

AMERICAN UNIVERSITY OF BEIRUT

MACHINE LEARNING FOR NETWORK
RESILIENCE

by
ALI HUSSEIN

A Dissertation
submitted in partial fulfillment of the requirements for
the degree of **Doctor of Philosophy** of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
January 2020

AMERICAN UNIVERSITY OF BEIRUT

MACHINE LEARNING FOR NETWORK RESILIENCE


by
ALI HUSSEIN

Approved by:

_____ 

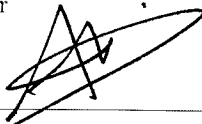
Dr. Ali Chehab, Professor Chairman

Electrical and Computer Engineering, AUB

_____ 

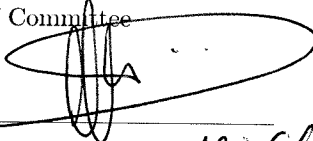
Dr. Ayman Kayssi, Professor Advisor

Electrical and Computer Engineering, AUB

_____ 


Dr. Imad Elhajj, Professor Member of Committee

Electrical and Computer Engineering, AUB

_____ 


Dr. Muhammad Imran, Professor Member of Committee

Electrical and Communication Engineering, University of Glasgow, UK

_____ 
c/o Prof. Imran

Dr. Kassem Fawaz, Professor Member of Committee

Electrical and Computer Engineering, University of Wisconsin, USA

_____ 
c/o Prof. Fawaz

Date of dissertation defense: January 24, 2020

AMERICAN UNIVERSITY OF BEIRUT

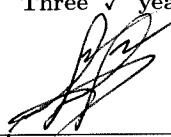
DISSERTATION RELEASE FORM

Student Name: Hussein _____ Ali _____ Imad _____
Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my dissertation.**
Two ___ years from the date of submission of my dissertation.
Three years from the date of submission of my dissertation.



Signature

30-01-2020

Date

Acknowledgements

First, I want to convey my honor to have worked under the supervision of my advisor Prof. Ali Chehab and my co-advisors Prof. Ayman Kayssi and Prof. Imad Elhajj throughout these years. Thus, I would like to express my sincere gratitude for their continuous support and motivation, their patience, and offering their tremendous knowledge. Their guidance have helped during my studies and will forever be a key motivation for me to move forward. Having them as my mentors is the key factor for being the person I am today.

I would also like to thank the rest of my committee: Prof. Muhammad Imran and Prof. Kassem Fawaz, for their insightful comments and experienced knowledge which led me to inspect my research topic from many perspectives.

To my friends, thank you for supporting me along the way, listening and offering me advice, which had a good impact on my work. Having faith-full and honest friends is a great motivation one can have in any part of our lives.

Finally, I would like to thank my family especially my parents and my wife for supporting me in any way they could throughout these long years of study and research to have reached this point. A special appreciation for my wife for

always believing in me and inspiring me to always be better.

An Abstract of the Dissertation

of

Ali Imad Hussein for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: Machine Learning for Network Resilience

Resilience is taking networks a step further beyond security. Security is one of the main concerns facing the improvement of new networking and communications systems. Another important challenge is verifying whether or not a system is working exactly as specified, hence ensuring its consistency. We argue that a resilient network is both a secure and consistent one. It is from this point that we start our thesis research. On the other hand, advances in Artificial Intelligence (AI) technology have opened up new markets and opportunities for progress in critical areas such as network resiliency, health, education, energy, economic inclusion, social welfare, and the environment. AI is expected to play an increasing role in defensive and offensive measures to provide a rapid response to react to the landscape of evolving threats. Software Defined Networking (SDN), being centralized by nature, provides a global view of the network. It is the flexibility and robustness offered by programmable networking that lead us to consider the integration of these two concepts, SDN and AI. Inspired by the fascinating tactics of the human immunity system, we aim to design a general hybrid Artificial Intelligence Resiliency System (ARS) that strikes a good balance between centralized and distributed security solutions that may be applicable to different network environments. Another objective is to investigate and leverage the state-of-the-art AI techniques to enhance network performance in general and resiliency in particular.

Being able to describe a specific network as consistent is a large step towards resiliency. Next to the importance of security lies the necessity of consistency

verification. Attackers are currently focusing on targeting small and crucial goals such as network configurations or flow tables. These types of attacks would defy the whole purpose of a security system when built on top of an inconsistent network. Another important goal of our work is to propose a new AI-based consistency verification system, which will be part of the overall ARS solution.

Throughout this work, we discuss a new architecture that integrates both, a double layer security system alongside a consistency establishment technique, while preserving data privacy. We show results related, on one hand, to the architecture tests including the accuracy of multiple AI techniques for both security layers, and on the other hand, to the attack mitigation and consistency tests. Finally, we present a new distributed AI-based security enforcement technique as part of the ARS system.

Our results showed our centralized security ensemble, which is formed from the random forest (RF) technique for anomaly detection and the deep neural network (DNN) for attack identification, providing a 99% and 98% accuracy respectively. Our distributed neural network overlay for anomaly detection provided a 94.8% accuracy, while our consistency verification convolutional neural network (CNN) system provided a 96% accuracy. Also, both our systems passed the unknown attack detection tests when integrated with our AI optimization module. Our system managed to ensure network security and consistency maintaining data privacy and decent processing and traffic overhead, as discussed in the results.

Contents

Acknowledgements	v
Abstract	vii
1 Introduction	1
1.1 Motivation	3
1.2 Introduction to Software Defined Networking	4
1.2.1 SDN Overview	4
1.2.2 SDN Architecture and Components	8
1.2.3 Communication Protocols	9
1.3 History of programmable networking	12
1.3.1 Programmable Networking	12
1.3.2 OpenFlow History	14
1.4 Introduction to Artificial Intelligence	15
1.4.1 Machine Learning	16
1.4.2 Artificial Neural Network	17
1.4.3 Multilayer Perceptron	18
1.4.4 Radial Basis function	19
1.4.5 Self-organizing Map	20
1.4.6 Adaptive Resonance Theorem	20

1.4.7	Genetic Algorithm	20
1.4.8	Fuzzy Logic	21
1.4.9	K-nearest Neighbors	22
1.4.10	K-means	22
1.4.11	K-medoids	22
1.4.12	Naive Bayes	23
1.4.13	Decision tree	24
1.4.14	Support Vector Machine	24
1.4.15	Random forest	24
1.4.16	Deep Learning	25
1.4.17	Overall Summary	26
2	Literature Review	28
2.1	SDN Security Overview	28
2.2	SDN Security and vulnerabilities	29
2.2.1	SDN Communication Security	30
2.2.2	SDN General Vulnerabilities	32
2.3	Attacks on SDN	35
2.3.1	Physical attacks	36
2.3.2	Plane Based Attacks	37
2.3.3	OpenFlow Security Problems	45
2.3.4	SDN Specific Security Challenges	49
2.4	SDN security Solutions	50
2.4.1	Application Layer Solutions	51
2.4.2	Control Layer Solutions	63
2.4.3	Hybrid Controller Architecture Solutions	64

2.4.4	Data Layer Solutions	74
2.4.5	OpenFlow Proposed Solutions	81
2.4.6	General Solutions	82
2.5	SDN Network security enhancements	88
2.5.1	DISCO (Distributed Control Plane)	89
2.5.2	Parelo-based Optimal Controller Placement (POCO)	89
2.5.3	Network Security Architectures	90
2.5.4	Control-Data Plane Intelligence Tradeoff	91
2.6	Security Discussion and Future Directions	97
2.6.1	SDN Security Summary	100
2.7	Artificial Intelligence for network security	102
2.7.1	AI for Network Security	104
2.8	Tables	112
2.8.1	Intelligent Network Applications	113
2.9	AI for Network Quality	115
2.9.1	Time Line Overview	120
2.9.2	Machine Learning Hybrid techniques	120
2.9.3	Intrusion Detection Selected Summary	122
2.10	Consistency and verification for network resilience and quality	123
3	Previous Work	135
3.1	SDN Security Plane	135
3.2	SDN Verification plane	136
3.3	SDN and MPTCP	137
3.4	SDN and VANET Security	137
3.5	SDN and QUIC	138

3.6	SDN Security review	139
3.7	SDN and 5G	139
3.8	Diameter Security	141
3.9	Machine Learning for network Resilience	142
4	ARS Introduction	143
4.1	The Road Towards ARS	143
4.2	SDN Security Plane	144
4.2.1	DDoS Prevention Technique	148
4.2.2	Testing and Simulation	151
4.3	SDN Verification Plane	153
4.3.1	SDN Verification Tool	156
4.3.2	Testing and Simulation	159
4.4	ARS Architecture	163
5	ARS Security System	165
5.1	System Overview	166
5.2	AI for Network Security	167
5.2.1	Applying Artificial Neural Networks for anomaly detection	167
5.2.2	Designing a two-step process for anomaly detection and attack identification	168
5.2.3	Designing an Ensemble of AI techniques	169
5.3	ARS Architecture OverView	169
5.3.1	The $D \sim C^2$ architecture can be described as follows: . . .	170
5.3.2	The $D^2 \sim C$ architecture can be described as follows: . . .	170
5.4	ARS System discussion	173

6	ARS Consistency System	176
6.1	General ARS Architecture Overview	177
6.2	Research Investigation Phases	178
6.3	Proposed Consistency System	179
6.4	Consistency system analysis	181
7	ARS AI Optimization	184
7.1	AI Optimization	184
7.1.1	Binary classifiers	186
7.1.2	Probabilistic classifiers	187
7.1.3	Class Calibration (CC)	187
7.1.4	Probabilistic Calibration (PC)	187
7.2	General Discussion	188
7.2.1	Dropout	188
7.2.2	Overfitting	194
7.3	ARS Optimization using stochastic regularization techniques (SRT)	199
7.3.1	Classification	200
7.3.2	Uncertainty estimate discussion	202
8	System Simulation and results	204
8.1	System Security Simulation and Results	204
8.1.1	System Datasets	205
8.1.2	System Simulation	206
8.1.3	System Training	209
8.1.4	Tests and Results	211
8.2	System Consistency Simulation and Results	218
8.2.1	Data collection	218

8.2.2	Data Preprocessing	219
8.2.3	Classification Results	222
8.2.4	System Consistency Tests	223
9	Conclusion	228
10	Future Work	230
A	Abbreviations	231

List of Figures

1.1	Conceptual Architecture of Software-Defined Networking	7
1.2	SDN Architecture Planes	9
1.3	Programmable Networking Timeline	12
1.4	Three layer Neural Network	18
1.5	Radial Basis Function Architecture	19
1.6	Machine Learning Overall Summary	27
2.1	Research Projects Regarding SDN Security Solutions At Different Layers/Interfaces	51
2.2	SDN Based DDoS Detection and Mitigation Solutions	65
2.3	Intrusion Detection Techniques Overall Summary	103
2.4	AI and ML Publications Since 2010	121
2.5	IDS Publications for Different AI Techniques	121
4.1	SDN Security Plane Architecture	145
4.2	Security Module – Controller REST Connection	147
4.3	Overall Procedure Scheme	149
4.4	Attack Detection FlowChart	150
4.5	SDN Verification Architecture	155
4.6	Verification Module – Controller REST Connection	156

4.7	Overall Verification Scheme	158
4.8	SDN Topology State Machine in UPPAAL	160
4.9	Switch States in UPPAAL	161
4.10	SDN Topology View of the Verification Tool	163
5.1	Proposed $D^2 \sim C$ Architecture	171
5.2	Proposed $D^2 \sim C$ Architecture	171
5.3	Simple Neural Network Architecture	172
5.4	Distributed NN Overlay Architecture	172
6.1	System architecture Distribution	177
6.2	System Consistency Establishment Architecture	178
6.3	Data extraction and processing phases	182
7.1	Applying dropout to a simple Neural Network (Hidden Layers)	189
7.2	dropout interval distribution at $d=0.5$	190
7.3	dropout interval plot for different dropout values	191
7.4	dropout interval distribution at $d=0.375$	192
7.5	Recommended method of dividing the data set	196
7.6	Change of accuracy values in subsequent epochs during neural network learning	198
7.7	Graphical analysis of the top 3 classifications of the CNN model for each input	202
8.1	Attack/Detection Test Scenario	213
8.2	RandomForest Confusion Matrix for Unbalanced Data	214
8.3	RandomForest Confusion Matrix for Balanced Data	214
8.4	Attack Mitigation Process	216

8.5	Consistency Results Visual Data Representation	221
8.6	Consistency Verification Test Scenario	225
8.7	S1-S5 Graphical Consistency Comparison	225

List of Tables

1.1	SDN Security Advantages Over Traditional Networks	6
1.2	Main SDN Interfaces	11
2.1	SDN Security Issues At the Communication Level	33
2.2	Main SDN Security Research Timeline Summary	37
2.3	Main SDN Security Research Timeline Summary (continued) . . .	38
2.4	Security Threats in SDN Networks (1)	46
2.5	Security Threats in SDN Networks (2)	47
2.6	Classification of SDN Security Attacks	50
2.7	Classification of The-State-of-The-Art Security Solutions on Dif- ferent SDN Layers/Interfaces (1)	52
2.8	Classification of The-State-of-the-Art Security Solutions on Differ- ent SDN Layers/Interfaces (2)	53
2.9	Classification of The-State-of-the-Art Security Solutions on Differ- ent SDN Layers/Interfaces (3)	54
2.10	Summary List of Recent Deep Learning Based IDSs	112
2.11	IDS Accuracy of Detection and Data Set Used	122
2.12	Summary Table of Selected Reviewed Papers	124
2.13	SDN Verification for Network Quality (1)	129
2.14	SDN Verification for Network Quality (2)	130

2.15	SDN Verification for Network Quality (3)	131
2.16	SDN Solutions for Network Quality (1)	132
2.17	SDN Solutions For Network Quality (2)	133
2.18	SDN Solutions for Network Quality (3)	134
4.1	Selected Policies	147
4.2	Query Policies	148
4.3	Sample of Invariant Verification	151
4.4	Sample of Default Policies Being Checked	151
4.5	UPPAAL Parameter Samples	157
8.1	Attack Names Included in Attack Categories	205
8.2	Numbers of Class Records in NSL-KDD Dataset	205
8.3	Attack Types and Related Features	207
8.4	Attack Types and Related Features Continued	208
8.5	Test Results Summary Based on NSL-KDD	213
8.6	Test Results Summary Based on CICIDS2017	214
8.7	Consistency Attack Classes	219
8.8	Deep Learning Consistency Verification Results	223
8.9	Switch and Corresponding Attack Classes Scenario	224
8.10	Consistency Test Scenario Results	226

Chapter 1

Introduction

Future network technologies are on a fast track towards finding solutions to the ever-increasing challenges resulting from the exponential increase in connected devices and network traffic. The lack of seamless scalability, programmability, and remote management led to the rise of new networking paradigms such as SDN, which promises to offer the above three features, in addition to other advantages as well. It provides flexible control of computer networks by orchestrating switches in the network data-plane through a centralized controller. Nevertheless, the progress is still slow towards achieving a reliable and consistent network security solution. Alongside the evolution of networks is the evolution of network attacks, and the major concern is that traditional security protocols are failing at providing proper protection to these future networks. Recent advances in artificial intelligence, machine learning, deep learning, and big data are providing opportunities to address fundamentally the concerns about the security of networks and their correct behavior.

Even though networks started the transition to SDN, it is anticipated that SDN would co-exist with traditional networks for a certain time period. Accord-

ingly, our work discusses the integration of AI and SDN to provide security as well as consistency and correctness of hybrid networks. Our solution consists of a multilevel distributed security architecture with a Central Artificial Intelligence (CAI) based controller. This is possible through featuring multiple levels of security and consistency measures throughout different nodes over the network.

Network administrators handle various applications on the control side to perform different management tasks such as firewall, monitoring, routing, and others. Most of these applications have complex interactions among each other, creating difficult challenge when reasoning about their behaviors. We argue that there is no security without consistency, and enforcing security policies to a network without the ability to verify any misbehavior is a challenge by itself.

Our fundamental motivation was the fascinating tactics of the human immunity system, which is based on a double layer of defense. The first layer is responsible for the deflection of unfamiliar attackers at various exterior contact points in the body without the necessity of identifying their type. The second layer is responsible for the identification of the attacks that were flagged by the first layer, by distributing watchers throughout the blood stream, keeping an eye on any abnormalities. This system earned its efficiency from one layer using minimum energy, and a second layer requiring distributed techniques and more processing.

Another inspiration was our brain functioning as a single control unit which, aside from its millions of functions, controls the consistency of our body by keeping in touch with all our organs to make sure that they are functioning correctly. In simple words, our brain handles this task by knowing ahead the correct function of each organ, then alerts our awareness if any part of the body is malfunctioning after comparing the intended function and the actual one.

The rest of our introduction focuses on introducing both software defined networking and artificial intelligence. Both these essential developments form the foundation of our research and thus, it is important to discuss them in details.

1.1 Motivation

The introduction of cloud services, financial technologies, and server virtualization are among the factors driving the networking industry to reassess the design of traditional networks. One such approach is Software Defined Networking (SDN) [1].

Software Defined Networking (SDN) has opened a new horizon in the field of networking by introducing a separation between the control and data planes. This separation allows the centralization of network logic and decision making in the Network Operating System (NOS) or the control plane, which lowers OPEX and on the other hand, it allows for the simplification of the network forwarding devices into network switches, which lowers CAPEX. Although the introduction of NOS provides a programming abstraction that is leveraged by network application developers to control the network data plane devices, it has serious implications at the level of network security [2]. For example, the centralization of the controller introduces a new concern because it provides the attackers with the ability to control the entire network and with a single point of failure.

In the past few years, SDN/NFV deployment has grown significantly. According to Arbor Network, 11% of respondents reported that they are already implementing SDN/NFV in 2016. However, 39% indicated that they are now testing the technologies, with only 28% having no plans to implement SDN or NFV in the next few years [3].

In terms of the domains where these technologies are having the most impact, data centers are the clear leaders. 75% of respondents deployed these technologies in their data centers, with 42% respondents using SDN or NFV in value-added services infrastructure. Besides, about one-third of respondents deployed it within their fixed-line infrastructure [3].

Regarding the barriers that prevent the deployment of SDN and NFV, half of respondents indicated that both interoperability and operational/business support system integration are key barriers. Around 33% cited security and vendor support as top concerns, and less than one-quarter mentioned scalability [3].

The amount of investment contributed to knowledge discovery for Software Defined Networking (SDN) and Network Functions Virtualization (NFV) by 2020 is estimated to be \$21 billion [4]. However, the area of security requires further attention.

1.2 Introduction to Software Defined Networking

1.2.1 SDN Overview

SDN is an emerging networking paradigm supported by several big names like Google [5] and Cisco Systems [6]. It is based on physically decoupling the intelligence of a network (i.e. Control Plane) from network forwarding devices (i.e. Data Plane) such as switches, hubs, routers etc. Specifically, the control plane is implemented on a dedicated central controller to abstract it from all the underlying structures of the network. Consequently, the controller keeps a global view of the network and dictates the entire network behavior [7]. The data plane

remains located on network devices, which forward traffic based on forwarding rules in flow tables. If there is no corresponding rule for a network packet, the packet is forwarded to the controller as a *PacketIn* message. Forwarding rules are defined by applications running on top of the controller or modules within the controller itself. As a result, SDN offers an application-programming interface (API) where the data plane can be modified by external applications. SDN networks have increased control capabilities over traditional networks. This is due to the implementation of the flow-based structure in SDN, where several header fields delineate how packets should move in the network rather than depending only on the destination address, like in traditional networks. Therefore, the SDN controller can perform better traffic differentiation based on several header fields. Another characteristic that differentiates SDN from traditional networks is the fact that it is provided with self-healing techniques. For example, conditional rules can be installed on switches by the controller and activated if a certain condition is satisfied. These conditional rules are related to statistics gathered by switches and state how the switch should react when the specified condition is encountered. The reaction could be to drop packets or redirect them, providing automatic resiliency against attacks. These security pros are summarized in Table 1.1.

The advantages of SDN have been demonstrated in many cases such as in data centers [5]. First, the SDN model makes the modifications of the network policies less prone to errors since it is software based. Second, the control program can react automatically to any change in the state of the network. Third, the development of sophisticated network functions is simplified.

However, a number of challenges exist [8]; one fundamental area is security. The SDN model provides both opportunities and concerns when it comes to secu-

Table 1.1: SDN Security Advantages Over Traditional Networks

SDN characteristic	Attributed to	Security Usage
Global view of the network	Centralization Traffic statistics collection	Network-wide intrusion detection Network forensics
Better control capabilities	Flow-based forwarding structure	Access control
Self-healing techniques	Conditional rules Switch's collected statistics	Reactive packet dropping Reactive packet redirection

rity. On the positive side, the SDN architecture can be used to improve network security with the delivery of reactive security monitoring, analysis and response system. Well-known network security mechanisms have been implemented in order to realize network security by leveraging SDN [9, 10, 11, 12, 13, 14, 15, 16].

The first aspect of SDN that comes up in the context of security is the centralized controller. Although the single point of failure is troubling to any security professional, this centralization can be an enabler for some security mechanisms. Anomaly-detection methods and traffic analysis can benefit from a global view and control of the network. Nodes in the network can generate security-related data, which can be regularly transmitted to the central controller. Controller applications can then analyze this feedback from the whole network enabling better intrusion detection whereby intrusion prevention can be executed at a granularity, not possible pre-SDN.

On the negative side, SDN networks inherit most, if not all, vulnerabilities of traditional networks. Furthermore, SDN has introduced new vulnerabilities through its unique architecture and southbound/northbound protocol implementations. Examples of vulnerabilities include those related to the OS running the controller, spoofing of open flow rules [17], [18] in the absence of IPSec [18], and

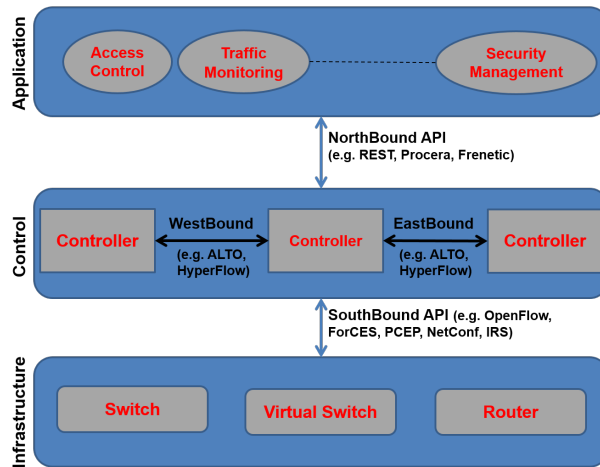


Figure 1.1: Conceptual Architecture of Software-Defined Networking

DoS / spoofing / hijacking of controllers. In addition, the northbound API is a risk, if not secured properly. All this is compounded by the lack of experienced developers and users in this domain, which will also increase the likelihood of misconfigurations.

Moreover, SDN lends itself to new attack vectors as compared to a traditional network topology due to the centralized control and programmability, limited physical security, and threats from compromised nodes inside the network. The primary focus of this work is to provide a survey on different types of security concerns and opportunities.

The architectural design of SDN (Figure 1.1) shows the different layers and communication protocols that are the base of any SDN network. As discussed earlier, each of the planes (data and control) are responsible of a separate task and are controlled by several application modules on the centralized controller. There could be more than one controller, especially in large networks. These controllers synchronize their tasks via east/west-bound APIs. The data plane interconnects via the south-bound API to the control plane.

1.2.2 SDN Architecture and Components

As per ONF [19], the SDN architecture comprises two planes with their respective interfaces as shown in Figure 1.2. The Application layer contains the diverse applications and services such as load balancers, deep packet inspection (DPI), access controls, intrusion detection system (IDS) and intrusion prevention system (IPS). This layer interconnects with the control layer through an application programming interface API (Northbound communication) [1].

The control layer, also known as the central layer, consists of the controllers and it is considered the brain of SDN and is responsible for creating and terminating flows, monitoring the network, and programming the behavior of the physical equipment.

The data plane layer, also known as infrastructure layer, contains the forwarding equipment (switches, routers, etc.). It implements the management operation of the controller through SDN-enabled switches to forward the data, collect the network information, and send it to the controller. The data layer connects to the controller layer through the southbound interface [1].

It is worth noting that the southbound communication can be deployed in two different scenarios:

1. *In-band*: The communication between the controller and the physical interface depends on the flow rules like all other traffic.
2. *Out-of-band*: The southbound communication is implemented through a path that is not affected by SDN flow rules, but via VLAN configuration that separates the networking devices without *OpenFlow* rules, as shown in Figure 1.2.

The adoption of this layered architecture offers many advantages: First the

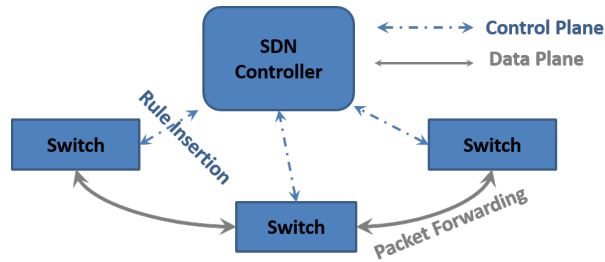


Figure 1.2: SDN Architecture Planes

separation of the planes provides the desired flexibility to combine the benefits of system virtualization and cloud computing. Second, the centralization of the intelligence allows a total view of the network, which helps in decision making and improves network management. Consequently, SDN could be a solution for security services that protect the information system.

1.2.3 Communication Protocols

Southbound Interface

OpenFlow and *OpFlex* are the first SDN protocols that defined the communication between the control layer and architecture layer. The key variance between the two protocols is that *OpenFlow* is agent-less compared to *OpFlex* which requires agents to be installed. *OpenFlow* defines the communication between one or more control servers with switches.

OpenFlow is the de facto standard for SDN; it was standardized by the Open Networking Foundation (ONF) [18], and it is the leading south-bound API. The controller connects to the applications (e.g. SDN management applications) via a north-bound API. The ONF has started a Northbound Interface Working Group but there is no actual standardized northbound API.

Rules installed on the *OpenFlow* switch by a controller application define the

behavior of the switch as a switch, router, network address translator, firewall, etc. An *OpenFlow* switch has one or more tables of packet handling rules. Each rule has a pattern, a list of actions (e.g. dropping, flooding, forwarding, modifying a header field, or sending the packet to the controller), and a priority to distinguish between rules with overlapping patterns. When an *OpenFlow* switch receives a packet, it identifies the highest-priority matching rule, makes the corresponding actions and increments the counters.

The *OpenFlow* protocol comprises three different types of messages:

1. Messages sent by the controller to the switch (handshake, flow table configuration, switch configuration, etc.)
2. Asynchronous messages sent by the switch to the controller (packet-in message, port status message, flow removed message, etc.)
3. Symmetric messages can be sent by both the controller or the switch (Echo Request, Echo Reply, Hello, Experimenter)

Another alternative for the southbound interface is the Forwarding and Control Element Separation (*ForCES*) [20], [21] which was standardized by IETF. *ForCES* defines separate control and data plane functionalities. It is more powerful and more flexible than *OpenFlow* since it can handle complex forwarding mechanisms.

The *SoftRouter* [22] can also fulfill the role of the southbound interface. It also separates the data plane and the control plane and provides dynamic bindings between data elements and control plane elements.

Other standards include the Network Configuration Protocol (*NetConf*), the Locator/ID Separation Protocol (*LISP*) promoted by ONF [23], and the Path Computation Element (*PCE*) [24].

Table 1.2: Main SDN Interfaces

Interface	Description	Examples
Northbound APIs	These APIs enable programmability and are used to communicate between the SDN Controller and the applications layer.	Floodlight Rest API [25, 26, 27], Openstack , Vyatta
Southbound APIs	These API are used to communicate between the control layer and the infrastructure layer. They can be open or proprietary	Openflow, ForCES) [20, 21, 22], SoftRouter , NetConf, LISP , and PCE
Eastbound APIs	These APIs are used to integrate traditional IP networks with SDN networks.	ALTO [28]
Westbound APIs	These APIs enable management of distributed SDN Architecture, and are used to send information between various SDN controllers in different domains.	Hyperflow [29]

Northbound Interface

A standardized universal Northbound API between the controller and the business applications does not exist. The form and frequency of exchanged information depends on the network and applications and therefore universal APIs would not be useful. Table 1.2 summarizes the main SDN interfaces.

The rest of this review is divided as follows: The history of programmable networking is presented in section II. Then, a discussion on SDN security including the different vulnerabilities is given in section III. Section IV provides a review of the main attacks that exploit SDN vulnerabilities. Section V discusses known SDN security solutions. A presentation of network security enhancements that make use of SDN is included in section VI. In section VII, we present a discussion and analysis of SDN security. Finally, we conclude in VIII.

