined network can be described in three levels; application, control and infrastructure levels. The most significant level in SDN is the control level (controller) that is responsible for implementing the network control logic. It implements the southbound, northbound and east/west interfaces for intercommunication between the SDN application, controller, and infrastructure at different levels.

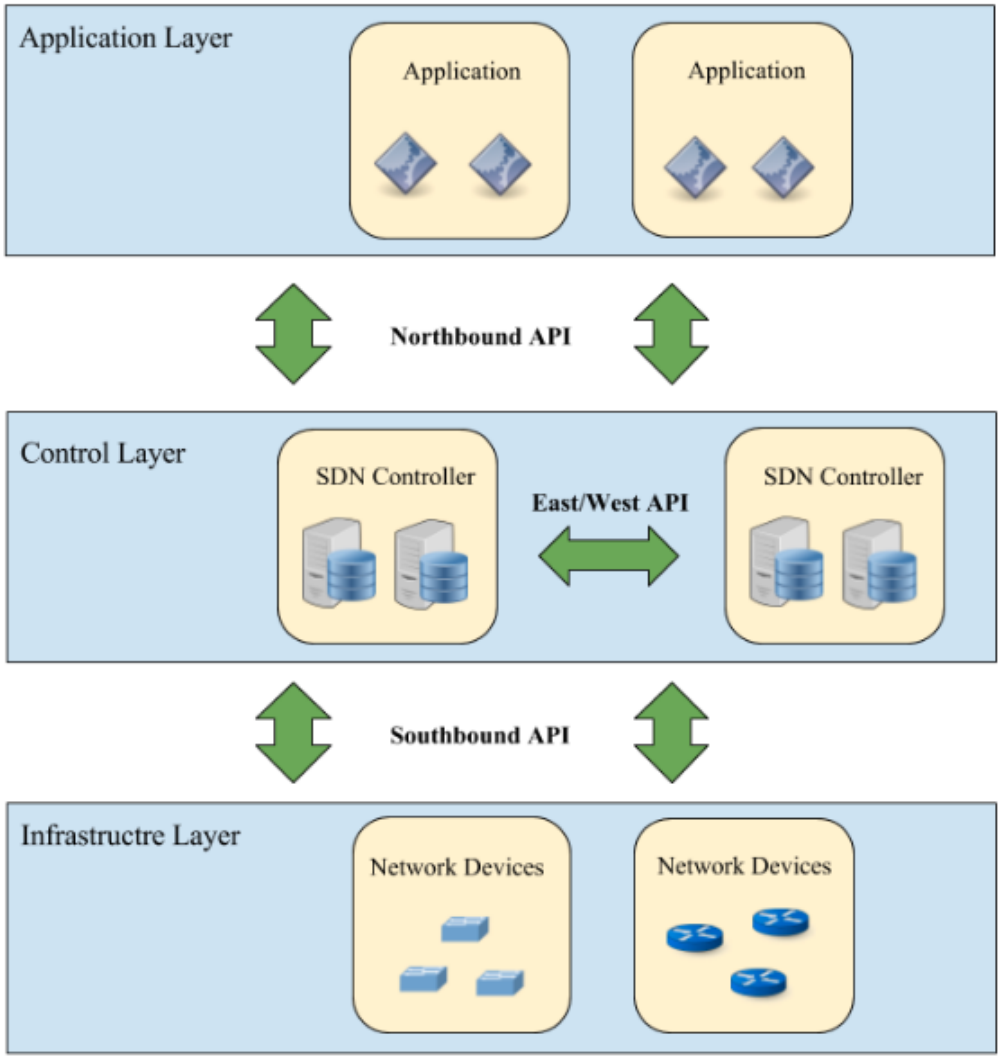


Figure 3.1: SDN architecture.

The southbound interface is between the controller and infrastructure levels and mainly specifies functionalities to manage switching devices residing at data plane. Advising packet forwarding rules and taking feedback of current network status could be example capabilities accomplished over this interface. Similarly, for communication between the application level and the controller, the northbound interface establishes a service access generally through an API.

For example, using the information gathered from lower levels (from infrastructure level or parallel controller devices), SDN applications may run management and tun-

ing algorithms whose outcomes determine controllers' behavior. Synchronicity between the application and the controller is performed via the northbound interface. Apart from "vertical" bridges between different layers, east and west interfaces are lying between multiple controller devices which are the practical issue for large network domains [59].

As a relatively new technology presenting different concepts and design issues, SDN has a number of open-research challenges. While it stands the test of time for further research, security is also a weight matter to be investigated. From the security viewpoint, SDN has three main characteristics with their upside and downside:

- First, logically centralized intelligence enhances the security of the network by enabling network managers to unify security policies and adapt them automatically and consistently. However, the centralized controller is also prone to security attacks and new attack vectors.

- Moreover, even though programmability enhances the level of security by facilitating efficient deployment of security functions as applications on controllers employing northbound interface, it eases exploiting vulnerabilities and compromising controllers abusing control functions.

- Lastly, abstraction enhances security by blanketing physical assets; in contrast, an attacker does not need physical access to the system anymore.

In the sequel, the rest of the chapter organized as follows: in the next section, problems and limitations of traditional networks are presented. The general concepts presented by SDN against those problems are also discussed in the same section. Then, we focus on SDN security: firstly, "SDN for security" is the concept that provides solutions to secure the networks by using SDN's benefits and secondly, "security for SDN" is the concept which aims at securing SDN assets, especially the controller. Besides, we clarify our discussion with an example network design considering SDN properties.

3.2 Limitations Of Traditional Networks And Solutions

While the technology grows exponentially in both quality and quantity, the majority of the networks still follow the legacy architectural principles. Even though the traditional network architectures have relatively fulfilled their mission heretofore, the entailment of a new system for future networks is currently debated considering quite a few points.

In this section, the reasons that triggered those controversies are briefly clarified by revealing the main problems of traditional network architectures. Besides, the solutions SDN propose to related problems are introduced by comparing the current network structures with the SDN. Security issues and concerns of traditional networks are also discussed in detail together with the primary solutions in the next section.

3.2.1 Problems of Traditional Networks

Independent of an interest area that technology advanced, network structures considerably affect and are affected by numerous innovations. From cloud computing to the Internet of Things, the evolution in technology trends continuously burst and this progression is integrating the technology into people's lives. As a result, an excessive usage promoting technological demand at both enterprise and single end-user levels arise.

The increasing demand also stimulates flexible and manageable service requirements in parallel with increasing traffic load and technological advancements. In this sense, SDN comes into the picture as a novel networking architecture that reinterprets extensively active and connected networks such as 4/5G mobile networks on which brand-new technologies are built to utilize the resources with maximum capacity and efficiency.

In this subsection, the weaknesses and limitations of traditional networks in a highly demanding and positively fluctuating technological world are specified under four major points [60]. What SDN offers for those weak points in current network architectures is presented next.

The network devices which are currently being used in today's network architecture handles routing and forwarding through control and data planes as shown in Figure 3.2. The control plane is mainly responsible for configuration of the network element and routing in the network layer. Avoiding cyclic paths in a local area network or quality of service (QoS) based packet or service differentiation are also functions of the control plane [61].

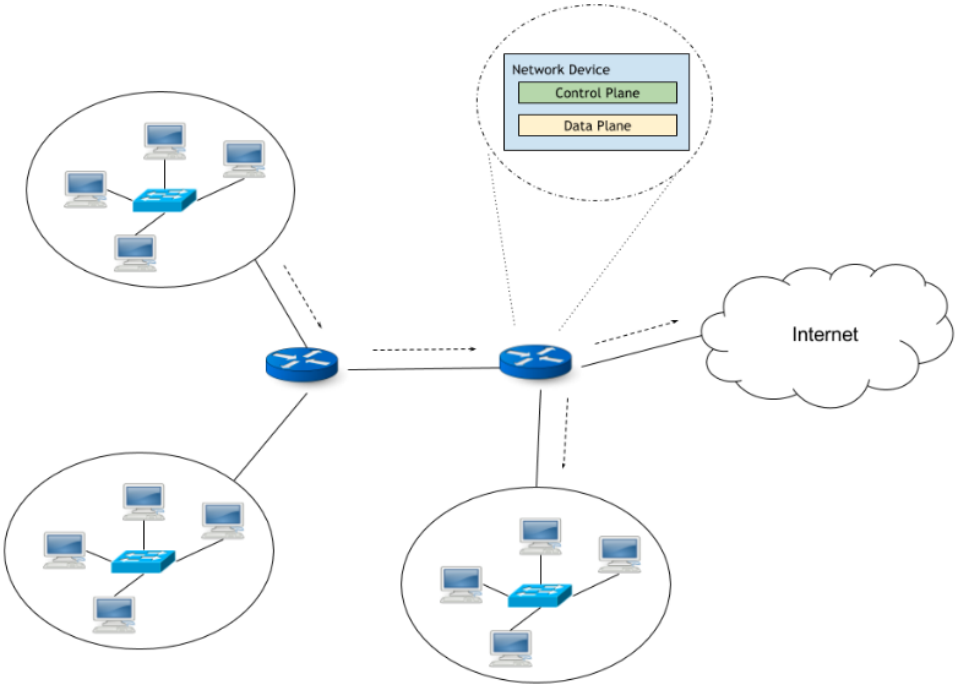


Figure 3.2: The basic architecture of traditional networks.

Once the possible routing paths have been determined, data are passed to the data plane and typically stored in device hardware. The data plane basically determines suitable paths among available routing options provided from the control plane and forwards packets through the decided route. This mechanism has traditionally been quite effective: the path-decision process on hardware is quite fast and reducing latency while the control plane handles the "processing burden" and focuses on other requirements. Both planes are implemented in the firmware of routers and switches and each network device makes routing and forwarding based on its own calculations and decisions using such autonomous methods in those planes. However, the architecture is far from being flawless when modern network requirements are considered.

The distributed, autonomous approach explained in the previous subsection developed when networks were predominantly static and end-systems are placed on fixed locations. Eventually, these characteristics fall short to conduct today's dynamic, complex and weighty network composition. In this sense, general problems of the traditional network could be collected under four main headlines [62].

3.2.2 Static and Complex Architecture

Commonly, the typical enterprise network architecture is simply constructed by local tree structures of Ethernet switches with routers connecting to relatively larger Ethernet LANs and connecting to the main providers such as the Internet and WAN facilities through those Ethernet LANs. This approach is perfectly convenient for basic client/server architecture that is the major network model in the enterprise environment. With this model, interaction and network traffic mostly take place between a single client and server. Considering this type of connection, networks could be set up and configured with the relatively static client and server locations and predictable -mostly pre-determined- traffic volumes throughout the network.

However, in modern systems, network requirements for many enterprises, public and private systems have fundamentally changed to be able to cope with more dynamic traffic patterns. For instance, since mobile device usage has become widespread, including bringing your own device (BYOD) policies, users are enabled to access to content and applications in a time, device, and location independent manner. As a consequence, this mobile traffic is becoming an increasingly significant fraction of network traffic. Speaking with statistics, according to Cisco's latest forecast illustrated in Figure 3.3, the expected number of smartphones (including tablets) will be nearly 50 percent of global devices by 2020 and this fact indicates future networks will be required to be even more flexible [1].

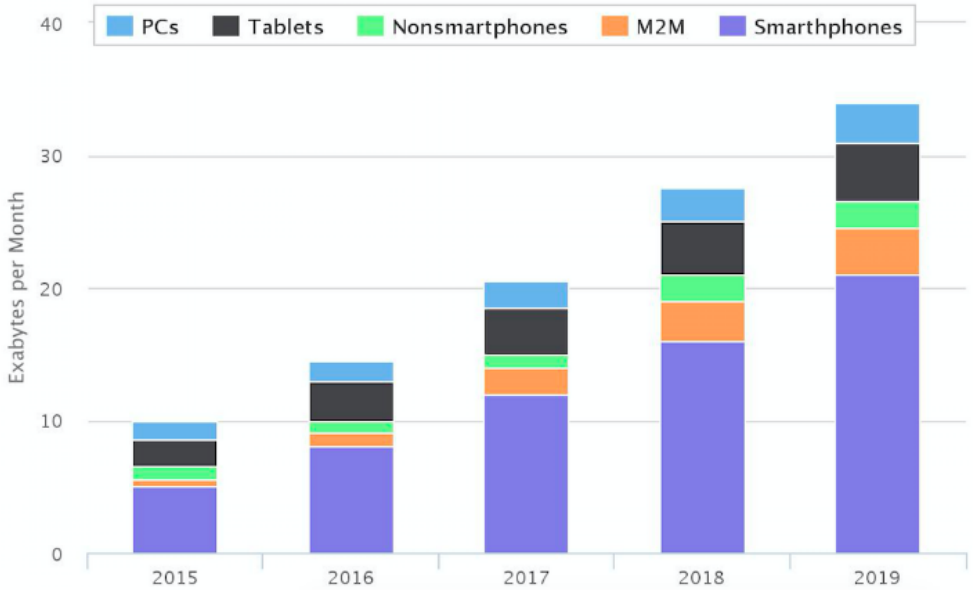


Figure 3.3: Cisco's forecast for mobile network load for near future [1].

Apart from mobility, the most preferred cloud, virtualization, and other networks services are supposed to be utilized with different Quality of Service and security requirements for distinct clients. In order to respond to demands in such varying levels of QoS, heavy and fluctuating traffic volumes and different security requirements, networking technology has become more complex and difficult to maintain. Especially in case of device addition and translocation, necessary changes to configure parameters of multiple network elements such as routers, firewalls, and switches have become compulsory. This obligation has created an extra need for increasingly sophisticated and agile fashion in network architecture.

3.2.3 Difficulty in Scaling

Demands on networks -and network components- are also growing rapidly in volume in-difference with service variation as shown in Figure 3.4. Scaling a network with more switches, routers and indirectly transmission capacity using multiple different vendor equipment becomes more and more difficult due to complex and static nature of the network.

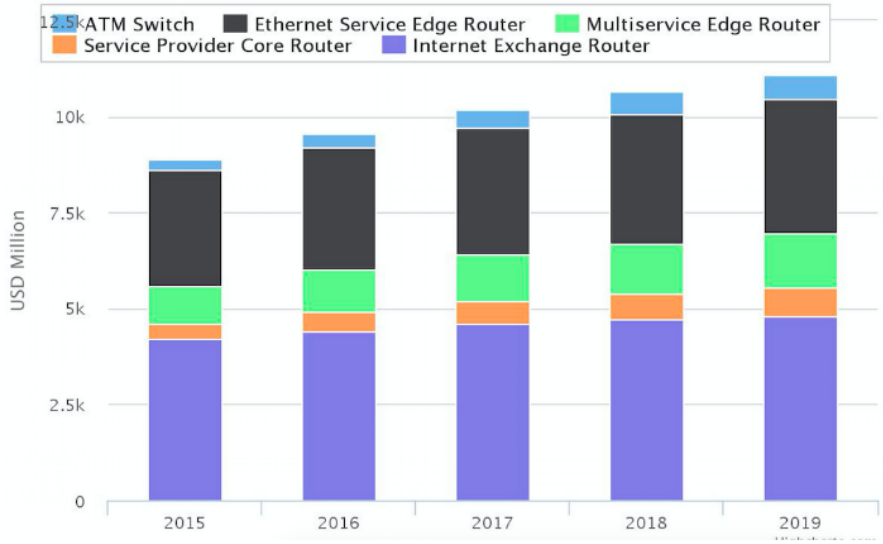


Figure 3.4: Increasing router and switch demand in the United States.

Commonly, enterprises oversubscribe network links considering pre-analyzed traffic density and flow and since this approach is mainly based on predictions, for dynamic traffic patterns in virtualized networks, it provokes a significant problem that is especially a subject in highly parallelized service-provider networks and collocated datasets in their computing pools. All in all, configuring thousands of network devices and millions of virtual servers dynamically are drawn almost impossible by manual configuration. However, growing attention towards a significant shift in necessity from hardware-based networking architectures to software-based systems probably shapes an important challenge to the traditional hardware-based companies. [1].

3.2.4 Vendor Dependency

Vendor lock-in basically means the incompatibility between devices that are designed and produced by different manufacturers, or namely vendors. It complicates usage and deployment of proprietary technologies that are incompatible with those of competitors.

Considering today's traffic demands on modern networks [63], parties need to customize their networks with new features and services quickly for evolving business

necessities [2]. However, notwithstanding that any structural changes and modifications in core network devices are nearly impossible for network administrators. Considering a number of vendors in the market shown in Figure 3.5 (and a significant number of relatively small vendors which are omitted), it is also quite challenging to deploy different network devices provided from different vendors to customize overall network structure.

Besides, apart from modification, automation and controllability are profoundly restricted because of vendor dependency. Thus, vendor lock-in utterly restricts modifications and customizations to networks built on commercial products. Eventually, a lack of standardized network interfaces and the competition stem from vendor lock-in restricts the pace of enterprises' progress with relatively slow product cycles of commercial vendor equipment.

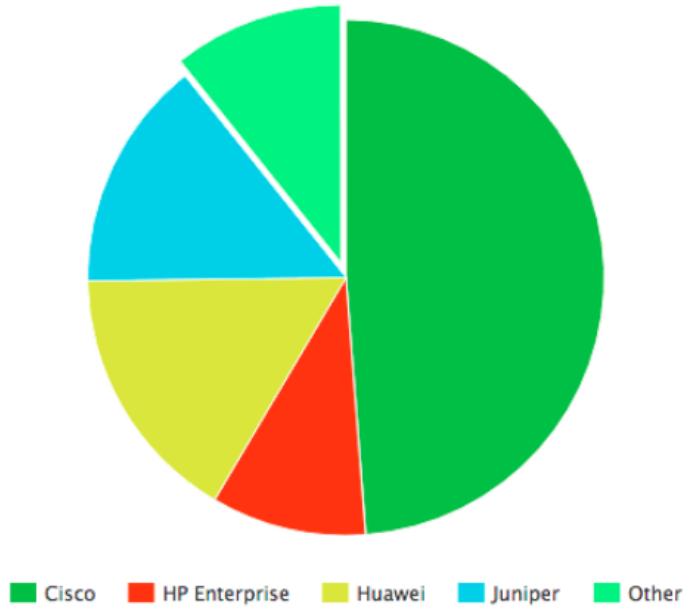


Figure 3.5: Companies dominating the market according to 2016 surveys [2].

In addition, due to those reasons, the ability of network operators to tailor the network becomes scarcely any. Those conditions create an unacceptable environment for future requirements of rapidly advancing network architecture with varying demands.

3.2.5 Security Issues

In this subsection, major security issues of traditional networks are gathered under three topics. These are especially the points that are naturally targeted by SDN architecture and features for the future networks which are desired to have more scalable, trusted and stable security systems [64].

- **Policies:** Network managers have to work with all protocol layers and interfaces in the standard network model that is ineligible to manage and troubleshoot together since they are designed in isolation by their nature. As a result of this complexity, to configure any network policy in a particular range, network managers may have to deal with adjustments thousands of devices and mechanisms to ensure robustness of the overall system. This makes implementing and deploying consistent policies for access control, security, and QoS quite challenging on that scale.

- **Perimeter-based Security:** In traditional network security precautions, the traffic labeled as "safe" is mostly based on the limited perimeter that statically defines features or borders of "safe" data. That perimeter security simply includes a firewall at the network level and a commercial antivirus system at the host level in most of the systems. Besides, most of the perimeter-based firewalls and generic antivirus solutions predicate manual rules and signatures to identify threats. In today's world, each and every day, tons of zero-day vulnerabilities are revealed and the time required for patching the system against those vulnerabilities without existing signatures are much more than modern systems could tolerate. As a result, it has become more and more difficult for traditional perimeter-based solutions to detect threats targeted current networks.