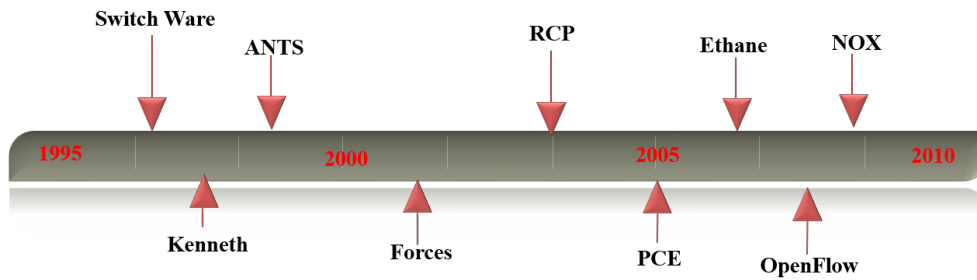


Figure 1.3: Programmable Networking Timeline

1.3 History of programmable networking

1.3.1 Programmable Networking

Programmable networks laid down the foundation path towards SDN [30]. It took its origin in Active Networks when the Internet was just being introduced. The aim of active networking, from mid 1990s to early 2000s, was to introduce a networking API that allows network managers to control the resources available at network devices such as switches and routers. Initial attempts were to introduce functionalities that would apply to a set of packets that pass through network nodes. Active networking evolved into two programming models:

1. Programmable router/switch model where the program to be executed on the network nodes was achieved through out-of-band mechanisms [31].
2. Capsule mode where the network nodes' executable code was carried out through packets of data [32].

What motivated the rise of active networks was the proliferation of middle-boxes, which were deployed separately and programmed differently based on each vendor. The issue also arose due, and not limited to, the large-scale experimentation, time consuming deployment of new devices, and complex management.

Active networks addressed the difficulty of introducing innovation in networking by introducing data plane programmability (ex: NFV) and leading to software programs that depend on the header of observed packets.

The second milestone towards the development of SDN was the era of separation of control and data planes between 2001 and 2007. This was motivated by the fact that network operators had difficulties in network management and administration, particularly traffic engineering, where choosing a particular routing path over another and controlling it was very complicated in terms of the available tools. These led to the introduction of:

1. Open interface between control and data planes such as *ForCES* and *Netlink*
2. Logically centralized controller such as *RCP* and *PCE*

This separation addressed the barriers in network management by having the centralized controller use an open interface with the data plane, and distributing the management of the network state between multiple controllers to prevent single point of failure.

OpenFlow and the Network Operating System (*NOS*) were the latest innovations that led to the birth of SDN, which lasted from 2007 to 2010. It started with the 4D project [33] that consists of the following four layers: data plane which forwards traffic based on forwarding rules; discovery plane to gather network traffic statistics and network topology information; dissemination plane to install traffic-handling rules; and decision plane which represents the centralized controller of the network. *SANE* [34] and *Ethane* [35] took a step further towards the introduction of *OpenFlow* API and switches by taking the high-level architecture introduced by the 4D project and applying it to functionalities other than routing such as access control. An immediate impact of the introduction of *OF*

was the innovation of centralized controllers called Network Operating Systems (*NOS*), which allowed for the development of control applications. *OpenFlow* simplified switch deployment, which led to its adoption in academic research and industry due to the programmable nature of switches as compared to commodity switches.

Nowadays, SDN technology has been deployed in a number of networks [36], [37], which raises the concern about its security. Security of SDN networks is not well explored yet and presents many challenges.

Several working groups have recognized the importance of SDN security and they were established since the beginning of 2013. In the Open Networking Foundation (ONF), some groups were dedicated specifically to promote security in SDN. In the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) and the Internet Research Task Force (IRTF), general SDN study groups have investigated the security issues in SDN. It is essential to increase the focus on security in order for SDN to support the emerging related capabilities such as Network Functions Virtualization (NFV) [38].

Figure 1.3 shows the programmable networking history timeline from 1995 till 2010.

1.3.2 OpenFlow History

SDN is part of a long history of efforts to make networks programmable. *OpenFlow* was the first released SDN standard and it has gone through several revisions before becoming broadly deployed by networking vendors. In this section, we explain briefly the history of programmable networks prior to OpenFlow.

From the mid-1990s to the early 2000s, active networks were presented to include programmable functions in the network to support better innovation. From

2001 to 2007, various efforts were introduced to separate the data and control planes by creating an open interface between them. In the mid-2000s, network experimentation at scale emerged to deploy data-control plane separation by introducing the *OpenFlow* API and network operating systems. In March 2008, the original concept for *OpenFlow* was presented in a white paper by the research team Cleanstate of Stanford University. One month later, the paper appeared as an editorial note in ACM SIGCOMM Computer Communication Review, given the importance of the work. The design of the *OpenFlow* API [39] was followed by the creation of controller platforms like *NOX* [7], which allowed the introduction of several control applications. Directly after the introduction of the first OpenFlow switch specification in December 2009, there was greater interest in the networking market. In April 2009, NEC was the first to present a commercial switch with built-in *OpenFlow* support. Also, companies like Big Switch, Vello, Plexxi and Pica8 started to offer SDN-ready solutions. In 2011, Google declared the adoption of SDN inside their data centers backbones. In the same year, The Open Networking Foundation (ONF) [18] was established by Deutsche Telekom, Microsoft, Google, Facebook, Verizon, and Yahoo to promote the concept and operation of SDN and *OpenFlow*-based networks by standardizing *OpenFlow*. Now, ONF has more than 95 members including numerous major vendors.

1.4 Introduction to Artificial Intelligence

By definition, AI is “the study of mental faculties through the use of computational models” [40]. Main fields of AI research include reasoning, knowledge representation, automated planning and scheduling, Machine Learning (ML), natural language processing, computer vision, robotics and general intelligence.

1.4.1 Machine Learning

ML deals with the construction and generalization of algorithms (i.e. learn) from limited sets of data. Such algorithms operate by building models based on input and using those models to make predictions or decisions, rather than following only explicitly programmed instructions. Having such characteristics makes them ideal candidates for network security. There are three common problems that ML tries to solve: classification, regression and clustering. Classification involves identifying group membership, with its output labels being class labels. While clustering involves a set of inputs that are divided into groups where members have similar characteristics, with its output labels being subsets (i.e. clusters). Regression, on the other hand, involves estimating or predicting a response, with its output labels being continuous numerical values. Thus, regression is applicable for prediction type of problems as opposed to classification and clustering. Depending on ways of learning, ML can be further split into three main categories: supervised, unsupervised and reinforcement learning. Supervised learning is commonly used in classification problems where the goal is to get the computer to learn a classification system that we have created. Unsupervised learning seems much harder; the goal is to have the computer learn how to do something that we do not explicitly teach it how to do. Unsupervised learning is a powerful tool for identifying structure in unlabeled data, thus reflecting the statistical properties of the overall collection of input patterns. Reinforcement learning is performed by interacting with an environment where the learning agent learns from the consequences of its actions through trial and error, rather than from being explicitly being taught. There is no single AI algorithm that can achieve the best accuracy for all situations. Hence, one way to improve results is the fusion of multiple algorithms to obtain a better quality of reasoning rather than using

a single algorithm. There are generally two approaches: ensemble and hybrid [41]. In case of ensemble classifiers, multiple but homogeneous weak models are combined, typically at the level of their individual output, using various merging algorithms (e.g. majority voting). Hybrid algorithms, on the other hand, combine completely different, heterogeneous, AI approaches (e.g. through cascading). In the following section, we discuss some machine learning algorithms, each falling under a different category from the ones mentioned above [42].

1.4.2 Artificial Neural Network

Artificial Neural Network (ANN) [43] is an algorithm in machine learning. It is inspired by biological neural networks in the brain. ANN is presented as a system of forward computation of “neurons” in multi layers where each pair of neighboring layers is connected. The “neurons” in the same layer have no associations with each other. Solving an ANN involves optimizing the weight parameters between two neighboring layers and a bias parameter, and then using the model with optimal parameters for the real data [44].

Supervised learning

In supervised learning, the neural network learns the mapping process from inputs x to outputs y given a labelled set of inputs-output pairs:

We refer to d as the training set and N the number of training examples. It is assumed that y_i is a categorical variable from some infinite set; $y_i \in 1 \dots C$ [45].

Unsupervised learning

In unsupervised learning, the neural network is only provided with input data without conceptualizing the output: it discovers patterns within the data au-

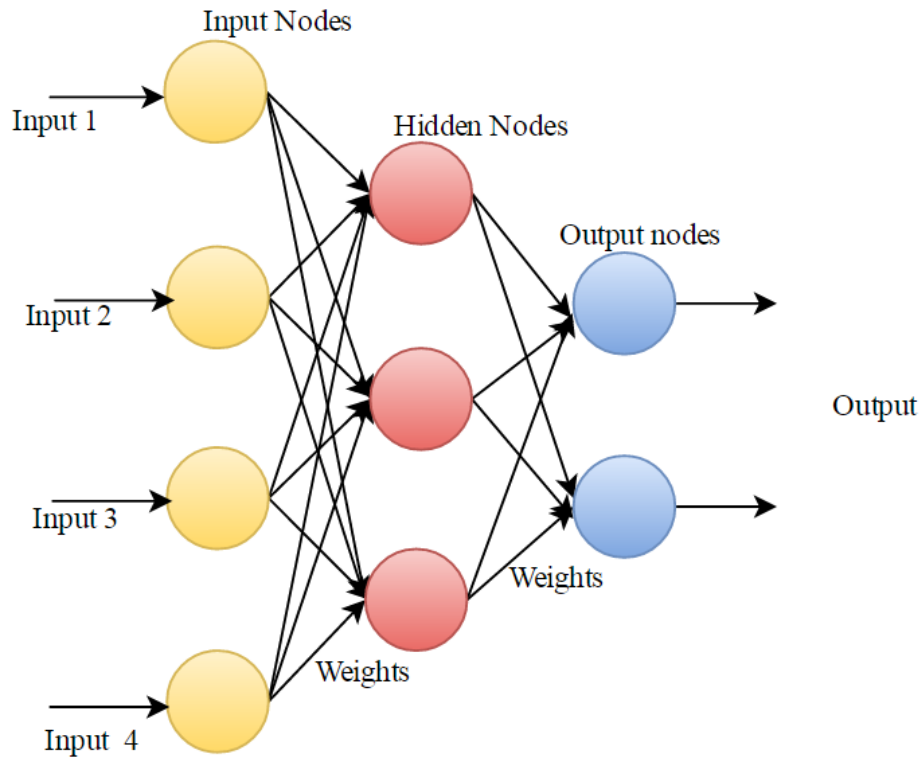


Figure 1.4: Three layer Neural Network

tonomously. The data yet to be discovered is called unlabeled data [45]. Two typical unsupervised learning are Self-Organization Maps (SOMs) and Adaptive Resonance theory (ART).

1.4.3 Multilayer Perceptron

When used without qualification, the terms “neural network” (NN) and “artificial neural network” (ANN) usually refer to a multi-layer perceptron (MLP). MLP is a supervised learning algorithm based on the feed-forward neural network with one or more layers between the input and output layers. Feed-forward means that data flows in one direction from input to the output layer (i.e. forward). This type of network is trained with the error back-propagation learning algorithm.

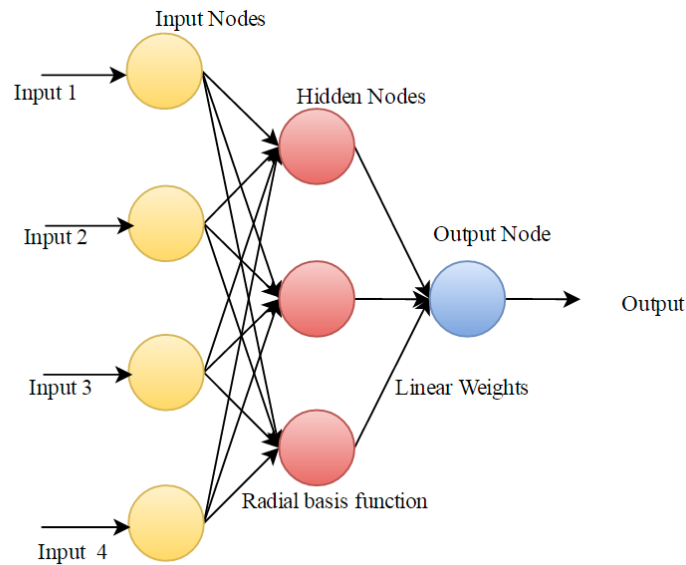


Figure 1.5: Radial Basis Function Architecture

The true power and advantage of MLP lies in its ability to represent both linear and non-linear relationships, and in its ability to learn these relationships directly from the data being modeled. Traditional linear models are simply inadequate when it comes to modeling data that contains non-linear characteristics [46], [47].

1.4.4 Radial Basis function

Radial Basis function (RBF) is another feed forward neural network. It classifies by taking a measurement of the distance between the inputs and the center of hidden neurons [48]. Figure 1.5 is an RBF architecture showing the input nodes, hidden nodes and an output node. Each RBF has different parameters with an input vector. The network output is thus a linear combination of the radial basis function's output. The input and hidden nodes' weights are always 1 since the transfer function of the network is a Radial basic function. This allows an adjustment on the weight between the hidden nodes and the output [49].

1.4.5 Self-organizing Map

Kohonen's self-organizing map (SOM) is an unsupervised learning algorithm producing a low dimensional (typically two-dimensional), discretized representation of the training samples' input space, simply called a map. It consists of components called nodes or neurons. Each node is associated with a weight vector of the same dimension as the input data vectors and a position in the map space. Nodes in a 2-D layer learn to represent different regions of the input space where input vectors occur. In addition, neighboring neurons learn to respond to similar inputs, thus each layer learns the topology of the presented input space. SOM provides a way of representing multidimensional data in much lower dimensional spaces, where this process of reducing the dimensionality of vectors is essentially a data compression technique known as the vector quantization [46], [50].

1.4.6 Adaptive Resonance Theorem

ART is an unsupervised learning model but, as a hybrid, it performs supervised learning. It functions as a pattern recognition and prediction tool. The unsupervised learning models consist of ART-1, ART-2, ART-3 and fuzzy Art. The supervised models consist of ARTMAP, Fuzzy ARTMAP and Gaussian ARTMAP [48]. In general, the models compare each input vector to a single neuron's weight (weight vector) [48], [51].

1.4.7 Genetic Algorithm

Genetic algorithm (GA) is a search algorithm that works similar to the process of natural selection. It begins with a sample set of potential solutions which then evolves toward a set of better solutions. The algorithm repeatedly modifies

a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population “evolves” toward an optimal solution. Within the sample set, solutions that are poor tend to die out while better solutions integrate and propagate their advantageous traits, thus introducing more solutions into the set that boast greater potential (the total set size remains constant; for each new solution added, an old one is removed). A little random mutation helps guarantee that a set won’t stagnate and simply fill up with numerous copies of the same solution [46].

1.4.8 Fuzzy Logic

Fuzzy logic (FL) is a form of many-valued logic, where one deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets where variables may take on true or false values, FL variables may have a truth value that ranges in degree between 0 and 1. FL proponents claim this generality allows greater flexibility, freedom, accuracy and compactness when representing real world situations. All the usual properties of Boolean algebra can be extended to FL, and probability’s degree of belief in a Boolean variable becomes a fuzzy variable’s degree of truth. FL provides a principled way to encode expert knowledge or heuristics into algorithms that mimic a human’s approach to solving challenging problems by weighing different sources of information and making judgments based on a preponderance of the evidence [46], [52].

1.4.9 K-nearest Neighbors

K-nearest neighbors (kNN) [53] is a traditional non-parametric supervised learning algorithm. It stores all available cases and classifies new ones based on a similarity measure (e.g. distance function) calculated between feature vectors. Classification is made by a majority vote of object's k nearest neighbors, where k is typically chosen to be a small positive integer (e.g. 3). For example, if k is chosen to have a value 1, then object is assigned to the class of its nearest neighbor. It has to be noted that if k is chosen to be considerably large, it will result in additional classification time [46].

1.4.10 K-means

K-means [54] aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean which serves as the prototype of the cluster. Given a set of data (x_1, x_2, \dots, x_n) , where each data is a d-dimensional real vector, K-means clustering [55] aims to partition the n observations into $(k \leq n)$ clusters $C = C_1, C_2, \dots, C_k$ to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

$$\sum_{i=1}^K \sum_{x \in C_i} \|x - C_i\|^2 \quad (1.1)$$

1.4.11 K-medoids

K-medoids algorithm [56] shares the properties of the K-means algorithm. Instead of calculating the mean of items within the cluster, a representative item or

medoid is selected from each cluster at each iteration. Medoids for each cluster are calculated by finding object i within the cluster that minimizes the following equation:

$$\sum_{j \in C_i} d(i, j) \quad (1.2)$$

Where C_i is the cluster containing object i and $d(i, j)$ is the distance between objects i and j

1.4.12 Naive Bayes

Naive Bayes (NB) [57] is a supervised learning algorithm based on Bayesian theorem with the “naive” assumption of independence between every pair of features. Classification is made by combining prior probabilities and likelihood, to form a posterior probability using the so-called Bayes’ rule.

$$P(C_k|x) = \frac{P(C_k)P(x|C_k)}{\sum_{i=1}^k P(x|C_i)P(C_i)}, \quad P(C_k|x_1, \dots, x_n) \quad (1.3)$$

Despite its simplicity, NB can often outperform more sophisticated classification algorithms in both speed and accuracy [57]. Since it uses probability the accuracy of classification is increased; however, it takes a longer time for classification as the training data increases. Also, NB is particularly suited when the dimensionality of the input is high, and when the attributes are independent of each other [46], [58] .

1.4.13 Decision tree

Decision tree (DT) is a supervised learning algorithm based on flowchart-like structure in which internal nodes represent a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label. Decision is taken after computing all attributes. The paths from root to leaf represent classification rules. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features (i.e. attribute). DT is simple to understand and to visualize, making its explanation for the resulting value easily explainable by Boolean logic [46].

1.4.14 Support Vector Machine

Support vector machine (SVM) is a supervised learning algorithm based on the concept of decision planes that define decision boundaries. An SVM constructs the hyperplane, or a set of hyperplanes in a high-dimensional space, that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes. It often happens that classes can’t be linearly separated. For this reason, the original finite-dimensional space can be mapped into a much higher-dimensional space, using what is called the “kernel trick”, presumably making the separation easier [46], [59] .

1.4.15 Random forest

Random forest (RF) is a supervised learning algorithm that is based on a collection (ensemble) of tree predictors, rather than a single classification tree, where

to grow each tree a random selection is made from the examples in the training set. Each tree gives a classification and we say that the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest), with basic principle that a group of “weak learners” can come together to form a “strong learner” [46].

1.4.16 Deep Learning

Deep learning (DL) is a branch of machine learning based on a set of algorithms. Some of the most successful deep learning methods involve artificial neural networks, such as Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Deep Belief Networks (DBN) and Stacked Auto Encoder (SAE) [60]. In recent years, DL has gained popularity and it was applied to computer vision [61], speech recognition [62], and natural language processing [63]. Studies show that deep learning completely surpasses traditional methods in most areas. Surprisingly, the error rate fell from 26% to 15% in ImageNet Challenge 2012 [64]. The most important advantage of deep learning is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction; where auto encoder is a perfect example. It aims to learn an efficient, compressed representation for a set of data [65]. The state-of-the-art on machine learning breakthrough comes from deep learning which has been predicted to cause a powerful improvement in artificial intelligence field. Numerous complex applications have been accomplished by deep learning. One of the distinguished applications is AlphaGo from Google that uses Convolutional Neural Network. AlphaGo beat the Korean world champion in the “Go” game recently by showing superman-like capabilities in remote machine learning. The advancements on this learning algorithms may improve IDS ability to reach high

detection rate and low false alarm rate [66]. Deep learning methods such as deep belief network, restricted Boltzmann machine, deep Boltzmann machine, deep neural network, auto encoder, etc., are highly recommended for IDSs.

1.4.17 Overall Summary

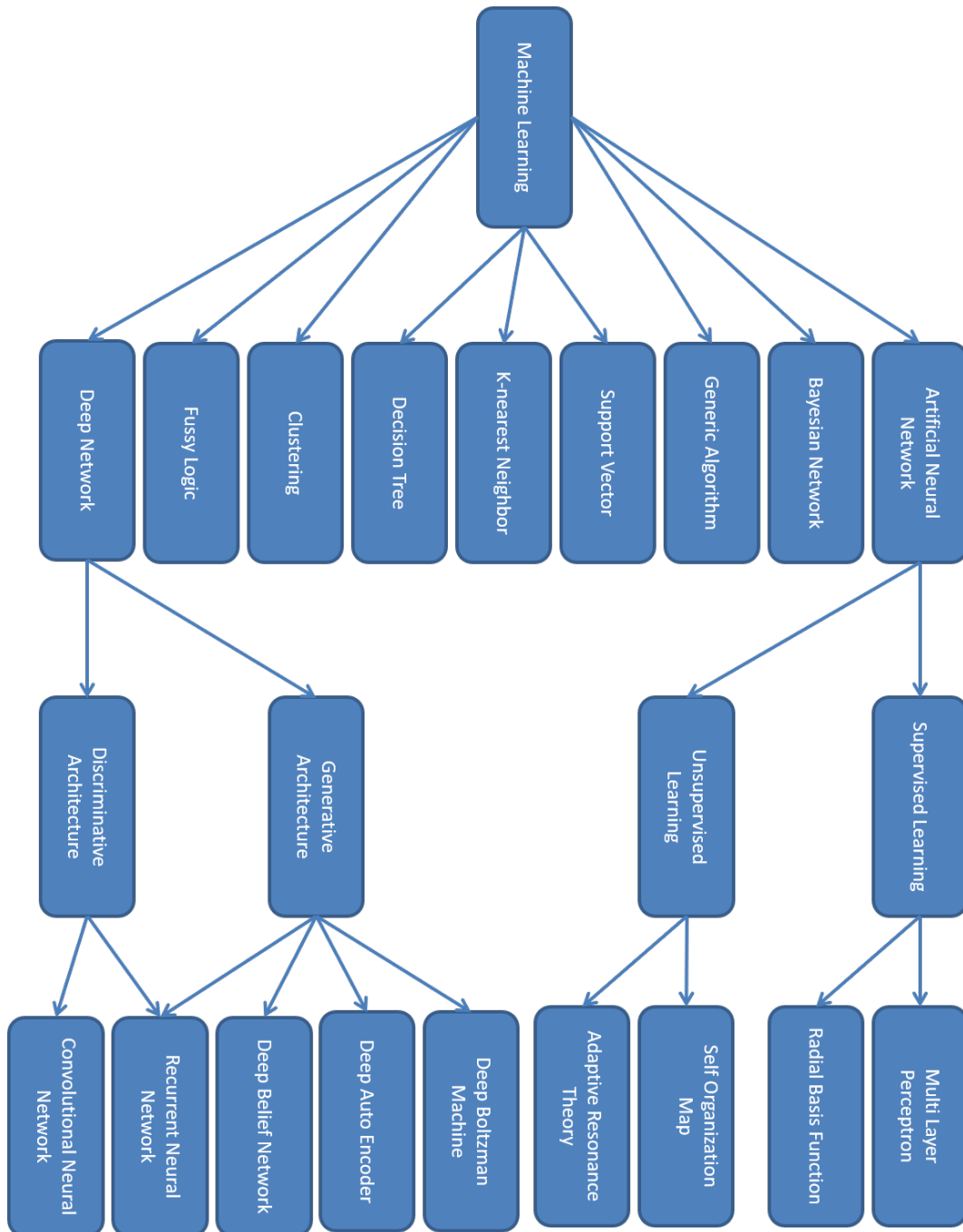


Figure 1.6: Machine Learning Overall Summary

Chapter 2

Literature Review

Network security and consistency have taken a part of the research domain for their necessity and importance. Different solutions, techniques, and propositions have been presented and discussed for different purposes in the network domain and others. We will discuss and comment on the literature in these domains through the years, thus maintaining a solid information background on these topics that would benefit our work.

2.1 SDN Security Overview

SDN is likely to replace current traditional networks with several innovative features. However, SDN adoption is still hindered by many security-related concerns. In this paper, we analyzed the security vulnerabilities, attacks and solutions of the current SDN stage. SDN features expose the network to a number of new attacks. Therefore, it is essential to make them more robust and secure to adapt to the requirements of these networks. This leaves SDN open to research to meet the security challenges. The research on SDN security is still in its early stages;

current proposals are typically attack-oriented, that is, they first recognize some security threats and improve on the existing protocol or propose a new protocol to impede such dangers. Because the solutions are intended explicitly for certain attack scenarios, they work well in the presence of such attacks but may fail under unanticipated attacks.

Although the amount of investment and research is increasing in the SDN domain, the security community is slow in embracing and deploying the SDN technology. It is hard to find studies that examine the practicability, feasibility, effectiveness and efficiency of network security applications based on SDN technology.

Therefore, a more ambitious objective for SDN security is to develop a multi-fence protection solution that is embedded into every element of the network, resulting in depth security that deals with multiple lines of defense against both known and unknown security threats.

2.2 SDN Security and vulnerabilities

SDN security is the main concern facing the future of SDN deployment. Researchers such as [67, 68, 69, 70] focused on reviewing SDN security in terms of network attack detection and defense, challenges and opportunities.

Tao discussed how the repositioning of the control plane as an external entity in SDN would redefine the security weaknesses. Although they found similar exposures in both architectures, yet they indicated that SDN cannot depend on edge-based filtering to protect its control plane, which is the primary defense in conventional networks. Their analysis was based on different network properties such as basic forwarding, loop-free forwarding, link redundancy, device redun-

dancy and scalability. They discussed control functions, attacks and defenses for both types of networks for these different properties. Their security analysis suggests that a distributed SDN architecture that supports fault tolerance and consistency checks is important for the SDN control plane security [71].

A security analysis of SDN was made including integrity challenges, weaknesses and vulnerabilities on plane-specific bases [72]. The research highlighted the control plane, control channel, and data plane specific attack points. A review on the proposed security models in an SDN environment was also discussed at different levels including Identification and authentication, state table management, conflict resolution, use of TLS and Flow checking.

SDN security can be divided into different sections that include communication security and general vulnerabilities which are reviewed below.

2.2.1 SDN Communication Security

The communication channel between the controller and switches is supported over a TCP connection that can optionally be secured by TLS with mutual authentication [73]. It is important to note that TLS encryption affects the performance of the controller due to extra processing and delays. Based on the experiments performed in [74], the authors deduced that a large delay is added by the switches on the first packet-in message, where it differs from one vendor to another based on CPU power.

Linyan et al. analyzed the trust boundaries between the different SDN layers and endpoints i.e. terminal-switch, switch-switch, switch-controller and controller-administrator [75]. They modeled and analyzed the layered SDN architecture using attack trees and petri nets (AAMPA). The model divided the SDN architecture into user access, data transmission, and control command distribu-

tion; it constructs the Petri net of each structure and the attack tree, and this showed the possible vulnerabilities that may occur between Petri net locations and transitions, which represent the vulnerabilities of different entities and their corresponding connections in SDN.

Others have worked on a trust management framework for network applications within an SDN environment. The research focused on the authentication and authorization of network applications that control network behavior (unlike the traditional network where network devices like routers and switches are autonomous and run proprietary software and protocols to control the network). The paper proposed a mechanism to help the control layer authenticate network applications and set authorization permissions that constrict manipulation of network resources [76]. The main security weaknesses at the different levels are:

At the northbound communication level:

1. The weak authentication at this level may allow spoofing attacks and spoofed northbound communications.
2. Incorrect authorization that may lead to malicious access on the applications.

At the southbound communication level:

1. The lack of encryption of traffic between switches and the controller may cause eavesdropping and spoofing of southbound messages.
2. The weak authentication between the controller and switches may allow spoofing and Man-In-The-Middle attacks, which allows the attacker to eavesdrop and analyze traffic.

3. Incorrect authorization that may lead to unapproved access. For instance, if a user demands a service that causes the controller to make a route and cause packets to follow that route, it must be verified that the user is allowed to do so.
4. The Southbound communication is vulnerable to attacks on flow rules especially in In-band placements.

2.2.2 SDN General Vulnerabilities

Originally, security gaps were found throughout traditional networks, and most of them have not been resolved yet. As the SDN era began and the transition process is on the correct path, some security issues have been solved, others have been introduced, and the rest were inherited from traditional networks. Different research works have been done on the security analysis of SDN including SDN specific and non-specific vulnerabilities such as [77], which focused on the security of stateful SDN data planes. Others, such as [78], took part of identifying the general security issues that SDN is facing on the road of evolvement. The main vulnerabilities of SDNs include:

Centralized control

The centralized control of SDN denotes a single point of failure and makes SDN vulnerable to attacks and disruptions [79]. In fact, the centralization of management makes the controller an attractive target for an intruder. Anyone with access to the controller is able to manipulate directly each flow and as such controls the whole network. Also, if the controller is not available due to an attack (e.g. DoS attack), or to a misconfiguration, switches become blocked since they

Table 2.1: SDN Security Issues At the Communication Level

Communication Level	Malicious Behavior	Reason
Northbound	Spoofing attack	Weak authentication
	Malicious access on the applications	Incorrect authorization
Southbound	Eavesdropping	Lack of encryption
	Spoofing	
	Eavesdropping	Weak authentication
	Spoofing	
	Man-In-The-Middle attacks	
	Unapproved access	Incorrect authorization

are only able to apply predefined rules and cannot handle correctly new packets that do not match any existing flow entry. Being the focus of SDN security, a solid solution is still being researched to protect the controller from such events. Along side multiple backup controllers waiting as a last fallback in case of a major breakdown.

Programmability

Since SDN networks are mainly programmable, they are vulnerable to targeted attacks. A malware targeting specific network architectures, could compromise the operation of the network devices by modifying their configurations, which will have dramatic consequences on the network.

Open Source Standards

SDN is implemented through open standards such as *OpenFlow*, which has open source implementations, and hence, attackers have the advantage of identifying vulnerabilities.

Flow Table Security Risk

A flow table can be managed through the main controller and a Backup controller. The latter should be subject to lower security levels, and therefore it is important that the flow table entries remain consistent and protected from a malicious update emanating from a compromised controller. Currently, *OpenFlow* does not account for such consistency unlike *FlowVisor* [80].

Lack of Important Functionalities

Important security functionalities are not inherently defined within the SDN architecture such as firewalls or Network Address Translation (NAT). In currently operating networks, many of these functionalities are realized in the form of added devices (middle boxes). However, the implementation of some functions is not easy and has some limitations such as deep packet inspection in the *OpenFlow* protocol.

Dynamic Reaction Design

As mentioned earlier, the data plane forwards incoming packets based on flow rules in its flow tables. When it receives new packets with no defined flows, these packets are forwarded to the controller that generates new forwarding rules to deal with them dynamically and not pro-actively. An attacker could benefit from this dynamic reaction design of the controller to get information about network states.

Lack of TLS Requirement in OpenFlow

Current *OpenFlow* specifications (v1.5.0) propose that Transport Layer Security (TLS) protocol may be established to secure communication channel between

switches and controllers. However, these specifications make TLS optional [81], leaving an opportunity for adversaries to infiltrate OpenFlow networks.

Lisa et al. showed in [82] that the security benefits of SDN outweigh the threats that are either SDN specific or inherited. They assessed and compared information security of SDN and conventional networks with respect to the impact on the network from the perspective of authenticity, integrity, confidentiality, availability and consistency. Furthermore, they evaluated the information security or network characteristics provided by SDN and conventional networks from the perspective of network management, costs, and attack detection and prevention. As a result of these performed evaluations, the authors concluded that SDN is a step forward for network security since the advantages exceed the available threats.

2.3 Attacks on SDN

The security Analysis section addresses the different type of security issues and attacks that threaten SDN security and some related countermeasures. First, we discuss attacks that physically affect the controllers and switches. Then, we discuss the attacks that affect the different SDN planes, Application, Control, and Data planes.

Tables 4 and 5 present a time-line based summary of the Main SDN Security research, including both SDN solutions and attacks. Also, a general view of the physical and plane based attacks is provided in tables 6.

2.3.1 Physical attacks

Universal Plug & Play (UPnP) security Drawback

The authors of [83] presented an SDN-based solution to overcome the vulnerabilities due to physical attacks. In particular, universal plug and play (UPnP) plays a crucial role in connecting heterogeneous devices to enable information exchange between them. UPnP has inherent security-related drawbacks because it is operated using simple service discovery protocol and user datagram protocol (UDP). UPnP is a protocol in which users can be automatically allocated an address without network settings being inserted.

This study proposed a method of automatically detecting illegal traffic using an SDN-based switch by inserting rules to prevent DDoS attacks in an UPnP environment. Upon receiving an alert, the alert is matched for a specific attack based on the alerts (rules) that have been matched previously.

Controller hijacking

The controller is prone to Controller Hijacking attack disturbing the overall security of SDN and leading to the compromise the entire network. An attacker taking over the controller would have full control over SDN defined policies, and could easily gather sensitive information such as communication data, passwords, etc. This would allow the adversary to tamper with the data and redirect traffic. In practice, controller hijacking is a result of malicious applications that leverage the vulnerabilities of northbound API to take control of the network controller. Hong et al. discussed an attack called host location hijacking, where the identity of a target host is spoofed to hijack its location information inside *OpenFlow* controllers [84].

Table 2.2: Main SDN Security Research Timeline Summary

Type	2009	2010	2011	2012	2013
Solutions	[85] Flow Visor [24]	[86, 87, 88] FlowChecker [89]	[90]	FortNox [91] VeriFlow [92] LiveSec [93], [94]	POCO [95] NSV [96] Fresco [97] Avant- Guard [80], [98, 99, 100, 101]
Attacks				FortNox [91]	

The Malicious Administrator Problem

A network administrator can mis-configure controllers, intentionally or not, and ultimately damage forwarding, routing, or network availability.

2.3.2 Plane Based Attacks

Application Plane

The application plane in SDN is vulnerable to different types of attacks primarily due to the lack of authentication and authorization. Since different modules and applications are installed on the controller, with a different function assigned to each, all applications are able to install flows into the switches without flow authentication. Another problem is the absence of verifications or consistency checks of the installed updates, which may lead to unwanted behavior that could affect the security of the network. Moreover, no access control mechanisms are installed on the controller to restrict the interaction between the applications or prevent applications from accessing or altering the controller's resources.

Another issue in the SDN Application layer is the lack of global standards

Table 2.3: Main SDN Security Research Timeline Summary (continued)

Type	2014	2015	2016	2017
Solutions	Flow [102]	SPINX [112]	HoneyMix	DELTA [166]
	Payless [103]	DaMask-D	[145]	DaaS [167]
	Rosemary [104]	[113]	BGPSEX [146]	SODM [143]
	DBA [105]	IB-AKA [114]	SHIELD [147]	Distributed-
	OrchSec [106]	Operetta [115]	UNISAFE	SOM [168]
	FlowGuard	Mynah [116]	[148]	SLICOTS [70]
	[107]	BigTap [117]	CPP [149]	WedgeTail [169]
	[105],	EnforSDN	DDoS	[67], [141],
	[108, 109, 110]	[118]	Detection[150]	[170, 171, 172,
	Firewell [111]	FlowMon [119]	OpenSec [151]	173, 174, 175]
		CINDAM [120]	NFG [152]	Perm-Guard
		SDSNM [121]	SD-anti-DDoS	[176]
		SN-Security	[153]	
		DDoS De-	DHC [154]	
		tection	BroFlow [155]	
		[122, 123, 124]	CIPA [156]	
		[96], [113], [125,	HogMap [157]	
		126, 127, 128,	PBS [3], [83],	
		129, 130, 131,	[141], [158, 159,	
		132, 133, 134,	160, 161, 162,	
		135, 136, 137,	163, 164, 165]	
		138, 139, 140,		
		141, 142, 143]		
		OFX [144]		
Attacks	[80], [109]	[81], [84], [116],	[75], [159],	[182], [183]
		[122], [177, 178,	[180], [181]	
		179]		

or open specifications to facilitate open APIs for applications to control network services and functions; this can pose serious security threats to network resources, services, and functions.

Since SDN is an open project, different vendors and third-party applications are developed in different environments and in different programming modules and languages leading to interoperability issues and security policy collisions.

Control Plane

Spoofing Attack: OpenFlow networks are subject to spoofing [184], which is not specific to SDNs, however, it can have a larger impact on it. For example, the attacker could have control over the whole network by simply spoofing the address of the controller. A spoofing attack on the controller could last for few seconds just the time needed to program rules that are dedicated for malicious purposes.

In the absence of TLS or switch authentication, a State-Spoofing attack is possible where a compromised *OF* switch spoofs its state to the controller i.e. statistics information, interfaces, and flow table. Furthermore, a compromised switch can create virtual switches or links in the network and thus change the topology view of the network at the controller, which can change the paths taken to route traffic, create congestion or reserve bandwidth for the attacker himself [109].

Man-in-the-middle Attack: The channel between the controller and the switches is established over a TCP connection that is vulnerable. As indicated by the OpenFlow specification, Transport Layer Security (TLS) with mutual authentication can be used optionally to secure the communication. Hence, it is

possible for the controller and switches to communicate using plain text. TLS requires mutually authenticated certificates signed by a private key corresponding to a mutually trusted public key. However, TLS requires many steps to be configured. Also, many switches and controllers lack the support of TLS. As such, TLS is most of the times left off allowing adversaries to penetrate OpenFlow networks and insert fraudulent rules or modify existing ones [99].

Brooks et al. aimed to exploit two SDN specific vulnerabilities; non-secure communication channel between SDN switches and controller, and controlling the network by compromising the controller [178]. They were successful in exploiting these vulnerabilities by performing MITM attack using *ettercap* through ARP cache poisoning on the controller, traffic sniffing, and capturing the login credentials of Open Daylight (ODL) web interface.

Denial-of-service attack: The authors of [185] indicated that a challenging issue of SDN is the bigger potential of denial of service (DoS) attacks due to centralized controllers and flow tables limitation. DoS attacks can target the routing information and therefore disrupt the entire operation of SDN networks. Basically, since the data plane is separated from the control plane, the data plane will normally ask the control plane to find flow rules when the data plane gets new packets. An attacker could specifically produce fake flow requests from the data plane, which has two consequences: 1) it can make it difficult for the centralized control plane to handle all requests (control plane resource consumption) and 2) the crafted flow requests can yield too many unusable flow rules that need to be stored by the data plane (data plane resource consumption) [186].

Data Plane

Scanning Attack: An adversary can remotely scan SDN networks by sending probes to random IP addresses in the network. When a target replies, it can then be identified or attacked. If no one responds, the probed IP addresses are considered unused.

Fingerprint SDN networks: The authors of [98] proposed a method for an attacker to identify whether or not a particular network is SDN. Almost all traditional networks have a pre-configured forwarding table and hence they do not need extra time to process and generate a flow entry for a newly received packet. In contrast, the SDN controller takes some time to create a new flow entry for a new incoming packet. Based on this information, attackers can recognize whether a network is SDN or not by testing the difference between the response times of the first and subsequent packets.

A different approach was taken by the authors of [128] where active and passive fingerprinting was performed through RTT and packet-pair dispersion features to identify whether an interaction is occurring between a switch and a controller. When a new packet arrives at an *OF* switch, a *PacketIn* message is sent to the controller which installs the corresponding rules on the switches, however, when a flow rule already exists, the packets are simply forwarded. The delay that is added due to the communication with the controller is an evidence of this interaction that is exploited by the authors. To measure the accuracy of their procedure, they used the Equal Error Rate Metric (EER), which is the rate at which False Match Rate (FMR) and False Non-Match Rate (FNR) are equal. As EER decreases it can be clearly distinguished whether a rule installation has been triggered or not. The packet-pair dispersion is a stable feature and is not

affected by changes in network configuration, whereas RRT leads to an increase in EER when network size and configuration changes, if measured over a long period of time.

Information Disclosure: An attack scenario is described in [184] where the attacker derives information about active flow rules. This is achieved by timing the TCP setup of two connection attempts. If the second connection attempt is considerably faster than the first one, the attacker may conclude that a new flow rule has just been installed and did not exist before the connection attempt. On the contrary, if there is no substantial variance, then a flow rule existed previously. Hence, the attacker exploits the flow aggregation to discover the content of flow tables by observing differences in controller response times.

By injecting fake flow rules into switches via malicious applications on controllers, hackers can bypass the firewall, and invade the network system. This is called Dynamic Flow Tunneling.

Tampering Attacks: An important type of tampering attacks in SDN is Fraud flow rules due to the lack of verification and consistency checks. A concrete example of tampering is dynamic flow tunneling: an attacker might try to orchestrate several rules, where no single flow violates any firewall rules but they can indeed violate firewall rules in a collaborative way. The authors of [91] designed such an attack and indicated that OpenFlow controller can generate optimal flow routing rules from remote clients to virtually spawned computing resources.

TCP Level Attacks: Forwarding devices connect to the remote controller over TCP. Consequently, an attacker could launch known TCP-level attacks to break into the network. Regardless of the use of SSL over TCP, there are many TCP-

level attacks threatening the security of SDN. These attacks could be in the form of ICMP attack, reset attack, SYN attack, sequence prediction attack and DoS attack.

However, *OpenFlow* specification indicates the applicability of user datagram protocol (UDP) with datagram transport layer security (DTLS), but there are no full considerations of how it could be implemented and used[101].

Cache poisoning on flow tables and controller state: This type of attack refers to unauthorized modification of rules in flow tables. Attackers can add fraudulent flow rules that may cause network misbehavior. Hong presented an attack of Fake LLDP Injection, in which an attacker produces forged LLDP packets into an OpenFlow network to declare bogus internal links between two switches. By observing the traffic from OpenFlow switches, the attacker can get the real LLDP packet [84].

Lack of Authentication: Kang discovered a new SDN specific attack that occurs due to the lack of authentication of the data plane [143]. Particularly, the absence of data plane identifier (DPID) authentication causes network insecurity and instability. The reaction against DPID duplicates is based on the implementation of the controller where some send packets to all switches that have the same DPID while others replace an existing connection with another having the same DPID.

A study was made of the vulnerabilities in *ONOS*, *Floodlight* and *ODL* through developing a network topology sniffer masquerading itself as a load balancer by leveraging the lack of authentication between controller and applications [180].

A malicious application modifies *Floodlight PacketIn* listener component to make itself the first to receive *PacketIn* messages, removes the payload and passes

the message to the next interested application, which raises an exception once no payload is found and thus the *Floodlight* controller disconnects itself from the switch. In *ONOS* and *ODL*, the malicious application changes the parameters of the properties/services configured for a certain target application thus disallowing it to install flow rules on the switch, which leads to a *PacketIn* message generation for every received packet thus overwhelming the controller.

Control Channel Hijacking: This occurs when a compromised *OF* switch detaches itself from an authentic controller and attaches to a malicious one, and through flow table modification, it redirects control channel traffic to this malicious controller, which in itself spoofs messages to the honest controller [109].

Saturation Attack: The authors of [141] proposed an approach to defend against switch saturation attacks. First, they added a miss-matched packet cache module in the *OpenFlow* switch, which can temporarily cache the packets that don't match in the flow table. Then, they applied the mechanism of separating the header and payload of packets in the cache queue once the switch detects the volume of cache queue exceeding the threshold of the cache size. In addition, the switch can classify the packets headers and send it in an alert message to the SDN controller for further processing.

Freeloading: Park et al introduce a new SDN specific attack called Freeloading where an attacker intercepts the traffic and spoofs its IP/MAC address to one of the hosts of an already established communication link, sends malicious packets and avoids detection by the controller by making use of existing flow rules [159].

Eavesdropping: Other work is addressing the issue of how to cope with eavesdropping attacks in the SDN data plane by using multiple routing paths to reduce the severity of data leakage. While this existing approach appears to be considerably effective, their analysis discussed that without a proper strategy of data communication, it can still lead to total data exposure [182].

2.3.3 OpenFlow Security Problems

Several SDN security advantages are exploited in OpenFlow as the standard implement the flow-based forwarding scheme and necessitate switches to collect traffic statistics. However, the protocol presents several security concerns which are examined in the following.

The lack of TLS adoption makes OpenFlow vulnerable to man-in-the-middle attack whereby attackers can penetrate OpenFlow networks and remain undetected. Once an attacker places a device between the controller and the switch to capture OpenFlow traffic, he could insert fake additional rules into the switch or modify the existing ones. The authors of [80] discussed the possibility for the controller and *OpenFlow* switches to use plain text traffic. Even worse, if the traffic is encrypted, the man-in-the-middle attack is still feasible between switches and controller [99]. In this case, an attacker can intercept the traffic to compromise the security and privacy of an SDN network. The threat of these types of attacks is greatly worsened if a switch is left configured with a passive listening port. In this mode, the switch will accept unauthenticated connections from any network source and simply accepts all commands. This mode of operation is used to write rules to switches and read information from them for easy debugging purposes. However, it presents a main vulnerability since it has no integrated authentication. The attacker can modify or delete flows without conducting the

Table 2.4: Security Threats in SDN Networks (1)

Targeted Level	Threats type	Caused by	SDN specific
Physical attack	DDoS attack in UpnP environment [122]	Illegal traffic generating when connecting heterogeneous devices for IoT services to enable information exchange between them.	Yes
Control plane	Controller hijacking [84]	Malicious application leveraging vulnerabilities of northbound API	Yes
Application plane	Threats from applications	Lack of authentication and authorization Absence of verifications or consistency checks of the installed updates	Yes
Control plane	Spoofing [184]	Compromised switch due to absence of TLS or switch authentication	No
Control plane	Man-in-the-middle attack between controller and switches [80, 99, 178]	The communication channel is not secured without TLS	No
Control plane	Denial of service attack to saturate flow table [100, 186]	Centralized controllers Flow tables limitation	Yes
Data plane	Fingerprinting SDN networks [98, 128]	Difference in time to process packets between traditional and SDN	Yes
Data plane	Information disclosure [184]	Difference in time to process packets which reveals information about the content of flow rules.	Yes

Table 2.5: Security Threats in SDN Networks (2)

Targeted Level	Threats type	Caused by	SDN specific
Data plane	Tampering attack using fraud flow rules [91]	lack of verification and consistency checks	Yes
Data plane	ICMP attacks, reset attack, SYN attacks, sequence prediction attack and DoS attacks.	Inheritance of TCP level attacks from traditional networks.	No
Data plane	Cache poisoning attack against the flow table and controller state [84]	Inserting forged packets	Yes
Data plane	Control Channel hijacking [109]	Compromised OF switch	Yes
Data plane	Freeloading [159]	Spoofing IP/MAC address to one of the hosts of an already established communication link.	No

initial man-in-the-middle attack [99].

OpenFlow protocol is analyzed using the STRIDE [18] threat analysis methodology [187]. The authors successfully executed Information Disclosure and DoS attacks.

On the other hand, although OpenFlow supports controller replication, the standards do not specify where to replicate controllers or how to select the master controller [81]. Without security standards for such replications, different security issues would arise as a result of this process.

Also, there are no *OpenFlow* security applications that can manage, secure, and authenticate the traffic between the controller and switches. Therefore, there are no mechanisms to detect malicious or malformed packets exchange.

Finally, another shortcoming of OpenFlow is that it supports proactive instead of reactive rule caching. Yet, the standard totally disregards how the incoming packets should be dealt with when some of their headers are hidden due to encryption.

The authors of [179] discussed the security, or rather lack of security, of the current SDN topology discovery mechanism, and its vulnerability to link spoofing attacks. While there is no official standard for the SDN topology discovery mechanism, there is a de-facto standard, which is sometimes informally referred to as Open Flow Discovery Protocol (OFDP). The basic security problem with OFDP is the lack of authentication of LLDP control messages. Any LLDP packet received by the controller is accepted and link information contained in it is used to update the controller's topology view. They presented and evaluated a flaw based on HMAC authentication and proposed a solution by adding a cryptographic Message Authentication Code (MAC) to each LLDP packet, providing both authentication and packet integrity.

The authors of [183] investigated the vulnerability of link discovery service in SDN controller. They also discussed the potential attacks on link discovery service. They took into consideration both the service (LDS) and protocol (LLDP) using *Packet_In* and *Packet_Out*. The vulnerabilities they tackled include fabricated LLDP injection and LLDP replay. Also, they researched the proposed defenses and tested them.

Another work on topology discovery and the associated security implications in SDN is presented in [188]. The authors discussed the possible threats relevant to each layer of the SDN architecture, and highlighted the role of the topology discovery in the traditional network and SDN. Also, they presented a thematic taxonomy of topology discovery in SDN, and provided insights into the potential threats to the topology discovery along with its state-of-the-art solutions.

2.3.4 SDN Specific Security Challenges

We present in Table 7 our classification of SDN attacks and the corresponding vulnerability exploited to realize it. A connection is drawn between the type of attack and its categorization as SDN-specific (second column) or inherited. We abbreviate the lack of TLS requirement in *OpenFlow* by L-TLS-RO. We can see that SDN has brought new threats that were not present in traditional networks. Moreover, the impact of threats that already existed may be higher and consequently may need to be dealt with differently. Considering the vast potential of security attacks identified in this Table, it is crucial to increase the efforts in order to find prominent solutions to these threats. Also, there is a gap between the challenges and the existing solutions. Research in SDN security has been focused on using SDN features to enhance security of networks. Little work has been carried out on exploring SDN security challenges and finding solutions to

Table 2.6: Classification of SDN Security Attacks

Security Attack	SDN	Inherited Vulnerability exploited
Scanning		X <i>L-TLS-RO</i>
Spoofing		X <i>L-TLS-RO</i>
MITM		X <i>L-TLS-RO</i>
Fingerprint	X	Dynamic reaction design
DoS	X	Centralized Controller Flow tables limitation
Information Disclosure	X	Dynamic reaction design
Malicious Administrator	X	Auth.
Data Plane ID Duplication	X	Auth.
Control Channel Hijacking	X	Auth.
Freeloading	X	Auth. and Integrity
universal plug and play (UPnP)		X Address allocation using UDP and without auth.
Tampering (fake rules & LLDP)	X	Lack of Auth.

these threats. There is additional potential in this area to analyze SDN standards, identify their flaws and correct their limitations. This needs to be applied between the different SDN layers.

2.4 SDN security Solutions

Different SDN security reviews have been published aiming to shed light on the amount of work done in this domain. Different security issues lead to different solutions which we discuss next. Tables 8 and 9 provides a general summary of the SDN security solution section.

Veena et al. analyzed security in SDN and included a comprehensive study on different security architectures such as *CloudWatcher*, *FRESCO*, *Procera*, *FlowSec* and others regarding used mechanisms, advantages and disadvantages [189]. The SDN security solutions are divided between application, control, and

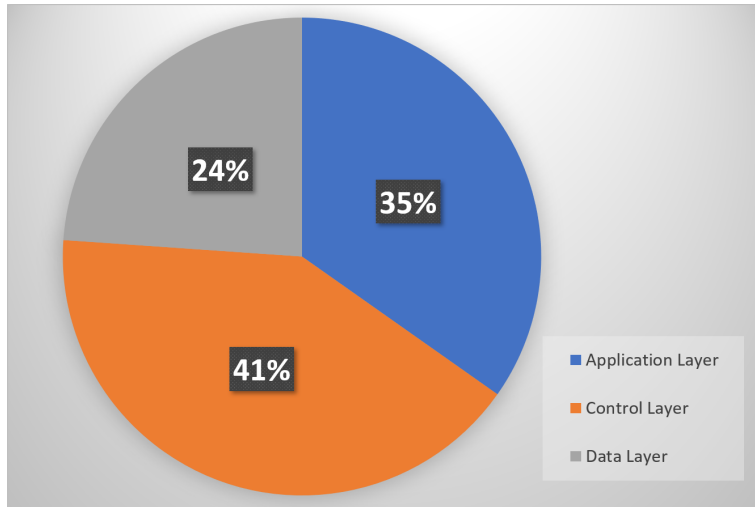


Figure 2.1: Research Projects Regarding SDN Security Solutions At Different Layers/Interfaces

data layer solution, as shown in the figure.

2.4.1 Application Layer Solutions

There are different security applications that aim to handle access control and to verify and debug SDN applications to stay consistent with updates and changes:

Payless [103], a low cost and efficient network statistics collection framework, is a flexible and extendable monitoring framework for SDN. The efficiency of *Payless* is due to its proposed scheduling algorithm for the collection of flow statistics. The algorithm uses a variable frequency flow statistics collection technique that maintains a low polling frequency for flows that do not have a high utilization of the link and a high polling frequency for flows that have a high utilization of the link. This approach maintains a balance between network overhead and the accuracy of the collected network statistics. The results showed that the algorithm achieved high statistics collection accuracy compared to the existing applications and the resulting overhead was 50% of the periodic polling strategy

Table 2.7: Classification of The-State-of-The-Art Security Solutions on Different SDN Layers/Interfaces (1)

layer/interface	Technique	Threats addressed
Application layer	Payless [190]	Unbalanced network overhead
Application layer	FRESCO [97]	Secure application development and integrated structure of Detection and mitigation security modules
Application layer	PemOf	Policy enforcement of access control
Application layer	VeriFlow [191]	Verification of network-wide correctness using forwarding graphs
Application layer	The stateless firewall application [126]	Enforcing ACL on Openflow enabled switches
Application layer	NIPS application	Accommodate SDN to IPS/IDS applications
Application layer	FlowGuard [107]	Verification of Security Policy
Application layer	PERM-GUARD[176]	Flow rule production-permission authentication scheme
Application layer	SD-Anti-DDoS [153]	Act against DDoS attacks
Application layer	Double hopping comm. (DHC) [192]	Solve the problem of sniffing attacks
Application layer	BroFlow[155]	Intrusion Detection and Prevention System based on Bro traffic analyzer and on the global network view of the software-defined networks (SDN)
Application layer	[130]	Anomaly detection through traffic analysis Deep packet inspection for attack signature recognition.
Application layer	CIPA[156]	Intrusion detection solution
Application layer	ReflectorNet	Module for threat detection
Application layer	OFX [144]	Balance between performance and deployment of OF switches
Application layer	SHIELD [147]	Suspect malicious applications

Table 2.8: Classification of The-State-of-the-Art Security Solutions on Different SDN Layers/Interfaces (2)

layer/interface	Technique	Threats addressed
Control Layer	SE floodlight	Implementing a North-bound API between Application and data levels Providing a Module for flow verification Inserting security policies on Packet_In and Packet_Out
Control Layer	[122]	Detect DDoS attacks based on the entropy variation of the destination IP address and probability calculation between different hosts
Control Layer	[123]	Zombie hosts detection, by keeping ingress port (source IP) mapping to detect spoofed IP addresses.
Control Layer	[150]	Distributed DDoS detection
Control Layer	[124]	DoS attacks detection using packet sniffing
Control Layer	[90]	Spoofing prevention by validating the source address of all arriving packets to the switch
Control Layer	SPHINX [112]	Security attack detection and prevention module in SDN. It is based on mapping (host: switch: port) in order to detect fake edges and trusting only messages from the controller to the switches.
Control Layer	[193]	Attack detection based on Graph model
Control Layer	Flow [193]	Verification of security policy
Control Layer	[85, 129, 194, 195, 196]	DoS and DDoS attacks detected using flow-based techniques
Control Layer	Avant-Guard (AG) [186]	The problem of DoS attack. It Integrates connection migration to stop the threats of the saturation attacks, and actuating trigger that introduced condition-triggered statistics information push capability in the SDN switches.
Control Layer	Rosemary [104]	Malicious applications that corrupt the OF controller
Control Layer	OPERETTA, [115]	Verification of legacy of a host
Control Layer	Mynah controller [116]	DPID problem mitigation
Control Layer	[127]	Provide trusted domain authentication between SDN controllers

Table 2.9: Classification of The-State-of-the-Art Security Solutions on Different SDN Layers/Interfaces (3)

layer/interface	Technique	Threats addressed
Data Layer	FortNox [91]	The problem of rule conflict and provisioning of role-based authorization
Data Layer	[108]	detect malicious application activities that are enforcing false rules into the switches
Data Layer	Flow checker [89]	Detection of inconsistencies in OpenFlow flow rules within the switches of the SDN network
Data Layer	DaMask-D [193]	Attack detection module based on anomaly detection
Data Layer	[158]	Adaptive anomaly detection based on the mechanism of flow counting to maintain the balance between performance and efficient detection.
Data Layer	[93]	Protection from scanning attacks
Data Layer	[197]	Scanning attacks identification
Data Layer	[109]	Control channel hijacking detection
Data Layer	[198]	Fingerprinting of SDN networks prevention
Data Layer	FlowMon [119]	Detection of compromised switches
Data Layer	[159]	A watermarking-based encoding technique to mitigate Freeloading

that was being used.