- **Static Environment:** In traditional networks, security appliances such as intrusion detection and prevention systems (IDS/IPS) and firewalls are deployed at fixed locations assuming a fixed perimeter or a set of certain points that are possible targets for external threats. However, considering the mobility of users, ad-hoc devices, and scaling systems, predicting network traffic pattern and origin of possible threats become nearly impossible. Moreover, the adoption of

the virtual networks and cloud systems creates another dynamic launch area that demolishes predictions. Consequently, the ability of consistently and accurately identify all threats based on the traditional deployment of network security systems has been the severely broken approach.

3.3 SDN and Solutions for Traditional Networks' Problems

In this part, we discuss how SDN offers solutions to the traditional networks' problems presented in the previous section. The solutions for traditional networks' problem are mostly originated from the logically centralized and programmable nature of SDN [65].

3.3.1 Adaptability

Centralized control of a network means that system-wide changes can be quickly applied without any side-effect on network operation since the system is able to be continuously monitored from a central perspective. Reducing time and effort needed to maintain the network and deploy new resources or applications may greatly increase an organization's agility and technological adaptability. For instance, without individual adjustments for each device in the network, a network administrator can easily integrate a new device to the network and configure the rest of the network from the central controller without causing a discontinuation in the network operation.

The adaptability of SDN can be considered as a solution for the static structure of traditional networks. Centralization that SDN adopted as a key feature provides a significant mechanism for dynamic, scalable and easily controllable network architecture. Adaptability of SDN is the means for establishing flexible future networks.

3.3.2 Management and Orchestration

Adapting and deploying new resources become significantly easy with SDN compared to traditional networks. Another simplicity procured by the SDN technology

is in the management and orchestration of the network. A central management console with an utter monitoring ability on network resources considerably simplifies adjusting and maintaining of both real and virtual networks.

For physical networks constituted in varying complexity, management and orchestration are main concerns for availability and performance for network administrators. Especially in networks with dynamic traffic patterns, direct manipulation and intervention are very critical. Considering the network tendency growing with mobile devices and sensors, SDN provides great flexibility to manage entire network and diverts data traffic using a centralized controller. Occasional peaks in data traffic and dynamic network participation are handled with SDN with ease in that manner.

For virtual networks and cloud deployments, scalability is one of the key points, especially for enterprises. SDN has a promising structure for this kind of networks which requires flexibility in both size and functionality with central controller and API-programmability at both end bounds.

3.3.3 Programmability

Programmability is the key concept of SDN. The idea of "programmable networks" is originated from modern networking characteristics such as fluctuating and dynamic traffic patterns, distributed and virtual networks and cloud computing [66]. Those and other network trends mentioned throughout the section require a significant amount of automation and orchestration. By using software-driven programmable controllers to automate and manage functions of a network, SDN can significantly boost efficiency and stability; also provide an opportunity for more specialized and robust security functions. Therefore, businesses can focus on innovation rather than operational tasks and costly maintenance.

Programmability of northbound and southbound provides a great opportunity for custom and modified network models. In this manner, programmable network devices and centralized controller are natural solutions for vendor lock-in at the network level (Haerick & Gupta, 2015). Network managers and administrators are able to develop on-demand device orchestration models without dependency on vendor-

Table 3.1: Advantages of SDN over traditional networks.

	Traditional Networks	**SDN**
Adaptability	Static and highly complex for multi-device mobile environments.	Adapting, easy-to-configure for different network requirements.
Scalability	Unsustainable due to complexity.	Easily scalable using centralized and agile management.
Transposability	Limited control and modification since changes are subject to vendors' product cycle.	Vendor-agnostic, overall control mostly belongs to enterprises or users, programmable.
Security	Limited, hard-to-manage and statically deployed security systems.	Flexible, programmable and centrally observable dynamic security systems.

provided device-specific software management. For unprecedented requirements of future networks, programmability is one of the core features that render SDN as the most promising network technology for the future networks (Astuto, Mendonca, Nguyen, Obraczka, & Turlett, 2014).

All in all, traditional networks have done the pretty good job so far, however, they have already given the signals of inadequacy for the future networks. SDN offers quite promising features and far-reaching solutions for current network limitations. The interpretation of those features is summarized in Table 1 by providing a very lean comparison of traditional networks to SDN (Portal et al.,2015).

Throughout the chapter, SDN's importance in terms of security is broadly discussed. In order to digest the place and importance of SDN for the future networks considering both security issues and other weaknesses of traditional networks, this section can be taken as a reference while reasoning on the arguments about security presented in the next sections.

3.4 Security Approaches on SDN

3.4.1 SDN for Security

The term "SDN for Security" means what security measures SDN could bring to the security domain. Every new concept and solution has its own advantages and disadvantages in terms of security. Those concepts that draw less attention to security issues, will introduce a larger number of threats. Even if the security is the top priority in all systems, we may end up with some serious problems and incidents. Security is a continuous process in the lifetime of a system.

The essence of SDN lies on the logically centralized architecture and the ability of deep programmability (Ahmad et al., 2015). While the community tries to leverage SDN, they have to develop novel security measures for SDN. In this section, we will focus on the contributions of SDN on security. Centralization and network programming are great advantages of SDN. However, from a different perspective, they may also be the downside for SDN. Any security measure that is not well designed and established may introduce weaknesses. Therefore the new features of SDN may also introduce new security vulnerabilities.

3.4.1.1 Logically Centralized Architecture in Terms of Security

Centralized control of a network means that system-wide changes can be quickly detected, analyzed and performed without any side-effect on network's operation since the system is able to be constantly monitored from a central perspective. Continuously monitoring overall network and collecting statistics of almost all considerable parameters such as packet source, delivery frequency, route-trace etc., identification and mitigation of different attack vectors have become observably effective and manageable in comparison with solutions in traditional networks. Besides, the flexibility that logical centralization gives has a great impact on maneuverability in case of anything suspected on any part of a software-defined network (Shin, Xu, Hong, & Gu, 2016). The mechanism, the logical central controller, enables direct manipulation and intervention on a pinned-down critical point considering its relations with other

network entities in the system.

Even though the logically central controller is a single feature of software-defined networks, it leads more other significant capabilities especially in terms of security. For instance, the simplified and dynamic management ability stems from the separation of control and data planes and has naturally permitted deployment of utterly specialized network security devices (Yegneswaran, Shin, Porras, & Gu, 2013). Moreover, with a central orchestration, they are able to work in collaboration to create a complete "defense and intervention" system taking account of multiple parameters. As a result, it has become quite possible to take advantage of different security entities in different pinpoints like distinctly located but globally related group of forwarding devices which are under control of a central controller. Figure 6 shows a very convenient example emphasizing this point. In the figure, using SDN-specific monitoring tools operating in-between controller and forwarding levels, possibly variant packet traffic could be observed and classified from a central perspective. Additionally and more importantly, another relatively more intelligent and sophisticated tool that is able to use widely collected network data and analyze this bulk for deducting more significant results about partial or entire network could easily be deployed on both southbound and northbound interfaces (Tantar, Palattella, Avanesov, Kantor, & Engel., 2014). In this sense, the ability to collect "big data" from the whole system through a central controller also creates both possibility and necessity of this kind of network-wide multipartite security deployments.

Apart from having an overall awareness on the whole network, centralized controller approach provides vast opportunities to make important changes and decisions even on heterogeneous networks considering all their effects to other elements lying in network's itself (Infonetics, 2015). Besides, this awareness and ability to make network-wide adjustments give an "orchestrator" role to a central controller that is responsible for management and supervision. Those characteristics mainly pave the way for orchestration of three major security requirements (Scott-Hayward, Natarajan, & Sezer, 2016):

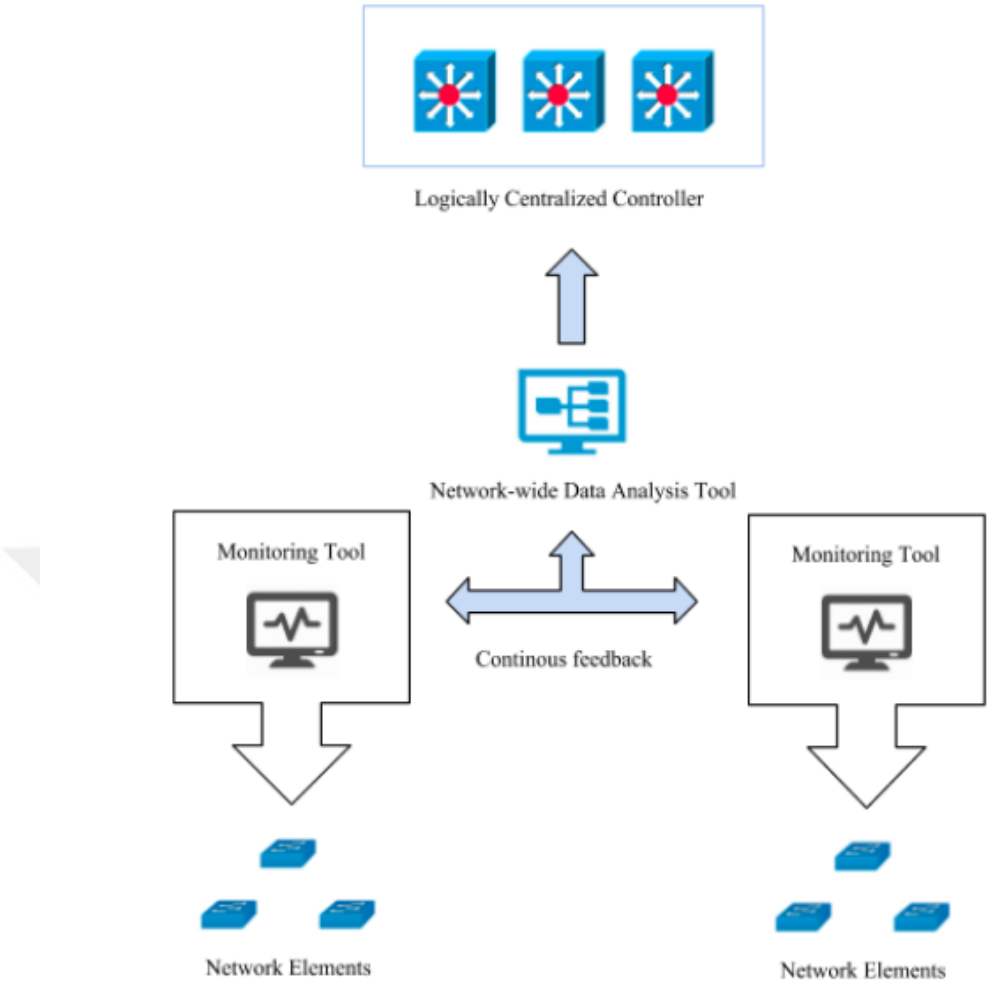


Figure 3.6: SDN with multipartite security tools taking advantage of the overall network view.

- **Authentication and Authorization:** Independent from network architecture, authentication and authorization are indispensable for a secure and reliable communication. Controller centralization in SDN comes with advantages of its design, for example, its own central architecture is available to be used as an authentication server coupled with a RADIUS server which conducts as network traffic orchestrator based on authentication and indirectly authorization (Mattos & Duarte, 2014). In addition, exonerating controller-RADIUS combination, since this available "central by-default authentication" approach of

SDN renders infrastructure-level network elements as a “controlled variable”, each and every device and their related sub-network could be easily audited and accounted using authorization cognizance. It also decreases the success chance of impersonation and masquerade of those simple data-plane switches without the sanction of the SDN controller (Kuliesius & Dangovas, 2016).

- **Security Policies:** Deploying consistent policies over large-scale networks for QoS and security is a real challenge. On the other hand, managing and providing their reliability are totally different concerns which are needed to be seriously taken consideration. Logically centric structure of SDN is quite promising remedy with different solutions developed on both northbound and southbound interfaces in its nature (Porras, Cheung, Fong, Skinner, & Yegneswaran, 2015). Those policies have become flexibly manageable using SDN applications considering feedback acquired from lower-level (controller and infrastructure levels) monitoring and analyzing tools. Actively using its monitoring ability on the overall system, central SDN controller dynamic migrate and reconfigure security policy inconsistencies possibly occurred from very basic firewalls to advance IDS/IPS solutions in a heterogeneous network environment.

- Dynamic and Occasional Intervention: Prevention, detection, and mitigation of a variety of security threats are primary capabilities of traditional network security devices. However, seeing the big picture that is all relationships between each element of a network is required for an advance potency to preside. The centrally-constructed controller of SDN presents and uses this potency to prevent network attacks by taking occasional precautions. For example, with a proactively installed set of rules, DDoS attacks targeted very distinct points of a network could be mitigated and as a result, bandwidth allocations of the overall network could be persistently protected using application and infrastructure level tools collaboratively (Lim., Ha, Kim, Kim, & Yang, 2014). This also leads new generation networks to more dynamic and moment-oriented IDS/IPS solutions with relatively higher orchestration capabilities of SDN networks.

3.4.1.2 Contributions of Programming Paradigm in Terms of Security

The idea of "programmable networks" is originated from modern networking characteristics such as fluctuating and dynamic traffic patterns, distributed and virtual networks and cloud computing [66]. Those and other network trends mentioned throughout the section require a significant amount of automation and orchestration. By using software-driven programmable controllers to automate and manage functions of a network, SDN can significantly boost efficiency and stability, and eventually, provide an opportunity for more specialized and robust security functions.

In traditional networks, security functions are typically defined by commercial software solutions, hardware middle-boxes or soft built-in features coming with network devices. These security approaches by predefined precautions are increasingly being elapsed since they have become insufficient to attain fluctuant QoS and security requirements of modern networks. In addition, while predicting disposition of attack threats on different parts of distributed networks is getting more and more challenging, expecting an ultimate protection from those non-flexible solutions are too precarious. In SDN, programmability of northbound and southbound interfaces provides a great opportunity to develop custom and modified security modules for different purposes and use cases [67].

By virtue of the multi-level structure of SDN (application, controller, and infrastructure levels), there shows up various solutions collaborating in-depth. Two different approaches directly related to programmability feature of SDN are Designing Security Interfaces for QoS and Custom Security Applications in Heterogeneous Networks.

Designing security interfaces for QoS are based on different systems with different topologies and necessities, security precautions depend on a wide range of parameters to be considered. This requisite leads the idea of defining flexible and configurable programming interfaces to be used for creating and managing targeted security policies by network owners. SDN presents a very convenient "programmability" feature which makes use of custom security frameworks at both high-level SDN application layer [68] and low-level infrastructure layer [69] possible.

This also brings the term network function virtualization (NFV) as a complementary of SDN architecture. Using those frameworks and NFV approach on top of the network elements, entire classes of custom network functions could be virtualized into building blocks that may chain them together to create more abstract services [70]. Advancing it, even in heterogeneous networks, this kind of frameworks and services draw an easy cross-device communication via a unified "language" among network elements or logically virtualized groups. Besides, abstraction of this approach triggers a more important advantage coming from SDN programmability, which is a vendor-independent security application for dealing with network-specific policies, QoS, and requirements.

Custom security applications in heterogeneous networks are directly related to the programmability of northbound and southbound interfaces effaced the vendor lock-in at the network level [71]. To be more precise, must-tools used for different intendments such as monitoring, intrusion detection and prevention and rule-based packet management have gone beyond commercial products with limited features.

Instead, for enterprises, developing their own network security products and modifying and deploying open-source products which are natural results of network programmability and specialized programming interfaces for security policies have become prior solutions [72]. When it is possible to use their operation-specific tools, enterprises reduce time and effort needed to defend their overall network structure protected by to-the-target network security deployments.

For example, an open-source network monitoring tool that could be personalized to concentrate on different parameters on different points of the network would be quite effective to identify DDoS attacks targeted elements with varying critical level [73]. Moreover, customization of this tool enables to collaborate it another custom tool relying on infrastructure or controller level via a predefined interface.

3.4.2 Security for SDN

The infrastructure of information systems has rapidly transformed into cloud computing technologies recently by providing important innovations on data centers. This

evolution affects the user behavior and users claim to access their data from anywhere and anytime.

Besides, network operations are evolved from operator based management approach to more autonomous management and control [58]. Security concerns are highlighted as an important obstacle in this transformation. On the one hand, securing user data is very important; on the other hand mobility and virtualization technologies create new threats. Furthermore, human factor leads to unnecessary flaws, redundant expenditures, and unauthorized accesses.

From the security point of view, present solutions are strained to deploy, manage, program and scale the security throughout heterogeneous equipment from multiple vendors since network policies are tightly bundled to physical resources instead of services and applications. Moreover, the management of consistent security policies is very difficult with today's networks in computing, storage, and networking domains. To solve these problems, there are an increasing demand for software-defined networking (SDN) solutions that are mostly preferred for factors like rapid error eliminations and easy installations of new versions [74].

In SDN, the control plane is separated from the underlying network traffic forwarding devices. Then, the control logic transforms into a logically centralized controller that simplifies policy enforcement, network reconfiguration, and evolution. As emphasized in the previous part, SDN's fundamental concept could potentially have some disadvantages. Especially, SDN's centralized architecture could cause single-point-of-failure in case of any compromise of the controller. To secure the controller and provide the security of the software network, mechanisms such as middle-boxes can be developed and deployed using the northbound application programming interface (API) of the software-defined network controller. One of the candidate solutions for the middle-boxes is the intrusion detection system (IDS) that is either signature- or anomaly-based [75].

In this part, we will first present OpenFlow protocol to explain SDN's communication mechanisms with network devices. Then potential cyber-attack vectors in SDN environment will be mentioned. Finally, we will put forward the use of intrusion detection systems (IDS) in SDN to secure the network.

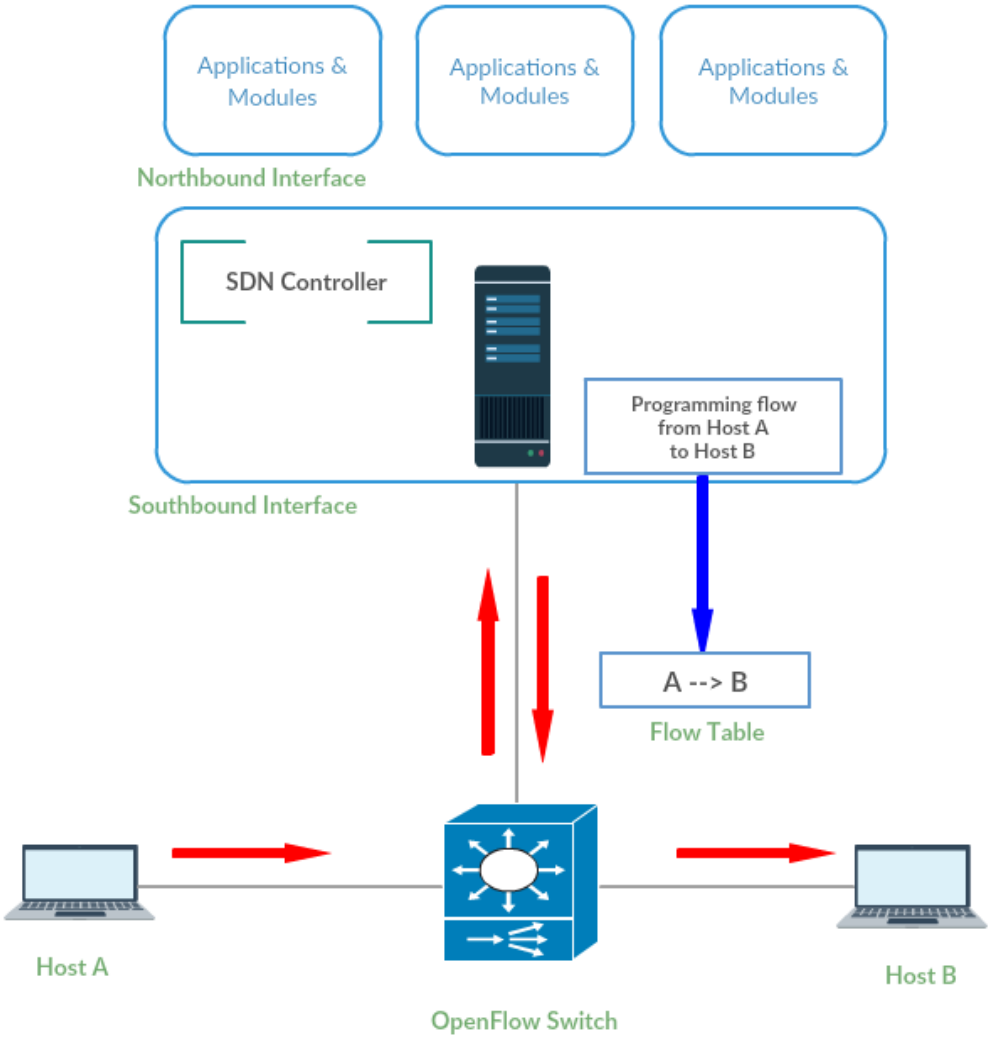


Figure 3.7: Packet forwarding scheme on SDN.

3.4.2.1 OpenFlow Protocol

One of the dominant protocols used in the southbound API of SDNs is the OpenFlow protocol. The OpenFlow protocol is used by SDN controller to communicate with network devices and network traffic is managed by the controller with respect to the defined rules. Both the SDN controller and the OpenFlow based switch store the flow table which has programmed rules such as the packet forwarding methods.

Figure 3.7 shows the operation order of packet flow which has been set by the SDN controller. If the OpenFlow switch has no entry in flow table when the packet is re-

ceived, it sends the packet to the controller. What to do with the packet is decided by the modules and applications that run on the controller. Then, the new entry is transmitted to the switch's flow table by the controller. As a result, the running algorithms of the modules and applications on the controller establish the network traffic.

3.4.2.2 SDN Specific Attack Vectors

Typical attack vectors are presented in Figure 3.8. There are a lot of threat vectors that are defined for the SDN architecture [76]. Most of these attacks, such as exploiting logically centralized controllers, capturing the entire network through compromised controller and malware deployment through controllers are associated with the compromise of the controller. Because the control plane is more vulnerable to cyber threats than data plane and it is prone to single point-of-failure due to SDN's centralized nature.

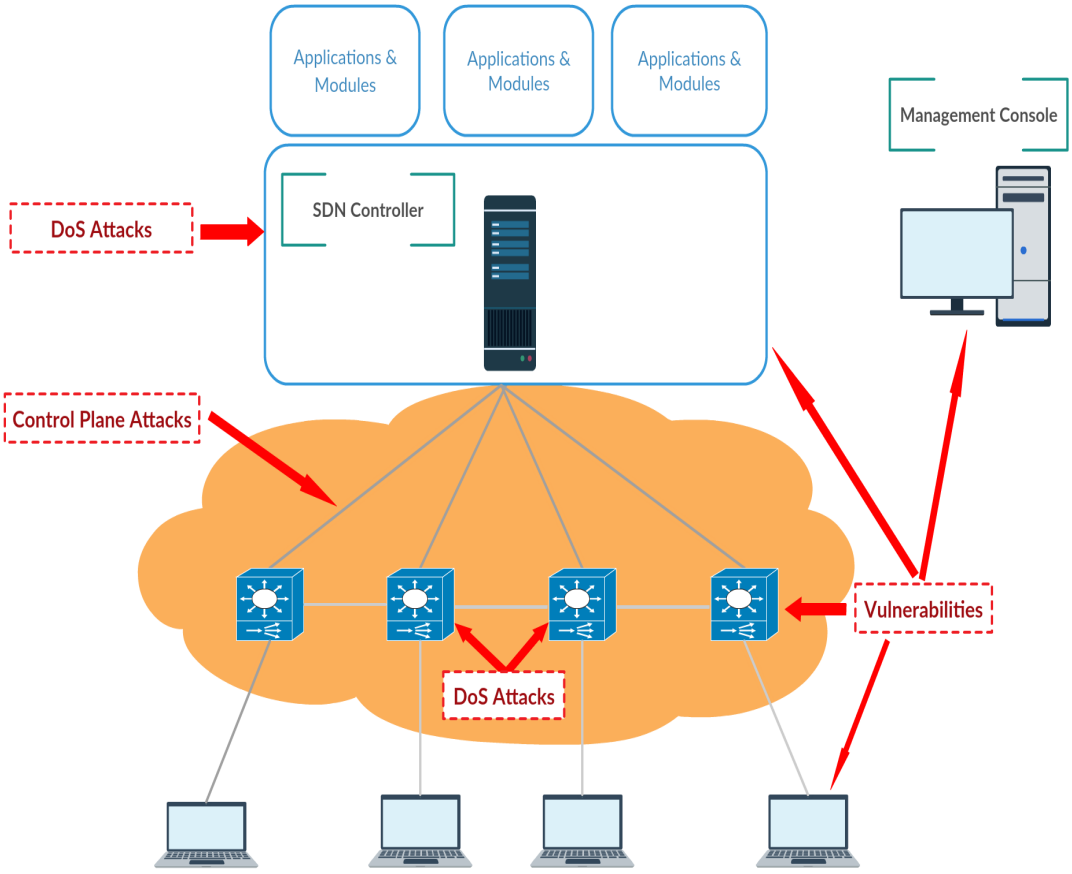


Figure 3.8: Potential attacks in SDN architecture.

In a computer network, intrusion detection systems detect malicious network traffic. Then, some security mechanisms isolate the insecure network devices from the network. When a potential threat is discovered, the security issue is identified and resolved by those mechanisms with the automatically downloaded patches. The infected device is allowed to re-connect to the network after the threat is removed. The security problem, which normally cannot be detected by network's any single device, can be easily and effectively eliminated by this active security mechanism.

In today's networks, the mentioned security solutions are generally installed as proprietary systems in which each device fulfills its functionality based on its limited knowledge of the network's state and not knowing how other devices act. These solutions are generated for stationary network traffic and have to monitor real-time ingress flow. Especially for the future's data-centers, this approach is not flexible since these data-centers have virtualized workloads, highly dynamic network traffics and multiple synchronous security policies. Also, future's high-speed network connections make this mechanism hard to be operated. From the elasticity viewpoint, software networking will pave new ways for better security solutions.

3.4.2.3 Intrusion Detection Systems

Network IDSs are mainly categorized into the two types: signature-based IDSs and anomaly-based IDSs. In signature-based intrusion detection systems, pattern (signature) matching approaches are employed. Abnormal behaviors in the network are defined by signatures. The downside of signature-based IDSs is the insufficiency for new attack types. In anomaly-based intrusion detection systems, the normal behavior in the network is modeled and exceptions are identified using machine learning techniques. Anomaly-based IDSs also have the ability to detect novel attack types relative to the signature-based IDSs [77]. From this perspective, anomaly-based IDSs are considered to be more powerful.

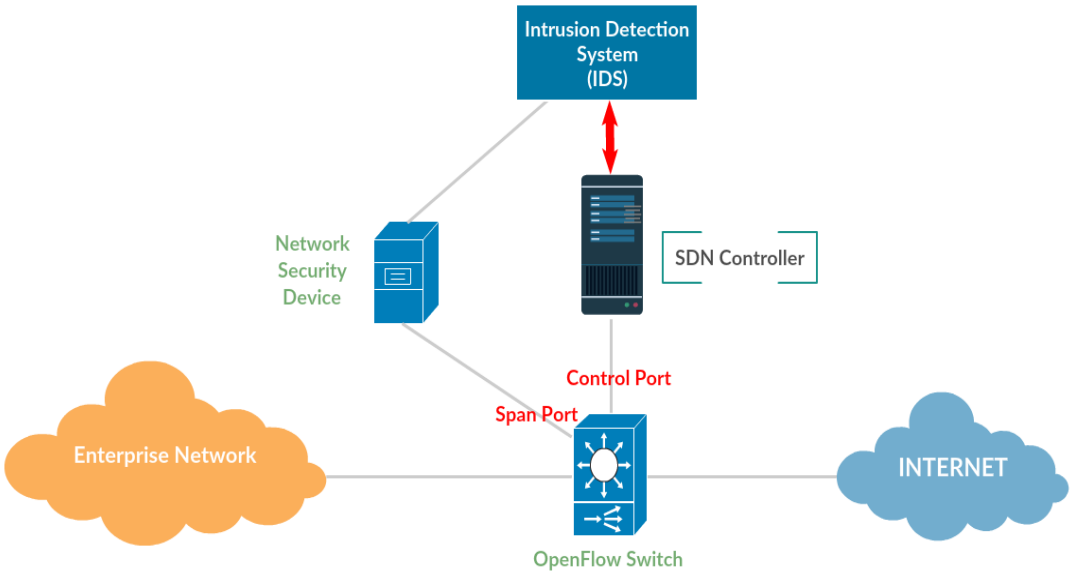


Figure 3.9: IDS usage in SDN environment.

3.4.2.4 Example Anomaly-based IDS Implementation in SDN

Anomaly-based IDSs in SDN environment is a new paradigm [78] and a more flexible solution for IDSs can be enabled with SDN as shown in Figure 3.9. Anomaly-based IDS functionality is centralized on the SDN controller where an efficient security processing is provided. In this environment, IDS can isolate and monitor only suspicious network traffic instead of completely isolating networking devices.

The flow-based paradigm is particularly well suited for this scenario because it can manage policies and coordinate service-chaining efficiently. In addition, it could benefit from the elasticity achieved through SDN. This example anomaly-based intrusion detection scenario consists of two stages:

- **Infection:** An end user in enterprise network clicks on an attachment embedded in a phishing e-mail and malware is installed on user's host machine.
- **Security breach:** The malware begins the probing attacks and tries to botnet's connect command and control server outside the enterprise network.

The anomaly-based IDS would proactively respond to this malware attack as follows:

- **Observe:** The anomaly-based IDS observes the traffic pattern from the malware to the outside network devices. This traffic has the stream of port scans executed on the enterprise network from the malware on the infected host.
- **Detect:** According to the traffic profile that has the information of who was communicated to whom, the anomaly-based IDS detects that the malware on the infected hosts.
- **React:** The anomaly-based IDS creates an infection profile and sends a quarantine notification to the centralized controller device.
- **Respond:** Controller device converts the quarantine notification to pre-defined OpenFlow rules and pushes them to the OpenFlow switch. As a result, user's malware hosting machine is isolated from the enterprise network with respect to predefined rules.

Policies are dynamically and transparently applied to an individual switch port with this solution by using device (or user) port information. The response time required for security threats is reduced thanks to automatic reconfiguration. Also, the necessity of having networking personnel to define and apply a policy to administrate network access is removed. The requirement of network's user policy application and manual configuration are minimized by this approach.

This solution does not need any networking software/hardware other than the basic OpenFlow-enabled switch or networking device. Thus, it is fully interoperable with all vendor implementations. Moreover, it has the potential to minimize operating expenditures, provide automatic re-configuration of edge-port security parameters and allow the users' mobility on the edge of the network.

CHAPTER 4

PROPOSED ARCHITECTURE

4.1 Overview

Recent network security approaches have already started utilizing SDN for mitigating certain types of attacks such as DDoS [79] and target link flooding [80], using statistical properties of flows.

In this chapter, an SDN-assisted security architecture for IoT networks, which utilizes an ensemble of Prototypical Networks and SVM for intrusion detection, is proposed. The proposed methodology is a more generalized approach capable of detecting attacks of various types with machine learning-based network intelligence, even in the absence of large training samples for particular classes of attacks. The proposed approach allows for the system's dynamic extensibility to detect new types of attacks as they arise by continuous updates to the intelligence model with new attack data.

In the proposed SDN-assisted intrusion detection architecture, heterogeneous IoT devices are connected to the network through openflow-enabled networking devices managed by SDN, as explained by Wang et al. [81]. While the SDN controller here is responsible for providing agility and resource optimization in the IoT network through the dynamic configuration of flow rules, it is also responsible for isolating detected attacks through issuing flow rule updates.

The SDN controller dynamically manages the local traffic in the network, and it can also communicate with the wide-area network (WAN) through the application layer, updating its rules based on attacks detected at other SDN controllers in the WAN. The WAN links several different actuators, gateways, and tools that provide device-to-

cloud communications. The SDN should be compatible with the access, WAN, and cloud technologies as a network control paradigm. The proposed approach mainly uses three modules, namely *Statistics/Feature extractor*, *Traffic intelligence/Classification*, and *Mitigation*, at the SDN application layer to achieve intelligent security.

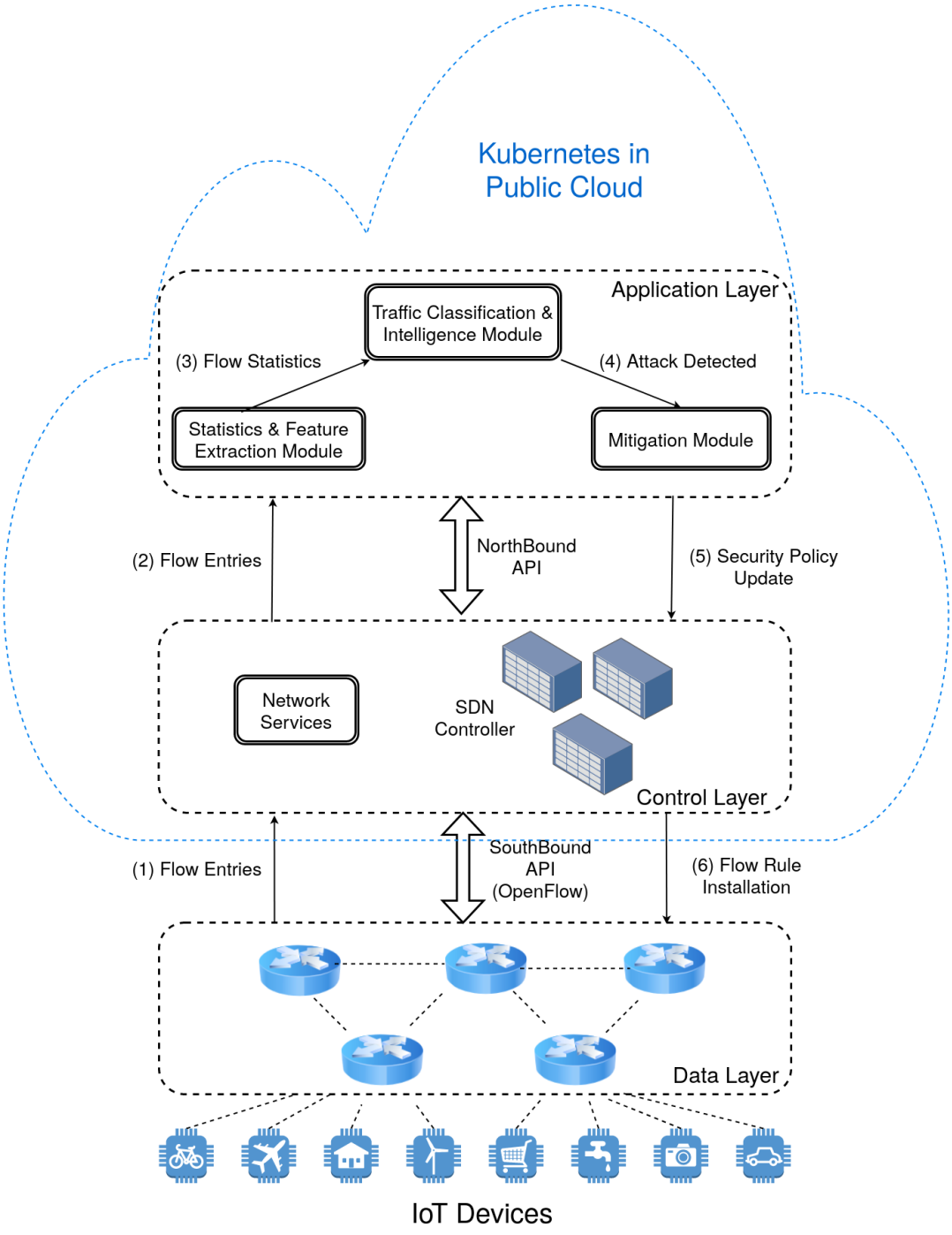


Figure 4.1: Proposed architecture.

The end-to-end operation of the proposed approach is shown in Figure 4.1. In this approach, the SDN controller and related application layer modules are installed in a Kubernetes cluster [82] in a public cloud. The OpenFlow access to the SDN controller is enabled with a load-balancer type Kubernetes service by which the data layer switches can connect to the controller. On the one hand, Kubernetes, as an orchestrator, provides high-availability; on the other hand, any data layer equipment from anywhere can access the controller as the Kubernetes service for OpenFlow has a public IP address. As to the best of our knowledge, there is no proposed solution in the literature where the SDN controller is deployed into a Kubernetes cluster in a public cloud.

The solution involves collecting data from the data plane switches at regular intervals using the controller. At such intervals, flow entries are collected from switches, and the feature values required by the intelligence module are calculated for these flows. Every flow record is classified by the intelligence module to figure out whether the flow is benign or corresponds to one of the attack classes in the trained machine learning model. In case of attack detection, the mitigation module takes an appropriate decision based on the nature of the detected attack and communicates its decision in the form of a security policy update to the controller.

Security policy updates are in the form of updates to flow rules in the affected switches, e.g., dictating the switches to drop packets from a host that has been deemed malicious due to the classification by the traffic intelligence module. The architecture provides the flexibility to update the trained machine learning models when additional training data becomes available and additional attacks are discovered. One data source for such training data is the threat intelligence database that can serve multiple SDN controllers.

Algorithm 1 summarizes the operation of the proposed approach. In the below subsections, we provide detailed descriptions of the three modules.

Algorithm 1 : Continuous Detection & Mitigation

```
1: L_S ← SDN switch list
2: while true do
3:     for each switch S in L_S do
4:         R ← Pull flow rules from S
5:         F ← Extract features from R
6:         L_A ← DETECT_ATTACKS(F)
7:         if L_A contains any attack then
8:             MITIGATION(L_A)
9:         end if
10:    end for
11:    sleep 1 second
12: end while
```

4.2 Statistics/Feature Extractor

This module retrieves flow entries from switches at regular intervals. To do that, first, the list of switches in the network is retrieved through the controller. In an infinite loop, the feature extractor application traverses the switch list and reads flow entries from each of them.

The feature extractor calculates the values of all required machine learning features (input data) for every flow using the flow entry fields as a data pre-processing step. (Note: The features are explained in detail in the Experimental Evaluation section.) Some of these features are calculated using all of the switch entries, e.g., the standard deviation of flow durations, total packets per second in a transaction, and the number of inbound connections per source IP.

These features are calculated with an initial pass over the flow entries. The extractor also creates a hash map and four hash sets, which contain some flow entry statistics. The hash map's keys are composed of source/destination IP addresses, and port numbers and values of the hash map are packet count, byte count, and the number of packets per second.

The hash map is used later for retrieving statistics of reverse flows instead of passing over all of the flow entries for each flow. The hash sets store source IP addresses, destination IP addresses, destination port numbers, and protocols. They are used to calculate the number of incoming connections per source/destination IP, the total number of packets per destination port, and protocol.

Other features are specific to each flow, e.g., flow duration, number of packets, number of destination to source (reverse flow) packets. In the second pass, flow specific features are retrieved using flow entry fields and the hash map that contains reverse flow statistics. The formed feature vectors are then sent to the intelligence module as input for the running intrusion detection algorithm.

One important thing for this module to be noted, it works on the flows, which only come from the hosts, not switches. Before the calculations start for a flow, if it came from a switch port connected to a host or another switch is checked first. If the flow is from another switch, the feature extractor module does not examine it and apply the forwarding logic. Otherwise, if it is from a connected host, the module is run for the flow.

4.3 Traffic Classification/Intelligence

This module receives feature vectors from the Statistics/Feature Extractor module and classifies them using the ensemble algorithm that utilizes the outputs of the Prototypical Network and SVM models trained for the network.