FRESCO [97] is an *OpenFlow* security application development framework with a primary target to facilitate the quick design and modular composition of *OpenFlow* enabled security services. The motivation for its design comes from the challenges in information deficiency, security service composition and threat response translation. Its strength is due to its ability to reprogram the underlying network infrastructure by efficient usage of *OpenFlow* to protect the network from emerging attacks. *FRESCO* currently supports sixteen modules, which form its basic processing units that can be joined together to form complex network security applications in comparison to applications that support simple halting or forwarding flows. It has a scripting API that allows application developers to write intrusion monitoring and detection algorithms as libraries, which can be joined with other modules to produce more complex and capable security applications. As a response to any intrusion detection, *FRESCO* based applications can reconfigure the network switches by changing their flow tables rules. Different applications, e.g. reflector net and cooperating with legacy security applications, have been developed and tested by the authors using *FRESCO* which provided promising results in terms of minimal overhead introduction when deployed in real networks.

PermOF is a permission system that introduces 18 different permissions belonging to different categories to control the privileges of SDN applications. They introduce an isolation mechanism by providing access control through a shim layer and separating applications from the kernel of the controller to prevent applications' direct access to the resources of the controller (resource isolation).

Khurshid proposed *VeriFlow*, which is a layer between the controller and the switch that checks network-wide correctness with low latency [92]. This layer

consists of a graph of the whole network, and as an update is inserted, it queries the graph for inconsistencies. Although this scheme showed low delays and good performance; however, traversing the whole graph each time introduces extra processing delays.

Other general security measures would be securing the applications themselves, through defending them from outside attacks and unauthorized access in addition to authenticating all connections with other applications. On the other hand, since an SDN controller allows access from third party applications through the REST interface, security measures and protocols should be established to differentiate between third party and user applications in terms of access control, resource sharing, and trust.

Similar to access control policies between applications, policies should be implemented to monitor the privileges given to each application to alter the switches data and install flows. Also, policies should control the application access to the data received from the switches to the controller to specify which applications can read this information.

Some projects have researched SDN from a firewall point of view. The firewall rules are maintained in a table in memory at time of initialization. A stateless firewall application [126], which is capable of enforcing ACL on *OF* switches, captures the packets and compares the header of each packet against the rule table and if a match is found, it stores the matching rule action and sends it to the forwarding module, which uses *PACKET_OUT* messages to allow or deny the forwarding of the packets. Besides the stateless firewall, the authors implement a stateful firewall which dynamically tracks the state of all valid TCP connections and passes any derived connections.

In a different approach, the switches act as a firewall based on firewall rules

installed by the controller. They improve on distributed firewall approaches by introducing the following two concepts: Selective Firewall Rule which removes redundancy by installing firewall rules for a directly connected host, and Reverse Flow rules on the source and destination such that when a traffic is dropped from a certain source to a destination, it is useless for that destination to send any traffic to a source which cannot respond, hence similar firewall rule is also installed on switches connected to the destination [139].

The authors pinpoint that the separation between data and control planes through SDN becomes inefficient in IPS/IDS applications because only packet header information can be sent to the control plane whereas IDS and IPS require the whole packet for analysis. They have designed a NIPS application that is the combination of three modules, Forwarding, Packet Inspection and Packet Handler. When a new packet arrives, the forwarding module captures it, creates flow rules and installs them on the switch to forward the traffic (whole packets and not only header) through a second dedicated network interface to the NIPS application. The NIPS application obtains the packets, analyzes and detects any maliciousness. If there is a match, an alert is raised and sent to the host, and if no match occurs the packets are returned to the network via the dedicated second network interface.

Seeber et al. created an IDS that has low false positive rate and correct flow rules installations to countermeasure malicious traffic [199]. Based on the user required flow rules, the SDN switches act as lightweight IDS systems by collecting traffic information and reporting it to the controller. The latter uses a white list, a black list, and a geolocation database, to compute a score called CVSS indicating the probability of malicious traffic and accordingly updates the flow rules. To protect users' privacy, suspicious traffic is forwarded to the user for

further analysis and the results dictate the update of the switches flow rules. For example, malicious traffic such as DDoS is redirected to a DDoS washing machine that can handle large data. Another approach is proposed in [200] where self-organized maps of Neural Networks classification method are used, which depend on 11 features extracted from the statistics of the flows that are collected from the OVS switches. A third approach on IDS was the work of Nguyen in [201], which proposed an anomaly-based Intrusion Detection architecture integrated as an *OpenFlow* switch.

The authors of [167] introduced detection as a service (DaaS), benefiting from network function virtualization (NFV) and cloud computing. They worked on two approaches to implement their architecture. The first (or any consequent) packet that arrives to a switch would be mirrored into a clustering system and will be forwarded normally towards the destination. The mirrored packet will be forwarded for analysis to a corresponding DaaS node, which decides whether the traffic is malicious or normal. If it tags the traffic as malicious, it will inform the SDN application running on top of SDN controller that the flow should be blocked. The two approaches differ in the cluster processing where the second one is distributed to multiple components for load balancing. Another cloud based solution for DDoS detection is [202], which discussed both auto correlation and alert generation methods. In addition to a countermeasure based on a selection mechanism which is built on analytical models and reconfigurable virtual network-based countermeasures.

The authors of [143] discussed the importance of SDN in network security as it brings new opportunities to defeat DDoS attacks in cloud computing environments. They reviewed the DDoS attacks on SDN and the various methods to guard against them. The research focused of the different types of DDoS attacks:

Application, control, and infrastructure layers.

The authors of [203] proposed a solution to overcome the limitations of robust firewalls since in OpenFlow-based networks, the network states and traffic frequently change. They introduce *FLOWGUARD*, a comprehensive framework that checks network flow path spaces to detect firewall policy violations when network states are updated. The firewall checks violations at the ingress switch of each flow, it also tracks the flow path and then clearly identify both the original source and final destination of each flow in the network. The authors of [176] proposed a solution to manage and authenticate flow rules between the application layer and the control layer. They presented *PERM-GUARD*, a flow rule production-permission authentication scheme that introduces an identity-based signature scheme to ensure that the controller can verify the validity of flow rules. Their technique was possible by enabling each application to digitally sign its flows, and an application on the controller can validate each flow.

The authors of [153] proposed SD-Anti-DDoS, a mechanism consisting of four modules, attack detection, attack trigger, attack trace back and attack mitigation. The attack trigger is used to quickly act against DDoS attacks and reduce the overload on the controller and switches. The DDoS attack detection method is based on Back Propagation neural network that extracts features from packet flows, while the trace back modules aims to find the path taken by the attack through querying the switches over the whole topology. Furthermore, the attack mitigation includes attack blocking through inserting blocking rules with high priority on the source switch to stop the attack traffic. Another DDoS mitigation approach is [204], which proposed SDN One-packet DDoS Mitigation (*SODM*) scheme with an *OpenFlow* switch functioning as a gateway to protect the inner server infrastructure. This mechanism is an anomaly mitigation method that

analyzes traffic statistics over a given observation duration to retrieve attack indicators. *SODM* checks if there is DDoS attack among a bundle of incoming flows.

The authors of [154] presented an SDN-based double hopping communication (DHC) approach to solve the problem of sniffing attacks. In this technique, the packets as well as the routing paths are changed dynamically. The traffic will be distributed to multiple flows and transmitted over different paths, and moreover, the data from multiple users is mixed. Although controller bottlenecks can occur in large scale networks, but it makes it difficult for attackers to obtain and recover the communication data.

The authors of [155] indicated that internal users in any network are the main causes of anomalous and suspicious behaviors that lead to network outages or leakage of sensitive information. The proposed *BroFlow*, an Intrusion Detection and Prevention System, is based on Bro traffic analyzer and on the global network view of software-defined networks:

1. Dynamic resource provisioning and traffic analysis mechanisms.
2. Real-time detection of DoS attacks through simple algorithms implemented in a policy language on top of the SDN controller.
3. Immediate reaction to DoS attacks, dropping malicious flows close of their sources.
4. One drawback is the lack of consideration for the switches rules establishment in the detection mechanism.

The authors in [130] presented an approach that integrates distributed network traffic Monitors and Attack Correlators supported by Open Virtual Switches

(OVS). These Monitors perform anomaly detection through traffic analysis, while the Correlators perform deep packet inspection for attack signature recognition. They both take advantage of the topology overview and information availability on both the data and control planes in SDN. These modules concentrate on network flooding attack signatures such as the TCP flood attack. Other work on handling flooding attacks is Distributed-SOM in [205], a bottleneck handler in large scale SDN networks. The approach integrates Distributed Self-Organizing Map (*DSOM*) system to OpenFlow Switches instead of using a standalone SOM.

The authors of [156] proposed *CIPA*, an intrusion detection solution which is deployed as a virtual network of an artificial neural net over the substrate of networks. Using low computational power, each programmable switch virtualizes one to several neurons. The whole neural net functions like an integrated IDS / IPS. This allows *CIPA* to detect distributed attacks on a global view. The same technique was tested over *OpenFlow* based SDN switches to create a complete neural network for intrusion detection and it was tested with different types of attacks

The authors of [111] presented a firewall detection and prevention technique *FireWell*, which is integrated as a module on top of an SDN controller, and it models firewall policies as formal predicates to validate, detect and prevent conflicts in firewall policies.

A *ReflectorNet* Module consists of two modules: Threat detection and Forwarding module. Reflector Module captures any newly arriving packets at the switch and sends the packet for analysis to the threat detection module which is a blacklist-based detector. When no threat is detected, the forwarding module generates rules to allow the forwarding of the flow to the target host. However, if a threat is detected, the forwarding module changes the source-to-destination

and destination-to-source flows to redirect the traffic to the honeynet [97].

To detect network anomalies, the authors have implemented an anomaly detection module; it contains a thread that periodically requests network statistics information from the switches and analyzes them using a detection algorithm to raise alerts if any anomaly is detected. The scan detector algorithm is based on an anomaly score algorithm which computes the anomaly score for each destination host whereas the DDoS algorithm keeps track of the number of packets and bytes to calculate the packet and byte rates from the collected data and compares them to a predefined threshold [206].

Current security functionalities in SDN networks suffer from performance issues either due to their implementation on the controller, which adds latency, or due to deployment issues on the switches, which requires redesign of the switches. OFX, proposed in [113], is an *OF* extension that implements the security functions as software by balancing between performance and deployment. An OFX enabled controller application can dynamically install the security functions as modules on the OFX software agents running on current SDN switches. But the CPU usage of switches didn't perform any better when under flood of pings.

An SDN security architecture was designed based on implementing policy-based security applications on the controller side and the use of security agents to enforce these policies in the switches in the data plane [164]. The approach extended the *OpenFlow* protocol to enable the communication of the security policies between the security applications at the controller.

A list of critical APIs was constructed in [147] that allows us to suspect whether an application is malicious or not. To determine the maliciousness, they analyze the order by which these APIs are called by the application. *SHIELD* consists of:

1. Source Code Tracer that analyzes the source code of the application and generates the control-flow graph (CFG).
2. Control Flow Analyzer that parses every node of CGI, sorts API calls, finds the critical sequences (flows) based on the list of critical APIs, and stores them in the SHIELD database.
3. Result Manager that generates the behavior graph based on the critical flows stored in the database.

2.4.2 Control Layer Solutions

Securing the control plane starts by guarding against malicious or faulty apps. Enhanced (SE floodlight) was implemented to secure the control layer. It implements a North-bound API between Application and data levels, provides a module for flow verification, and inserts security policies on *Packet_In* and *Packet_Out*.

An effective security approach would implement a security monitoring application that tracks security related events, and making them available to other security modules to analyze them and take action accordingly. This approach divides the security load among the controller applications and minimizes the overhead by minimizing the number of applications that capture or request specific types of traffic. This provides a single security related database available to different applications.

SLICOTS is proposed to mitigate TCP SYN flooding or what is known as control plane saturation attack. *SLICOTS* takes advantage of the dynamic programmability nature of SDN to detect and prevent attacks. It is implemented on the controller; it surveils ongoing TCP connection requests, and blocks malicious hosts. Upon receiving a new TCP packet, it first checks the type of the packet;

if the packet is SYN, it extracts required information (source MAC, destination MAC, source TCP port, and destination TCP port). This information is used to track and analyze incoming TCP connections regardless of the source IP to prevent IP spoofing. Nevertheless, *SLICOTS* is not applicable to networks in which the SDN controller installs forwarding rules proactively. Moreover, *SLICOTS* does not consider flow rule aggregation and installs two paths between the client and server for each individual TCP session, i.e., one for forward path and the other for backward path.

Xiaofeng et al. presented a security controller-based software defined security architecture in which a modularized security controller is placed in the control plane and interacts with other components through APIs [170]. The security controller completes the control function for security services together with the SDN controller. The architecture is based on security information collection, security policy resolution, security device management, security resource scheduling, service chaining and others. The module was tested for these services under different security situations.

Figure 5 summarizes the different DDoS detection and mitigation solution at different SDN planes.

2.4.3 Hybrid Controller Architecture Solutions

Two general types of controllers exist: Proactive controllers which set rules before the flow arrives and Reactive controllers which set the rules after the flow arrives.

Statistics show that DoS and DDoS attacks are the most used attacks to easily strike active servers, and researches have shown that the SDN controller is a perfect target for these types of attack and with drastic impact. Since an SDN network cannot tolerate the failure of the controller, many researchers have

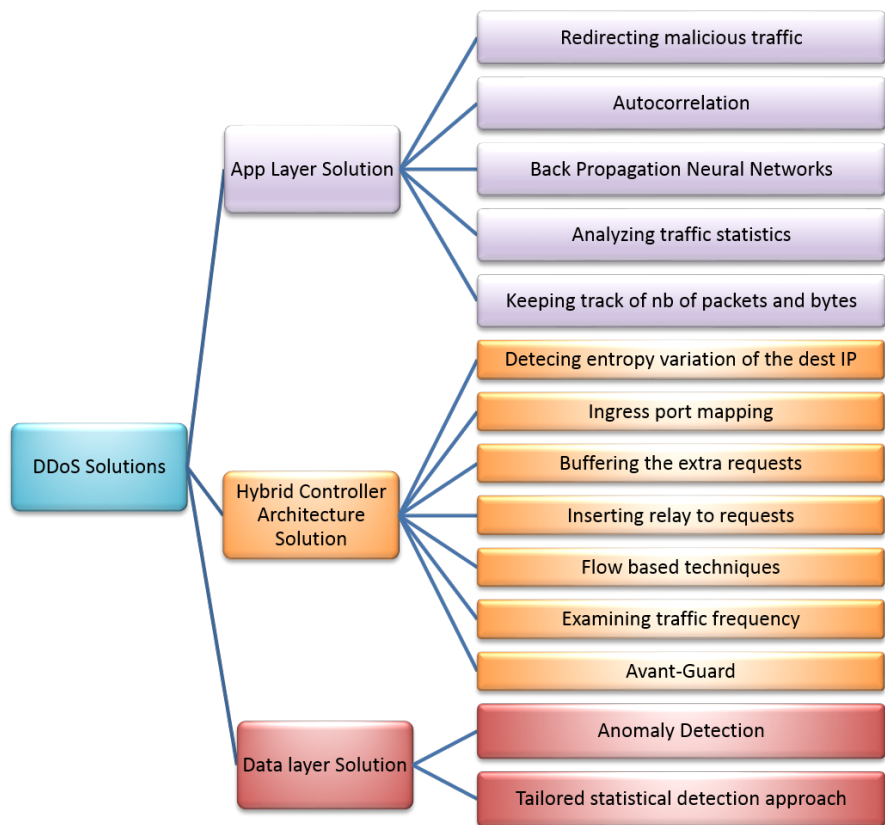


Figure 2.2: SDN Based DDoS Detection and Mitigation Solutions

designed different solutions to prevent DoS and DDoS attacks and mitigate their effects.

Mohammad Mousavi et al. showed in [122] how DDoS attacks can exhaust controller resources and provided a solution to detect such attacks based on the entropy variation of the destination IP and probability calculation between different hosts. The approach assumed that all hosts in the network have relatively equal traffic rates, and the solution was based on forwarding all traffic to the controller for analysis. This introduces additional overhead on the controller and hence makes it more vulnerable to future DoS attacks especially that no prevention technique was proposed.

Hyo-Bin Bae et al. proposed in [123] a model for detecting zombie hosts used for launching the DDoS attacks, by keeping ingress port (source IP) mapping to detect spoofed IP addresses. Again, this approach is based on redirecting the traffic to the controller to calculate the occurrence frequency of each packet, and only targeting a small angle of such attacks.

Kostas Giotis et al. investigated in [150] the applicability of inserting an *Open-Flow* middlebox to detect distributed DDoS attacks. The approach proposed a module for flow-based monitoring statistics, harvested from the edge router and a second module that analyzes the dataset, searching for abnormal traffic patterns. As discussed before, neither redirecting all the traffic to the controller nor depending on middleboxes alone is efficient enough to secure the network.

Attack mitigation is concerned with minimizing the impact of an attack rather than preventing it completely. Sandeep Singh et al. proposed in [124] a packet sniffing scenario to detect specific types of DoS attacks. The mitigation is based on buffering the extra requests so the server is not overloaded, and if high traffic still exists, the source IP is blocked. However, this approach inserts network

delays and doesn't hold as a solution in case of IP spoofing.

The authors of [90] proposed a design to prevent users spoofing and to validate the source address of all arriving packets to the switch. When an incoming packet does not match any rule, it is forwarded to the controller to validate whether or not that source address matches a valid flow. A rule is created to stop the traffic if spoofing is identified.

S. Lim et al. proposed in [194] a detection tool based on analyzing request traffic at the server and triggering a redirect order to another server when an attack is detected. The redirect alert forces the server to change a specific service to a different IP address and notifies the connected hosts. Such a tool will only delay the attack and mitigate its effect, in addition to redirecting the packets to the controller to keep track of the requests.

Sungheon Lim et al. suggested in [125] a scheme for modifying the controller model such that the single request processing queue at the controller is subdivided into k queues, each of which corresponds to a single switch. Then, the controller serves these queues with a scheduling order. Here, the suggested scheme aims to limit the effect of DDoS attacks by inserting a delay to the requests.

Attack prevention is a very interesting and useful feature in security, whereby not only the attack effects are mitigated but also attacks are prevented from taking place in the first place. Mohan Dhawan et al. proposed in [112] *SPHINX*, a security attack detection and prevention module in SDN. It is based on mapping (host: switch: port) in order to detect fake edges and trusting only messages from the controller to the switches. Also, the number of packets/flows should be consistent through a specific path. The number of packet flows/sec should not exceed the normal limit of the network by more than a threshold. Although the proposed scheme was secure enough, yet the prevention mechanism, which was

based on blocking the source IP, was not implemented and the detection still uses the normal control plane.

Bing Wang et al. proposed in [193] an attack detection system that stores known traffic patterns as a relational graph between patterns and their labels to distinguish between normal and abnormal traffic. The prevention technique was based on blocking the source IP of the attack. This model followed the same strategy of sending all packets to the controller via a path based on the graph model.

K. Giotis et al. proposed in [102] *Flow*, a technique to analyze flow statistics in order to reveal anomalies triggered by large scale malicious events such as DoS attacks. The proposed module acts in three stages: data collection, anomaly detection, and anomaly mitigation. The module is still based on forwarding all traffic to the controller.

DoS and DDoS attacks are mostly detected using flow-based techniques [85], [129], [194], [195], [196]. These techniques are based on extracting some traffic flow features and analyzing them. The variation is in the number and type of features chosen in the analysis. They use Self Organizing Maps (SOM) to categorize the traffic as normal or malicious. However, detection tools installed on the controller without proper collection of network traffic could overload the communication between the data and control planes. In [129], a hybrid of hard threshold and Sugeno-type Fuzzy Inference system is used. Other possible countermeasures propose redundancy to guard against DoS attacks. In [94], the authors discussed the number and placement of redundant controllers. Their technique reduces the timeout of flow table entries and drops infected packets. In [87], the authors proposed a scheme that examines the traffic frequency; the controller recognizes a DDoS attack if the frequency exceeds a specific threshold, and thus begins

dropping packets. In [149], Controller Protection Protocol (CPP) was proposed to solve the imbalance of work between a host (simple packet transmission) and controller (route calculation, flow rule insertion, processing. . .) that is leveraged by attackers to perform DoS attacks. They introduce a pseudo-random proof-of-work (POW) that must be computed by the host, sent with the first packet, and verified by the controller to prevent the drop of traffic, thus increasing the computational cost at the attacker, which makes DoS attacks computationally expensive. Only few researchers took the work a step further to trace back the attack to the original switch and proposed techniques to prevent the attack from that source, hence minimizing the number of rule insertions into the network switches. An example of such work is found in [131], which is a non SDN specific puzzle based scheme that verifies the requester and identifies the true IP of the source.

Avant-Guard (AG) [186] integrates connection migration to stop the threats of the saturation attacks, and an actuating trigger that introduces into the SDN switches the push capability of condition-triggered statistics information. Due to the high *OF* messages that are exchanged between the data and control planes, the control plane can be exploited by performing DDoS and scanner attacks. This is possible because an external input stream is the one that initiates interaction between the data and control planes, and hence an attacker may exploit this situation by producing many new unique flow requests to quickly overwhelm the control plane. To solve this problem, connection mitigation is introduced, which is an extension of the *OF* data plane. Enabling the control plane to quickly detect and respond to changing flow dynamics within the data plane is the second challenge. So far, *OF* only allows applications to pull or poll statistics information from every switch, which is not enough for monitoring applications that need

switch statistics to track, detect and respond to malicious attempts. Actuating Trigger is added to the existing data plane services to address this challenge. The control plane adds triggers into the data plane to register asynchronous call backs and to insert conditional flow rules, which are activated when a trigger condition is satisfied. *Avant Guard's* actuating trigger module allows the delivery of packet payload to the control plane. This is possible through its ability of defining conditions that involve header fields that it suspects and wants to investigate; these are passed from the control plane to the switch. The switch reports all condition matching packets to the *Avant-Guard* application. This was not possible in the absence of this application because the data plane sends only network header information to the control plane.

Rosemary [104] is a new NOS that creates an independent sandbox for each application. Its aim is to prevent applications from performing malicious operations that can corrupt the *OF* controller as is the case in many controllers. *Rosemary* introduces the idea of micro-NOS, where each application is developed within an independent instance, where it is monitored and controlled with resource utilization controls. The authors evaluated the absence of currently existing controllers' robustness and security by using an application that silently crashes the controller through an exit function, an application that leads to out of memory error and an application that changes the internal data structures of the controller instance. To mitigate these types of attacks, *Rosemary's* architecture is based on separation of applications from the NOS kernel, compartmentalization of the NOS kernel modules, control of resource utilization by applications, access control and authentication, monitoring the NOS, safely restarting the NOS and performance considerations.

The authors of [137] proposed an adaptive technique to balance between com-

plexity of detection policies and monitoring overhead of the controller. They defined a flow counting range that is based on a linear prediction formula. They capture aggregates based on IP prefix, and then for each aggregate, they calculate the flow counting value and check if it lies within the prediction range. If it is less than the range, then they decrease the collection interval based on a header space divider. If it is greater, then they decrease the number of samples collected to reduce the monitoring overhead by ignoring less important entries. However, they did not clearly define the header space divider and sampling rate.

Kim et al. described a framework to secure network resources through SDN-based security services using Network Security Functions (I2NSF) with the aims of providing fast reaction to new attacks, autonomous defense from network attacks and network-load aware resource allocation [206]. They introduced a Security Controller that has an interface, called Capability Interface, (i.e. YANG) within the SDN application layer to call the functions of security applications, and another interface, called Service Layer Interface, (i.e. REST CONF) with Clients & Application Gateways Layer (lying above Security Controller) to define high level security policies. They present the centralized firewall system and the centralized DDoS-attack mitigation system that is based on their framework, since traditional systems suffer from lack of flexibility and administration costs and hence arises difficulties in autonomous operation, management and fast configuration. However, the proposed framework has not been implemented yet and the details of the algorithms were not provided.

In *OPERETTA* [115], the controller verifies whether a certain host is legitimate or not. When a host wants to establish a TCP connection, it sends a SYN packet. If a rule for that particular user does not exist in the SDN switch, it is forwarded to the controller, which replies with a SYN-ACK and expect an ACK

from the source. If an ACK is received, a RST message is sent by the controller and flow rules are installed on the switches, hence allowing a user to recover a TCP connection by re-performing the TCP Three-Way handshake but now with the destination. However, if the ACK is not received and the SYN packets are sent multiple times exceeding a well-chosen counter threshold (tradeoff between efficiency and performance), the MAC is blacklisted for a certain time interval by deploying rules on the switches that drop any SYN packet received from the matching blacklisted host. However, if MAC is spoofed along the IP address, the algorithm will fail because it uses a counter of the number of SYN packets received from a certain MAC address and hence blocks it when it exceeds a threshold. *OPERETTA* performs poorly when there are no ongoing attacks compared to current controls with no SYN-Flooding mitigation (900% increase in delay).

To mitigate the DPID problem, the *Mynah* controller was developed in [116] and it consists of: Extended Vendor *OF* Library, Encryption modules to encrypt and decrypt using public or symmetric algorithms, *Mynah* Controller module and *Mynah* Switch Module. To perform data plane authentication and prevent MITM and DoS attacks, they extended the switch-to-controller Echo Request message to contain a session key that includes switch DPID, a timestamp and a sequence number. *Mynah* handles DPID duplication by adding a modifier to the same DPIDs to distinguish them from one another and maintain uniqueness. As for preventing DPIDs, if a session key fails to pass validation, the corresponding connection is rejected. However, if it is valid and another DPID is already registered in the controller, then this connection is rejected. To handle DPID duplication, apps using DPID should be aware of the changed notation in the DPIDs.

Zhou et al. implemented in [127] a new protocol to provide trusted domain authentication among SDN controllers through extending the controller with Trusted Measurement Module (TMM) and Controller Communication Module (CMM). Eight messages are exchanged between the two controllers to authenticate them through controller platform certification and controller software certification. However, it is not clear how certain parameters are exchanged.

ATTAIN is an attack injection framework for *OpenFlow*-based SDN architectures [181]. *ATTAIN* consists of an attack model, an attack language, and an attack injector. The attack model is defined for relating system components to an attacker's presumed capabilities to disrupt the control plane state. The attack language is defined for writing control plane attacks, subject to the attack model. The attacks are modeled in stages, called attack states, and represented graphically. Each state consists of a set of rules governing conditions under which actions are taken in an attack. Both the attack model and language are implemented using an attack injector, which interposes *OpenFlow* control protocol messages in the network's control plane to affect attacks and allow practitioners to understand how such attacks manifest in controller, switch, and end host behavior.

The authors of [207] introduces a new approach to identify attacks on SDNs that uses: 1) similarity with existing attacks that target traditional networks, 2) an inference mechanism to avoid false positives and negatives during the prediction process, and 3) a packet aggregation technique which aims at creating attack signatures and use them to predict attacks on SDNs. Their approach used regular labeled flows to analyze different samples of incoming *OpenFlow*-based flows that are generated by SDNs.

DELTA is a security Assessment framework for SDN [166]. It focuses on au-

tomating and standardizing the vulnerability identification process in SDNs. The authors developed a security assessment framework, *DELTA*, that re-instantiates published SDN attacks in diverse test environments. Then, they enhanced their tool with a protocol-aware fuzzing module to automatically discover new vulnerabilities. Their evaluation successfully reproduced 20 known attack scenarios across diverse SDN controller environments and discovered seven novel SDN application mislead attacks.

2.4.4 Data Layer Solutions

FortNox [91] is a security enforcement kernel which is an extension of the *NOX OF* controller providing role-based authorization and security constraint enforcement. It has a live rule conflict detection engine, which intervenes with any *OF* rule insertion request and analyzes the conflict before enforcing it. The rule conflict analysis is performed by an Alias Set Rule Reduction algorithm that can detect rule contradictions even if the set action, which can rewrite a packets header, is used. The authorization of an *OF* application is determined by the role-based authentication and to ensure the integrity of the mediation process, it uses the principle of least privilege. To resolve rule conflicts, every application signs its flow rule insertion request by using digital signatures representing its authorization role. When a conflict is detected, *FortNOX* decides to accept or reject the new rule based on the authorization level (administrator, security application or non-security application) of the new rule. The results of the experiments performed by the authors show that the conflict evaluation overhead is in the worst case linear with respect to the number of rules.

To mitigate fake rule injection, *FortNox* [91] prevents dynamic flow tunneling by comparing the newly inserted candidate flow rule and the existing one to detect

conflict between them. The rules are converted into a representation called alias reduction rules (ARR) and the analysis for conflict is performed on these ARR. After obtaining the alias set rules, validity checks are performed between the candidate ARR cRule and the active flow ARR fRule.