The utilization of an ensemble of machine learning models rather than simply Prototypical Networks enables the system to benefit from improved performance when the training dataset size increases sufficiently to allow SVM to achieve high performance. We use a weighted average ensemble model in this module to find the machine learning model with the best attack detection performance.

A weighted average ensemble [53] is an extension of a model averaging ensemble, where each model's performance weights each member's contribution to the final class prediction. Model weights are small positive values, and the total of all weights

is equal to one, which allows weights to indicate the expected contribution or level of trust from each member.

The module first reads the set of all target classes. The pre-trained models of Prototypical Networks and SDN are retrieved afterward. Then, the optimum weight value of the Prototypical Networks model, E_W, is read. E_W is calculated in the model construction phase for the best F1 score of the ensemble model on held-out data for all the possible values between 0.1 and 0.9 (by 0.1 increments).

The optimum weight value for SVM is obtained by subtracting E_W from 1. A data normalization step is applied to the feature vectors to generate a more usable format for the classifiers. In this step, the data is normalized in between zero and one, and categorical features/labels are converted to numeric values. The distance matrix is found by giving the normalized features to the Prototypical Networks model. The softmax function is applied to the distance matrix to calculate a probability distribution for the model.

Also, the class probabilities are retrieved from the SVM classifier. All the target classes are traversed one-by-one to construct the weighted average ensemble predictions. For each target class, the prediction probabilities of Prototypical Networks and SVM are multiplied with their own optimum weight value. The prediction probability of the ensemble model is the weighted sum of the prediction probabilities of the included models. In the final step, the class predictions are determined by the arguments of the maxima (*argmax*) over the class predictions of the ensemble.

Upon detection of an attack, that is, if the output of the ensemble model is not equal to the normal traffic; the prediction and the other relevant fields of flow such as switch id, source/destination switch port, source/destination IP, etc. are added to the detected attacks list. Then, the traffic Classification&Intelligence module informs the mitigation module to perform the needed security policy updates.

The intelligence module's machine learning model is not static; it evolves in time to integrate new attack data that either provide additional samples for existing attacks or completely new (zero-day) attacks that were never observed in that network before.

Integration of new attacks into the model is achieved through a threat intelligence

database shared by multiple networks, which the intelligence module periodically checks for updates. A possible extension of the intelligence module involves humans-in-the-loop, who approve/reject traffic classification results of the running machine learning model to provide feedback that will improve the model, i.e. perform active learning with human-in-the-loop.

A particular flow entry classified by the model as a specific type of attack with high confidence can be added to the model as additional training data. Algorithm 2 summarizes the operation of the detection process.

Algorithm 2 : Attack Detection

1: **procedure** DETECT_ATTACKS(F)
2: $C \leftarrow$ Set of all target classes (including normal)
3: $PROTONET \leftarrow$ Prototypical Network model trained for the network
4: $SVM \leftarrow$ SVM model trained for the network
5: $E_W \leftarrow$ The optimum value for the weighted average ensemble of PROTONET
6: $F\prime \leftarrow$ Normalize Features(F)
7: $DIST \leftarrow$ Get distance matrix for each class in PROTONET($F\prime$)
8: $P_{pro} \leftarrow$ Calculate softmax over distances ($DIST$)
9: $P_{svm} \leftarrow$ Retrieve class probabilities from SVM($F\prime$)
10: $P \leftarrow$ Initialize the probabilities array
11: **for** each c ϵ C **do**
12: $P_\prime \leftarrow P_{pro}[c] * E_W$
13: $P_{\prime\prime} \leftarrow P_{svm}[c] * (1 - E_W)$
14: $P[c] \leftarrow P_\prime + P_{\prime\prime}$
15: **end for**
16: $prediction \leftarrow$ argmax(P)
17: **if** $prediction$ is **not** equal to "Normal" **then**
18: $L_A \leftarrow$ Add $prediction$ to the attacks list
19: $L_A \leftarrow$ Add relevant fields of attack flow
20: **end if**
21: return L_A
22: **end procedure**

4.4 Attack Mitigation

The attack mitigation module is informed by the intelligence module when an attack is detected. This module is responsible for making security policy update recommendations to the controller based on the detected attack type. Algorithm 3 summarizes the operation of the mitigation process that takes place after being notified by the intelligence module.

Algorithm 3 : Attack Mitigation

```
1:  procedure MITIGATION(L_A)
2:      T ← Set the threshold value
3:      H ← Initialize the hash-map
4:      for each attack A in L_A do
5:          SwitchSrcPort, SwitchDstPort ← Extract from attack (A)
6:          H′ ← Increase by 1 for SwitchSrcPort, SwitchDstPort pair
7:          if H′ is bigger than T then
8:              M ← Extract from attack (A)
9:              Install drop or redirection rule in switch for M
10:         end if
11:     end for
12: end procedure
```

For example, if the intelligence module has ruled that a volumetric DoS attack sourced from a single IP address is in progress, the mitigation module can recommend blocking that source IP address, routing traffic from that IP address to an analysis endpoint (e.g. a honeypot) or limiting the sending rate of the suspected attacker.

In case of MAC or IP address spoofing, blocking might be done based on the physical port of the switch that the attacker is connected to. The attack mitigation module installs flow rules with a higher priority than normal flow rules. Therefore, malicious packets are matched with flow rules installed by this module rather than passing over existing flow rules or triggering new flow rule installations. The mitigation module counts the attack predictions per source switch port - destination port pairs. Once the number of detections per pair exceeds the predefined threshold value, a drop or

redirection rule is added to the switch to prevent/redirect the related traffic.

Id:	790000ffd6483a
Match:	IPV4_SRC
Actions:	OUTPUT:1
Priority:	100
State:	ADDED
Table:	0
Duration:	35
Packets:	10563
Bytes:	1647828

(a) Packet redirection.

Id:	790000ffefa4aa1
Match:	IN_PORT
Actions:	DROP
Priority:	200
State:	ADDED
Table:	0
Duration:	27
Packets:	143455
Bytes:	8607300

(b) Packet drop.

Figure 4.2: Fields of sample flow rules installed by the mitigation module after attack detection.

Figure 4.2 shows examples for fields of flow rules installed by the mitigation module. The rule in 4.2a is installed when the mitigation module decides to redirect the traffic coming from a source IP address. All of the packets with the specified IP address are directed to the output port regardless of destination IP, source and destination port numbers. The rule in 4.2b is installed when the mitigation module decides to drop packets coming from the host connected to the specified switch port. All of the packets coming from the specified switch port are dropped regardless of source and destination IP addresses and port numbers.

4.5 Highly Available SDN Controller and NFV Services

Kubernetes is the most common container-orchestration platform built under the Cloud Native Computing Foundation (CNCF) [83] umbrella. It helps to build, scale, and manage state-of-the-art applications throughout their entire life cycles.

The typical way to make a service in Kubernetes highly available is to use several replicas of a pod such that if one pod fails, another pod is running and can serve

up traffic in its place. A Kubernetes deployment object is an entity that handles a collection of pods and holds as many replicas as needed. A crucial piece of this is to provide service discovery and constant health checking that can predict a pod's malfunction and reroute the traffic to healthier pods.

Regular health checks are critical for applications so that Kubernetes can decide whether the pod is failing. When a health check fails, Kubernetes will either interrupt the network traffic or restart the pod. Furthermore, nodes can die without warning, and Kubernetes can also detect the type of failure. A node sends the master a normal heartbeat, and if the master does not receive a heartbeat, it is assumed that the node has crashed. The pods operating on the failed node are rescheduled to other nodes.

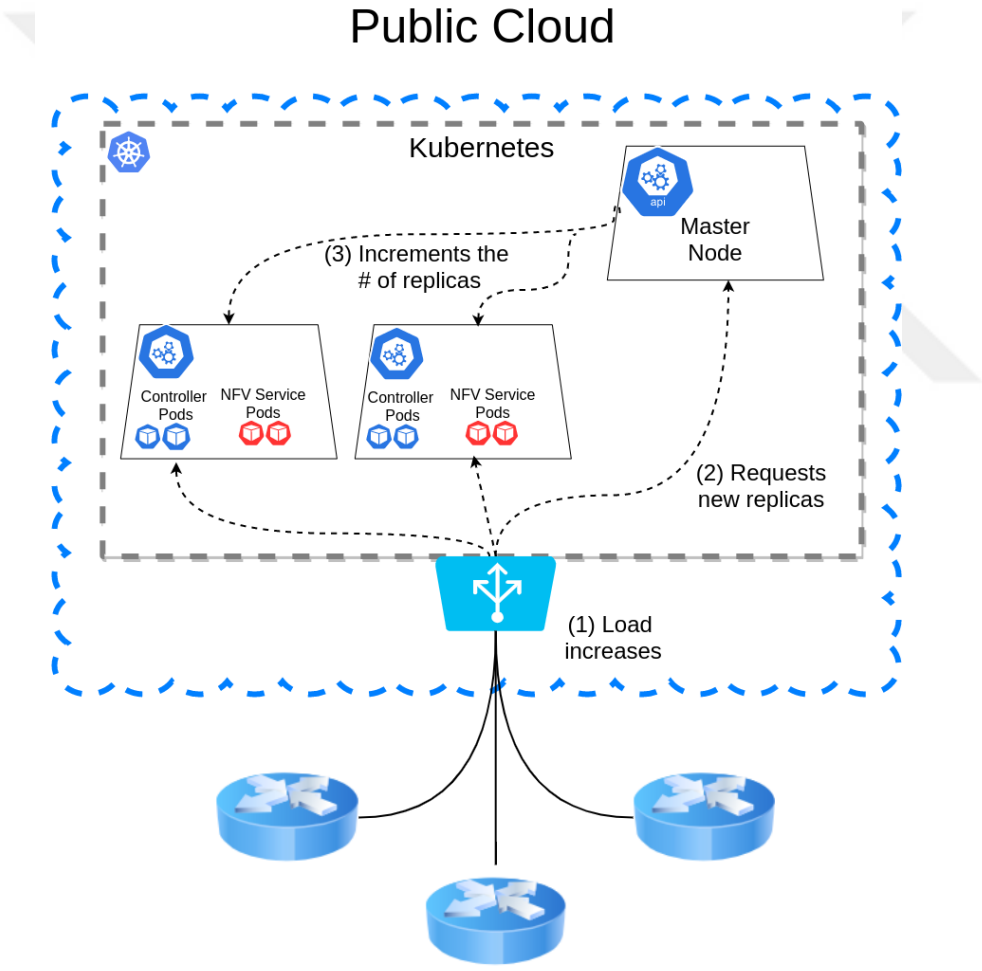


Figure 4.3: Auto-scaling SDN controller and NFV services on increased loads.

Using pod replication for high availability means that an application can execute mul-

tiple pod instances. Such replications can be quickly produced with Kubernetes' auto-scaler features. Mainly, there are three types of auto-scaling features:

- **Cluster Auto-scaler:** The Kubernetes Cluster Auto-scaler updates the number of nodes in the cluster automatically when pods fail to start due to the lack of resources or when nodes in the cluster are underused, and their pods may be rescheduled to other nodes in the cluster. The cluster auto-scaler is a Kubernetes utility that increments or decrements the size (by adding or deleting nodes) of a Kubernetes cluster, depending on the size of pending pods and node usage metrics.
- **Horizontal Pod Auto-scaler:** The Kubernetes Horizontal Pod Auto-scaler (HPA) dynamically adjusts the number of pods in a set of deployments, replication controllers, or replicas depending on CPU use of that resource. HPA can also be configured to take custom or external metrics-based scaling decisions. HPA is a perfect tool for ensuring that critical services are elastic and that they can scale up to meet that demand and scale down to ensure optimal resource usage.
- **Vertical Pod Auto-scaler:** The Kubernetes Vertical Pod Auto-scaler sets the CPU and RAM reserves for the pods dynamically to support the applications "right-size." This will help to make the best use of the cluster and free up CPU and memory for other pods. VPA seeks to reduce the overhead management for access request configurations and container limits and raise cluster space usage.

Figure 4.3 illustrates the auto-scaling of SDN controller and NFV services. The SDN controller and NFV services are deployed into a Kubernetes cluster in a public cloud to provide the high-availability. OpenFlow switches connect to the SDN controller with its load balancer type IP address through the southbound API. The NFV services, namely Feature Extractor, Traffic Classification, and Mitigation Modules, are connected to the SDN controller via the northbound API.

When the load balancer faces an overload in the requests, it notifies the Kubernetes API, which lives in the master node. The master node implements auto-scaling by increasing the number of SDN controllers and NFV service pods. Furthermore, in a

situation where a custom/external metric is below a predefined threshold, the master node scales-down the cluster by reducing the number of nodes and pods.

CHAPTER 5

EXPERIMENTAL EVALUATION

5.1 Machine Learning Experiments

5.1.1 Datasets and Experimental Setup

Various publicly accessible datasets were created in the past two decades to evaluate the performances of intrusion detection models. While the KDD Cup'99 dataset [84] and a refined version of it, namely NLS-KDD [13] are the most commonly used, they are fairly old and contain no contemporary normal and attack behaviors, which may bias the assessment of intrusion detection models. In this thesis, the BoT-IoT and UNSW-NB15 datasets, along with our self-generated SDN dataset to evaluate the performance of our proposed ensemble learning model are used.

5.1.1.1 Bot-IoT Dataset

The first dataset utilized for the evaluation of the proposed intrusion detection model is the very recently released BoT-IoT [85] dataset, which was generated in an IoT testbed and represents real IoT network flows. This dataset includes more than 73 million network flow records, which include samples for OS Fingerprint, DDoS, DoS, Data Theft and Service Scan attacks in addition to normal (attack-free) data.

Since the original size of the Bot-Iot dataset is too large, we downsampled it into an extracted dataset. We have randomly selected 20000 samples from DDoS, DoS, Service_Scan, and OS_Fingerprint categories. We took all the samples of Normal and Theft classes, because they have low volumes. Table 5.1 shows the distribu-

tion of flow samples through categories from both the original Bot-IoT and extracted datasets.

Table 5.1: Distribution of records in the Bot-IoT datasets.

	Original Dataset		Extracted Dataset	
Category	Size	%	Size	%
DDoS	38.1M	52.51	20,000	21.9
DoS	32.9M	44.98	20,000	21.9
Service_Scan	1.46M	0.19	20,000	21.9
OS_Fingerprint	358,275	0.05	20,000	21.9
Normal	9,543	0.001	9,543	10.47
Theft	1,587	0.0002	1,587	0.17
	73.3M		91,130	

5.1.1.2 UNSW-NB15 Dataset

The UNSW-NB15 [86] dataset was developed in 2015 by the Australian Cyber Security Center (ACCS) research community. Approximately 100 GB of raw data was created, reflecting authentic normal and attack records. 49 features were developed using various network extractor tools. There are 9 different types of attacks in the data collection, including Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.

The total number of records in the original dataset is 2,540,044. Different partitions from this dataset were configured as a training set and testing set. In our experiments, we removed the Analysis, Backdoor, Shellcode, and Worms categories from both the training and test datasets, because of their rare numbers, which would affect the statistical significance of the results. The compositions of the original training/test and extracted training/test sets are shown in Table 5.2.

Table 5.2: Number of records in the UNSW-NB15 datasets.

	Training Dataset		Test Dataset		Extracted Datasets			
Category	Size	%	Size	%	Size	%	Size	%
Normal	56,000	31.9	37,000	44.9	56,000	32.8	37,000	45.8
Generic	4,000	22.8	18,871	22.9	4,000	23.4	18,871	23.3
Exploits	33,393	19.1	11,132	13.5	33,393	19.6	11,132	13.8
Fuzzers	18,184	10.3	6,062	7.3	18,184	10.6	6,062	7.5
DoS	12,264	7.1	4,089	4.9	12,264	7.2	4,089	5.1
Reconnaissance	1,491	6.1	3,496	4.2	1,491	6.1	3,496	4.3
Analysis	2,000	1.1	677	0.8				
Backdoor	1,746	0.09	583	0.7				
Shellcode	1,133	0.06	378	0.4				
Worms	1,30	0.007	44	0.05				
	175,341		82,332		17,332		8,650	

5.1.1.3 SDN Dataset

Using the Mininet [87] simulation tool, we created a simulation environment similar to the BoT-IoT network for the third machine learning experiment set. Figure 5.1 displays the layout of the testbed. Five hosts simulating IoT devices transmit tiny volumes of data to a server in the network at frequent intervals.

Two benign hosts send large volumes of data to the server, and four malicious computers organize attacks on the server and IoT devices. An Open vSwitch links all the hosts. We have used ONOS as the SDN controller device, and the controller has been attached to the switch to manage the network.

The BoT-IoT [85] dataset has nearly 9500 normal traffic samples, and we have analyzed these records to establish our own normal dataset traffic to have identical packet transmission speeds, packet sizes, and protocols. Simulated IoT hosts send small packets by using TCP with an interval of 12 to 15 seconds to the server, randomly chosen at the start time.

The default time-out on the flow rule is 10 seconds. So each packet sent causes a new version of the flow rule to be installed. One host has transmitted packets using TCP to

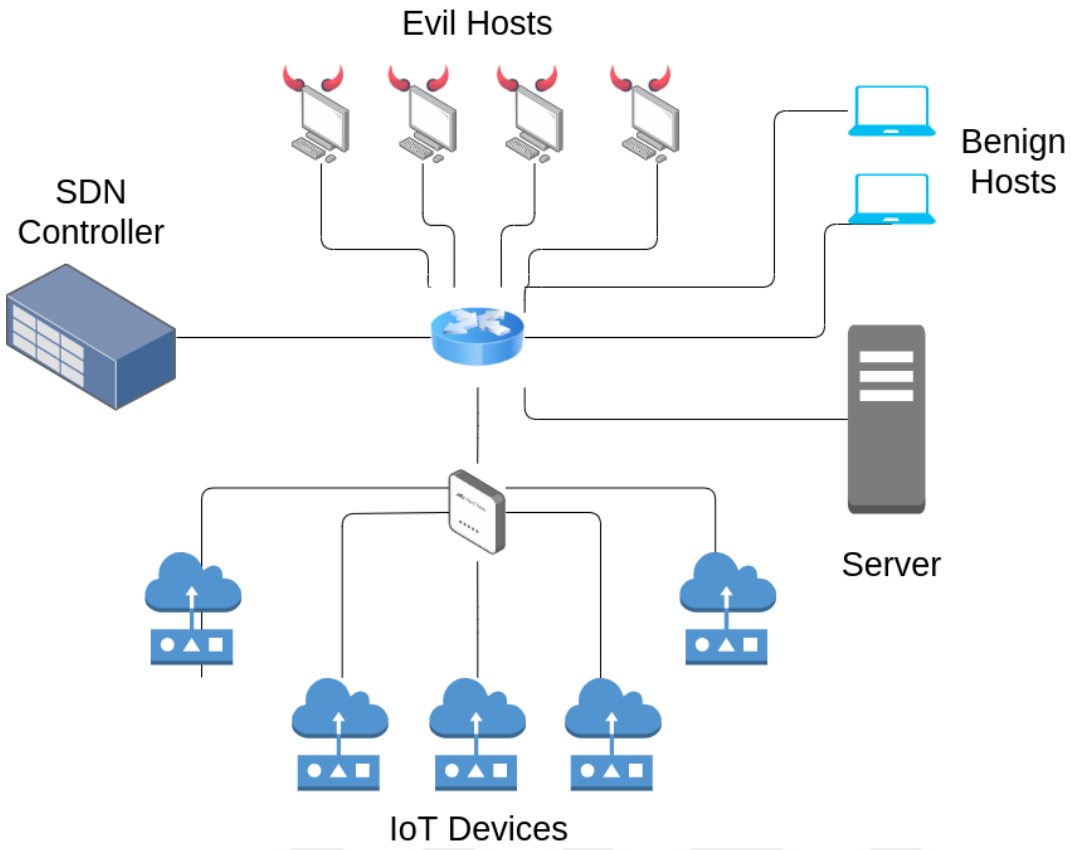


Figure 5.1: Testbed setup.

the server, and the other one transmitted packets using UDP protocol. We employed four sending rates and two packet sizes for TCP packets. Two separate sending rates and packet sizes have been utilized for UDP packets.

We have chosen send rates and packet sizes in proportion to the rates selected in the BoT-IoT dataset. We have recorded normal traffic during attacks are implemented and attack-free scenarios. To generate intrusion records, we employed hping3 and Nmap attack utilities, as did in the BoT-IoT dataset.

Hping3 tool with TCP SYN flood and UDP flood options was used for creating DoS and DDoS attacks. Attackers targeted the server and IoT nodes, by sending 4000, 6000, 8000 and 10000 packets per second, with and without spoofing the source IP addresses. For port scan analysis, and operating system identification attacks the Nmap tool-set was used. The Boofuzz software with HTTP/FTP fuzzing options was used for fuzzing attacks. One attacker host targeted the HTTP and FTP servers

running on the server.

An ONOS program was developed to take the flow rules out of each OpenFlow switch every one second. Then, the ten features depicted in Table 5.3 extract to calculate the values of each flow. These 10 features were determined in order to put less load on the system and to obtain the best detection performance with the least number of calculated features. Because the ONOS controller fails to retrieve some features that BotIot dataset owns, only measurable features were created using statistics on the flow entries.

Table 5.3: Selected 10 features from the SDN dataset.

Feature name	Feature description
Dur	Record total duration
Mean	Average duration of aggregated records
Stddev	Standard deviation of aggregated records
Sum	Total duration of aggregated records
Min	Minimum duration of aggregated records
Max	Maximum duration of aggregated records
Spkts	Source-to-destination packet count
Dpkts	Destination-to-source packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count

Our dataset includes more than 20 million traffic samples, consisting of 5.92 million Distributed Denial-of-Service (DDoS), 4.85 million Denial-of-Service (DoS), 1.15 million fuzzing, 1.05 million Operating System and service inspection, and 6.41 million port scan attacks, as well as 750 thousand normal traffic records.

Considering that, to the best of our knowledge, since there doen not exist a publicly accessible SDN-specific intrusion detection data-set, the development of an SDN-based IDS dataset is very valuable in terms of accurate simulation of the operating environment along with a capacity evaluation to extract certain traffic flow attributes.

5.1.2 Experiment Results

A weighted average ensemble [88] is a strategy that enables multiple models to act in proportion to their confidence or expected output towards a prediction. Our proposed intrusion detection model, namely Proto-S, is also based on a weighted average ensemble technique, and consists of the Prototypical Networks and Support Vector Machines (SVM) models. Our experiments show that the proposed model outperforms state-of-the-art ML models that were previously shown to achieve very high accuracy on large datasets, especially when trained with much fewer data records.

We evaluated the performances of the different classifiers for different types of attacks in the network. For comparison, we have chosen 6 other state-of-the-art machine learning models, namely (a) SVM [89] due to its long-standing success in generalization and intrusion detection tasks; (b) One-Dimensional Convolutional Neural Networks (CNN) [47] as a representative of deep neural networks, also demonstrated to be highly successful in intrusion detection; (c) Naive Bayes (NB) [90] for its lightweight nature, which could speed up the detection process, and various approaches including (d) Adaboost [52], (e) Deep Auto-Encoders (DAE) as a deep learning model, with its unsupervised feature learning, also demonstrated to be very successful in intrusion detection tasks [91].

The proposed Prototypical Networks - SVM ensemble (Proto-S) and other models were trained and tested on the BoT-IoT, UNSW-NB15 and SDN traffic datasets. In addition, k-fold cross-validation (10 times in our experiments) was used to achieve high performance and decrease the bias of the machine learning techniques. All detection models were implemented in Python 3.6.8, and the classifiers were developed using the PyTorch [92], Keras [93], and Sklearn [94] libraries. For each k-fold iteration, the best tuning parameters for SVM and NB were selected using sklearn library gridSearchCV.

For generating a 1D-CNN model, we utilized a sequential model developed with Keras [93]. Figure 5.2 illustrates the 1D-CNN model architecture used in experiments. We define the model as having two consecutive convolution layers, followed by a dropout layer for regularization, then one pooling layer. A dense layer with a 100

unit long output occurs at the end of the network, followed by a softmax layer. After the convolution and pooling layers, the learned features go through a fully connected layer and are flattened to one long vector. Finally, the output layer makes a prediction.

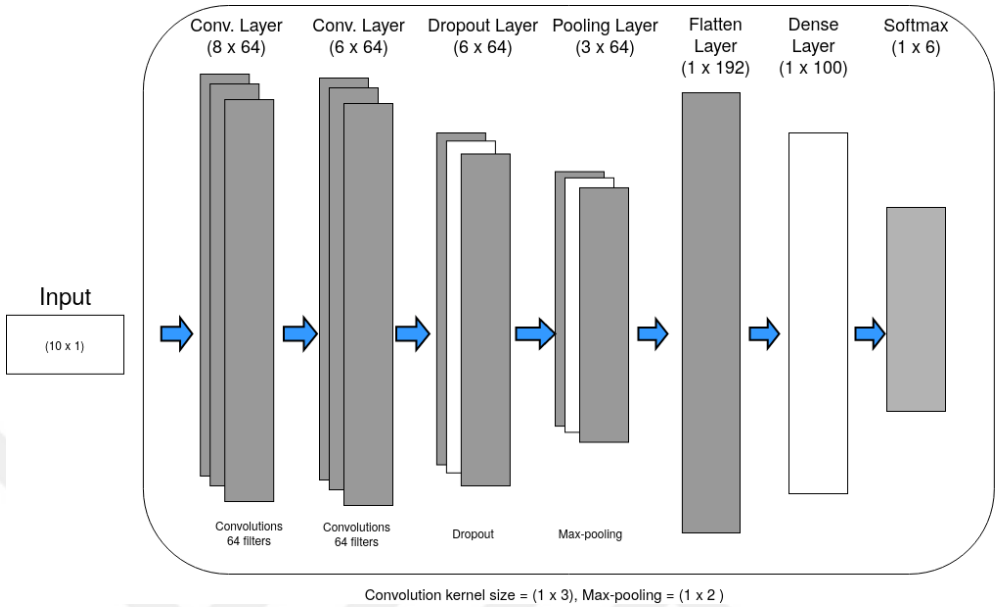


Figure 5.2: One-dimensional convolutional neural network (1D-CNN) model used in experiments.

The 1D-CNN model uses a kernel size of 3 and a standard configuration of 64 parallel feature maps. Also, the model employs an efficient Adam version of stochastic gradient descent to optimize the network, as well as the categorical cross-entropy loss function given that the aim is to solve a multi-class classification problem. To attain the best performance, we employed hyper-parameter optimization for the 1-D CNN model on the following parameters; batch_size = [10, 20, 40, 60, 80, 100] and the number of epochs = [10, 50, 100].

For Deep Autoencoders (DAE) to be used as classifiers, they need to perform an unsupervised feature learning and then a supervised fine-tuning. To do that, the encoder part (un-supervised) is copied to a separate model and modified to give classification results (supervised). Briefly, the encoder part takes the training data without using input labels and generates a compressed representation of input data. Afterward, a supervised model takes the encoder part's output as the input and is trained with training data labels. Finally, a softmax layer classifies the attack categories from the

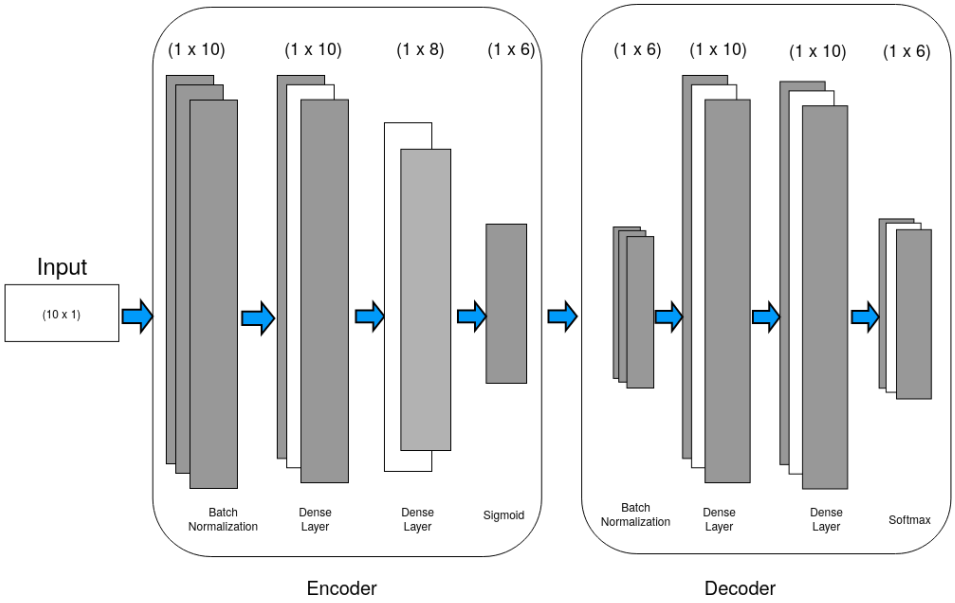


Figure 5.3: The Deep Auto-encoder (DAE) model used in experiments.

input dataset in the last hidden layer. We created a DAE model that satisfies the aforementioned conditions, following the work of Farahnakian et al. [91].

Farahnakian et al. achieved over 94% accuracy on the KDD'99 dataset [84]. That is why we decided to follow their work in our experiments. The network architecture of the deep auto-encoder model used in the experiments is given in Figure 5.3. The network takes input from our SDN dataset and represents all the 10 features. The encoder part of the DAE model consists of a batch-normalization, two fully-connected (dense) layers, and a sigmoid layer. On the other hand, one batch-normalization, two consecutive dense layers, and a softmax layer comprise the decoder part of the model. The output layer is used as an output of the entire DAE model and it represents six available classes of the SDN dataset. We applied hyper-parameter optimization for the DAE model on the following parameters; batch_size = [10, 20, 40, 60, 80, 100] and the number of epochs = [10, 50, 100].

In the experiments' scope, a multi-class version of the support vector machines (SVM) model is utilized by setting the *slack penalty* equals to 10, and *gamma* equals 1, and the *kernel* is as RBF. Two parameters need to be considered when training an SVM classifier with the Radial Basis Function (RBF) kernel: the *slack penalty* and

the *gamma*. The *slack penalty* parameter is common to all SVM kernels and trades against the simplicity of the decision surface from training examples' misclassification. While a low *slack penalty* value makes the decision surface smooth, a high *slack penalty* aims at correctly classifying all samples of training. On the other hand, the *gamma* value defines how much influence a single training example has. That is why a larger *gamma* value affects the other examples more.

For the development of SVM, Naive Bayes (NB), and AdaBoost clasifiers which are used in experiments, we benefited from the Sklearn [94] library. The NB model employs a list of hyper-parameters that enables the additive smoothing by setting the *alpha* parameter to 1; it also adjusts the *fit_prior* parameter to "True" for the model to learn prior class probabilities. Also, the AdaBoost model uses five *estimators* and sets the *learning rate* parameter to 0.3 to construct an optimized classifier.

5.1.2.1 Evaluation Metrics

Most existing work in the field of intrusion detection utilizes the accuracy, precision, recall, and F1-score metrics to evaluate the goodness of detection models. We also have utilized the same metrics in this work. Table 5.4 shows the confusion matrix for the definitions utilized in metric calculations.

Table 5.4: Intrusion detection confusion matrix.

	Attack	**Normal Traffic**
Attack	Correctly predicted attack samples True Positives (TP)	Incorrectly predicted normal samples. False Negatives (FN)
Normal Traffic	Incorrectly predicted attack samples False Positives (FP)	Correctly predicted normal samples True Negatives (TN)

- **Accuracy** (AC) is the ratio of accurately categorized normal and attack records to the total samples.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.1}$$

- **Precision** specifies what percentage of successful predictions is accurate.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall** (Detection Rate) is the ratio of correctly classified attack samples.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- F_1 **Score** is an overall measure that takes up both Precision and Recall metrics to see how well the model successfully detects the attacks.

$$F_1 = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{5.4}$$

It is important to select the correct metric for finding the best model. Precision is a good measure when the cost of false positives are high, whereas the recall shall be the model metric when there is a high cost associated with false negatives. If we want to find a balance between Precision and Recall, the correct metric to use is the F1 score. Therefore, for our experiments, we chose F1 as the primary metric to perform comparisons between the different models.

5.1.2.2 Hyper-parameter Selection

The embedding network of Proto-S model is similar to the one proposed by Snell et al. [46]. Every CNN block contains a convolution, a batch-normalization, a non-linearity of Rectified Linear Unit (ReLU) which directly outputs the input if it is positive (otherwise, it outputs zero), and an avg-pooling layer. It is worth mentioning that our network uses avg-pooling [95], because it converges faster with it, instead of the max-pooling which is proposed in the original paper [46]. To determine the best encoding size, the number of CNN blocks, and the number of support/query sample sizes, we applied the following experiments on the BotIot dataset.

1. **Encoding Sizes:**

   The network proposed by Snell et al. [46] results in a 64-dimensional encoding when applied to the support and query data points. We experimented the encoding sizes 16, 32, 64, 128 and 256 on our network to choose the correct

parameter for the model. Table 5.5 shows that the best weighted F1 score performance is taken with the encoding size 128.

Table 5.5: Encoding size experiments.

Encoding Size	Precision	Recall	F1	Accuracy
128	0.86	0.80	0.82	0.80
256	0.85	0.78	0.81	0.78
16	0.79	0.74	0.76	0.74
32	0.80	0.73	0.76	0.73
64	0.75	0.70	0.69	0.70

2. **The Number of CNN Blocks:** On the second step of hyper-parameter optimization, we experimented with different CNN block sizes between 1 and 4. Table 5.6 compares the performances of Prototypical Networks with various CNN blocks & encoding sizes.

   We observed that 3 and 4 CNN blocks show the same weighted F1 score performance with 0.82. Because the model with 4 CNN blocks outperforms the other model in terms of weighted precision metrics, we decided to use the model comprised of 4 CNN blocks.

Table 5.6: The number of CNN blocks experiments.

# of CNN Blocks	Encoding Size	Precision	Recall	F1	Accuracy
4	128	0.86	0.80	0.82	0.80
3	128	0.84	0.80	0.82	0.80
4	256	0.85	0.78	0.81	0.78
2	64	0.84	0.79	0.81	0.79
2	128	0.83	0.78	0.80	0.78
3	64	0.81	0.78	0.76	0.78
4	16	0.79	0.74	0.76	0.74
4	32	0.80	0.73	0.76	0.73
4	64	0.75	0.70	0.69	0.70
1	64	0.63	0.62	0.62	0.62
1	128	0.62	0.59	0.60	0.59

3. **The Number of Support/Query Samples:** Snell et al. [46] propose that, for prototypical networks, it is usually best to train and test with the same "shot" (support/query) number. To experiment with the suggestion, we utilized a various set of support/query samples.

Table 5.7 presents the performances of ProtoNet models with 128 encoding size, 4 CNN blocks, and different support/query sets. The models with 5-support/5-query and 10-support/10-query showed the same F1 scores with 0.83. Since the model with 5-support/5-query is better than the other one in terms of precision performance, we decided to use 5 samples from each support/query sets in the subsequent experiments.

Table 5.7: The number of Support&Query samples experiments.

# of Support/Query	Precision	Recall	F1	Accuracy
5s_5q	0.87	0.82	0.83	0.82
10s_10q	0.86	0.82	0.83	0.82
10s_90q	0.85	0.81	0.82	0.81
15s_15q	0.85	0.81	0.82	0.81
45s_45q	0.85	0.81	0.82	0.81
60s_40q	0.85	0.81	0.82	0.81
20s_20q	0.85	0.80	0.82	0.80
30s_30q	0.85	0.80	0.82	0.80
35s_35q	0.85	0.80	0.82	0.80
95s_5q	0.85	0.80	0.82	0.80
50s_50q	0.84	0.80	0.82	0.80
20s_80q	0.84	0.80	0.81	0.80
25s_25q	0.84	0.80	0.81	0.80
40s_40q	0.84	0.80	0.81	0.80
70s_30q	0.84	0.80	0.81	0.80
30s_70q	0.84	0.79	0.81	0.79
90s_10q	0.84	0.79	0.81	0.79
80s_20q	0.83	0.79	0.80	0.79
40s_60q	0.80	0.75	0.74	0.75

5.1.2.3 Bot-IoT Dataset Experiments

In the first set of machine learning experiments, the proposed intrusion detection model (Proto-S) and the baselines were constructed utilizing the 16 network flow features (Table 5.8) from the BoT-IoT dataset. To make the feature selection SDN-compatible, we paid attention to using the features we could obtain from the SDN controller.

Table 5.8: Selected 16 features from the Bot-IoT dataset.

Feature name	**Feature description**
Pkts	Total count of packets in transaction
Bytes	Total number of bytes in transaction
State	Transaction state
Dur	Record total duration
Mean	Average duration of aggregated records
Stddev	Standard deviation of aggregated records
Sum	Total duration of aggregated records
Min	Minimum duration of aggregated records
Max	Maximum duration of aggregated records
Spkts	Source-to-destination packet count
Dpkts	Destination-to-source packet count
Sbytes	Source-to-destination byte count
Dbytes	Destination-to-source byte count
Rate	Total packets per second in transaction
Srate	Source-to-destination packets per second
Drate	Destination-to-source packets per second

The number of samples in the BoT-IoT dataset is very high. However, the records have not been evenly distributed across the dataset, that is, the dataset is imbalanced. This imbalance is an important problem for most classifiers, therefore we down-sampled labeled records in the dataset to the size of 20,000, except the Normal and Theft classes, as their sizes are small (9543 and 1587 respectively). Moreover, to

generate a more usable format for the classifiers, we normalized the data (to between [0,1]) and converted categorical features/labels to numeric values.

The experiments were conducted by splitting the extracted dataset into different sizes of training, validation and test sets. We have randomly selected 100, 200, 400, 800, and 1000 samples from each class for the training set, 100 samples for validation, and the rest for testing purposes.

Table 5.9 presents the F1 scores obtained for each model per number of training samples. The results reveal that the proposed Proto-S model exhibits high performance even for small training dataset samples. Also, CNN, as a deep learning model underperforms in the low amount of training data volumes. With increasing the sample size from 100 to 1000, it is seen that CNN's F1 score improves.

Table 5.9: F1 scores per model for different training sizes of the Bot-IoT dataset.

	# of training samples				
Models	100	200	400	800	1000
Proto-S	**0.84**	**0.85**	**0.85**	**0.85**	0.84
ProtoNet	0.83	0.84	0.84	**0.85**	0.83
AdaBoost	0.65	0.70	0.76	0.75	0.75
SVM	0.62	0.63	0.66	0.73	0.73
CNN	0.57	0.62	0.64	0.64	**0.86**
DAE	0.43	0.54	0.52	0.58	0.62
NB	0.38	0.39	0.38	0.37	0.33

SVM's performance is also rising when the volume of training data is increasing. While the F1 score of SVM is higher than that of CNN for 100, 200, 400, and 800, CNN surpasses SVM on 1000 training samples. This result proves that deep learning models require more extensive training data for better detection rates. Also, the DAE model is not improving as much as CNN with more training records. We suspect as the reason for that, the input size, which is 16 feature values for a single input, is too small for the deep auto-encoder to extract features and decode it into a prediction.

Table 5.10 compares all models' performances for all the classes in the dataset for

the training size 100. For all the five attack categories, the Proto-S model produces almost the best F1 scores. We see that our model achieves 90% F1 scores for DoS and DDoS attacks. Because DoS attacks have been getting enlarged surfaces in IoT networks in recent years, this result also proves the proposed model's importance. Only in the Theft category, all the classifiers (except AdaBoost classifier) do not show a reasonable detection rate. The number of test records for this category is 1387, while each of the other classes has 10000 test samples. We suppose that this imbalance affects statistical significance, and thus all the classifiers show poor performance.