For dynamic flow detection, the authors in [108] proposed the creation of an adjacency matrix according to the topology of the network, hence containing the connections that are available between any two nodes in the network. To detect malicious application activities that are enforcing false rules into the switches, the authors have developed an algorithm that checks if the information saved in the adjacency matrix matches the policies of the firewall. If there is a match, then there is no malicious activity, however if there is no exact match, then an attack alert is raised. The authors do not mention the location where the adjacency matrix is stored, which cannot be on the controller because it is assumed to be running a malicious application that modifies switch flow rules. Additionally, the size of the adjacency matrix increases in the order of $O(n^2)$ as the number of nodes in the network increases thus adding delays and degrading performance.

FlowChecker [89] is a configuration verification tool that aims to detect and identify inconsistencies in *OpenFlow* flow rules within the switches of the SDN network. *FlowChecker* can function as an *OpenFlow* application or a standalone master controller in order to analyze, validate, and enforce *OpenFlow* end-to-end policies at run-time.

In [208], a framework is presented to enhance security in SDN datacenters through the integration of security middle boxes such as IPS or FW to block attackers. The authors depended on third party security agents known as middle boxes to inject into the controller relevant security analyses of the network to enable taking different security measures. This method showed good security

results, however, it has been proven to underperform when dealing with delay sensitive issues.

DaMask-D is an attack detection module based on anomaly detection, which has graphical probabilistic inference model as its core. The detection uses this graph system to differentiate between malicious and normal traffic. What makes *DaMask-D* an advanced anomaly detector compared to existing systems is the presence of automatic feature selection that allows the data to decide the relevant features based on a large set of candidates and efficient model update, which copes with the problem of dataset shifting. When a packet arrives at a switch, it is checked to which virtual network it belongs to. If it is an existing flow, the flow statistics is updated. If it is a new packet, then *PACKET_IN* message is sent to the corresponding controller, which generates a new flow rule and sends the flow statistics information to the anomaly detection module for further processing and analysis. If *DaMask-D* detects the packet as malicious, an alert is generated signifying a DDoS attack and is sent with necessary packet information to the reaction module for DDoS protection rule generation. *DaMask-M* is an attack reaction system that consists of a mapping table matching alerts with corresponding countermeasure to be taken. It provides an API that allows the application users to define their own countermeasures to different DDoS attacks. The three operations that are supported for defense are drop, modify and forward. Once the mapping between the attack and the corresponding countermeasure is done, the controller generates policies according to that decision, which are accordingly enforced on the switches to mitigate the DDoS attack. The flow table in a network entity has limitations. The authors of [186] described a mechanism where some minimal intelligence is added to the data plane devices to resolve DDOS issues.

Furukawa et al. proposed a new SDN architecture to prevent MITM attacks by implementing a new address translation method that does not require end-host security procedures and does not notify the receiving host of the address information of the sending host. The new architecture proposes a change in SDN switches by adding security related functions such as fragmentation, duplication and shuffling, and an application that allows the users to choose these parameters to control the level of security that they require. The address translation method, based on shared key negotiation, allows the SDN switches to shuffle and fragment the encrypted packets and transmits them on different routes to prevent eavesdroppers from collecting the data and reassembling them [136].

A virtual SDN network is modeled as a graph in [209], where the nodes are the components and edges represent data flows. They presented two functions where the first provides the risk level of the component, while the second represents the level of support of confidentiality, integrity and availability. A system is called secure when there are no interactions among nodes that have high risk and nodes that have low level of security (confidentiality, integrity and availability), else the need of an interface called *SecS* (Security Server) is required. But they did not indicate how to calculate these functions.

A method to detect disobedient forwarding in the flow table by compromising a switch was designed in [210]. To enhance detection efficiency and minimize additional network traffic, the authors reduced the number of necessary detection packets by aggregating the flow entries. This method selects the flow entries whose match fields can compose a valid packet from multiple switches. The switches, on which the entries are, form a path that allows the packet to travel through for rapid detection.

Garg proposed an adaptive anomaly detector that is based on the mechanism

of flow counting by maintaining balance between performance and efficient detection. The traffic obtained from the network is initially aggregated based on network prefix, and then divided into smaller chunks based on network requirements. A linear prediction formula is then used to calculate the prediction value of the aggregates (i.e. based on packet size). For incoming traffic, the aggregate value is calculated and compared with the predicted value, if it is greater than the predicted value, then more aggregates are needed with smaller chunks to be calculated to detect anomalies correctly. However, if it is less than the predicted value, then there are no anomalies and hence size of aggregates will be increased. This mechanism allows a balance between overhead and performance of the detector [158].

Dmitry et al. focused on the confidentiality in SDN security [142]. They proposed a model that makes reason about confidentiality and checks if confidential information flows do not interfere with non-confidential ones. The model is based on mapping the network host addresses into low and high security groups and includes the security type in the matching process before taking any action. This is to insure the separation between different security levels of traffic.

The authors of [93] proposed a random host mutation method to protect from scanning attacks. A virtual IP address is assigned to each host for data transmission and is randomly mutated, which hides the real IP addresses from intruders and defeat their scanning threats. In [197], the authors developed an SDN application for identifying network scanning attacks. First, the algorithm keeps a queue of new connection initiations that do not receive a response. Every time one of these connections times out with no reply or gets a TCP RST, the algorithm removes it from the queue and increases a certain probability ratio of the host that started the connection. Otherwise, when a successful response is

obtained, this probability ratio is decreased. When the probability ratio for a host surpasses a certain threshold, it is declared as infected.

One solution for information disclosure attacks, is to implement randomization in the *OpenFlow* protocol. Randomizing the timeouts of the established flow rules would prevent the attacker from creating a clear view of the network state. Another possible solution could be to establish proactive flow rules rather than installing rules in response to a new request.

To detect control channel hijacking, the authors of [109] proposed a detailed analysis of the network through latency measurement and active traffic shaping. Additionally, if an honest controller suspects the presence of a compromised switch, it can start delaying classes of data-layer traffic and analyze the effect on the latency of messages that reach the compromised switch to verify that it has been compromised.

To prevent fingerprinting of SDN networks, the authors of [198] proposed the addition of delay on the first couple of packets of old flows by modifying the selection logic of the SDN switches' group table. This increases the difficulty of fingerprinting through measuring packet-pair dispersion and RTT features because the attacker cannot identify if this delay is introduced naturally when new flow packets are sent to the controller or by a countermeasure against fingerprinting. More naïve and costly countermeasures delay every received packet or delete flow rules and reinstall them through the controller every time a *PacketIn* message is received.

FlowMon [119] detects switches that have been compromised and are acting maliciously, by analyzing the port statistics and actual forwarding paths that are obtained periodically from the switches. It identifies a switch as malicious when it intentionally drops packets and/or routes flows to wrong ports. To detect if

packets are intentionally dropped on switches and/or links, it computes the drop rate based on the packets sent and received on a port and/or packets transmitted from the first switch and received by the second simultaneously and then compares the values with predefined thresholds. To detect packet swappers, they compared the expected output port obtained from the controller with the actual output port obtained from the statistics information.

To mitigate Freeloading, a watermarking-based encoding technique is proposed in [159], which provides signature of ownership of flow rules and the origin of packets, thus providing integrity between two communicating parties, where the secret configuration information can be provided by the controller during flow-rule establishment. The receiver verifies every packet by decoding it and dropping any packet that does not contain the required established watermark.

The authors of [211] proposed to add a mismatched packet cache module into the *OpenFlow* switch which can temporarily cache the packets that do not match the flow table. Also, they applied a mechanism for separating the header and payload of packets in the cache queue once the switch detects that the volume of cache queue exceeds the threshold of the cache size. In addition, the proposed technique could classify the packets' headers and sends it in an alert message to the SDN controller for further processing. The authors of [162] took advantage of software defined networking for security policies enforcement. They introduced a 2-layer open flow switch architecture to implement security rules, considering the flow table size limitation and load balancing between switches. Also, they proposed a safe way to configure these switches and proposed an update scheme in a tree model.

Detecting and mitigating DoS attacks against the Data plane was the topic of the research in [172]. The authors discussed DoS attacks against the data plane

and their impact. They proposed a tailored statistical detection approach as well as a lightweight countermeasure. The detection process depends on localizing the fixed header fields of the attacking flow. The approach uses a table of counters with the different header fields as columns. The table is regularly inspected and the maximum entry would be large in case of an attack. The counter measure is based on installing low prioritized dropping rules on the switches along the attack path.

2.4.5 OpenFlow Proposed Solutions

Securing networks using *OpenFlow* is an active area of research. *FortNOX* [91] was developed to deal with malicious or incompatible *OpenFlow* applications. It is as an extension to the *NOX OpenFlow* controller where role based authorization and constraints are added to rules sent to switches. In a similar context, *FlowVisor* [212] acts as a transparent proxy between controllers and switches to put restrictions on the rules created by controllers. It rewrites the rules to limit their influence to a specific “slice” of the network. Both, *FortNOX* and *FlowVisor* focus on limiting untrusted controllers or applications running on controllers.

Another approach is aggregating flow rules to decrease the flow table size and hence reduce the possibility of information disclosure and DoS attacks.

The authors of [213] argued that knowing reactive rules, attackers can launch DoS attacks by sending numerous rule-matched packets which trigger packet-in packets to overburden the controller. They presented a novel method (*INSPIRE*) that discovers the flow rules in SDN from probing packets. The approach was based on evaluating the time delay from probing packets, classifying them into defined classes, and inferring the rules. This method involves three relevant steps: probing, clustering, and rule inference. First, forged packets with various header

fields are sent to measure processing and propagation times in the path. Second, it classifies the packets into multiple classes using k-means clustering based on packet time delay. Finally, the algorithm finds common header fields in the classes to infer the rules.

2.4.6 General Solutions

Information disclosure is one of the attacks that was inherited from traditional networking. In SDN, the solution becomes easier due to flexible programmability of the switches and their actions. Some proactive strategies suggested proactive flow rule establishment to randomize the actions of the flows in order to alter the statistical sequence for the attacker. Another method is the randomization or increase in the variance of measurable response times which can increase the statistical uncertainty for the attacker. A third method is attack detection; since any attack that is based on timing analysis is likely to exhibit a distinctive, repetitive pattern, the controller security applications can use these patterns to detect such attacks.

The aim of Network Security Virtualization (NSV) [214] is to benefit from the pre-existing middle boxes by maximally utilizing these resources and to use these security resources to provide dynamic and on-demand security services to the users, without the user's knowledge about the actual location of the devices. With NSV, tenants do not need to be security experts and do not need to worry about information, operating security functions and underlying network architecture. The solution provided by the authors does not require the re-architecture of the middle-boxes.

There are two techniques to fulfill the availability of a NSV:

1. Transparently and automatically redirecting flows to the desirable security middle-boxes when needed, regardless of the actual location of the middle-box.
2. Enable network security response functions on a network device.

To test the feasibility of NSV, the authors implemented a prototype for cloud networks calling it *NetSecVisor* which benefits from the cloud operators pre-installed security devices to provide dynamic, transparent, and on-demand security services to cloud network tenants. In addition, it enables basic response functions such as network isolation for network devices.

The proposed framework in [132] is based on the assumption that collaboration exists between the ISP and the customer, in addition to a controller at the customer, besides the one at the ISP. Every flow that enters the ISP network is tagged by a tag generation algorithm running on the access switches generated by the authors to reduce overhead on the controller and impose end-to-end policy. *OF* switches update the controller with traffic statistics information, hence any suspicious behavior is alerted by the detection engine at the customer controller. This leads to the generation of new *OF* rules to be inserted at customer routers. Furthermore, the customer controller passes this alert and mitigation requests to the ISP controller, which redirects the suspicious flow to a specialized middle-box for further processing by a mitigation algorithm developed by the authors, which if detected as malicious, will be redirected to a sink.

Identity-based Authenticated Key Agreement (IB-AKA) protocol has been introduced in [114] to secure the southbound and east/west bound communication channels in distributed SDN rather than TLS because it is simpler, better in performance and requires less memory for public key storage. IB-AKA is used to

create the symmetric session key that secures SDN communications. For south-bound communication, the PKI is implemented by the controllers and hence for large networks, it requires large memory to store the keys that are periodically renewed.

Current distributed management approaches suffer from latency, appliance and network overload due to the implementation of policy configuration on controllers and policy resolution and enforcement on middle-boxes (FW, IDS, IPS). To solve these issues, the authors implemented *EnforSDN* in [118]; a new management design that separates policy resolution from enforcement by centralizing resolution in the middle-boxes while the enforcement at the controller. Initially, the controller installs the flow rules on the switches to forward new flows to middle-boxes. The middle-box analyzes the first couple of packets and then takes a resolution decision that is sent to the controller, which in turn enforces new policy rules on the switches to either drop or forward the packet to the destination. This mechanism removes the responsibility of this flow from the firewall compared to distributed management where the firewall is responsible for enforcing the rules on each packet after processing it. It is important to note that an authentication scheme is required between firewall and controller because a MITM attack could lead to installing wrong rules and diverting traffic from the required destination.

Shin et al. proposed the utilization of middle-boxes found in traditional networks to provide virtualized security functions to cloud networks through SDN [214]. Based on the security policies demanded by the cloud tenant, the network traffic is forwarded by installing correct flows rules, through an optimal path containing SDN switches to the corresponding middle-box (IDS, IPS, Firewall). This allows the dynamic detection of any malicious traffic and thus the isolation

of any infected nodes by installing the appropriate rules onto the switches connected to the infected machine. They implemented *NetSecVisor*, a prototype that provides these functionalities through 5 modules: 1) Device and Policy Manager that consists of available registered security device information and translates the security requests of tenants to security policies; 2) Response Manager that generates the corresponding flow rules based on the input of security devices; 3) Routing Rule generator that generates the optimal route, 4 different algorithms, to the security devices based on the security policies required by the user; 4) Flow Rule Enforcer that installs the flow rules generated by response manager on SDN switches; 5) Data Manager that temporarily holds the packets of an ongoing flow until the security middle-box outputs the result. However, the proposed routing algorithms add relatively high overhead for small size networks.

The authors of [140] proposed a controller independent web-based northbound interface that provides TLS certificate-based authentic, confidential and accountable secure communication link between the controller and applications. The applications use the REST-like interface that uses standard HTTP to connect and register with the web-based controller service and thus use its resources according to the agreed-upon access policies, where permission sets are mapped to allowed resources. Based on the acquired permission, an application can register for the appropriate event listeners to receive the required information.

BGPSecX [146] is an SDN based architecture that provides secure BGP. Different IXPs can use *BGPSecX* to create a secure communication channel. *BGPSecS* consists of ROA verification using RPKI, prefix filtering of malicious announcements, and it validates routing information by queries between IXPs. However, RPKI coverage is small and prefix filtering does not protect an AS itself.

A solution for the malicious administrator problem, presented in [110], consists of leveraging a threshold voting protocol to confirm that the network has one configuration. Another approach could be to allow the selection among many configurations created independently by each of the administrators.

UNISAFE [148] is software switch architecture that leverages the programmable nature of switches and extends them by implementing flexible security services. The proposed architecture is extensible because the identifiers of security functions are saved in a table, which can be extended by any developer who wants to implement a new function. When a certain security function/action is desired by an application, the identifier corresponding to that action is set in the action field of the flow rule. Hence, every security action has its own detection logic and data structures to analyze and detect a specific attack, and alert the user. Flow rules that require the same security function are differentiated by cluster ID, which is used as a key to a hash table to refer to a separate data structure in the detection logic. Multiple security actions can also be clustered as a single action in a single flow rule, where any flow rules that have the same security action and cluster ID share the same data structure (resources and statistics information/data status).

As for processing and delay due to copying lengthy rule arguments during packet arrival, they implemented a direct private connection between the user-space daemon and *UNISAFE* module in kernel-space through a *UNISAFE* private channel to copy the arguments directly rather than waiting for the arrival of a matching flow packet to trigger the copying process.

CINDAM [120] is a deception and attack mitigation mechanism that uses the SDN architecture and deceives attackers by periodically changing and eluding the topological view of the network for each host. Only *CINDAM* itself will always be aware of the real topology of the network and thus will provide DHCP, DNS,

NAT, ARP and MPLS services to have a normal functional network (receive and transmit packets) noting that each host has a different view of the network. Adding and/or removing resources (topological change), constantly changing IP addresses and adding honeypots, can invalidate network history, that is highly used by the attackers, and thus preventing them from detecting honeypots in the network.

OpenSec [151] is a security framework that simplifies policy enforcement for network operators by abstracting the configurations and description of the network policies (through *OF* rules) by providing a user-friendly interface and human-readable description language. To prevent the controller from overloading, the user security services are performed by processing units (specialized security middle-boxes) that analyze the automatically redirected traffic obtained through the *OF* rules specified on the switches. Furthermore, *OpenSec* provides automated reactions, alert or quarantine or drop through new *OF* rule installation, to any detected suspicious traffic.

The authors of [160] proposed an intrusion prevention system for SDN cloud networking. It works on mechanisms such as botnet/malware blocking, scan filtering and honeypot. Malicious traffic is isolated because bot-infected VMs are removed efficiently from the private cloud. The scanning behavior can be filtered at a very early stage of prevention, making the VMs less exploitable. A honeypot mechanism is also deployed to trap attackers by creating a VM to pretend to be a victim, then when attacked, it alerts the controller.

The authors of [157] proposed *HogMap* a software-defined infrastructure for the purpose of measuring and monitoring of cyber-threat activities. The cyber monitoring landscape was created by integrating some SDN-enabled capabilities:

1. Intelligent in-place filtering of malicious traffic

2. Dynamic migration of interesting and extraordinary traffic
3. Software-defined marketplace where various parties can opportunistically subscribe to and publish cyber-threat intelligence services in a flexible manner.

They are based on a traffic filtering scheme and rule flow analyzer that checks incoming packets and allows rule insertion accordingly.

The authors of [173] presented a dynamic policy enforcement mechanism that allows ISPs to specify security policies to mitigate the impact of network attacks by taking into account the specific requirements of their customers. This mechanism depends on: Monitoring Component (MC), Policy Database (PDB), Policy Decision Point (PDP), and a Policy Orchestrator and Implementer (POI). Another policy based approach is the work of [215], which implements Policy Based SDN application for attack mitigation. The authors implemented a policy manager application that communicates with the controller over the northbound interface. The manager includes a policy creator, handler, and evaluation engine, and repositories to map AS domains.

The authors of [177] proposed *SN-SECurity* Architecture (*SN-SECA*) to ensure effective security evaluation and integration on the SDN/NVF designs and implementations. Their technique uses symbolic analysis to preview traffic process flow behavior across an infrastructure with SDN and NFV frameworks.

2.5 SDN Network security enhancements

Different solutions have been proposed to enhance SDN network security. The following solutions were based on either SDN security applications or SDN security architectures, that gave a boost to the SDN security domain.

2.5.1 DISCO (Distributed Control Plane)

A security monitoring application implemented on top of floodlight. It uses advanced message queuing protocol *AMQP* to monitor data traffic at two levels:

Intra-domain: Monitor and manage flow prioritization and dynamically reacting to network issues

Inter-domain: Manage communication between controllers which is composed of a messenger and agent, where the messenger discovers controllers and the agent exchanges data.

One approach to minimize the effect of security attacks is Reliable Controller placement. Research was done in this domain using the Simulated Annealing (SA) algorithm; this generic probabilistic algorithm has been favored as the most optimal algorithm for controller placement. The aim of this approach is to maximize the network resilience through efficient controller placement. Another algorithm is based on a minimum-cut based graph partitioning, which acts on minimizing the expected percentage of control path loss, but this approach introduced a tradeoff between reliability and latency.

2.5.2 Pareto-based Optimal Controller Placement (POCO)

SDN architectures have raised questions regarding reliability, scalability and performance when compared to traditional networks. Consequently, the controller placement problem has gained a lot of attention from the research community to estimate the optimal number and locations of distributed control elements inside SDN networks. Hock et al. discussed a resilient algorithm to optimize the placement of controllers with regard to failure tolerance [95]. They considered latency between controllers and traffic load balancing to get the optimal number of

controllers and their corresponding locations. The proposed algorithm took into account different failure events such as node failures including controller failure.

2.5.3 Network Security Architectures

LiveSec [216] is an OpenFlow-based architecture for network security management; it implements a new access switching layer besides the LiveSec controller and legacy switches to ensure distributed load balancing, end-to-end traffic control, and application monitoring through historical traffic replay and live traffic monitoring.

OrchSec [106] is an orchestrator-based architecture that uses SDN Control functions and Network Monitoring to enhance network security. The architecture separates the application development procedure from the SDN controller in order to mitigate different types of attacks such as cache poisoning/ARP spoofing and DoS attacks. In *OrchSec*, the controllers are only distributing control messages (e.g., flow rules), while network monitors examine traffic. Also, to enable independence among applications from controllers, applications are implemented as northbound applications and are not developed inside the controller.

Furthermore, in [217], an SDN application is proposed to mitigate ARP request and reply spoofing attacks by checking certain conditions on IP and MAC addresses based on the IPs leased by the DHCP server running on the controller, and ARP DoS attacks by port monitoring through statistics collection from switches. If an attack is detected by the controller, the corresponding flow rules are installed on the switches to drop these packets.

2.5.4 Control-Data Plane Intelligence Tradeoff

Total dependency on the controller has proven not to be practical. Therefore, some researches introduced intelligent tradeoff between the controller and switches aiming to increase availability, and reduce controller overhead.

Devoflow is one such approach that modifies the *OpenFlow* module in order to minimize controller-data planes interaction. It uses wild carded *OpenFlow* rules and the switches can take local routing decisions where per-flow vetting by the controller might not be necessary.

Netcore divides packet-processing responsibilities among controller and switches with the help of compilation algorithm coupled with the run-time system.

An SDN Security plane was designed and implemented for the purpose of detecting security attacks, and introduced a mechanism for attack source trace back in order to efficiently block the attack from the source. The security plane is designed to exchange security-related data between a third-party agent on the switch and a third-party software module alongside the controller in order to collect and analyze security related traffic and inform the controller to install a new flow to block the attack. The testing showed the capability of the proposed system to enforce different levels of real-time user-defined security with low overhead and minimal configuration [218].

The authors in [133] are extending the actuating trigger concept that was introduced in [186] by adding the handling of UDP packets besides TCP. To mitigate UDP DDoS attacks, they proposed traffic percentage trigger and amplification rate trigger. The traffic percentage trigger is fired when the amount of outgoing UDP packets relative to overall traffic (from a certain source) exceeds a predefined threshold in a certain window of time. Whereas the amplification rate trigger is fired when a large number of incoming UDP packets relative to outgo-

ing traffic exceeds a predefined threshold in a very short period of time. When a condition is triggered, predefined flow rules that are registered by the controller are installed onto the switches to prevent DDoS attacks or further analysis is performed at the control plane through forwarding the packet payload. However, traffic percentage trigger might not be enough to evaluate whether a DDoS is actually occurring or not. It might be that a server is generating a lot of legitimate requests at that particular time period. Additionally, traffic percentage trigger will not be fired when the attacker is spoofing its IP address.

The algorithm in [134] is run as an application on the controller and decides whether the received ARP packet is malicious or not. Additionally, the authors claim that it can be used to prevent similar threats such as MIM, eavesdropping, DoS and MAC Flood. The algorithm works in both dynamic and static IP assignment scenarios. When the dynamic DHCP IP assignment is used, the controller creates the IP-MAC matching table by obtaining the information from the option field of the DHCP reply messages. Similarly, when the static IP assignment is used, the IP-MAC pairs of all hosts connected to the switch are recorded in the table. When an ARP reply is received at the switch, the IP-MAC carried is checked against the table in the controller. If both MAC and IP match, it is forwarded to the destination, else the ARP packet is dropped. However, if an ARP proxy is used, the algorithm might fail by detecting ARP requests as malicious when it is not actually the case. Additionally, a large MAC-IP matching table is needed at the controller as the size of the network increases.

In [96], SDN-based architecture of Ad-hoc networks, SDN-based architecture for IoT networks and security functions to interconnect different domains is proposed. They divide a network into multiple extended SDN domains where each domain is an SDN-based IoT network, having a single border or root controller.

The different SDN domains connect to each other through the border controllers and are responsible for the security and routing functions of their own domains. To prevent attacks generated from inside or outside the network, the border controller would handle the authentication of both end nodes each belonging to a different domain. It is important to note that the border controllers, which are heavily loaded are acting as routers by forwarding the traffic between different domains, which is contradictory to the functionality of an SDN controller.

STRIDE was proposed to evaluate the security level of *sFlow*, an SDN network monitoring and measurement tool, and *BigTap*, a commercial monitoring controller [117]. *STRIDE* is used to evaluate only whether a system is vulnerable to a certain threat or not. *sFlow* consists of an agent and collector, where an agent is installed on a switch and sends statistical information to the collector installed on a monitoring host for analysis. The authors stated that if the monitoring and measurement network is not in the same network as the production network, and if TLS is used to secure the communication between agent and collector, *sFlow* agent can be considered secure, else it is vulnerable to various attacks such as tampering and information disclosure. *sFlow* is inherently not secure and its security depends on its deployment in a secure environment. Whereas *BigTap* is a SDN monitoring controller that applies routing policies and filtering for the incoming traffic from the production network through the switches. *BigTap* provides RBAC and TACACS+ security mechanisms which mitigate tampering, spoofing and information disclosure. Its only vulnerability is the unencrypted communication channel between the controller and the switches which can be secured using TLS; else it can be subject to tampering and information disclosure. However, TLS is not the only solution and it is cryptographically very involved.

The authors of [145] proposed the utilization of SDN in to solve the weaknesses in current honeynet architectures suffering from fingerprinting attacks and lack of fine-grained data control because only a single service can run at any given time. To solve these problems, *HoneyMix*-enabled controller was introduced which keeps a map of all available services on each honeypot at each host; it multicasts the incoming traffic to different honeypots based on the service being used by the attacker. If multiple honeypots offer the same service, connection weights are calculated and the one with the highest weight is selected to respond to the attacker. If a honeypot response includes indicators that can be used by attackers to detect the existence of honeynet, Response Scrubber, which is part of *Honeymix*, will remove these indicators.

Lim et al. proposed a mechanism to mitigate botnet based HTTP Flood DDoS attacks on servers by implementing an SDN based DDoS blocking application (DBA) [194]. When the server detects that it is under a DDoS attack, it alerts the DBA, which in turn provides the server with a new IP address. The server uses the new IP, by creating a new socket, to serve the client requests by sending a redirect message, where the information carried by the redirect message is costly to be computed by the bot (i.e. CAPTCHA). The procedure hence prevents botnets from flooding the new address and accordingly new flow rules are installed on SDN switches to forward the traffic to the server. Additionally, switches collect the counter for new flows that are being directed to the old address. If the counter exceeds a certain threshold, the flow generator is classified as a bot and its corresponding packets are dropped. However, they do not mention which API is used for DBA and server interaction and whether it is secure. Additionally, SDN is only being used to install the flow rules, and not for any security mechanism. It is the server that is acting as the DDoS detector.

The authors of [138] proposed the detection of port scan and fingerprinting attacks by installing specific flow rules. The proposed mechanism creates fake hosts alongside the authentic hosts. When the attacker scans the network, and tries to perform OS fingerprinting, based on a threshold on the number of packets sent to a specific number of fake host, flow rules are installed to prevent the attack. As for port scanning, fake ports are inserted into real hosts, and when the attacker scans a certain number of fake ports in a specific period of time, corresponding flow rules are installed to prevent the port scan attack. However, the source IP and MAC of the attacker is assumed to be always the same, and the attacker can study the behavior of the defense mechanism and can change certain parameters to bypass and perform the attack by spoofing its IP and/or MAC.

Network Flow Guard (NFG) [152] is a security application that detects rogue DHCP servers found in the network by analyzing every DHCP offer that is received at the switches, comparing it against a predetermined whitelist of DHCP servers and deciding either to drop or to forward the packet by installing the corresponding flow rules on the switches.

The authors of [219] proposed a software-defined networking (SDN) policy-based scheme for an efficient security architecture. The proposed scheme considers four policy functions: separating, chaining, merging, and reordering.

1. Separating: this divides the virtual services and decreases the size of the attack flows using the load balancer.
2. Chaining: this links many VNFs to prevent various attack flows and constructs big security systems.
3. Merging: this combines unnecessary VNFs to optimize the security system

and the system's resources.

4. Reordering: this reorders current VNFs depending on the type and strength of the current attack flows.

The authors claim that if SDN network functions virtualization (NFV) system managers use these policy functions to deploy a security architecture, they only submit some of the requirement documents to the SDN policy-based architecture. After that, the entire security network can be easily built.