Table 5.10: Class-based F1 scores of the models trained with Bot-IoT training set size 100.

	DDoS	DoS	OS_Fingerprint	Service_Scan	Theft	
# of rec.	(10000)	(10000)	(10000)	(10000)	(1387)	Weighted Avg.
DAE	0.62	0.00	0.56	0.33	0.04	0.43
NB	0.62	0.09	0.43	0.12	0.02	0.38
AdaBoost	0.94	0.92	0.00	0.52	**0.82**	0.65
ProtoNet	0.94	0.91	0.71	**0.73**	0.35	0.83
SVM	0.87	0.36	0.62	0.60	0.21	0.62
CNN	0.81	0.41	0.57	0.44	0.18	0.57
Proto-S	0.97	**0.93**	**0.72**	**0.73**	0.36	**0.84**

The proposed Proto-S model not only shows better performance in the F1 score but in other metrics as well. Table 5.11 compares the efficacies of the models in terms of all the metrics for the training dataset having 100 samples. For all the evaluation metrics except for precision, the proposed model outperforms the other baseline machine learning models. These results mean that, in a real-world scenario where the IDS is deployed as a north-bound application into the SDN environment, more than eight attacks out of 10 would be successfully detected and mitigated, even when only 100 training samples are used from each threat category to create the model.

Table 5.11: Performance evaluation of different models for 100 training samples of the Bot-IoT dataset.

Models	Precision	Recall	F1	Accuracy
Proto-S	0.88	**0.82**	**0.84**	**0.82**
ProtoNet	0.87	0.81	0.83	0.81
AdaBoost	0.61	0.71	0.65	0.71
SVM	0.67	0.63	0.62	0.63
CNN	0.68	0.58	0.57	0.58
DAE	0.52	0.52	0.43	0.52
NB	0.55	0.42	0.38	0.42

5.1.2.4 UNSW-NB15 Dataset Experiments

We performed our second set of machine learning experiments on the UNSW-NB15 dataset. It is important to note that this dataset was prepared for legacy networks and does not exhibit IoT characteristics. We still provide results on this dataset as a sanity check of the relative performance of our model compared to others.

In pre-processing of the dataset, we first selected 16 SDN-compatible features out of the 49. The list of selected features and their definitions are given in Table 5.12. UNSW-NB15 comes along with pre-defined training/test splits. Table 5.2's first two parts display the distribution of records in original training and test datasets between the various levels. We removed the samples of the classes Analysis, Backdoor, Shellcode, and Worms to get a more balanced dataset.

Afterwards, we split the extracted training dataset into different sizes to be used in the experiments. From the extracted training set, 100, 200, 400, 800, and 1000 records for each class were randomly selected for the training process, while another random 100 samples were spared for the validation steps. The whole extracted test dataset is reserved as hold-out data for testing purposes.

Table 5.13 provides a comparison of the models in terms of the F1 scores. The per-

Table 5.12: Selected 16 features from the UNSW-NB15 dataset.

Feature name	**Feature description**
proto	Transaction protocol
dur	Record total duration
sbytes	Source to destination bytes
dbytes	Destination to source bytes
sload	Source bits per second
dload	Destination bits per second
spkts	Source to destination packet count
dpkts	Destination to source packet count
smeansz	Mean of the flow packet size transmitted by the src
dmeansz	Mean of the flow packet size transmitted by the dst
is_sm_ips_ports	If source IP addresses equal to destination IP addresses, and source and destination port numbers are equal, this variable takes value 1, else 0.
ct_dst_ltm	No. of connections of the same destination address in 100 connections according to the last time.
ct_src_ltm	No. of connections of the same source address in 100 connections according to the last time.
ct_src_dport_ltm	No of connections of the same source address and the destination port in 100 connections according to the last time.
ct_dst_sport_ltm	No of connections of the same destination address and the source port in 100 connections according to the last time.
ct_dst_src_ltm	No of connections of the same source and the destination address in 100 connections according to the last time.

formance metric has been measured for each training data size. While the proposed model produces better results in low numbers of training samples, the CNN model evolves with increased train data sizes. Except for the training set sizes 400 and 1000, our model surpasses the baseline algorithms in terms of F1 score performance. However, a CNN model trained with 400 and 1000 traffic flows shows better performance than the proposed model. As the UNSW-NB15 does not possess the SDN characteristics and is designed for legacy networks, we can say that our model would work well in non-SDN environments as well. Our Proto-S model outperforms other all classifiers in all training data distributions.

Table 5.13: F1 scores per model for different training sizes of the UNSW-NB15 dataset.

	# of training samples				
Models	100	200	400	800	1000
Proto-S	**0.58**	**0.60**	0.60	**0.63**	0.64
CNN	0.57	0.58	**0.62**	0.60	**0.67**
ProtoNet	0.56	0.59	0.59	**0.63**	0.64
SVM	0.56	0.56	0.56	0.54	0.55
NB	0.42	0.43	0.44	0.44	0.43
AdaBoost	0.42	0.48	0.54	0.34	0.53
DAE	0.32	0.35	0.52	0.48	0.46

In Table 5.14, it can be seen that the proposed Proto-S approach is better than or equal to all other baseline models in terms of accuracy, precision, recall, and F1 on average.

Table 5.14: Performance comparison of the different models for 100 training samples of the UNSW-NB15 dataset.

Models	Precision	Recall	F1	Accuracy
Proto-S	**0.77**	**0.52**	**0.58**	**0.52**
CNN	0.72	**0.52**	0.57	**0.52**
ProtoNet	**0.77**	0.50	0.56	0.50
SVM	0.76	0.49	0.56	0.49
NB	0.66	0.37	0.42	0.37
AdaBoost	0.64	0.37	0.42	0.37
DAE	0.55	0.27	0.32	0.27

Table 5.15 compares the performances of all models for all the classes in the dataset. Except for the Fuzzers and Generic attack types, the proposed model results in better F1 scores than the other baseline models. For those classes, the performance of CNN is only 1% higher than that of our model. In the other three categories, DoS, Exploits, and Reconnaissance, our model surpasses the CNN model by about %8.

Table 5.15: Class-based F1 scores of the models trained with UNSW-NB15 training set size 100.

# of rec.	DoS (4089)	Exploits (11132)	Fuzzers (6062)	Generic (18871)	Reconnaissance (3496)	Weighted Avg.
DAE	0.19	0.21	0.05	0.68	0.13	0.32
NB	0.22	0.26	0.16	**0.75**	0.18	0.42
AdaBoost	0.22	0.02	0.19	0.70	0.32	0.42
ProtoNet	0.29	0.45	0.18	0.73	0.53	0.56
SVM	0.29	0.45	0.17	0.73	0.33	0.56
CNN	0.27	0.42	0.20	**0.75**	0.37	0.57
Proto-S	**0.31**	**0.49**	0.19	0.74	**0.51**	**0.58**

5.1.2.5 SDN Dataset Experiments

As to the best of our knowledge, there does not exist any publicly available SDN-based intrusion detection dataset yet. Furthermore, the public datasets like Bot-IoT and UNSW-NB15 do not fully exhibit SDN characteristics. In order to address this issue, we developed cyber attacks in an SDN testbed, and captured them into CSV files. The dataset's features along with the descriptions are given in Table 5.3. By using these features and class sample sizes of 100, 200, 400, 800, 1000, and 10000 we produced training sets, prepared the validations datasets, created a large enough (60K samples) test dataset.

Table 5.16 shows that the proposed Proto-S model is better than or equal to the other models between training sizes 100 and 1000. For 10K training samples, the CNN as a deep learning approach gives better F1 scores than our model.

Table 5.16: F1 scores per model for different training set sizes of the SDN dataset.

	# of train samples					
Models	100	200	400	800	1000	10K
Proto-S	**0.68**	**0.70**	**0.70**	**0.72**	**0.72**	0.73
ProtoNet	**0.68**	0.69	**0.70**	0.71	0.71	0.72
SVM	0.50	0.53	0.52	0.54	0.55	0.64
CNN	0.45	0.57	0.61	0.65	0.66	**0.75**
NB	0.43	0.42	0.42	0.41	0.41	0.42
AdaBoost	0.39	0.49	0.48	0.50	0.55	0.56
DAE	0.37	0.38	0.42	0.49	0.52	0.51

Table 5.17 shows that the proposed Proto-S model by far outperforms other baseline algorithms for this set of experiments as well.

Table 5.17: Performance comparison of different models for 100 training samples in the SDN dataset.

Models	Precision	Recall	F1	Accuracy
Proto-S	**0.70**	**0.69**	**0.68**	**0.69**
ProtoNet	0.68	0.68	**0.68**	0.68
SVM	0.54	0.52	0.50	0.52
CNN	0.53	0.50	0.45	0.50
NB	0.51	0.49	0.43	0.49
AdaBoost	0.44	0.44	0.39	0.44
DAE	0.47	0.43	0.37	0.43

Table 5.18 compares the F1 scores of all models for all the classes in the dataset. In all categories, the proposed model produces better F1 scores than the other base-

line models. When we look in detail, we see that our model achieves above 60% F1 scores for DDoS, DoS, and Port_Scanning attacks, %84 for Fuzzing, and %72 for OS_and_Service_Detection and close to 70% F1 performance on average. The achieved results make the proposed approach promising for adoption in real-world SDN-assisted IoT networks, as the dataset exhibits SDN characteristics and has been prepared in an SDN environment.

Table 5.18: Class-based F1 scores of the models trained with the SDN dataset training size 100.

	DDoS	DoS	Fuzzing	OS_and_Service_Detection	Port_Scanning	
# of rec.	(10000)	(10000)	(10000)	(10000)	(10000)	Weighted Avg.
DAE	0.23	0.40	0.50	0.60	0.02	0.37
NB	0.25	0.09	0.66	0.57	0.52	0.43
AdaBoost	0.14	0.42	0.55	0.57	0.54	0.39
ProtoNet	0.61	0.60	0.83	**0.72**	0.64	**0.68**
SVM	0.24	0.42	0.73	0.55	0.48	0.50
CNN	0.30	0.07	0.69	0.55	0.52	0.45
Proto-S	**0.62**	**0.61**	**0.84**	**0.72**	**0.65**	**0.68**

5.2 Networking Experiments

In the scope on this section, we deployed an SDN controller into a Kubernetes [96] cluster. For the experiments, an Azure Kubernetes Service instance (AKS) [97] is used as a Kubernetes cluster.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: onos-onos
  name: onos-onos
spec:
  replicas: 1
  selector:
    matchLabels:
      app: onos-onos
  template:
    metadata:
      labels:
        app: onos-onos
    spec:
      containers:
        - image: ahdepe/onos:0.0.1
          name: onos
          resources:
            limits:
              memory: 1500Mi
          ports:
            - containerPort: 6653
              name: openflow
              protocol: TCP
            - containerPort: 6640
              name: ovsdb
              protocol: TCP
            - containerPort: 9876
              name: east-west
              protocol: TCP
            - containerPort: 8101
              name: cli
              protocol: TCP
            - containerPort: 8181
              name: ui
              protocol: TCP
          volumeMounts:
            - mountPath: /data
              name: onos-data
            - mountPath: /root/onos/apps/org.onosproject.fwd/active
              name: config
              subPath: active
            - mountPath: /root/onos/apps/org.onosproject.openflow/active
              name: config
              subPath: active
          env:
            - name: MY_POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
      volumes:
        - name: onos-data
          persistentVolumeClaim:
            claimName: onos-pvc
        - emptyDir: {}
          name: config

```

Figure 5.4: Kubernetes deployment definition for SDN controller.

ONOS [98] is selected as an open-source SDN controller. The mininet software-defined networking emulation environment has been used to simulate attackers, servers and IoT devices [87]. An Open vSwitch [99] instance is installed on a Ubuntu 18.04 operating system.

We deployed the ONOS controller into the AKS cluster as a Kubernetes deployment resource. The controller's deployment definition is given in Figure 5.4, where the following configurations have been set to run gracefully for the SDN controller; ports 6653 (OpenFlow), 6640 (OvsDB), 9876 (east-west), 8101 (CLI), and 8181 (GUI) have been opened, ReactiveForwarding and OpenFlow applications have been enabled, and the IP address of the pod has been assigned to an environment variable named MY_POD_IP in order for the mitigation module talk to the controller through its REST API [100].

To expose the SDN controller functionalities to the Internet, we have defined two Kubernetes services (Figure 5.5); the first is gui-lb-service that enables users to access the controller via a graphical user interface, and the latter is openflow-lb-service, which works for devices to connect to the controller through its south-bound API.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: gui-lb-service
spec:
  selector:
    app: onos-onos
  type: LoadBalancer
  ports:
  - name: http
    port: 8181
    nodePort: 32700
    targetPort: 8181
```

(a) Graphical interface service.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: openflow-lb-service
spec:
  ports:
    - port: 6653
      protocol: TCP
      targetPort: openflow
  selector:
    app: onos-onos
  type: LoadBalancer
```

(b) OpenFlow service.

Figure 5.5: Kubernetes services of the ONOS controller having load-balancer type IP addresses.

After the successful installation of Kubernetes services into the cluster, we can see that the services have taken public IP addresses. In Figure 5.6, the EXTERNAL-IP column shows the IP addresses of deployed services.

We have designed four evaluation scenarios as given in Table 5.19 to test the SDN