Another security approach to detect rogue controllers was proposed in [165]. The authors argued that SDN security can be polished via the applications of pre-existing network security solutions. These solutions include: public key cryptography, secured sessions (single sign-on services), firewalls, honeypots, and mitigation of DDoS attacks via the use of various algorithmic methodologies. The authors pointed out that the implementation of a central authority controller (network orchestrator), with exceeding administrative authority over a cluster of slave controllers, can exhumate threats and block a hazardous slave controller.

SDSNM is proposed in [121] to mitigate DDoS attacks by utilizing SDN architecture at the edge points of the networks where the core network is traditional IPsec network. The SDN architecture is leveraged at the edges to provide access control, registration and authentication services (applications) through the controller to install appropriate flow rules, provide authentic communication and forward non-malicious traffic to the core network. Furthermore, it detects and reacts to any attempt of DDoS attack by first dropping the packets through installing updated flow rules and then executing a trace-back mechanism to find the attackers and analyze the malicious traffic by using the recorded flows, which are stored in the cloud.

WedgeTail was presented in [169] as an intrusion prevention system (IPS) designed to secure the SDN data plane. *WedgeTail* maps forwarding devices as points within a geometric space and stores the path taken by packets when traversing the network as trajectories. It prioritizes forwarding devices before inspection using an unsupervised trajectory-based sampling mechanism. For each of the forwarding device, *WedgeTail* computes the expected and actual trajectories of packets and “hunts” for any forwarding device not processing packets as expected.

The authors in [161] are utilizing the concept and architecture of SDN to construct a Programmable BYOD Security (PBS) system to provide access policy and network management in BYOD networks. The PBS controller provides fine-grained application-level security policy control over the BYOD devices, the BYOD devices that run PBS-DROID (PBS client system on Android) mimic SDN switches and the application on the devices represent the hosts connected to the devices. Thus, PBS provides application-level access policy to BYOD networks with minimal performance overhead and battery consumption.

2.6 Security Discussion and Future Directions

After examining many research works done in different directions, we have developed a list of security items that should be addressed when considering SDNs.

From one point of view, intrusion detection and prevention became more complex through the introduction of SDN because the switches communicate only packet header information to the controller, which is not sufficient for intrusion analysis and detection. As routers were vulnerable in non-SDN topologies to different types of attacks due to protocol specifications such as source IP, BGP

protocol, wormhole, black-hole, etc., *OF* switches have similar vulnerabilities where an attacker can impersonate a legitimate controller and hence inserts his own flow rules to bypass middle-boxes or firewalls in the network. Interest about the potential impact of DDoS has started recently, because unlike in traditional networks where the functions were implemented in individual routers, these are centralized in SDN in the controller. Hence any success of DDoS attempt will eventually make the entire network dysfunctional because switches will not be able to communicate with the controller and thus fail to update their flow tables. Some efforts are exerted through academic research activities; however, companies are spending considerable time and money to make these networks as secure as possible by investigating on one hand the protocol- and design-related vulnerabilities, and by preventing huge financial losses as in traditional networks on the other.

We believe that the benefits of SDN security outweigh the risks. In fact, due to the intrinsic centralization of the control plane, SDN provides security by design and it allows for end-to-end view and control of the entire network topology. This simplifies the development of complex networking applications and offers the ability to detect and react quickly to fake variations of the network state. In fact, the community already came up with several applications that identify and mitigate threats by accessing information at the packet level, analyzing it, updating rules and then reprogramming the network.

In SDN, security policy alteration is done easily; SDN permits the definition of new security rules on all of the network devices, which reduces the frequency of errors and misconfiguration, and decreases inconsistent rules in the infrastructure.

Transport layer security (TLS) with mutual authentication at the control-switch interface can alleviate several security risks. A full specification of TLS

between the controllers and their switches is mandatory to secure the link and data transmitted.

Integrity checks on software applications can ensure that harmful software is prevented from being launched in the network. In addition to integrity checks, specific malware detection schemes need to be developed for SDN. A malicious network application is a significant security risk. Therefore, all third-party software applications should be scanned for vulnerabilities and malicious code.

We believe that major threats of SDN already exist in traditional networks. A first step towards securing SDN would be to apply well known concepts such as the addition of public key infrastructures to encrypt communication channels between different modules of SDN and to ensure authenticity.

We have seen in the literature that most of the problems, vulnerabilities and attacks have been address through the long era of SDN security researches. Nevertheless, the main limitation remains in the durability and adaptations of the proposed solution with respect to the threat revolution. In other word, the lack of completeness and intelligence of these solutions kept the floor open for more advanced in the area of SDN security architectures. Another important limitation facing most of the proposed solutions is relying on only security to defend the system, while failing to ensure consistency and correctness of the targeted system before applying security measures. In our opinion that this last point will provide a large step towards a solid and secure network.

It is true that we have an increase in awareness and importance of security due to realization of SDN paradigms in real networks, it still remains slow compared to the increase in the intrusions and the sophistication of the attacks performed. According to the categorization, we can observe that different techniques have been proposed to mitigate attacks only as a reaction to the attacks. Although

the proposed frameworks can reduce the possibility of attacks, however, they are mainly prototypes and only lately did we see further drives towards real implementations.

2.6.1 SDN Security Summary

From our point of view, a complete security system needs to be established. The solution lies in an independent security architecture that has access to available data to be analyzed and investigated with assigned computational power.

In a previous work [218], we proposed a security plane alongside the control plane, and above the data plane. The security was based on a security module on the controller receiving security-related traffic from a security agent implemented on the SDN switch. In turn, the security module forwards these packet headers to a third-party security engine to be analyzed and clustered. To achieve resource separation, the security plane is connected to separate ports and the security engine would run on a separate machine.

Our module was initially designed to block DDoS attacks on the controller, switches, or hosts. Also, it can trace back the attack to its source to be blocked from the initial ingress port without affecting the network. The trace back was successful through inserting specific rules on the SDN switches to trace the origin of the packets.

Since our design is a foundation for a security architecture, such a design could act as a base for any of the previously discussed security designs. Each solution could be independently implemented on the security engine and benefit from the high processing resources. Since we have control over the security module at the controller and the security agent on each switch, it is possible to provide each security solution with the type and amount of data it needs to finish its work.

Moreover, it is possible for security functions to be implemented on the security module and requested by a security solution to further minimize the delay and response time.

The security engine could communicate with the controller through the REST interface to alert the security module or other modules of a possible attack before the effect of the attack increases, and hence, mitigating the damage of an attack before blocking it completely. Blocking an attack is a process of inserting a blocking rule through a request from the security engine targeting a specific port and source.

Another important point that affects security and needs to be taken into consideration is consistency and verification. As previously discussed, different types of attacks act on altering flow rules or modifying them to infiltrate the controller or take control over the switches. Other attacks work on adding flow rules in order to redirect traffic for monitoring purposes or creating black holes or loops. Either way, flow rule consistency checks and update verification is an important feature that needs to be implemented in any future security SDN design. As do security checks, verification checks need to collect data traffic, intercept update action, and monitor topology changes. For these reasons, we have proposed a similar architecture for consistency verification to maintain a resilient network [220]. The architecture is based on a verification plane that enables the exchange of the necessary packet headers between the verification agent on the SDN switches and the verification module on the controller. The proposed system uses state machine illustration on a third-party verification engine to design a tree-based prototype of the network in order to verify each new update and monitor the network. This engine is able to communicate through REST interface with the controller to request approval or block a specific update. Through verifying the

existing rules in the network, the possibility of such attacks would decrease, and if possible, will be blocked as soon as possible.

This last point is possible through the interoperability and interaction of these two systems, which is a crucial feature to form a complete base system for any design to be able to contribute in increasing the security of the SDN network.

2.7 Artificial Intelligence for network security

Recently, the soft computing and artificial intelligence methods have started to play a significant role in most of modern systems such as intelligent networking. That gives us the chance to improve the performance of current computer networks. The integration between the abstraction concept in SDN paradigm and AI techniques can lead to more adaptive behavior of network elements. Also, it will introduce new mechanisms for dealing with both traditional network issues and new SDN related ones [221]. This section highlights the attempts of applying artificial intelligence for SDN security. Although results show significant improvement, yet, this remains an open topic. However, in our opinion, hybrid intelligent techniques could be the key for achieving more advanced behavior in SDN-based networks. Motivation statistics show that Ant Colony algorithms were successful in increasing the maximal Quality of Experience (QoE) by 24.1% compared with the shortest path routing approach. Neural network based intrusion prevention systems have shown a scalable performance with low false positive rate. Applying reinforcement learning based technique in adaptive video streaming system compared with the shortest path routing and greedy-based approaches has shown a decrease in the frame loss rate by 89% and 70%, respectively [221].

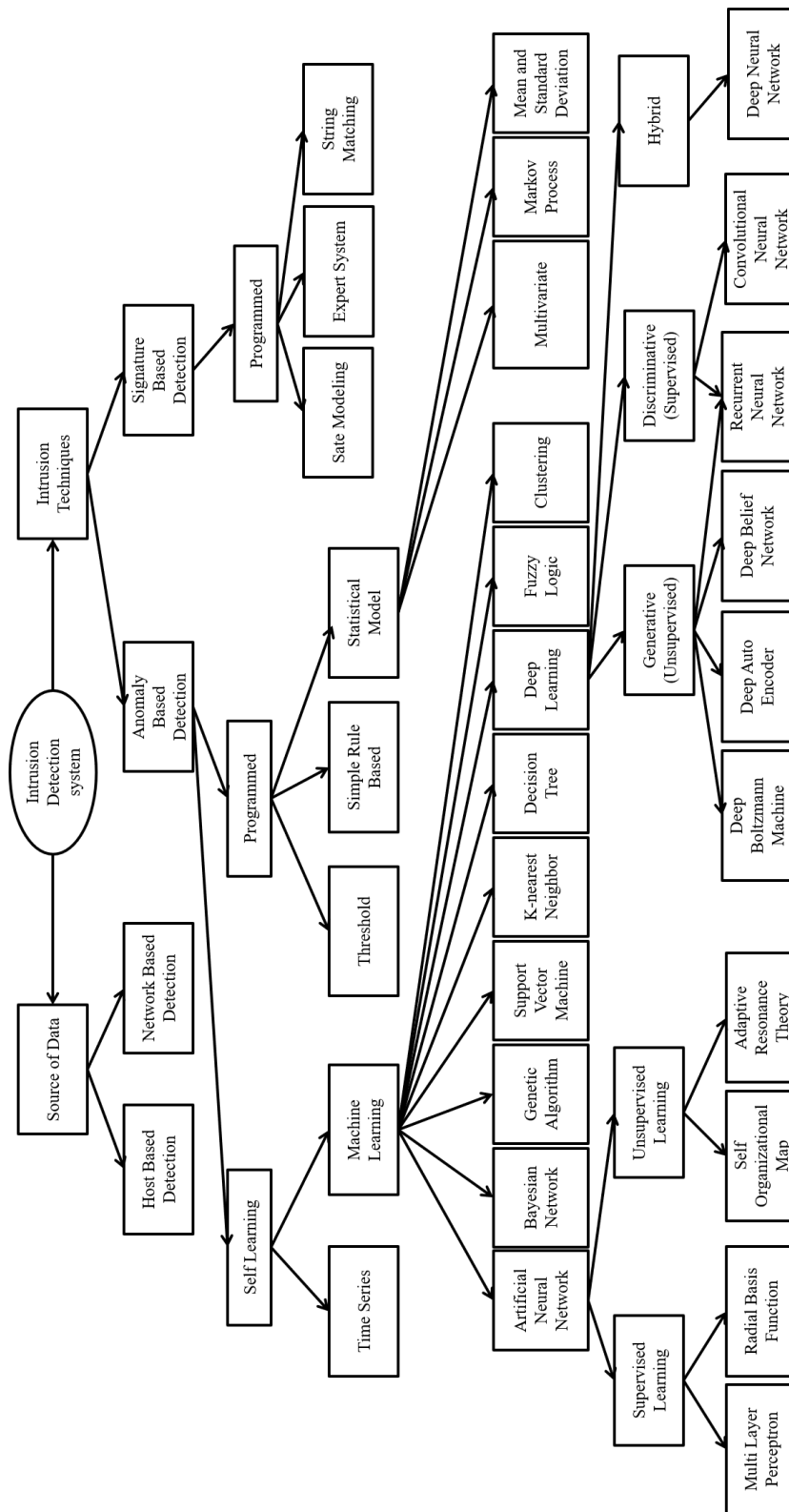


Figure 2.3: Intrusion Detection Techniques Overall Summary

2.7.1 AI for Network Security

New security threats in the networking and communication domains require rapid provisioning of security solutions capable of identifying and blocking known and unknown attacks. Different artificial intelligence-based techniques have been applied to enhance network security, quality, and intelligent network applications within SDN. The SDN approach introduces a set of new security challenges and it seems to be one of the biggest issues in SDNs. The potential threats include targeting the controller by programming vulnerabilities, error configurations and DDoS attacks on the secure channel [222]. Artificial intelligence and data mining techniques, which have been used before for solving routing problems and optimizing the performance of packet filters in conventional networks [223] [224] [225] seem to play a significant role in SDN-based networks after adding the programming ability, as the authors of [226] proposed an information security management system based on combination of fuzzy inference system and both of TRW-CB [227] and Rate Limiting [228] algorithms in SDN environment. The TRW-CB algorithm which detects the SYN Flooding, caused by a host based on the idea that the benign host will obtain a higher successful connection probability than a malicious one [9]. The input for the fuzzy logic module obtained by the mentioned algorithms and the degree of attack obtained as output. The decision-making system is implemented as an application for the SDN controller with short-term learning module. The proposed system has shown improved results compared with a non-fuzzy logic approach. The authors of [229] presented a classification of intrusion detection systems (IDS). Moreover, a taxonomy and survey of shallow and deep networks intrusion detection systems is discussed based on previous and current works. They reviewed machine learning techniques and their performance in detecting anomalies. Feature selection, which influences the

effectiveness of machine learning (ML) IDS was discussed to explain its role in the classification and training phase of ML IDS. At the end, the authors included a review of the false and true positive alarm rates to help researchers model reliable and efficient machine learning based intrusion detection systems.

Self-Organized Maps

A Self-Organized Maps (SOM) approach for DDoS attack detection has been proposed in [230]. SOM is a variant of artificial neural networks based on unsupervised learning. It can be used as a classification mechanism [231] when handling an unlabeled input vector. The training is based on a set of desired features from flow entries of the Open vSwitches. The detection loop consists of three stages: 1) Flow collection, which requests flow entries from all Open vSwitches; 2) Feature extraction, which takes the output of the flow collection module and extracts the most important features that forms a potential DDoS attack. These features include: Average of Packets per flow (APf), Average of Bytes per flow (ABf), Average of Duration per flow (ADf), Percentage of Pair-flows (PPf), Growth of Single-flows (GSf), and Growth of Different Ports (GDP) and 3) SOM classification, which is used as a classification method. These stages implemented as application-level modules in SDN controller. The proposed approach compared by different methods conducted on the well-known KDD-99 data set has shown a lower overhead [230], [232]. Srinivasan et al. [233] proposed an agent-based SOM IDS called SAPID, designed especially for wireless networks. They achieved an average DR of 97% on an undeclared dataset. Lichodziejewski et al. [234] achieved DR of 94% on DARPA 1998 dataset with a conclusion that hierarchically built unsupervised neural network approach is able to produce encouraging results. Rhodes et al. [235] analyzed SOM's potential

for intrusion detection. Their experiments showed that even a single SOM, when trained on normal data, will detect anomalous behaviors. Also, the ratio by which normal and intrusive packets differ has been greater by an order of magnitude. Their conclusion showed that IDS based on SOM should be particularly powerful because it never needs to be told what intrusive behaviors look like [46].

Multi-Layer Perceptron

MLP for anomaly detection was proposed as a single hidden layer neural network model [236]. The performance of this model tested on the DARPA 1998 dataset was a DR of 77% with 2.2% FP. Lippmann et al. [237] used selected generic keywords to detect the attack preparations and actions after the breaking. Results were used to train the MLP using backpropagation algorithm. This approach ensured a DR of 80% when it has been tested on the DARPA 1998 dataset. Debar et al. [206] stated that NNs are very slow in converging, they suffer from dimensioning and stability problems, training takes a lot of time to achieve a reasonable level of performance, and their adaptability is low because a partial retraining can lead to a network that forgets everything it has learned before [46].

Genetic algorithm

Li [238] proposed IDS with 57 genes in chromosomes, where each gene represents single connection feature, like: source IP address, destination IP address, source port, destination port, duration, protocol, number of bytes sent by originator, number of bytes sent by responder, etc. Due to the effectiveness of the evaluation function, the succeeding populations are biased toward rules that match intrusive connection. Mukkamala et al. [239] achieved 97.04% OA on DARPA 1998 dataset

for their implementation of IDS based on linear genetic programming (LGP). However, Teodoro et al. [240] concluded that main disadvantage of this kind of IDS is the high resource consumption involved [46].

Fuzzy Logic

Chimphlee et al. [241] proposed IDS based on rough sets theory and fuzzy c-means. The proposed system achieved a total of 93.45% OA on KDD'99 dataset, while reduced number of features resulted with enhanced performance. Dickerson et al. [242] developed FIRE IDS which uses simple data mining techniques to process the network input data and expose metrics that are particularly significant to anomaly detection. These metrics are then evaluated as fuzzy set for every observed feature and later used to detect network attacks [46].

K nearest means

Ma et al. [243] proposed kNN IDS based on similarity as a quantitative measure for distance and achieved DR of 90.28% (k=1) on KDD'99 dataset. Liao et al. [244] achieved DR of 91.70% (k=5) on DARPA 1998 dataset with their implementation of IDS based on kNN. They also concluded that kNN works well in dynamic environments where frequent updates of the training data are required, which makes it attractive for intrusion detection tasks [46]. The authors of [245] discussed the importance of network flow classification to network management and network security. They also emphasized on the challenge to classify network flows at very high line rates while simultaneously preserving user privacy. Machine learning based classification techniques utilize only meta-information of flow and have been shown to be effective in identifying network flows. In this work they analyzed a group of widely used machine learning classifiers, and observed

that the effectiveness of different classification models depends highly upon the protocol types as well as the flow features collected from network data. The proposed was vTC, a design of virtual network functions to select and apply the best suitable machine learning classifiers at run time. In their work they focused on K-nearest neighbor, SVM, Decision Tree, Adaptive boosting, Naïve Bayes, and MLP. The selected features were flag, logged_in, count, error_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count. Their experimental results showed that the proposed NFV for flow classification can improve the accuracy of classification by up to 13

Naive Bayes

Panda et al. [246] proposed NB based IDS and achieved OA of 94.90% on KDD'99 dataset. Amor et al. [247] made a similar proposition and achieved 91.52% OA. DDoS attack detection technique was discussed through the traces of the traffic flow [189]. The authors used different machine learning algorithms such as Naive Bayes, K-Nearest neighbor, K-means and K-medoids to classify the traffic as normal and abnormal. They tested how DDoS attacks can be detected by classifying the incoming requests using these techniques Machine Learning Algorithm.

Decision tree

Abbes et al. [248] compared DR of rule-based Snort and IDS based on DT on a custom dataset based on collected RPC protocol traffic. Snort detected only three, while proposed IDS detected all 46 different forms of attacks. Bouzida et al. [249] made comparison of DT with and without principal component analysis (PCA), a mathematical procedure that transforms a number of (possibly) corre-

lated variables into a (smaller) number of uncorrelated variables called principal components. They reduced computation time on dataset KDD'99 by a factor of approximately thirty, with slight loss of OA from 92.60% to 92.05% [46].

Support vector machine

Laskov et al. [250] concluded that the best performance is achieved by the non-linear methods, such as SVM, MLP and rule-based methods. Khan et al. [251] presented results where their solution, utilizing dynamically growing self-organizing tree (DGSOT), enhanced the training time of SVM on DARPA 1998 dataset from 17.34h to 13.18h and improved the accuracy from 57.6% to 69.8%. Chen et al. [252] achieved accuracy of 86.79% on KDD'99 dataset by using rough set theory (RST) to preprocess data and reduce the number of dimensions before forwarding it to SVM for intrusion detection. Mulay et al. [253] concluded that the integration of DT and SVM model gives better results than the individual models [46].

Random forest

Kim et al. [254] concluded that their approach based on RF has been able to show high DR, and figure out stable output of important features. Also, they stated that performance of RF based IDS turns out to be comparable to that of SVM. Zhang et al. [255] achieved average DR of 92.58% on original KDD'99 dataset and 99.86% on balanced dataset, where minority classes have been over-sampled and majority classes down-sampled to increase the DR of minority intrusions [46].

Deep Learning

The authors propose a deep learning based approach to implement such an effective and flexible Network Intrusion Detection System (NIDS) [256]. They used Self-taught Learning (STL), a deep learning based technique, on NSL-KDD. Self-taught Learning (STL) is a deep learning approach that consists of two stages for the classification. First, a good feature representation is learnt from a large collection of unlabeled data, x_u , termed as Unsupervised Feature Learning (UFL). In the second stage, this learnt representation is applied to labeled data, x_l , and used for the classification task. The authors discussed that the performance can be further enhanced by applying techniques such as Stacked Auto-Encoder, an extension of sparse auto-encoder in deep belief nets, for unsupervised feature learning and NB-Tree, Random Tree, or J48 for further classification.

The authors of [257] presented a survey on deep learning techniques for Intrusion detection systems. They examined the different DL categories:

- Unsupervised Learning: Auto Encoder (AE) and Boltzmann Machine (BM).
- Supervised Learning: Convolutional Neural Network (CNN).
- Hybrid: Deep Neural Network (DNN) and Deep Belief Network (DBN).

They claimed that deep learning is useful in IDS, especially for feature extraction and selection. The authors presented a new taxonomy of traffic classification from an artificial intelligence perspective, and then proposed a malware traffic classification method using convolutional neural network by taking traffic data as images [258]. The authors discussed that the method needed no hand-designed features but directly took raw traffic as input data of classifier. They claimed that this was the first time of applying representation learning approach to malware traffic classification using raw traffic data. The work discussed that trying to transform raw data into images and applying CNN to the results shows high clas-

sification results compared to other techniques. At the end the author included the need to test different ANN types for future work.

The authors introduced Deep Belief Networks to the field of intrusion detection, and proposed an intrusion detection model based on Deep Belief Networks, to apply in intrusion recognition domain [259]. The deep hierarchical model is a deep neural network classifier of a combination of multilayer unsupervised learning networks, which is called as Restricted Boltzmann Machine, and a supervised learning network, which is called as Backpropagation network. The authors showed that the experimental results on KDD CUP 1999 dataset demonstrated that the performance of Deep Belief Networks model is better than that of SVM and ANN.

A deep learning based multi-vector DDoS detection system in a software-defined network (SDN) environment was presented [260]. The authors implemented the system as a network application on top of an SDN controller. They used deep learning for feature reduction of a large set of features derived from network traffic headers. The detection system consists of three modules: i) Traffic Collector and Flow installer (TCFI), ii) Feature Extractor (FE), and iii) Traffic Classifier (TC). The authors emphasized that to minimize false-positives, the system relies on every packet for ow computation and attack detection instead of sampling flows. They claimed to achieve a detection accuracy of 99.82% with f-measure values as 99.85% and 99.75% for normal and attack classes, respectively, derived from the confusion matrix.

Below is a summary list of recent deep learning based IDSs:

Table 2.10: Summary List of Recent Deep Learning Based IDSs

Publication	Feature Extraction	Classification
[229]	Normalized Manually	DNN+Bayesian Calibration
[48]	Stacked Auto Encoder (SAE)	SAE/ Artificial Neural Network (ANN)
[230]	Auto Encoder (AE)	DBN
[231]	Stacked Auto Encoder (SAE)	Extreme Learning Machine (ELM)
[227]	Normalized Manually	DBN
[232]	Normalized Manually	Deep Neural Network (DNN), Recurrent Neural Network (RNN)
[233]	Monte Carlo Tree Search, MCTS	Convolutional Neural Network (CNN)
[234]	Normalized Manually	Restricted Boltzmann Machine (RBM)
[235], [236]	Normalized Manually	DBN
[237]	Stacked Denoising Auto Encoder	Logistic Regression
[238]	Deep Belief Network (DBN)	Support Vector Machine (SVM)
[239]	Normalized Manually	CNN

2.8 Tables

Commercial Products

Radware DefenseFlow, by cooperatively operating with software defined networks and leveraging the programmable and dynamic nature of SDN, presented a dynamic attack mitigation solution that mitigate network DDoS, Application DDoS and Advanced Persistent Threats. The solution operates using a continuous 4 stage service lifecycle [261]:

- Provision Security detection throughout the network by programming counters throughout the SDN nodes, by provisioning L4-7 Application Intelligence (AI) engines & by mirroring traffic to the L4-7 AI engines [261].

- Collect information from the entire set of provisioned information sources [261].
- Analyze network and application information in order to categorize behavioral patterns, maintain an ongoing behavioral baseline and identify any steep deviations from the baseline [261].
- Control traffic and service elements by blocking traffic, diverting traffic to dedicated attack mitigation engines and optimizing security policies [261].

2.8.1 Intelligent Network Applications

The integration between SDN and AI field opens the door for building more intelligent network applications. A reinforcement learning approach for adaptive video streaming in SDN paradigm was discussed in [262]. The controller represents a periodically decision maker that determines the time of selecting a new path and when the server needs to change the quality of the video. Markov decision process is used for modelling the actions of the decision making. The Q-learning technique is used in the case of unknown rewards for moving between the current and next state. The percentage of packet losses and the number of quality changes represent the most significant parameters to define the reward. The Q-values are updated and stored in Q-table where gamma and eta represent the discount factor and learning factor respectively [263].

The softmax function below represents the probability of selecting an action in state s time t .

Where T represents a random move already used in simulated annealing method to escape from the local optima problem. The controller can change the current path and/or adaptively extract/add the selected layers based on the available bandwidth to increase the QoE of the video streaming service. The

mentioned approach compared with the shortest path routing and greedy-based approaches has shown a decreasing of the frame loss rate by 89% and 70% respectively [263].

$$\hat{Q}(s_t, a_t) = \hat{Q}(s_t, a_t) + \eta(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (2.1)$$

$$P(a|s) = \frac{\exp[\frac{Q(s,a)}{T}]}{\sum_{b \in A} \exp[\frac{Q(s,b)}{T}]} \quad (2.2)$$

The authors presented a management framework to perform anomaly detection, classification, and mitigation [264]. The proposed framework is called ATLANTIC (Anomaly deTectioN and machine LeArNing Traffic classifIcation for software-defined networking), it combines the use of information theory to calculate deviations in the entropy of flow tables and a range of machine learning algorithms to classify traffic flows. Their results show that ATLANTIC was capable of categorizing traffic anomalies and using the information collected to handle each traffic profile in a specific manner, e.g., blocking malicious flows. The basic components of their work are the Statistical Layer that is responsible for collecting traffic flow statistics and comprises the following components: Statistics Manager, Features Selector and Network Driver. The information generated by the Statistical Layer is delivered to the Classification Layer, which comprises the following components: Anomaly Monitor, Flow Classifier, and Flow Manager [264].

An analyst-in-the-loop security system was presented in [265], where analyst intuition is put together with state-of-the-art machine learning to build

an end-to-end active learning system. The system has four key features: a big data behavioral analytics platform, an ensemble of outlier detection methods, a mechanism to obtain feedback from security analysts, and a supervised learning module. The authors discussed that when these four components are run in conjunction on a daily basis and are compared to an unsupervised outlier detection method, detection rate improves by an average of 3.41x, and false positives are reduced fivefold.

- Big data processing system: A platform that can quantify the behaviors (features) of different entities, and compute them from raw data.
- Outlier detection system: This system learns a descriptive model of those features extracted from the data via unsupervised learning, using one of three methods: density, matrix decomposition, or replicator neural networks.
- Feedback mechanism and continuous learning: This component incorporates analyst input through a user interface. It shows the top k outlier events or entities, and asks the analyst to deduce whether or not they are malicious.
- Supervised learning module: Given the analyst's feedback, the supervised learning module learns a model that predicts whether a new incoming event is normal or malicious.

2.9 AI for Network Quality

Network Consistency is initiated by maintaining a balanced and congestion free network before loading the verification phase. Load balance functions are a requirement for minimizing the latency and maximizing the throughput in computer networks that support multiple routing approaches. Load balancing also considered as a defense technique against some types of networks attacks such as

DDoS attack [266], [267].

The abstraction in SDN approach provides an important advantage which is the global view and discovering of topology of the network. A Back Propagation Neural Network was used for achieving real time dynamic load balance, which decreased latency by 19.3% compared to DLB and static Round Robin methods [268]. The input vector for the neural network contains path information such as: bandwidth utilization ratio, packet loss rate, transmission latency, and transmission hops [268].

A BPNN based approach for load balancing in data centers was also presented [269]. The BPNN applied internally inside the Open vSwitch in a way that reduce the time consumed for sending routing decision from the controller to the Open vSwitch. The input vector consists of: available bandwidth, and packet loss. The authors of [270], proposed a genetic algorithm in SDN based client server architecture. The fitness function defined by the Formula:

$$\min \frac{\sqrt{\frac{(\sum_{j=1}^K X[j]^2) - \left(\frac{\sum_{j=1}^K X[j]}{K}\right)^2}{K}}}{\frac{\sum_{j=1}^K X[j]}{K}} \quad (2.3)$$

Where K represents the servers and each one has X set of workloads. The performance in 33 has been compared with random and round robin methods and shows better performance. Another genetic algorithm was proposed for flow routing optimization in SDN based audio over IP network has been introduced [271]. The network described as a connected graph. The problem is to show that the graph meets the demand which is bandwidth and latency requirements of the source and destination. The fitness function given by the formula shown in

Equation.

$$\max \frac{\sum \text{Embedded demands}}{\sum \text{demands}} \quad (2.4)$$

The population size and non-allocation probability were the most important parameters to the algorithm. The advantage to use the genetic algorithm approach is to get a partial solution of the problem through the solving stage while this is not possible in linear program; this partial solution helps in evaluating other algorithms [271].

In another context, an Ant Colony Optimization (ACO) approach for QoE-aware flow routing is presented in [272]. ACO is a swarm intelligence method that uses metaheuristic optimization [36]. In computer networks the Quality of Experience (QoE) indicates requirements for the customers to measure the value of provided service from customer's perspective. In [272] the SDN applications deliver user session parameters to the controller which in turn runs the ACO algorithm on a weighted graph, where the weights between vertices are delay and loss rate for each network device. The fitness function depends on the flow type and estimated value of corresponds QoE model (i.e., audio, video or data). ACO has achieved 24.1% increasing for the maximal QoE value obtained by the shortest path routing approach. Other Kmeans and crossover based modified ACO algorithm was discussed in [263].

Another AI technique was developed for SDN QoE enhancement which proposed a machine learning approach combined with adaptive coding in order to provide a better QoE for video streaming services [273]. They discussed the benefits of a centralized architecture, where the totality of the network is known, to

predict its status. The solution calculates approximately the quality needed for a video to be streamed. Next the quality found by the ML-based algorithm will be combined with the network situation to choose the right coding. Their work is based on the measurement or prediction of image quality, based on calculating the structural similarities between two images, rather than the pixel to pixel difference.

In general, the total work done on AI techniques to enhance network quality in terms of verification, consistency, reachability, and correctness is far less than the effort done in the security domain. This integration in a remote management based network such as SDN is still a new trend with a promising future.

Cognitive learning based techniques have been proposed to enable dynamic network control and real-time management to future communication technologies [274]. Techniques such as self-organizing networks and autonomic network managements were discussed in addition to SDN in the context of solving network problems. The authors also emphasized the role on these techniques in supporting centralized traffic engineering and facing DoS attacks in SDN networks. A similar approach was discussed in [275] for applying cognition tools for reinforcing SDN both on the security and management domains. A new approach was considered to ensure network reachability over the best route. This technique was the Ant Colony Optimization (ACO) that was discussed earlier [276]. This method studied the ant's method of traveling for food and applied the concept for packet routing.

Other work on network quality was using Atoms to detect violations of network-wide reachability invariants on the data plane [277]. The authors proposed an amortized quasi-linear algorithm to do the job. Delta-net was a real-time data plane checker that incrementally maintains a compact representation about

packet flows of all packet in the network to support a broader class of scenarios and queries. Still these approaches require inspecting each packet in the network through forwarding them to the controller, and therefore introducing additional load.

A verified network can be achieved only if it is based on a verified system. A verified AI technique is essential to maintain a verified and secured network. Verifying any AI system is an important issue that is becoming an interest in the research domain. The authors of [278] discussed verified artificial intelligence that are provably correct with respect to mathematically-specified requirements. They described 5 challenges and 5 corresponding principles for achieving verified AI.

Challenges:

Environment Modeling, Formal Specification, Modeling Systems that Learn, Generating Training Data, Scalability of Verification Engine.

Principles for Verified AI:

- a. Introspect on the system to model the environment.
- b. Formally specify end-to-end behavior of the AI-based system.
- c. Develop abstractions for and explanations from ML components.
- d. Create a new class of randomized formal methods for systematically generating training/test data.
- e. Develop computational engines for run-time, quantitative, and learning-based verification. Other discussed and proposed different Verification and validation tools for AI software. The authors of [279] argued regarding the complexity and nature of AI techniques that makes them hard to be verified. In addition to

the different complex features, AI system can be nondeterministic which should be taken into consideration when trying to verify and validate such systems.

2.9.1 Time Line Overview

The evolution of the terms “artificial intelligence” and “machine learning” are shown for period 2010 till 2016 in the form of the number of intrusion detection related publications for each particular year. From given results it is visible that the usage of AI, and ML as its most prominent sub-field, has an increasing tendency. Greater values for ML can be explained by lack of references to AI in related studies [46]. A time line for different AI algorithms is shown for period 2010-2016. It is clear that NN is the most popular AI algorithm used for intrusion detection compared to other algorithms, but with slight decrease in popularity over the last couple of years. Then there are GA and SVM, where SVM is constantly increasing in popularity. Out of the rest, SOM seems to be the least popular, which could be explained by the lack of an intuitive approach to solution of intrusion detection problem. It is clear from observed trends that AI composes a significant part of the intrusion detection study, becoming more and more popular tool of choice [46].

2.9.2 Machine Learning Hybrid techniques

ML Intrusion detection systems have generally been used to detect attacks but in recent years, using two or more different techniques to form a hybrid system has improved the overall performance. The following table shows the accuracy of detection and data set used for selected reviewed ML/IDS papers from 2011 till 2016.

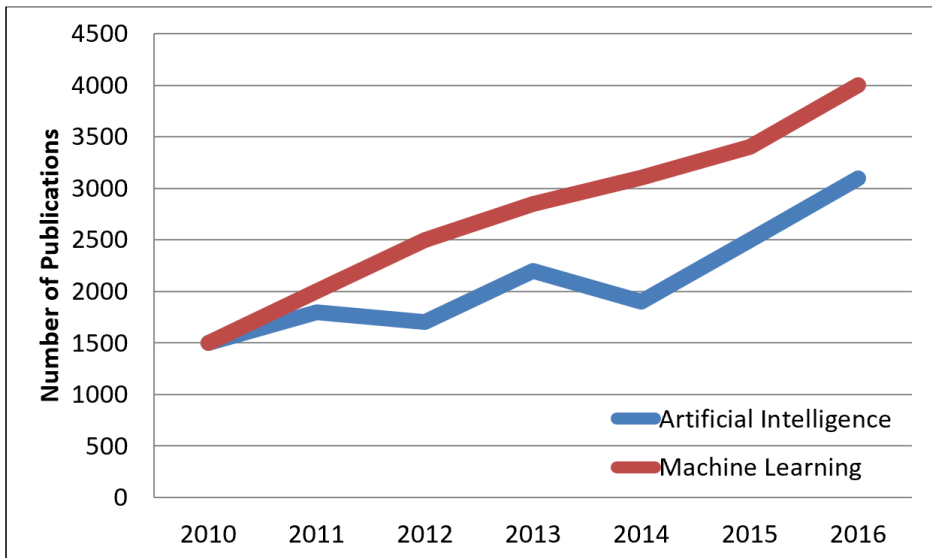


Figure 7 AI and ML Publications since 2010

Figure 2.4: AI and ML Publications Since 2010

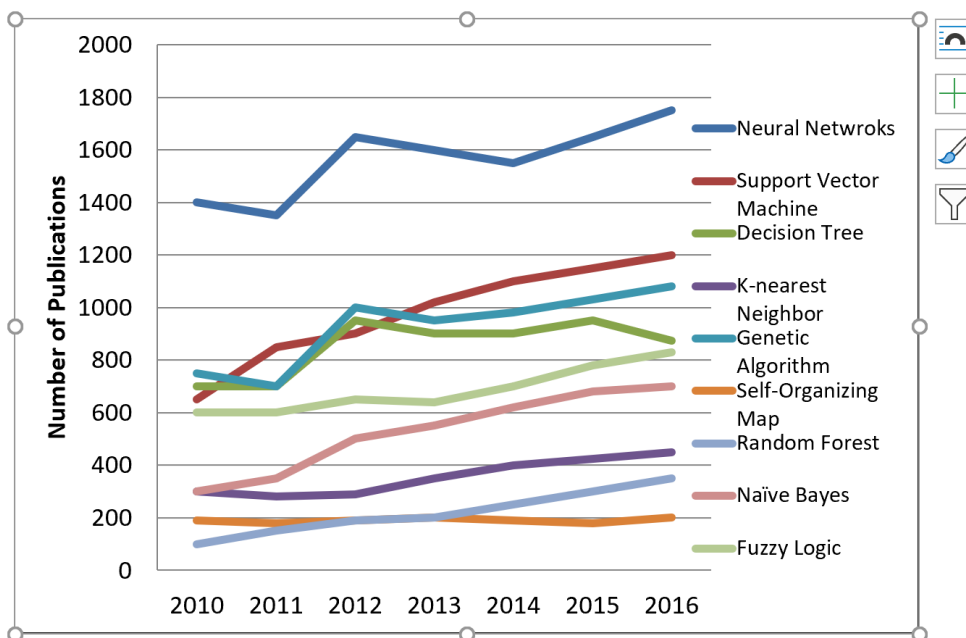


Figure 8 Intrusion Detection Publication for different AI Algorithm

Figure 2.5: IDS Publications for Different AI Techniques

Table 2.11: IDS Accuracy of Detection and Data Set Used

Reference	ML Technique	Attack Type	Data set	Accuracy
[31]	K-means +KNN+DT	R2L, U2R, DoS and probe	KDDCup99	96.5%
[32]	FL+GA	Dos and Probe	Real Life	97%
[33]	GA+SVM	Dos Probe U2R R2L	KDDCup99	99.1% 99.1% 97% 96.5%
[34]	SVM	R2L	KDDCup99	96.1%
[35]	SVM+BAT Algorithm	Malicious	NL-KDD	99.4%
[36]	GA	DoS, R2L, U2R and Probe	KDDCup99	92.6%
[37]	DT	Dos Probe	NSL-KDD	82% 64%
[38]	Cluster centre+ K-NN	Probe Dos	KDDCup99 (6 dimension)	99.9% 99.9%
[39]	SVM+K-NN	DoS,R2L,U2R and Probe	KDDCup99	87.4-91.7%
[40]	SVM + hierarchical clustering algorithm	DoS, R2L, U2R and Probe	KDDCup99	95.7%
[41]	ANN	DDoS/Dos	Real Life	99.4%

2.9.3 Intrusion Detection Selected Summary

The authors of [256] proposed a deep learning approach based on Self-taught Learning (STL) to implement a Network IDS. They tested their work on the NSL-KDD dataset.

The authors of [66] proposed a new method that illustrates network traffic as images using convolutional neural network for malware traffic classification. The authors argued that applying CNN on images rather than raw data results in higher classification accuracy.

Deep learning techniques showed much improvements in the DDoS detection domain. The authors of [260] proposed a deep learning multi-vector-based DDoS detection system in an SDN environment. They discussed feature reduction derived from network traffic headers. Their results show an accuracy of 99.75% between normal and attack classes.

A list of ML-based proposed techniques is presented in the following table. As shown, AI-based techniques are being used, offering high results. Thus, the problem remains in systemizing these techniques, making such solutions inflexible, and vulnerable. The main issue is that these solutions are focusing on increasing accuracy regardless of the other feature of a network that may be affected. First, through redirecting traffic to a fixed point for processing and analysis, such as the Controller, introducing high traffic overhead and security risks. Second, through redirecting only packet headers, which decrease the overhead, but still inherits similar weaknesses

A summary table of intrusion techniques and data used in selected reviewed papers between 2014 and 2016:

2.10 Consistency and verification for network resilience and quality

Ensuring consistency in a network is essential for any security system. To the best of our knowledge, this is the first attempt towards an AI-based consistency verification solution, in a privacy preserving architecture, aiming for more resilient networks. The following section describes the various consistency check techniques that were proposed in the literature for different purposes.

Maintaining a current view of the whole network and being able to handle a

Table 2.12: Summary Table of Selected Reviewed Papers

Reference	Year Published	Intrusion Technique	Application	Data set
[270]	2014	Anomaly	Cloud Computing	Real life
[271]	2016	Anomaly	Transport	Real life
[272]	2014	Anomaly	Substation	Real life
[273]	2015	Signature	Cloud Computing	Real life
[274]	2015	Anomaly / Signature	Computer systems	Real life
[275]	2015	Anomaly	Cloud Computing	ADFA-LD and KDD98
[276]	2015	Signature	Cloud Computing	Real life
[277]	2015	Signature	Internet of Things (IoT)	Real life
[278]	2015	Anomaly	Information systems	KDD99 Cup
[279]	2015	Anomaly	Cyber space	KDD98 and UNM
[280]	2015	Anomaly	Computer systems	KDD99
[281]	2015	Signature	Cloud Computing	Real life
[282]	2015	Signature	Transport	Real life
[283]	2016	Signature / Anomaly	Computer systems	Real life
[284]	2016	Anomaly	Telecommunication	Real life

large number of switches requires a significant amount of data exchange between the SDN controller and the switches. In addition, all unmatched traffic on each switch may get forwarded directly to the controller for it to analyze and insert corresponding rules. In return, the controller offers the ability to incubate a set of modules each having a different functionality. Each module is responsible for inserting flow updates onto the switches, for the module to fulfill its purpose.

Each SDN switch, and after the network has converged, will have a set of flows that control the packet exchange between switches. The flow updates process and the flow table created at each switch raises a consistency flag. How can one verify the update process and ensure the consistency of flows within and between switches?

SDN consistency entails several domains that need to be addressed to be able to answer the above question:

- Inter network consistency: The issue here stems from an update that affects several switches in the network at the same time. The inconsistency may occur if a switch is updated before another, leading in some cases to loops or other failures. Another issue is how to ensure that a newly received packet would be matched to the set of new rules only, when the packet passes during the update process [280].

- Another consistency issue is how to preserve the benefit of centralization when the model needs to decentralize in order to scale. How can one ensure that a set of controllers over a large SDN network, each having a set of different modules, would keep the network consistent and guarantee that each entering packet would exit from the correct switch port or reach its intended destination? In such a scenario, rules and policies are being installed from different controllers and different modules.

- Verification of the consistency of a flow table in the same switch is needed. This may be achieved by traversing the flow table to verify that no overlapping rules exist and no contradicting rules are installed. After the current state of the flow table is verified, the verification process will be limited to checking all new incoming rules.

- The domain of formal consistency verification includes translating the flow

tables from all switches in the network to a formal language to be fed to one of the real-time network verification tools. These tools work on querying the network to verify a set of preconfigured properties. After modeling the network into the verification scheme and configuring the desired properties, the verification becomes a reachability analysis such that the system is free of inconsistencies, there are no dead locks, and all required properties are satisfied.

The consistency related work in the literature focused on several main aspects. The authors of [281] worked on state synchronization to ensure that redundant controllers have the same state information. While, the authors of [280] focused on inter-flow consistency by proposing a scheduling process for multiple network updates to be checked for overlaps or contradictions that may cause network failure or even security threats.

The work of [282] proposed a third-party consistency verification check module that includes route correctness and network security isolations in a single domain. Other researches, such as [283], handled tunable consistency models to enable controllers to tune their own configurations to enhance the performance of applications running on top of them.

Several network verification tools, such as [284], [285], [286], have been developed to check for network inconsistencies such as: loops, dead ends and network unreachability alerts. The main drawback of most of these techniques lies in the security concerns of the solutions themselves, and their security side effects on the network, in addition to the additional network traffic being exchanged and redirected from one party to another.

The authors of [287] proposed employing online clustering techniques (sequential and incremental k-means), for the purpose of applying consistency and performance in a large scale SDN network governed by multiple simultaneous

controllers. Their results show that when using adaptive controllers, they can select a feasible value between consistency and performance according to network conditions.

The objective of SDN flow consistency verification is to ensure that a new updated rule inserted in the system will affect the system as intended, and no side effects will occur. Different techniques have been introduced into different parts of an SDN network to solve the different issues that inconsistencies may produce. Kang et al in [288] showed how UPPAAL can be used in formal modeling and analysis of SDN OpenFlow. Podymov and Popesko in [289] showed that by dividing the system into a set of all packet headers, a controller, a collection of switches, and a group of network channels, we can translate the SDN network to UPPAAL for verification. They also worked on verifying behavior scenarios using this tool, although such a tool was not initially designed to verify SDN networks.

After the introduction of SDN, several researchers have proposed different methods to verify the consistency of the entire network at once during the update process. Khurshid et al proposed VeriFlow in [290], which is a layer between the controller and the switch that checks network-wide correctness with low latency. This layer consists of a graph of the whole network, and as an update is inserted, it queries the graph for inconsistencies. Although this scheme showed low delays and good performance; however, traversing the whole graph each time introduces extra processing delays.

Ball et al presented in [285] VeriCon, a system for verifying that an SDN program is correct for different topologies and network events. The verification is done through expressing network-wide invariants as first-order logic formulas. Liu et al in [280] proposed an abstraction layer for inter-flow consistency to verify two relationships, spatial isolation and version isolation. They discussed an update

scheduling algorithm based on a dependency graph. The presented module still depends on traversing the whole graph and verifying only these two relationships.

Moshref et al in [291] discussed FAST (flow-level state transition), whereby the controller preinstalls a state machine, and switches can record flow state transitions by matching incoming packets to installed filters. This system enables the switches to take dynamic actions based on local information. Although it was not mentioned in this paper, but such a system can introduce inconsistency issues in terms of end to end verification, since it can only be used for flow table verification in a single switch.

Skowyra et al. proposed Verificare in [286], a verification platform to enable formal verification of SDN networks as components of a larger domain specific system. The purpose of this platform was to provide safety, security, and performance to SDN applications through formal verification. Mahajan and Wattenhofer in [292] presented an architecture for consistent updates in SDN. They proposed minimal algorithms to ensure a loop-free network after the update process. Depending on parent/child relationships, nodes could be updated in a consistent manner. Although this work showed promise, but it was limited to testing loop inconsistencies.

Sethi et al presented in [293] an abstraction for model checking controllers for a large number of packets exchanged in the network. They validated the utility of these abstractions through two applications: a learning switch and a stateful firewall.

Only few researchers took the work a step further toward designing a controller module that is the main gateway of all updates to be downloaded onto the switches. Such a module is presented in this paper, and it collects the necessary information to verify the consistency of each flow update using a third party for-

mal verification tool. Centralizing the update process makes it easier to verify the consistency of the flows in a single switch and in the entire network.

The following table summarizes the recent SDN solution for maintaining network quality by either debugging or verifying network consistency or correctness:

Table 2.13: SDN Verification for Network Quality (1)

Group	Solution	Main Purpose	Description
Debugging	ndb [137]	gdb alike SDN debugging	Basic debugging primitives that help developers to debug their networks.
	NetSight [138]	multi-purpose packet history	Allows to build flexible debugging, monitoring and profiling applications.
	OFRewind [139]	tracing and replay	OFRewind allows operators to do a fine-grained tracing of the network behavior. Operators can decide which subsets of the network will be recorded
	PathletTracer [140]	inspect layer 2 paths	Allows inspecting low-level forwarding behavior through on-demand packet tracing capabilities
	SDN traceroute [141]	query OpenFlow paths	Allows users to discover the forwarding behavior of any Ethernet packet and debug problems regarding both forwarding devices and applications.

Other SDN solutions aim to improve network quality through enforcing measurement and monitoring applications or introducing different traffic engineering modules. Recent works in this domain are summarized in the following table:

Table 2.14: SDN Verification for Network Quality (2)

Group	Solution	Main Purpose	Description
Verification	Assertion language [142]	debug SDN apps	Enables assertions about the data plane on the apps with support to dynamic changing verification conditions
	Cbench [143]	evaluate OpenFlow controllers	The Cbench framework can be used to emulate OpenFlow switches which are configured to generate workload to the controller
	FLOVER [144]	model checking for security policies	FLOVER provides a provably correct and automatic method for verifying security properties with respect to a set of flow rules committed by an OF controller
	FlowChecker [123]	flow table config verification	A tool used to verify generic properties of global behaviors based on flow tables
	FLOWGUARD [99]	verify security policy	Provides mechanisms for accurate detection and resolution of firewall policy violations in OpenFlow-based networks
	FlowTest [145]	verify network policies	Provides the means for testing stateful and dynamic network policies by systematically exploring the state space of the network data plane
	NetPlumber [146]	real time policy checking	NetPlumber uses a set of policies and invariants to do real time checking. It leverages header space analysis and keeps a dependency graph between rules

Table 2.15: SDN Verification for Network Quality (3)

Group	Solution	Main Purpose	Description
Verification	NICE [147]	remove bugs in controllers	Its main goal is to test controller programs without requiring any type of modification or extra work for application programmers
	OFCBenchmark [148]	evaluate Open-Flow controllers	Creates independent virtual switches, making it possible to emulate different scenarios. Each switch has its own configuration and statistics
	OFTEN [149]	catch correctness property violations	A framework designed to check SDN systems, analyzing controller and switch interaction, looking for correctness condition violation
	OFLOPS [150]	evaluate Open-Flow switches	A framework with a rich set of tests for OpenFlow protocol, enabling to measure capabilities of both switch and applications
	OFLOPS-Turbo [151], [152]	evaluate Open-Flow switches	Framework that integrates OFLOPS with OSNT [152], a 10GbE traffic generation and monitoring system based on NetFPGA
	PktBlaster [153]	emulation / benchmarking	Integrated test and benchmarking solution that emulates large scale software defined networks
	SDLoad [154]	evaluate Open-Flow controllers	A traffic generation framework with customizable workloads to realistically represent different types of applications
	VeriCon [155]	verify SDN apps	A tool for verifying the correctness of SDN applications on large range of topologies and sequences of network events
	VeriFlow [98]	online invariant verification	It provides real time verification capabilities, while the network state is still evolving

Table 2.16: SDN Solutions for Network Quality (1)

Group	Solution / Application	Main Purpose	Controller	Southbound API
Measurement & Monitoring	BISmark [156]	active and passive measurements	Procera framework	OpenFlow
	DCM [157]	distributed and coactive traffic monitoring	DCM controller	OpenFlow
	FleXam [158]	flexible sampling extension for OpenFlow		
	FlowSense [159]	measure link utilization in OF networks		OpenFlow
	Measurement model [160]	model for OF switch measurement tasks		OpenFlow
	OpenNetMon [161]	monitoring of QoS parameters to improve TE	POX	OpenFlow
	OpenSample [162]	low-latency sampling-based measurements	Floodlight	modified sFlow
	OpenSketch [163]	separated measurement data plane	OpenSketch	“OpenSketch sketches”
	OpenTM [164]	traffic matrix estimation tool	NOX	OpenFlow
	PaFloMon [165]	passive monitoring tools defined by users	FlowVisor	OpenFlow
	PayLess [166]	query-based real-time monitoring framework	Floodlight	OpenFlow

Table 2.17: SDN Solutions For Network Quality (2)

Group	Solution / Application	Main Purpose	Controller	Southbound API
Traffic Engineering	ALTO VPN [167]	on-demand VPNs	NMS [270], [271]	SNMP
	Aster*x [168]	load balancing	NOX	OpenFlow
	ElasticTree [169]	energy aware routing	NOX	OpenFlow
	FlowQoS [170]	QoS for broadband access networks	POX	OpenFlow
	Hedera [171]	scheduling / optimization	---	OpenFlow
	In-packet Origin filter [172]	load balancing	NOX	OpenFlow
	MicroTE [173]	traffic engineering with minimal overhead	NOX	OpenFlow
	Middlepipes [174]	Middleboxes as a PaaS	middlepipe controller	
	OpenQoS [175]	dynamic QoS routing for multimedia apps	Floodlight	OpenFlow
	OSP [176]	fast recovery through fast-failover groups	NOX	OpenFlow
	PolicyCop [177]	QoS policy management framework	Floodlight	OpenFlow
	ProCel [178]	Efficient traffic handling for software EPC	ProCel controller	---
	Pronto [179], [180]	Efficient queries on distributed data stores	Beacon	OpenFlow

Table 2.18: SDN Solutions for Network Quality (3)

Group	Solution / Application	Main Purpose	Controller	Southbound API
Traffic Engineering	Plug-n-Serve [181]	load balancing	NOX	OpenFlow
	QNOX [182]	QoS enforcement	NOX	Generalized OpenFlow
	QoS for SDN [183]	QoS over heterogeneous networks	Floodlight	OpenFlow
	QoS framework [184]	QoS enforcement	NOX	OF with QoS extensions
	QoSFlow [185]	multiple packet schedulers to improve QoS	---	OpenFlow
	QueuePusher [186]	Queue management for QoS enforcement	Floodlight	OpenFlow
	SIMPLE [187]	middlebox-specific “traffic steering”	Extended POX	OpenFlow
	ViAggre SDN [188]	divide and spread forwarding tables	NOX	OpenFlow

Chapter 3

Previous Work

The road towards integrating AI with SDN efficiently, and via new techniques, was motivated by our previous work on SDN security, verification, consistency, and the application of SDN in different fields such as VANET, MPTCP, and QUIC, aiming to enhance both security and network quality. The results of these studies were published in various articles as discussed below.

3.1 SDN Security Plane

SDN Security Plane: An Architecture for Resilient Security Services:

We proposed an SDN security design approach, which strikes a good balance between network performance and security features. We showed how such an approach can be used to prevent DDoS attacks targeting either the controller or the hosts in the network, and how to trace back the source of the attack. The solution was based on introducing a third plane, the security plane, in addition to the data and control planes. The security plane was designed to exchange security-related data between a third-party agent on the switch and a third-party

software module alongside the controller. Our evaluation showed the capability of the proposed system to enforce different levels of real-time user-defined security with low overhead and minimal configuration [294].

3.2 SDN Verification plane

SDN Verification Plane for Consistency Establishment:

We proposed an SDN verification layer based on formal techniques to establish flow consistency between SDN switches before the flow insertion process takes place. We showed how such an approach can be used to prevent loopbacks, deadlocks, security domain breaches, and to verify the time delay for a controller to update a switch versus the switch to forward a packet. This last point ensured that the update process is synchronized, and no packet would be checked against old rules during the update process. The solution was based on introducing a verification plane enabling our verification module to interact with a third-party verification tool (UPPAAL) translating the controller's view of the network to a state machine and verifying each flow before being installed. The verification tool checks each flow against a predefined set of rules by applying the new flow to the scheme and testing if a packet can pass from point A to B without violating these rules. Our evaluation showed the capability of the proposed system to enforce different levels of consistency verification in case of flow update and topology change in an SDN network [282].

3.3 SDN and MPTCP

SDN for MPTCP: An Enhanced Architecture for Large Data Transfers in Data-center Layer-2 Networks:

Multi-Path TCP (MPTCP) boosts network performance by aggregating bandwidth over multiple paths using sub-flows of the same TCP connection. However, MPTCP suffers from three limitations: 1) it is an end-to-end protocol with no control over the network routes, and sub-flows might end up traversing the same links, 2) it has no dynamic control over choosing the optimal number of sub-flows to achieve maximum throughput, and 3) its performance may degrade due to the large number of out-of-order caused by the heterogeneous paths traversed. Software Defined Networking (SDN), being centralized by nature, provides a global view of the network. When integrated with MPTCP, SDN improves resource utilization. We proposed an SDN-enhanced MPTCP that achieves higher data rates while transferring big-data in large-scale L2 networks such as those found in datacenters. Test results showed a 20% to 30% increase in the throughput over regular MPTCP [295].