```
$ kubectl get services
NAME                  TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
gui-lb-service        LoadBalancer   10.0.40.87    51.124.60.249   8181:32700/TCP
kubernetes            ClusterIP      10.0.0.1      <none>          443/TCP
openflow-lb-service   LoadBalancer   10.0.64.136   51.124.57.166   6653:31243/TCP
```

Figure 5.6: Deployed Kubernetes services.

controller pod's accessibility/availability and connectivity of legitimate hosts in the network. In the very first scenario, we just produced normal traffic. In the second scenario, we implemented attacks from the malicious machines while the legitimate traffic producer devices also connected to the servers. In the third stage, we enabled the security mechanism and produced normal traffic at the same time. In the last phase, the security mechanism is enabled, attacks are applied, and normal traffic is produced at the same time.

We developed Python scripts with which the legitimate machines create normal traffic. In those scripts, we emulated the BoT-IoT network's legitimate traffic using the Mininet [87] simulation tool's Python library. The BoT-IoT [85] dataset, which includes realistic IoT traffic traces, has nearly 9500 normal traffic samples, and we have analyzed these records to establish our legitimate traffic to have identical packet transmission speeds, packet sizes, and protocols. Simulated IoT hosts send small packets by using TCP with an interval of 12 to 15 seconds to the server, randomly chosen at the start time. We have chosen send rates and packet sizes in proportion to the rates selected in the BoT-IoT dataset. To generate attack traffic, we employed hping3 attack utility as done in the BoT-IoT dataset.

With normal traffic producer scripts, we simulate IoT devices by sending packages to an IoT server. At the beginning of the IoT simulation script, the packet size (1 or 2 packets), byte count (24-46 bytes), and period value (12-15 seconds) are selected randomly. In each of 100 iterations, the IoT devices send the packet(s) to the server and sleep during the period value.

After the normal traffic producer script's execution ends, we measure the total time passed and the number of successfully sent packets for each IoT device. Then, by dividing the successful packet count by the total time, we calculate the throughput value for each IoT host. Finally, we compute the packet success, packet loss, total

time, and throughput values by considering the avarage for all IoT devices.

Table 5.19: Experimented scenarios.

Scenario	Attacks	Security Mechanism	Normal Traffic
1	✗	✗	✓
2	✓	✗	✓
3	✗	✓	✓
4	✓	✓	✓

IoT devices send data to the servers they are connected to by waiting for a certain period. When this time is greater than the *FLOW_TIMEOUT* (10 seconds by default), the OpenFlow switch's flow disappears, and the switch asks the controller what to do for each new packet.

Especially in attacks against the controller, the controller cannot respond to these requests, and as a result, IoT devices cannot successfully send their packets to the destination servers. To also target the resources (CPU, RAM, etc.) of the SDN controller, we decided to apply Denial-of-Service (DoS) with source IP spoofing attacks to the target servers. In this way, we aimed at decreasing the availability of both the SDN controller and the target server.

In spoofed-DoS attacks, the attacker targets a specific host by the target's IP address, but this time, it aims to flood the target with spoofed packets. The attacker uses IP spoofing and performs the attack by running the related script. The attacker sends 40000 spoofed packets, sleeps for 45 seconds, and then repeats this process with increased number of packets. The number of packets is increased to 60000, 80000, and 100000 sequentially. The whole process repeats inside an infinite loop.

It is also worth mentioning why we have not defined a separate scenario for missed attacks. If the classification module cannot correctly detect a significant number of attacks, the result will be an extra load on the controller. We see the effects of missed attacks in scenarios 2 and 4. In the absence of the security mechanism, all the attacks are missed in scenario 2, and it causes a high packet loss in this scenario. Also, when

the security mechanism is enabled in scenario 4, we would expect a higher packet loss and lower throughput in case of a significant number of missed attacks.

5.2.1 Experiments with One-switch Network Topology

In Figure 5.7, the networking topology with one OpenFlow switch is given. In this topology, we randomly select one of the connected devices as the attacker, five of them as IoT devices, one of them as IoT server, and the remaining devices idle. In the scope of normal traffic, IoT devices connect to the IoT server and send legitimate packets to the destination server.

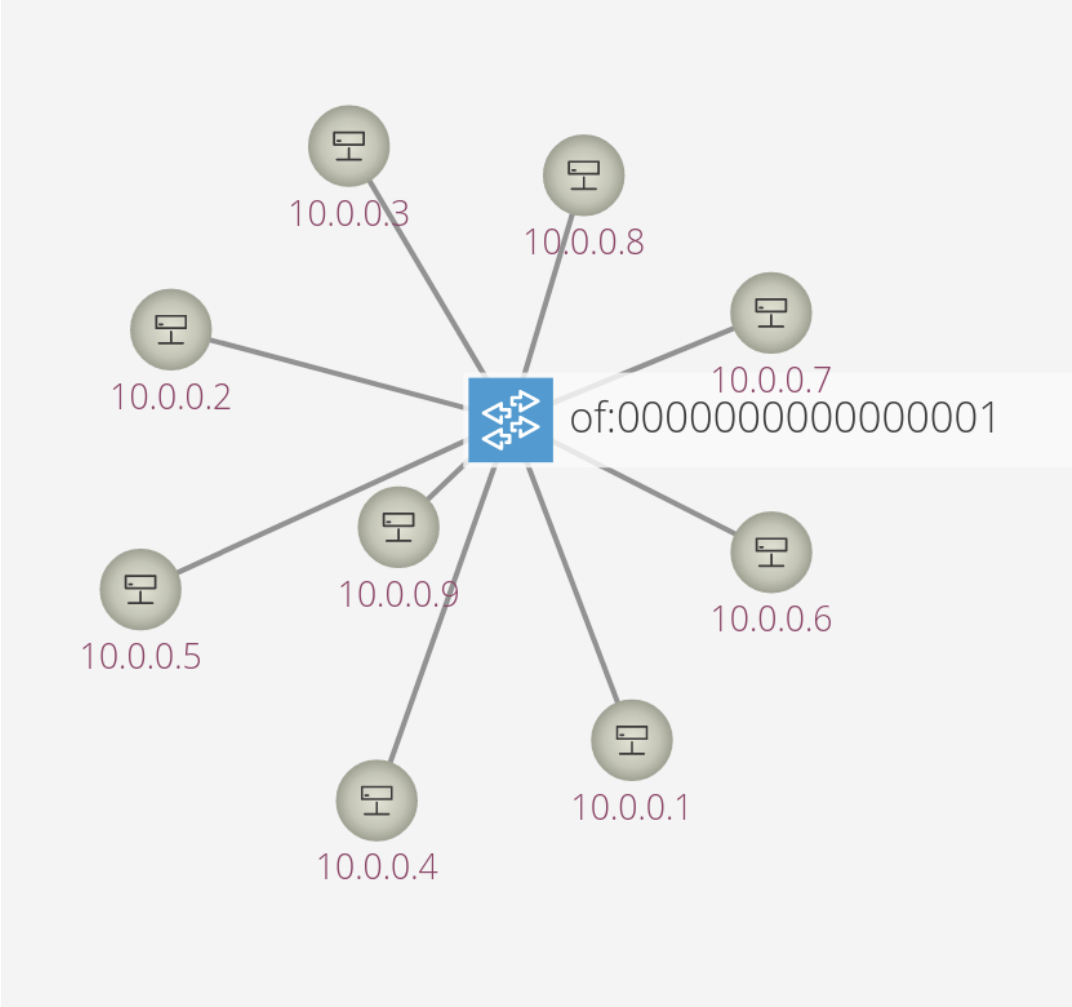


Figure 5.7: One-switch network topology.

The attacker selects its target as the IoT server. Subsequently, it starts to send source IP spoofed packets to the victim server. At the same time, our security mechanism receives flow entries and extracts required features from them. Then, it passes these features to the traffic classification module and reads the prediction in return.

When the number of detections per source switch port - destination port pairs exceed the predefined threshold value, a drop rule is added to the switch to prevent the traffic for that pair. In the fourth scenario, our security mechanism has added the drop rule only after 19 seconds (Table 5.20) than the attack starts.

Table 5.20: One-switch topology attack detection/prevention details.

Switch	Attack type	Switch Src Port	Switch Dst Port	Time to first detection (sec)	Time to block (sec)
of:0000000000000001	DoS	1	8	5	19

Table 5.21 shows the average packet success rate, average packet loss rate, total running time of the experiment and the average throughput per IoT device traffic for all scenarios in the one switch topology. In all the scenarios except the second one, there does not exist any packet loss and all the packets arrived at the destination successfully. In scenario-2, the attacker decreased the availability of the SDN controller in the absence of our security mechanism.

Also, the throughput is significantly dropped in this scenario. The results of scenario-3 show that our security mechanism did not prevent legitimate traffic, i.e., all legitimate packets arrived at that destination successfully. Another important point that stands out is that the security mechanism did not add an additional burden on the system, as demonstrated by the fact that the throughput values in scenarios 1 and 3 are very close to each other.

5.2.2 Experiments with the Two-switch Network Topology

Figure 5.8 depicts the two-switch network topology. In this topology, there exist nine hosts connected to each switch. Upon building the topology, in each switch, we

Table 5.21: One-switch topology performance measurements.

Scenario	Avg. Packet Success (%)	Avg. Packet Loss (%)	Avg. Total Time (sec.)	Throughput (packets/sec.)
Scenario-1	100	0	1451.41	0.14
Scenario-2	10.8	89.2	1746.53	0.01
Scenario-3	100	0	1391.45	0.15
Scenario-4	100	0	1427.61	0.11

randomly pick one host as the attacker, five of them as IoT devices, one of them as an IoT server, and the remaining devices idle. IoT devices generate normal traffic by sending packets to the IoT server.

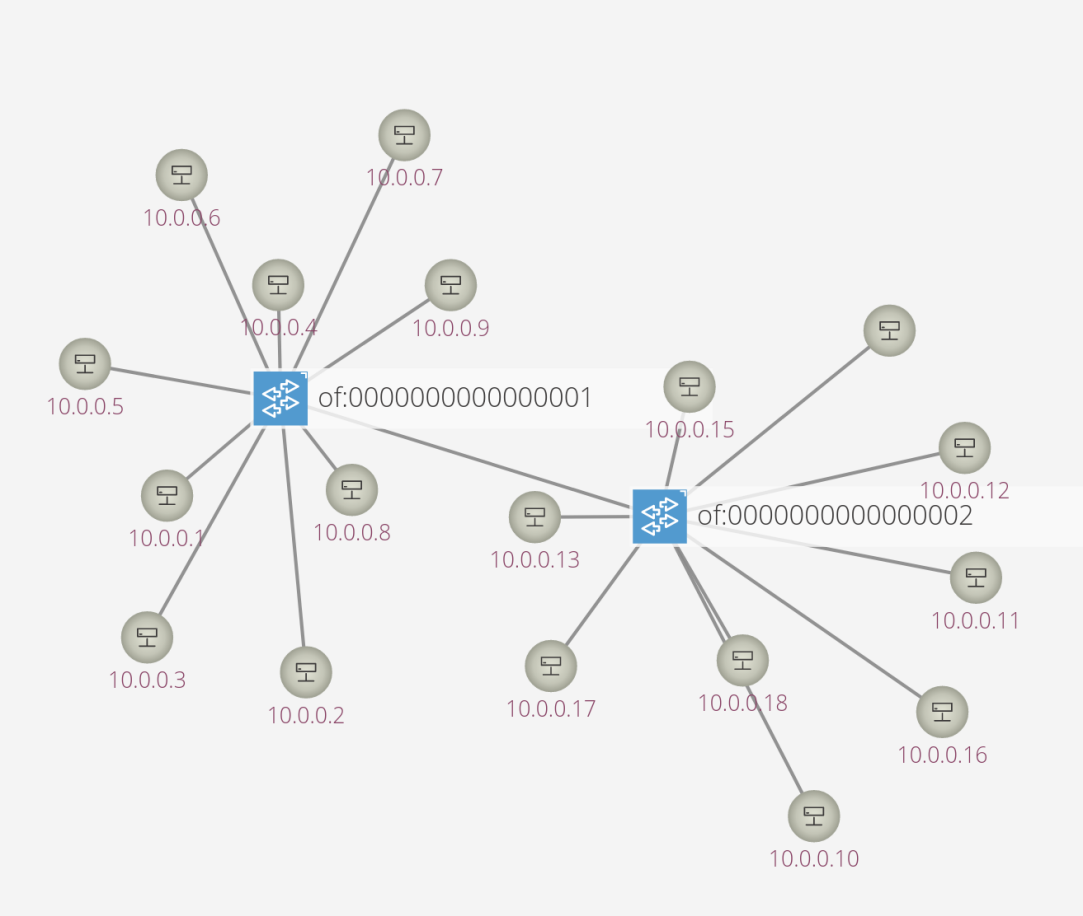


Figure 5.8: Two-switch network topology.

After the topology builds, attackers start to send source IP spoofed packets to the target servers. Each attacker aims at the server which is connected to the same switch. The feature extraction application first gets the list of switches in the network, then loops over them by reading flow entries in each switch. When any DoS attack is predicted for a switch source port - destination port pair, the number of detections for that pair is increased by one. Our security mechanism prevents attackers from sending any packets after the number of detections exceeds the threshold value for each pair.

Table 5.22: Two-switch topology attack detection/prevention details.

Switch	Attack type	Switch Src Port	Switch Dst Port	Time to first detection (sec)	Time to block (sec)
of:0000000000000001	DoS	4	2	47	79
of:0000000000000002	DoS	5	3	60	77

Table 5.22 shows the detection and prevention details for attacker machines in the two-switch topology. The proposed security mechanism prevents attackers under 80 seconds after the malicious network traffics start.

We also experimented with the effects of our security mechanism on the legitimate network traffic and whether it causes a burden on the network. Table 5.23 gives the details of each scenario defined in Table 5.19 for the two-switch topology. In the first scenario, we produced only normal traffic, and all the packets successfully arrived in the destination servers as expected. In scenario 2, attackers targeted the servers by generating spoofed source IP DoS attacks. In the absence of a security mechanism, above 94% legitimate packet loss, and thus a very low throughput (under 0.01) occurred in this scenario. In the third scenario, all packets arrived at their destination successfully, i.e. no normal traffic has been dropped by our security mechanism. In the final scenario, we enabled the security mechanism, and it successfully prevented the attacks without negatively affecting the connections of legitimate IoT machines. There is a %0.3 packet loss for the IoT devices in this scenario, which is quite low. As the throughput values of normal traffic in scenarios 1, 3, and 4 are very close to each other, we can clearly proclaim the security mechanism's efficacy.

Table 5.23: Two-switch topology performance measurements.

Scenario	Avg. Packet Success (%)	Avg. Packet Loss (%)	Avg. Total Time (sec.)	Throughput (packets/sec.)
Scenario-1	100	0	1357.21	0.12
Scenario-2	5.30	94.70	1688.83	0.005
Scenario-3	100	0	1368.27	0.13
Scenario-4	99.7	0.3	1344.44	0.13

5.2.3 Experiments with the Three-switch Network Topology

In Figure 5.9, we have tested the proposed security mechanism with three OpenFlow enabled switches along with their connected hosts. In this network topology, there are nine hosts connected to each switch. After constructing the topology, in each switch, we randomly pick one host as the attacker, five of them as IoT devices, one of them as an IoT server, and the remaining devices as idle. IoT devices generate legitimate traffic by sending packets to the servers. Attack detection works as explained for the previous experiments.

Table 5.24 demonstrates the results taken for the fourth scenario in the three-switch network topology. According to this table, our security mechanism has added the drop rules to the switch of:0000000000000001 in 20 seconds after the attack started, to switch of:0000000000000002 after 30 seconds, and to switch of:0000000000000003 after 102 seconds.

Table 5.24: Three-switch topology attack detection/prevention details.

Switch	Attack type	Switch Src Port	Switch Dst Port	Time to first detection (sec)	Time to block (sec)
of:0000000000000001	DoS	2	1	2	20
of:0000000000000002	DoS	5	3	14	30
of:0000000000000003	DoS	1	3	77	102

We also experimented with our security mechanism's effects on the legitimate net-

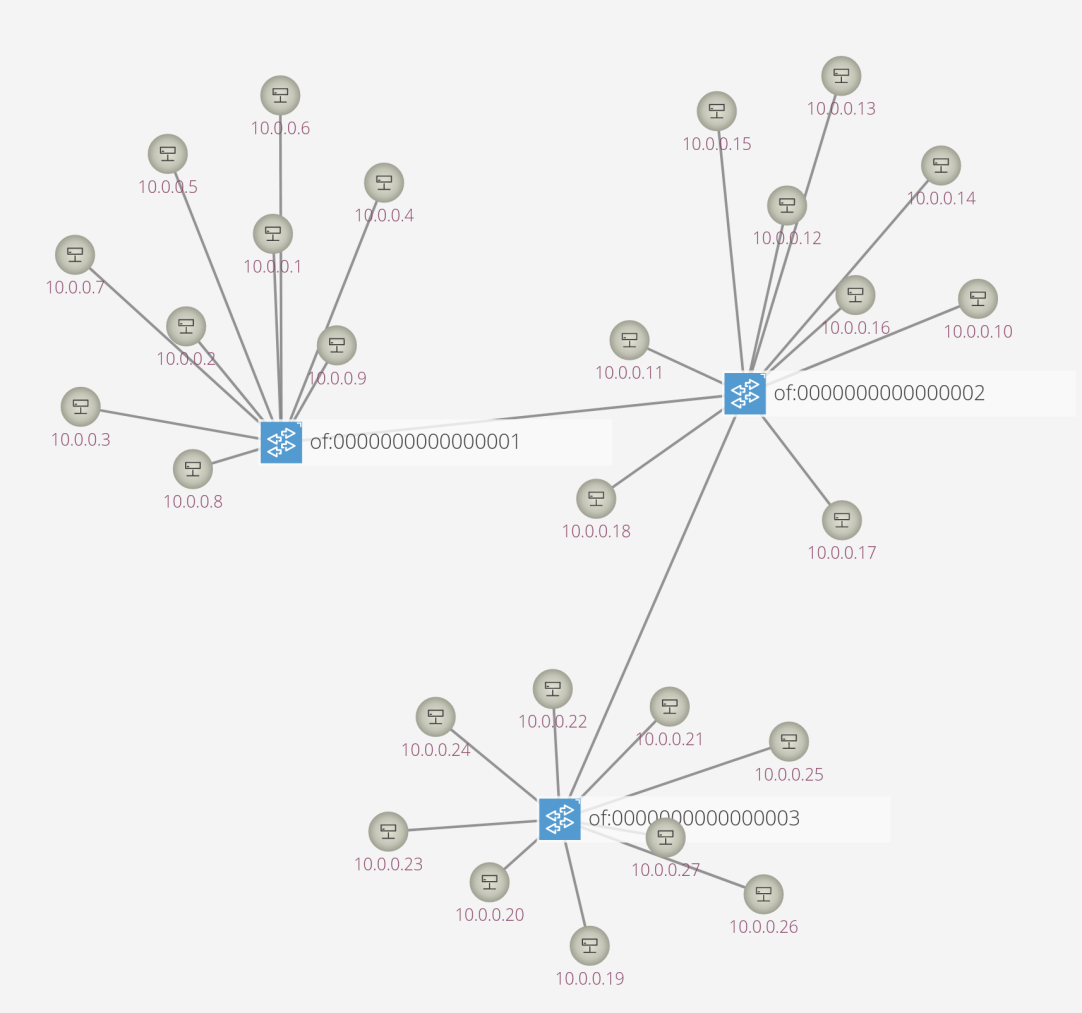


Figure 5.9: Three-switch network topology.

work traffic and whether it causes an extra load on the network. Table 5.25 gives the details of each scenario defined in Table 5.19 for the three-switch network topology. In the first scenario, we produced only normal traffic, and all the packets successfully arrived in the destination servers as expected. In scenario 2, attackers targeted the servers by generating spoofed source IP DoS attacks. In the absence of a security mechanism, above 97% legitimate packet loss, and thus a very low throughput (about 0.002) occurred in this scenario. In the third scenario, our security mechanism has dropped no normal traffic, thus all packets arrived successfully at their destination. In the final scenario, we enabled the security mechanism, and again there was no legitimate packet loss.

Table 5.25: Three-switch topology performance measurements.

Scenario	Avg. Packet Success (%)	Avg. Packet Loss (%)	Avg. Total Time (sec.)	Throughput (packets/sec.)
Scenario-1	100	0	1328.61	0.13
Scenario-2	2.27	97.73	1762.01	0.002
Scenario-3	100	0	1411.57	0.12
Scenario-4	100	0	1351.95	0.129

5.2.4 High Availability Experiments

Kubernetes Horizontal Pod Auto-scaler (HPA) can dynamically adjust the number of pods in a deployment object depending on the resource usage observed. The controller manager compares the metrics obtained from the Metrics Server API periodically and scales the deployment until the specified threshold is crossed.

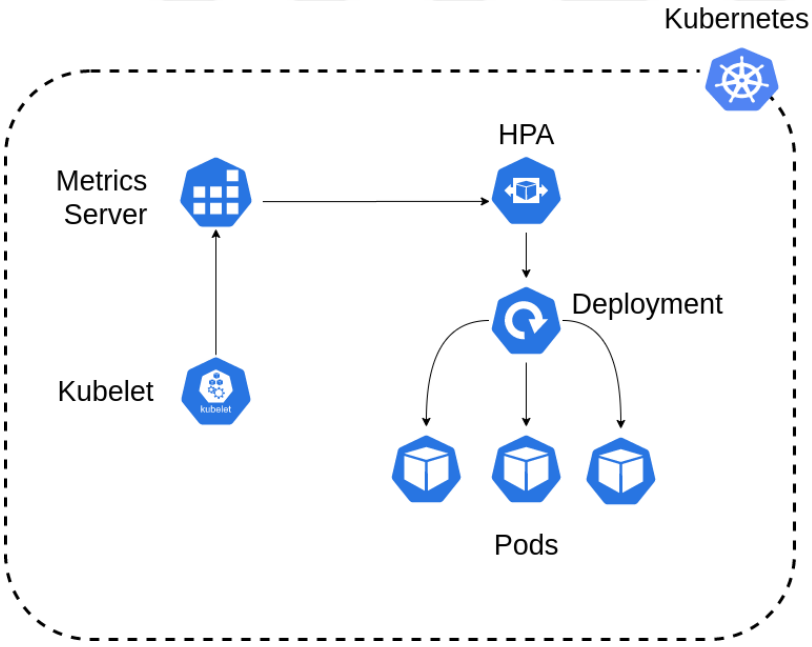


Figure 5.10: Kubernetes horizontal pod auto-scale process.

Figure 5.10 illustrates the operational order in the HPA process. The kubelet node agent periodically sends usage metrics like CPU, memory utilization, etc. to the Metrics Server. HPA reads the metrics and if an auto-scaling condition matches it

tells Deployment to produce new pod instances.

In the scope of the high availability experiments, we first measured the memory usages of the SDN controller in the network topologies experimented in the previous section. Figure 5.11 exhibits the SDN controller's memory use in terms of megabytes when only legitimate packets exist in the network.

With this experiment, we aimed to observe the controller's resource utilization in the absence of any cyber attacks. In each network architecture, the SDN controller's RAM requirement is nearly 1.15-1.25 GB at startup, and memory use does not exceed 1350 MB for legitimate traffic in a period of 1000 seconds.

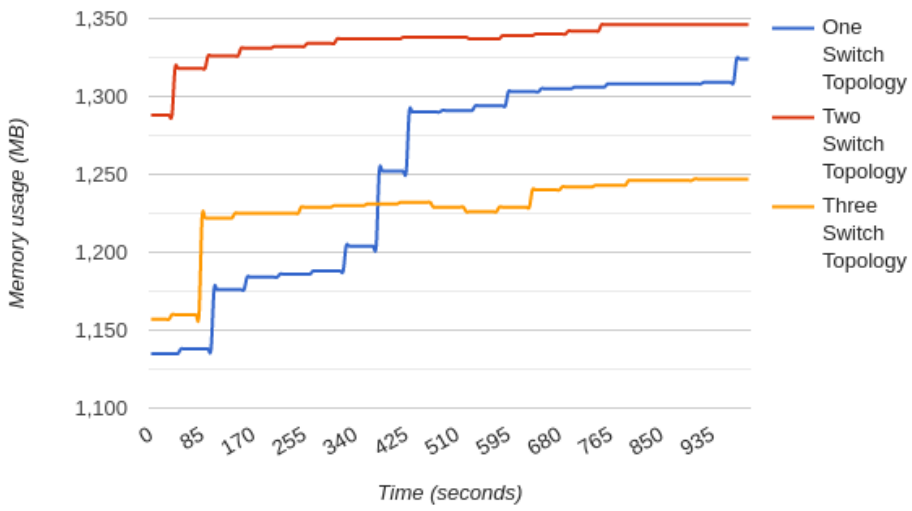


Figure 5.11: SDN controller memory utilization during normal traffic.

In the second step, we applied spoofed source IP DoS attacks on the networks along with normal traffic and measured the RAM utilization of the SDN controller. Because the attacker(s) generate random source IPs in each malicious packet, the OpenFlow switch's flow table does not have a corresponding entry for the packets, and the switch asks the SDN controller what to do with those packets for each of them. Attacker machines continuously send volumetric data to the target servers, and this situation also causes a high load on the SDN controller.

Figure 5.12 illustrates the SDN controller's memory usages in terms of megabytes. It

is seen that the controller requires just about 1.3 GB of memory at the beginning, but its RAM use sharply increases to approximately three gigabytes before 100 seconds pass after the attacks start.

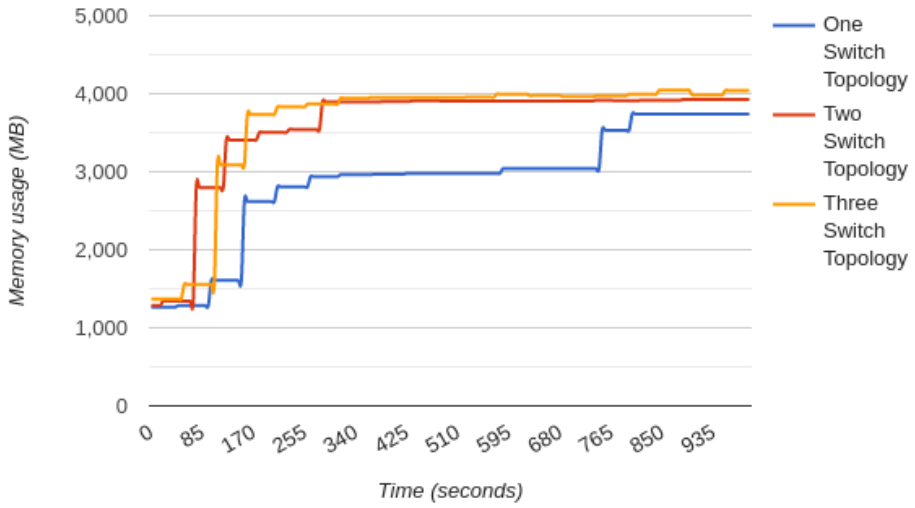


Figure 5.12: SDN controller memory utilization in spoofed-IP DoS attacks.

When the RAM use limit of the SDN controller is set to a value lower than 4 GB as we did in the previous section experiments, the controller cannot meet the switches' requests, and packet drops occur in the network. This situation causes a decrease in the system's availability, and the attackers achieve their goals as a result of the DoS attacks.

Thanks to the built-in auto-scaling capability of Kubernetes, the high-availability can be fulfilled even if under volumetric attacks. Using a Horizontal Pod Auto-scaler along with the proposed security mechanism, the architecture provides the high-availability until the mitigation module prevents attacks by adding required drop rules to the switches. After the switches start to drop volumetric cyber attack packets, they will not cause an extra load on the SDN controller, and the number of controller pod replicas will scale-down.

It is also clear in Tables 5.20, 5.22, and 5.24 that the proposed solution prevents cyber attacks 102 seconds after attacks start at the latest, in all the network topologies.

Thus, there is no need to measure the memory utilization when the mitigation module is enabled. Our module blocks the attackers before they create an extra load (in terms of RAM use) on the SDN controller.

```
1  apiVersion: autoscaling/v2beta1
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: onos-onos-ram
5  spec:
6    scaleTargetRef:
7      apiVersion: apps/v1
8      kind: Deployment
9      name: onos-onos
10   minReplicas: 1
11   maxReplicas: 5
12   metrics:
13   - type: Resource
14     resource:
15       name: memory
16       targetAverageUtilization: 66
```

Figure 5.13: Kubernetes Horizontal Pod Auto-scaler definition.

Figure 5.13 shows the Horizontal Pod Auto-scaler definition we deployed to the Azure Kubernetes Service (AKS) to provide high-availability in an unexpected load. According to the HPA definition, the number of SDN controller pods automatically increases once the controller's memory utilization passes over 66% of the defined limit, which is 1.5 GB in our experiments.

This raise continues until there exist five replicas in the condition of memory use exceeds the specified level. If the SDN controller's RAM utilization becomes lower than the 66% threshold, HPA starts to down-scale the replica counts. The down-scale process continues until there is only one replica so long as the average memory utilization stays under the defined level.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, we proposed an intelligent security architecture for SDN-assisted IoT networks. The proposed approach enables SDN-assisted IoT networks to detect and mitigate intrusions through intelligent processing of the network traffic data gathered from the data plane switches with machine learning models trained with previously observed examples of various attack classes. The proposed approach utilizes an ensemble of prototypical networks and SVM for constructing accurate classification models for intrusion detection from a small number of examples of actual attacks. We have performed experiments to evaluate the attack detection rate of the proposed intrusion detection model with a recently released realistic IoT intrusion dataset, Bot-IoT, and an SDN-based dataset containing a subset of the attack classes in Bot-IoT, which we generated in close resemblance to the architecture of Bot-IoT, as well as an intrusion detection dataset prepared for legacy networks (UNSW-NB15). We show that although classical machine learning models achieve high detection rates in the presence of sufficiently many training samples, their performance degrades significantly when fewer training samples are used. The proposed model is able to achieve superior performance in such cases. For example, the proposed Proto-S model achieves a good detection rate for our SDN dataset when trained with only 100 samples from each category. Because of the processing capacity-constrained existence of many IoT appliances, the proposed architecture would be an important help for the security of IoT networks. Also, the aggregation of flow rules from switches and extraction of features for each flow should not negatively affect the network performance to satisfy the real-time demands of SDN-assisted IoT. To experiment whether the proposed architecture results in a bottleneck in the network, we also experimented with the mitigation module's performance in a Kubernetes cluster in the

public cloud. We showed the success of the proposed solution with the attack detection tests and the attack prevention scenarios by constructing different networking topologies. Besides the time measurements in preventing cyber-attacks, we observed the effects of the proposed security mechanism on normal traffic. We proved that the proposed solution does not cause an additional burden on the SDN controller, which fits the low delay requirements of SDN-assisted ubiquitous IoT.

Our future work will include an extension of the intelligence module to make it capable of recognizing classes of attacks for which no training samples exist through zero-shot learning. We also plan to extend the generated SDN intrusion detection dataset with more samples, larger network structures, and different classes of attacks for the benefit of IoT network security researchers.

REFERENCES

[1] C. V. N. Index, "Global mobile data traffic forecast update, 2015–2020," *Cisco white paper*, p. 9, 2016.

[2] S. R. Group *et al.*, "Cisco's dominant share of switching & routers holds steady," *February*, vol. 24, p. 2016, 2016.

[3] A. H. Shamsan and A. R. Faridi, "Sdn-assisted iot architecture: A review," in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pp. 1–7, IEEE, 2018.

[4] T. Ninikrishna, S. Sarkar, R. Tengshe, M. K. Jha, L. Sharma, V. Daliya, and S. K. Routray, "Software defined iot: Issues and challenges," in *2017 International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 723–726, IEEE, 2017.

[5] W. An and M. Liang, "A new intrusion detection method based on svm with minimum within-class scatter," *Security and Communication Networks*, vol. 6, pp. 1064–1074, 2013.

[6] Y. Xiao, C. Xing, T. Zhang, and Z. Zhao, "An intrusion detection model based on feature reduction and convolutional neural networks," *IEEE Access*, vol. 7, pp. 42210–42219, 2019.

[7] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[8] S. Xu, Y. Qian, and R. Q. Hu, "Data-driven network intelligence for anomaly detection," *IEEE Network*, vol. 33, no. 3, pp. 88–95, 2019.

[9] M. Al-Hawawreh, N. Moustafa, and E. Sitnikova, "Identification of malicious activities in industrial internet of things based on deep learning models," *J. Inf. Secur. Appl.*, vol. 41, pp. 1–11, 2018.

[10] E. Ciklabakkal, A. Donmez, M. Erdemir, E. Suren, M. K. Yilmaz, and P. Angin, "ARTEMIS: an intrusion detection system for MQTT attacks in internet of things," 38th Symposium on Reliable Distributed Systems, SRDS 2019, Lyon, France, October 1-4, 2019, pp. 369–371, IEEE, 2019.

[11] J. Arevalo Herrera and J. E. Camargo, "A survey on machine learning applications for software defined network security," Applied Cryptography and Network Security Workshops, pp. 70–93, 2019.

[12] A. Mehmood, M. Mukherjee, S. H. Ahmed, H. Song, and K. M. Malik, "Nbc-maids: Naïve bayesian classification technique in multi-agent system-enriched ids for securing iot against ddos attacks," *J. Supercomput.*, vol. 74, no. 10, p. 5156–5170, 2018.

[13] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–6, 2009.

[14] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, "Deep recurrent neural network for iot intrusion detection system," *Simulation Modelling Practice and Theory*, vol. 101, p. 102031, 2020.

[15] Y. Otoum, D. Liu, and A. Nayak, "Dl-ids: A deep learning–based intrusion detection framework for securing iot," *Transactions on Emerging Telecommunications Technologies*, p. e3803, 2019.

[16] N. Ravi and S. M. Shalinie, "Learning-driven detection and mitigation of ddos attack in iot via sdn-cloud architecture," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3559–3570, 2020.

[17] M. E. Ahmed and H. Kim, "Ddos attack mitigation in internet of things using software defined networking," in *Proceedings of 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 271–276, 2017.

[18] D. Yin, L. Zhang, and K. Yang, "A ddos attack detection and mitigation

with software-defined internet of things framework," *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[19] J. Galeano-Brajones, J. Carmona-Murillo, J. F. Valenzuela-Valdés, and F. Luna-Valero, "Detection and mitigation of dos and ddos attacks in iot-based stateful sdn: An experimental approach," *Sensors*, vol. 20, p. 816, Feb 2020.

[20] P. K. Sharma, S. Singh, and J. H. Park, "Opcloudsec: Open cloud software defined wireless network security for the internet of things," *Computer Communications*, vol. 122, pp. 1 – 8, 2018.

[21] P. Bull, R. Austin, E. Popov, M. Sharma, and R. Watson, "Flow based security for iot devices using an sdn gateway," Proceedings of 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 157–163, 2016.

[22] J. Li, Z. Zhao, R. Li, and H. Zhang, "Ai-based two-stage intrusion detection for software defined iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2093–2102, 2019.

[23] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure iot architecture," *Internet of Things*, vol. 3-4, pp. 82–89, 2018.

[24] P. Amangele, M. J. Reed, M. Al-Naday, N. Thomos, and M. Nowak, "Hierarchical machine learning for iot anomaly detection in sdn," Proceedings of 2019 International Conference on Information Technologies (InfoTech), pp. 1–4, 2019.

[25] A. Muthanna, A. A. Ateya, A. Khakimov, I. Gudkova, A. Abuarqoub, K. Samouylov, and A. Koucheryavy, "Secure and reliable iot networks using fog computing with software-defined networking and blockchain," *Journal of Sensor and Actuator Networks*, vol. 8, p. 15, Feb 2019.

[26] G. Grigoryan, Y. Liu, L. Njilla, C. Kamhoua, and K. Kwiat, "Enabling cooperative iot security via software defined networks (sdn)," 2018 IEEE International Conference on Communications (ICC), pp. 1–6, 2018.

[27] A. Al Hayajneh, M. Z. A. Bhuiyan, and I. McAndrew, "Improving internet of things (iot) security with software-defined networking (sdn)," *Computers*, vol. 9, p. 8, Feb 2020.

[28] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357 – 374, 2012.

[29] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP*, pp. 108–116, 2018.

[30] A. Demirpolat, D. Ergenç, E. Ozturk, Y. Ayar, and E. Onur, "Software-defined network security," Enabling Technologies and Architectures for Next-Generation Networking Capabilities, pp. 232–253, IGI Global, 2019.

[31] A. Demirpolat, A. K. Sarica, and P. Angin, "Protédge: A few-shot ensemble learning approach to software-defined networking-assisted edge security," *Transactions on Emerging Telecommunications Technologies*, p. e4138, 2020.

[32] M. Betts, S. Fratini, N. Davis, D. Hoods, R. Dolin, M. Joshi, and Z. Dacheng, "Sdn architecture, issue 1, open networking foundation," tech. rep., ONF TR-502, 2014.

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

[34] T. Alharbi, S. Layeghy, and M. Portmann, "Experimental evaluation of the impact of dos attacks in SDN," Proceedings of the 27th International Telecommunication Networks and Applications Conference, ITNAC 2017, Melbourne, Australia, November 22-24, pp. 1–6., 2017.

[35] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, p. 14–76, 2015.

[36] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, p. 114–119, 2013.

[37] S. Deng, X. Gao, Z. Lu, and X. Gao, "Packet injection attack and its defense in software-defined networks," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 3, pp. 695–705, 2018.

[38] V. Lonker, "Thinking beyond the box – how software defined networks are changing the future of connectivity," May 2018.

[39] G. S. Aujla, R. Chaudhary, N. Kumar, J. J. P. C. Rodrigues, and A. Vinel, "Data offloading in 5g-enabled software-defined vehicular networks: A stackelberg-game-based approach," *IEEE Communications Magazine*, vol. 55, no. 8, p. 100–108, 2017.

[40] R. Chaudhary, A. Jindal, G. S. Aujla, S. Aggarwal, N. Kumar, and K.-K. R. Choo, "Best: Blockchain-based secure energy trading in sdn-enabled intelligent transportation system," *Computers and Security*, vol. 85, p. 288–299, 2019.

[41] G. S. Aujla, A. Singh, and N. Kumar, "Adaptflow: Adaptive flow forwarding scheme for software-defined industrial networks," *IEEE Internet of Things Journal*, vol. 7, no. 7, p. 5843–5851, 2020.

[42] M. D. Firoozjaei, J. P. Jeong, H. Ko, and H. Kim, "Security challenges with network functions virtualization," *Future Generation Computer Systems*, vol. 67, pp. 315–324, 2017.

[43] Stanford University, "Convolutional neural networks tutorial." http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.

[44] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," Proceedings of the 30th International Conference on Neural Information Processing Systems, p. 3637–3645, 2016.

[45] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017.

[46] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," Proceedings of Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pp. 4077–4087, 2017.

[47] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proceedings og 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1222–1228, IEEE, 2017.

[48] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[49] L. Lin, R. Zuo, S. Yang, and Z. Zhang, "SVM ensemble for anomaly detection based on rotation forest," in *Intelligent Control and Information Processing (ICICIP), 2012 Third International Conference on*, pp. 150–153, July 2012.

[50] S. Mukherjee and N. Sharma, "Intrusion detection using naive bayes classifier with feature reduction," *Procedia Technology*, vol. 4, pp. 119–128, 2012.

[51] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[52] W. Hu, W. Hu, and S. Maybank, "Adaboost-based algorithm for network intrusion detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 2, pp. 577–583, 2008.

[53] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in neural information processing systems*, pp. 231–238, 1995.

[54] M. Maternia, S. E. El Ayoubi, M. Fallgren, P. Spapis, Y. Qi, D. Martín-Sacristán, Ó. Carrasco, M. Fresia, M. Payaró, M. Schubert, *et al.*, "5g ppp use cases and performance evaluation models," *5G PPP*, vol. 1, 2016.

[55] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[56] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[57] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[58] M. McBride, M. Cohn, S. Deshpande, M. Kaushik, M. Mathews, and S. Nathan, "Sdn security considerations in the data center," *Open Networking Foundation-ONF SOLUTION BRIEF*, pp. 15–16, 2013.

[59] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[60] W. Stallings, "Software-defined networks and openflow," *The internet protocol Journal*, vol. 16, no. 1, pp. 2–14, 2013.

[61] H. S. Oluwatosin, "Client-server model," *IOSR J Comput Eng (IOSR-JCE)*, vol. 16, no. 1, p. 67, 2014.

[62] W. Stallings, *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2015.

[63] A. Ericsson, "Ericsson mobility report. ericsson," tech. rep., Sweden, Tech. Rep. EAB-17, 1. 1, 2017.

[64] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[65] K. Slavov, D. Migault, and M. Pourzandi, "Identifying and addressing the vulnerabilities and security issues of sdn," *Ericsson Technology Review*, vol. 92, no. 7, 2015.

[66] A. Hakiri and P. Berthou, "Leveraging sdn for the 5g networks: trends, prospects and challenges," *arXiv preprint arXiv:1506.02876*, 2015.

[67] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *2016 25th international conference on computer communication and networks (ICCCN)*, pp. 1–9, IEEE, 2016.

[68] 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*, pp. 1–6, 2010.

[69] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, M. Tyson, *et al.*, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*, Ndss, 2013.

[70] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow world congress*, vol. 48, pp. 1–16, sn, 2012.

[71] W. Haerick and M. Gupta, "White paper: 5g and the factories of the future," *5G-PPP, Tech. Rep*, 2015.

[72] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: an authoritative review of network programmability technologies*. " O'Reilly Media, Inc.", 2013.

[73] A. J. D. M. G. Chheda and V. Nethi, "Expedition: An open source network monitoring tool for software defined networks," 2015.

[74] D. F. Macedo, D. Guedes, L. F. Vieira, M. A. Vieira, and M. Nogueira, "Programmable networks—from software-defined radio to software-defined net-

working," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 1102–1125, 2015.

[75] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," tech. rep., Technical report, 2000.

[76] 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*, pp. 55–60, 2013.

[77] P. Amudha and H. A. Rauf, "Performance analysis of data mining approaches in intrusion detection," in *2011 International Conference on Process Automation, Control and Computing*, pp. 1–6, IEEE, 2011.

[78] D. Jankowski and M. Amanowicz, "Intrusion detection in software defined networks with self-organized maps," *Journal of Telecommunications and Information Technology*, 2015.

[79] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," Proceedings of IEEE Local Computer Network Conference, pp. 408–415, 2010.

[80] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao, and F. Yu, "Detecting and mitigating target link-flooding attacks using sdn," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 944–956, 2019.

[81] A. Wang, Z. Zha, Y. Guo, and S. Chen, "Software-defined networking enhanced edge computing: A network-centric survey," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1500–1519, 2019.

[82] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, pp. 70–93, 2016.

[83] "Cloud Native Computing Foundation," 2020. `https://www.cncf.io/`, Accessed on July 27, 2020.

[84] S. Hettich and S. Bay, "The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california," *Department of Information and Computer Science*, vol. 152, 1999.

[85] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *CoRR*, vol. abs/1811.00701, 2018.

[86] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network i intrusion detection systems (unsw-nb15 network data set)," Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6., IEEE, 2015.

[87] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, IEEE, 2014.

[88] R. M. Knegtel, I. D. Kuntz, and C. Oshiro, "Molecular docking to ensembles of protein structures," *Journal of molecular biology*, vol. 266, no. 2, pp. 424–440, 1997.

[89] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of svm and ann for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617–2634, 2005.

[90] A. Mehmood, M. Mukherjee, S. H. Ahmed, H. Song, and K. M. Malik, "Nbcmaids: Naïve bayesian classification technique in multi-agent system-enriched ids for securing iot against ddos attacks," *The Journal of Supercomputing*, vol. 74, no. 10, pp. 5156–5170, 2018.

[91] F. Farahnakian and J. Heikkonen, "A deep auto-encoder based approach for intrusion detection system," *International Conference on Advanced Communication Technology, ICACT*, vol. 2018-February, pp. 178–183, 2018.

[92] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

[93] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[94] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn," vol. 9 of *ICML Workshop on AutoML*, Citeseer, 2014.

[95] S. Zubair, F. Yan, and W. Wang, "Dictionary learning based sparse coefficients for audio classification with max and average pooling," *Digital Signal Processing*, vol. 23, no. 3, pp. 960–970, 2013.

[96] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure*. " O'Reilly Media, Inc.", 2017.

[97] H. Chawla and H. Kathuria, "Azure kubernetes service," in *Building Microservices Applications on Microsoft Azure*, pp. 151–177, Springer, 2019.

[98] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.

[99] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 117–130, 2015.

[100] W. Zhou, L. Li, M. Luo, and W. Chou, "Rest api design patterns for sdn northbound api," in *2014 28th international conference on advanced information networking and applications workshops*, pp. 358–365, IEEE, 2014.

CURRICULUM VITAE

Personal Information

SURNAME, NAME:	Demirpolat, Ahmed
PLACE AND DATE OF BIRTH:	
EMAIL:	

Education

JAN 2021	Doctor of Philosophy in COMPUTER ENGINEERING, **Middle East Technical University**, Ankara, Turkey
JULY 2014	Master of Science in COMPUTER ENGINEERING, **Ankara University**, Ankara, Turkey
JULY 2006	Bachelor of Science in COMPUTER ENGINEERING, **Ege University**, Izmir, Turkey

Research Interests

Computer networks, cloud systems, 5G, Internet of Things, cyber security, software-defined networking and application of machine learning techniques in the field of network security.

Publications

- Demirpolat, A., Ergenç, D., Ozturk, E., Ayar, Y., & Onur, E. (2019). Software-defined network security. In Enabling Technologies and Architectures for Next-Generation Networking Capabilities (pp. 232-253). IGI Global.

- Demirpolat, A., Sarica, A. K., Angin, P. (2020). ProtÉdge: A few-shot ensemble learning approach to software-defined networking-assisted edge security. Transactions on Emerging Telecommunications Technologies, e4138.