3.4 SDN and VANET Security

SDN VANETs in 5G: An Architecture for Resilient Security Services:

Vehicular ad Hoc Networks (VANETs) have been promoted as a key technology that can provide a wide variety of services such as traffic management, passenger safety, as well as travel convenience and comfort. VANETs are now proposed to be part of the upcoming Fifth Generation (5G) technology, integrated with Software Defined Networking (SDN), as key enabler of 5G. The technology of fog computing in 5G turned out to be the perfect solution for faster processing

in delay sensitive application, such as VANETs, being a hybrid solution between fully centralized and fully distributed networks. We proposed a three-way integration between VANETs, SDN, and 5G for a resilient VANET security design approach, which strikes a good balance between network performance and security features. We show how such an approach can secure VANETs from different types of attacks such as Distributed Denial of Service (DDoS) targeting either the controllers or the vehicles in the network, and how to trace back the source of the attack. High mobility in VANETs poses a major challenge in the face of fast processing and low delay, which was made possible by the integration of SDN and fog computing represented by the Road-Side-Controllers (RSC) architecture and the introduction of the security plan, alongside the data and control planes of SDN. Our evaluation shows the capability of the proposed system to enforce different levels of real-time user-defined security, while maintaining low overhead and minimal configuration [296].

3.5 SDN and QUIC

SDN for QUIC: An Enhanced Architecture with Improved Connection Establishment:

Quick UDP Internet Connection (QUIC) boosts web traffic performance by solving a number of transport-layer and application-layer problems experienced by modern web applications, while requiring little or no change to applications. QUIC provides key advantages to HTTP/2 such as reduced connection establishment latency, improved congestion control, multiplexing without head-of-line blocking, forward error correction, and connection migration. However, QUIC suffers from three limitations: (1) it performs poorly under high bandwidth and

large traffic conditions, (2) it suffers from some security issues, (3) it implements static Forwarding Error Correction (FEC) XOR “group” mechanism, which performs poorly under variable network conditions, influencing utilized bandwidth. Software Defined Networking (SDN), being centralized by nature, provides a global view of the network. When integrated with QUIC, SDN improves resource utilization and network security. We propose an SDN-enhanced QUIC that achieves higher bandwidth utilization and secure resiliency while transferring large amounts of traffic over the internet [297].

3.6 SDN Security review

Software-Defined Networking (SDN): The Security Review:

SDN provides network operators with significant visibility and granularity of their networks leading to more flexibility in programming these networks. With every new technology paradigm, two concerns would typically arise regarding security and resilience of the network. We provided a comprehensive SDN security review including the different vulnerabilities and attacks that SDN suffers from. The objective was to entice the SDN community to address such issues inherently and not as an afterthought. The paper also reviewed the different security proposals that have been presented or implemented for SDN and by SDN. A general discussion was included to shed light on the pending security issues and some proposed solutions were presented [298].

3.7 SDN and 5G

SDN & Edge Computing: Key Enablers towards the 5G Evolution:

“2020 and Beyond” was announced by the ITU to be the era of the next mobile network generation. After (LTE)/4G, 5G is promising to be a major evolution in the communication domain, not simply because of the acceleration of the data rate, but rather due to the new applications. The challenging objectives such as minimum user-plane latency, uninterrupted connectivity, high Quality of Service (QoS), high data rate communications and network capacity, while dealing with ubiquitous and heterogeneous network access call for a major overhaul of the whole mobile network architecture. The limitations of today’s mobile systems, derived from their dependency on hardware-based designs, led to inflexible and limited architectures. It is essential to have dynamic and flexible management systems at several levels, starting from the Radio Access Network (RAN), passing by the Evolved Packet Core (EPC), up to the application interfaces. These future demands and the requirement for a self-adaptive system can be realized by adopting the SDN paradigm, which leads to the integration of SDN in the network components of the upcoming 5G technology. SDN has its positive impact in the communication world from several aspects; it will provide 5G with a smooth transition and unified management among various wireless standards and among different RANs and wired core networks. Furthermore, SDN can optimally orchestrate the interference between cells, handovers, roaming process, routing, and signaling between access and core networks, management of the gateways, and even the management of user data. Furthermore, the new bandwidth and latency requirements, and the ability to support the innovative 5G applications, cannot be satisfied by centralizing the data in the cloud. Pushing the data to the user’s proximity will be vital for some time-critical applications, which is a requirement supported by edge computing. Therefore, the geo-distribution of data requires an optimal networking design for these edges. On top of this distributed data

layer, the Network Function Virtualization (NFV) coupled with SDN, promises easier management of such an infrastructure. We investigated how SDN can be integrated into the 5G network architecture at different levels (RAN, EPC, security, etc.), and highlighted the solutions, challenges and benefits resulting from such integration. Also, we presented the mobile edge computing concept, its integration with SDN, and its implications on 5G. We reviewed the designs and architectures that have been proposed in this area, and others that are under development, including the innovative applications and use cases that will be enabled by the SDN-5G combination. Our work was concluded by proposing an architecture for SDN-5G for telecom operators [299].

3.8 Diameter Security

Securing Diameter: Comparing TLS, DTLS, and IPSec:

The Diameter signaling protocol plays a critical role in mobile networks. Diameter manages a crucial function in mobile systems since it is designed for activity coordination between Internet Protocol (IP) network elements such as online charging systems, policy servers, mobility gateways, among others. As operators migrate their networks to LTE, Diameter provides new services and implements more sophisticated policy use cases through new signaling techniques. All this confirms the importance of securing Diameter. We performed a comparative study on three security protocols used to secure Diameter (TLS/TCP, DTLS/SCTP, and IPSec). The comparative analysis focused on three main aspects: transmission (header), connection establishment, and processing overhead. Each aspect in each protocol was investigated in details and the results showed that securing Diameter using TLS introduces fewer RTTs compared to DTLS,

with IPsec introducing the highest number of RTTs. On the other hand, DTLS requires the minimum processing overhead, TLS comes in second place with a relatively small difference, and IPsec introduces the highest processing delay [300].

3.9 Machine Learning for network Resilience

After all this work, we started our work on the final goal: Machine learning for Network resilience. From that day we have managed to publish a series of 4 papers on this topic. Each paper is an extension of its predecessor with a new contribution towards our ARS system. All the work and research in these papers and more will be discussed in details throughout this report. The papers were as follow:

- A- Machine Learning for network Resilience: A start of a journey [301].
- B- Machine Learning for network Resilience: A second Step towards ARS.
- C- Machine Learning for network Resilience: Midway towards ARS.
- D- Machine Learning for network Resilience and Consistency [302].

Chapter 4

ARS Introduction

Our vision towards a more resilient network was inspired by our initial work on the security and consistency of programmable networking. The following section shows the road map towards ARS.

4.1 The Road Towards ARS

Our work started with SDN and its features. We have researched, analyzed and contributed to the SDN community throughout the years. Our interest in SDN have grown, but with time our focus shifted from SDN itself to the contributions that SDN would offer as a programmable network architecture to other domains. Thus we started working on the security features that SDN can provide to other networking technologies. It is from this point, that we contributed in multiple of articles involving the integration of SDN with VANETs, QUIC, MPTCP, 5G. The goal was to provide new architectures for these technologies to overcome their security issues or create a more secure environment for them to deploy.

At this point, we were started arguing about the necessity of a new security

solution that is general enough to serve different networking technologies with minimal changes. Also, a solution that is specific enough to target a single or multiple attacks, regardless of the signature or features of that attack being large or small. With time we argued the necessity of network verification and maintaining consistent network. We researched and contributed in both these domains, where we contributed in the first SDN Security and Verification Planes. These publications were the inspiration of different researches in the future.

After that point, we knew that reaching our goal would require more than standard techniques. So, we started investigating Machine Learning and AI to fulfill our goal. We researched and tested multiple AI techniques for the role of intrusion detection and mitigation. We also used both supervised and unsupervised techniques to detect both known and unknown attacks. We also worked on the integration between SDN and AI to create a resilient network that would stand strong on both security and consistency basis. The following sections will briefly discuss our work on the SDN Security plane and SDN Verification Plane that lead us to the ARS architecture.

4.2 SDN Security Plane

The need for security became more critical after the introduction of SDN due to the centralized control and the different vulnerabilities that arise with any new network architecture and associated protocols. Our contribution in this paper is summarized as follows:

1. The proposal of a general scheme to be used as a third party for different exhaustive processing services in an SDN network.
2. The proposal of SDN real-time DDoS prevention technique based on the

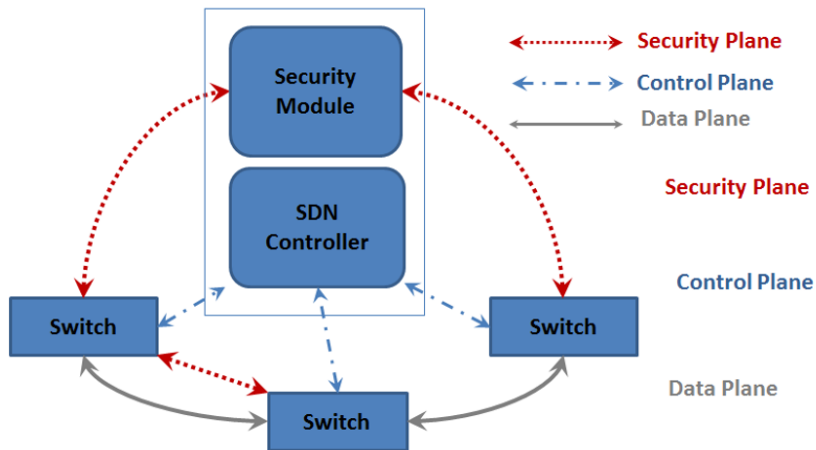


Figure 4.1: SDN Security Plane Architecture

previous scheme.

3. The proposal of trace back technique to identify the origin of an attack in case of IP spoofing and to block it from the source.

4. The implementation and test of the proposals in different environments.

Our proposal stems from the idea that security analysis requires both a processing engine and additional resources. Benefiting from what SDN has to offer in the area of software networking, we introduce a third plane in SDN relative to the data and control planes. The Security Plane, shown in Figure 1, provides a connection between the switches and the controller. An SDN software agent resides alongside the software switch, such as Open vSwitch (OVS), which connects to the agent using a virtual port, and a rule is inserted on each switch to forward traffic to the agent independent of the whole network. The agent on the other hand, establishes a connection to the controller via a port different than the control-plane listening port and from a unique source port. At the other end of this connection, a third party software module runs next to the SDN controller software, and possibly on a different computing platform.

Each agent holds a specific ID and is connected to only one OVS bridge

through a virtual port. The agent can either use the same physical port as the control plane or a different one, although the second choice is recommended for total separation. A different connection (from a different source port) is created for the security plane connection. The agent should be capable of running lightweight scripts to analyze part of the traffic locally before uploading it to the centralized security module. In this case, the processing load is balanced between the agent module and the security module thus less overhead is placed on the controller.

The most important part of the architecture is the security module at the controller side. It is responsible for collecting and analyzing all data traffic coming from all agents to detect abnormal events and to trigger alarms to the controller to take necessary prevention actions. For this last point, a connection with the controller is required to exchange rules and detection-related data. The security module communicates with the controller, which in our case is POX, through the REST API. The security module includes from one side as an interface with the downstream agents on the switch for the purpose of accepting connections and from the other side an interface for the detection engine to read incoming packets.

The detection engine is designed and written in Pyretic, an open-source programming language that enables programmers to specify network policies at a high-level of abstraction. The framework that runs the detection mechanism is Resonance, an SDN network management framework where operators define a network policy as a Finite State Machine (FSM). The transition between states is triggered by different types of dynamic events in the network. Samples of the states that we used are: Allow and Block. The term used for a Resonance platform implemented in Pyretic is PyResonance. The Pyretic monitoring and

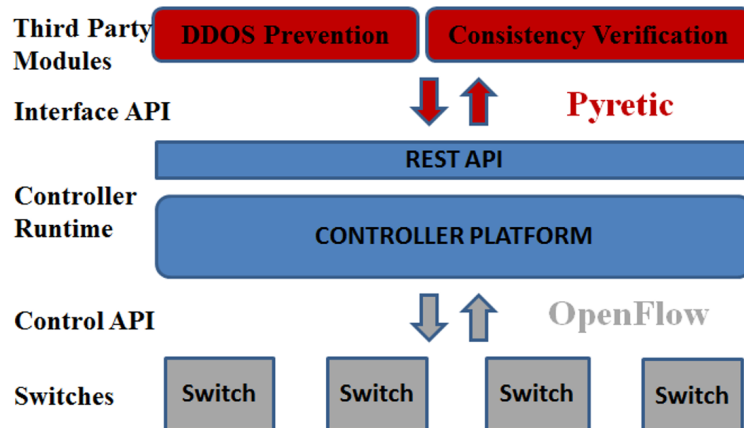


Figure 4.2: Security Module – Controller REST Connection

detection services are based on selected policies and query policies, as shown in the following tables, to calculate some predefined events. Here, any policy script can be implemented to perform an output action after analyzing the received data. The detection engine, upon a state change, triggers an alarm to the controller through REST and sends the appropriate rule for the controller to insert into the switches to block the attack. Another feature is the ability to send a trigger to the controller, which can then run a script to communicate with the switches through OVSDB to change, as an example, network interfaces or policy to start an IPsec session over a certain link.

Table 4.1: Selected Policies

Syntax	Description
identity	Returns original packet
none	Returns empty set
Match(f1=f2)	Identify if field f1 matches field f2 / None otherwise
Modify(f1=f2)	Returns packet with field f1 set to f2
Fwd(p)	Modify output port equal p
Flood()	Returns one packet for each local port on the network spanning tree

Table 4.2: Query Policies

Syntax	Description
Packets(limit=n, group_by=(f1,f2,))	Callback on every packet received for up to n packets identical on fields f1, f2,
Count_packets(interval=t, group_by=(f1,f2,))	Count every packet received Callback every t seconds providing count for each group
Count_bytes(interval=t, group_by=(f1,f2,))	Count bytes received Callback every t seconds providing count for each group

4.2.1 DDoS Prevention Technique

Malicious hosts and switches can launch DDoS attacks by flooding the network with traffic to arbitrary hosts or to the controller itself in order to exhaust resources or to change the total view of the network at the controller, thereby affecting the forwarding process in the data plane or denying the controller its services. IP spoofing is one of the most used techniques in these kinds of attacks that enable the real source to remain anonymous during the attack.

Our proposed technique aims at detecting DDoS attacks from malicious hosts or bots aimed at other hosts or at the controller. The proposed scheme is summarized as follows:

- a. Analyzing collected packets targeting a specific IP address and tracking its packet count/sec in order to compare it to a threshold.
- b. Keeping a database for mapping MAC-IP-Switch-Port binding for the purpose of detecting spoofed IPs and packets.
- c. Upon detection, an alert is sent along with the source IP of the attacker in order to be blocked.
- d. A trace back procedure is used to identify the origin of the attack Switch-Port so it can be blocked from its origin in case of IP spoofing.

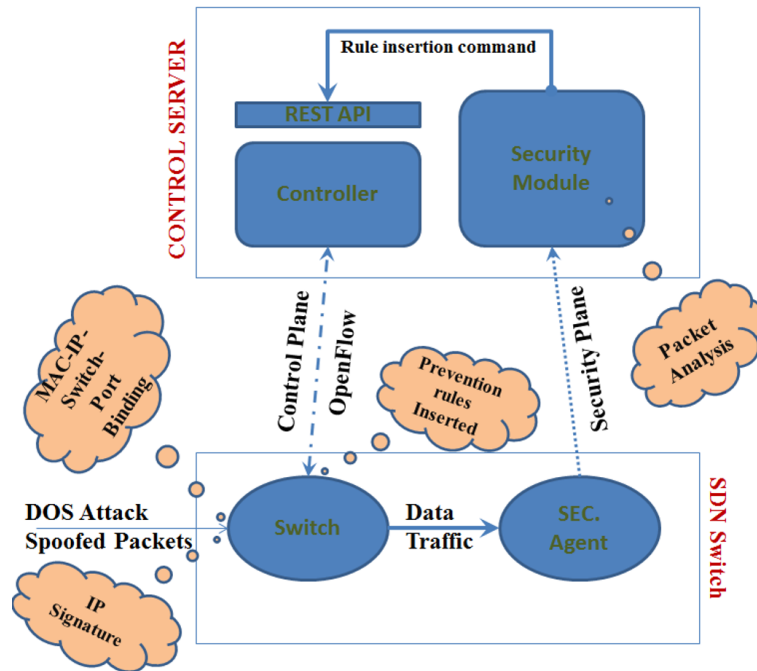


Figure 4.3: Overall Procedure Scheme

The overall procedure is described in the following steps:

At the switch level:

- a. The preconfigured MAC-IP-Switch-Port binding allows the controller to track each host in the network. An unauthorized (spoofed) packet coming from a certain port would trigger an alarm when unmatched to the binding rules on the switch and then sent to the security module at the server side.
- b. The security agent on the switch forwards all packet headers to the security module including unmatched spoofed packets through the security plane.
- c. The attack trace back policy is applied at the switch level by setting the source IP of the IP spoofed packets, that don't match the binding rules, to signature IP for only that ingress port. Only the modified packets will be sent to the security module.

At the server level: a. The security module listens for incoming packets from the agent; a sample flow chart is shown in Flow Chart 1. Upon receiving modified

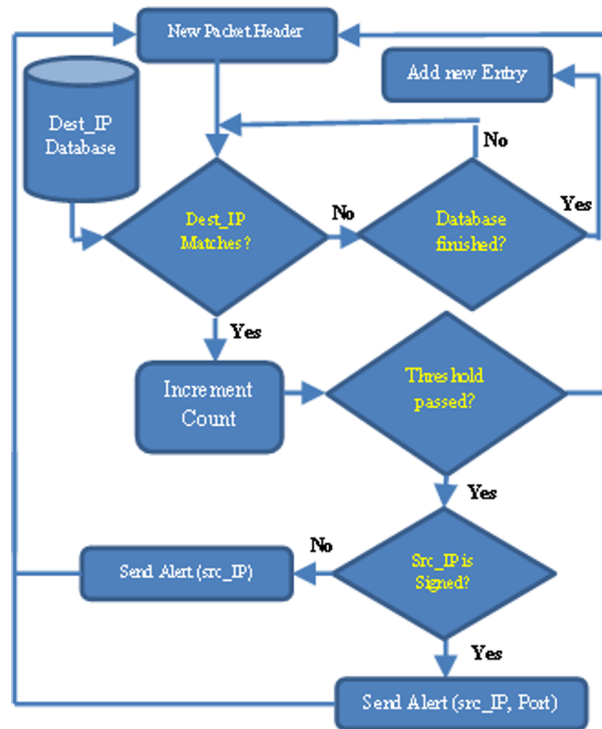


Figure 4.4: Attack Detection FlowChart

packets, it groups them according to signed IP addresses. This modification will allow the controller to identify the source port of the attack upon receiving such signed packets. These packets are considered as a DDoS attack packets only if their count exceeds the DoS threshold, else they will be treated as unauthorized packets and will be blocked. In both cases, the prevention method will be applied on that source port only.

b. After the detection process sees a state change, it forwards a command through the REST API to the controller to insert an IP blocking rule on all switches or on a single port in case of an IP spoofing DoS attack.

c. As mentioned in the previous sections and based on the sample in the following tables, Packet_In messages are being checked for invariances that would trigger an alert when a certain policy is violated by a group of packets.

Our proposed scheme is designed in such a way that it can detect an attack

Table 4.3: Sample of Invariant Verification

Data	Target	Invariant
Packet	Packet Spoofing	MAC-IP-Switch-Port PACKET IN

Table 4.4: Sample of Default Policies Being Checked

Trigger	Policy
PACKET_IN	Authorized MAC-IP-Switch-Port Binding ONLY

in real-time using minimal controller resources and with minimum configuration. Not only it proved to detect DDoS attacks, but to prevent malicious hosts from corrupting the controller’s view of the whole network by spoofing their IP addresses.

4.2.2 Testing and Simulation

We implemented and tested each module separately to verify its correctness. The next step was to test the connectivity layers between the switch and the agent from one side, and the security module and the controller from the other. The setup was done initially using a single Linux (Ubuntu 14.04.2) virtual machine using VirtualBox (3.2.10). We installed Mininet as a network emulator and used both POX and OVS version 2.4.0 as the controller and software switch, respectively.

At the switch level, we implemented the software agent that receives packets from one virtual port and sends them through another. At the server level, PyResonance was used to write the connectivity protocol with the controller through REST. After the initial connectivity is established, the agent opens a second connection to TCP port 6454 which is the security module port on the server. The security module listens for incoming connections on this port and

accepts connections coming from agents with authorized IDs registered in a pre-defined database. At this stage, the setup is completed and a rule is inserted, via POX, on all switches in order to additionally forward the incoming packets to the agent's port. Then, the packets are verified to have successfully reached the security module through the security plane. We tested triggering the blocking rule insertion on the controller by matching specific types of messages received from agents, and the tests proved to be successful.

In order to test DDoS prevention, we inserted POX code that binds the security key (MAC-IP-Switch-Port) together for every new incoming host, and that only accepts authorized connections after the first connection. We considered a number of ports to be host-only ports and applied the policy on these ports. Another feature that the controller provides is attack trace-back, which was achieved by creating a set of rules to check the authenticity of each packet coming from host-only ports, and if none of the rules matched, the default rule is to sign the packet by setting a signature IP, specific for each port, in the source IP field and then to forward the packet to the agent. A third feature of the POX controller is that it was extended to receive commands through the REST interface from the security module, and to insert flows in switches accordingly. These steps were successfully tested on a single and double switch networks.

Next, we test the DDoS analysis and detection in the security module. The code is based on receiving incoming packets from the agent and grouping them according to destination IP. A counter for each group was created to keep track of the frequency of these groups. The threshold of a DoS attack depends on the network environment. In our case, and for testing purposes, we used a threshold of 100 packets per second. When the threshold is exceeded, an alarm with the source IP is sent to the controller. In case of IP spoofing and upon receiving a

signed packet, the controller is informed of the presence of an unauthorized host to block the source host-only port. Each of the three stages showed successful responses in real-time and in the early stages of an attack.

We launched attacks in two different ways; first, we used ICMP flooding from different hosts within Mininet, with OVS connected to a local controller using the loopback IP 127.0.0.1 and destination port 6633. The second approach was to use three virtual machines, two of which are Ubuntu and the third was kali Linux in order to demonstrate different types of DoS attacks. The first Ubuntu machine was running Mininet and OVS, and it was connected to a remote POX controller on the second Ubuntu machine. We installed OVS on the kali Linux machine and it was successfully connected to the remote POX controller. Attacks using hping3 and hulk were demonstrated assuming that one of the hosts acts as a web server in the network. The testing of each of the stages over this setup was successful and the attacks were dropped as was shown using tcpdump on the hosts and represented in Figure 5. Blocking the attack in its early stages to enable real time protection is a matter of assigning a threshold relative to the maximum tolerance of the controller or the host under attack. The tests have shown an average response time 3s from the start of an attack till total blockage, while the controller kept functioning normally. The compromise between setting a low threshold or keeping a gap between the threshold and the victim's tolerance is an environment-dependent issue.

4.3 SDN Verification Plane

The need for flow consistency verification becomes more critical with SDN due to the centralized control, the various modules that run on the controller, the

network architecture, and the fact that the switches are unable to make dynamic decisions and should strictly apply the flow table rules. Our contribution in this paper is summarized as follows:

1. The introduction of a verification module to be used as a gateway for flow insertion from the controller downstream to all SDN switches.
2. The introduction of a verification plane linking the verification module to a third party formal verification tool.
3. A scheme to apply consistency verification on all new updates, including flow insertion and topology updates.
4. The implementation and testing of the proposals in different environments.

Our solution stems from the idea that consistency analysis requires both a full view of the network topology and intercepting each new update being installed. Benefiting from what SDN has to offer, we introduce a third plane in SDN relative to the data and control planes. The Verification Plane, shown in Figure 1, provides a link between the verification module and a third party verification tool. The verification module at first sends all necessary information to the verification tool, in our case UPPAAL, so a state machine representation of the whole topology could be sketched using this tool. This scheme will be divided into sub networks representing the security domains in our topology where a packet from one domain should not pass to the other. After the initialization stage is completed, the verification module acts by receiving all flow requests from all modules in the controller. The module will send each update to UPPAAL to be verified for consistency with the switch flow tables and to verify that no violations would occur if this update is installed.

The verification module will install the updates onto the destination switches after receiving a verified signature from UPPAAL. At the same time, the topolo-

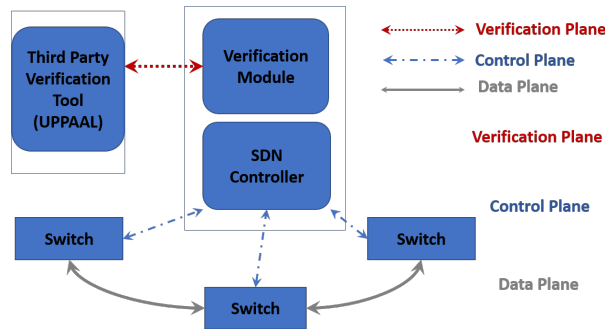


Figure 4.5: SDN Verification Architecture

gies at the verification tool are also accordingly updated.

The most important part of the architecture is the verification module on the controller side. It is responsible for analyzing all topology changes, benefiting from the controller’s view of the network, and sending them to UPPAAL to update its own topologies and hence to maintain a synchronized view. The verification module communicates with the controller, which in our prototype implementation is POX, through a REST API, as shown in Figure 2. The verification module consists of two main sub-modules:

- a. A verification sub-module which is part of the controller, responsible for collecting and sending data to the second sub-module.
- b. A verification layer that communicates with controller through a REST API accepting data from the first sub-module and forwarding them to the verification tool. This sub-module translates the data to UPPAAL format.

The verification module, upon receiving an inconsistency alarm from the verification tool, sends an alert to the controller along with the unverified update and discards the flow in question.

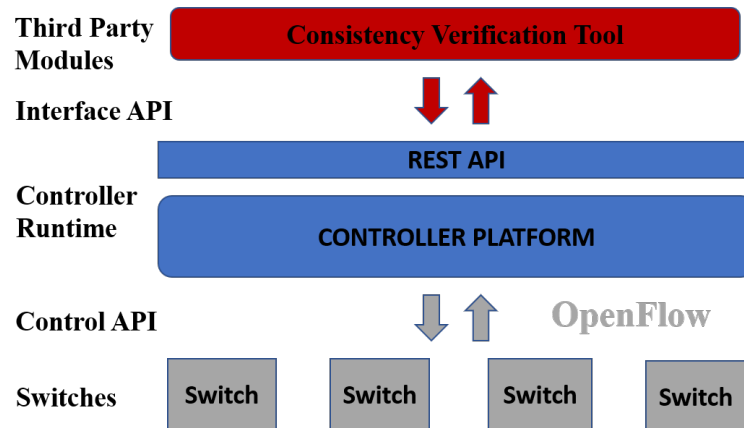


Figure 4.6: Verification Module – Controller REST Connection

4.3.1 SDN Verification Tool

UPPAAL is a model-checking tool that we used to verify real-time properties of an SDN network. The update verification is based on translating SDN topologies to a network of timed automata. UPPAAL illustrates networks of finite timed automata [26] as an input model. The modeling is based on state machines and is done according to the currently installed rules. Each node (switch) is guarding a matching rule on the incoming packet port and executes some updates on the network parameters according to the output decision. Different system parameters are shown in following table.

Our proposed approach aims to install the new update into the state machine topology and query this topology to verify that no predefined rules are violated. The proposed rules are summarized as follows:

- a. Verifying a loop-free network through setting a time limit for this packet to reach its destination.
- b. Checking deadlocks by checking that each node knows how to forward this packet, and that this packet eventually reaches its destination.
- c. Checking for security breach attempts on our security domains through set-

Table 4.5: UPPAAL Parameter Samples

Parameter	Description
sendingPacket?	A function triggering packet forwarding between nodes
CurrentPosition	A parameter that is set to next position when a certain rule is matched
packetRule	The rule being matched
RequestToController	A new flow being requested from the controller upon receiving a new unmatched packet
repltFromController	Flow update being verified
Finish!	Returning from HostB to HostA state upon receiving the packet to the final destination

ting fixed waypoints on the gateways between the security domains and verifying that no packet passes through these waypoints.

d. Verifying the update time delay by checking that the delay time for an update to reach a switch coming from the controller is always less than the time needed for the packet to be forwarded from the upstream switch to that same switch.

The most useful technique to verify network properties is to sketch a model of the entire network using the controller’s view, and then run queries on the model to verify these properties. However, checking the entire network’s graph every time a new flow is installed fails to provide real-time response and introduces extra processing overhead. Instead, and since most forwarding rule updates affect only a small portion of the overall topology, we could divide our topology into sub networks and query the relevant parts each time an update is to be checked. In our case, we choose to divide the network into security domains to further verify that our technique can also check the update for security misconfiguration.

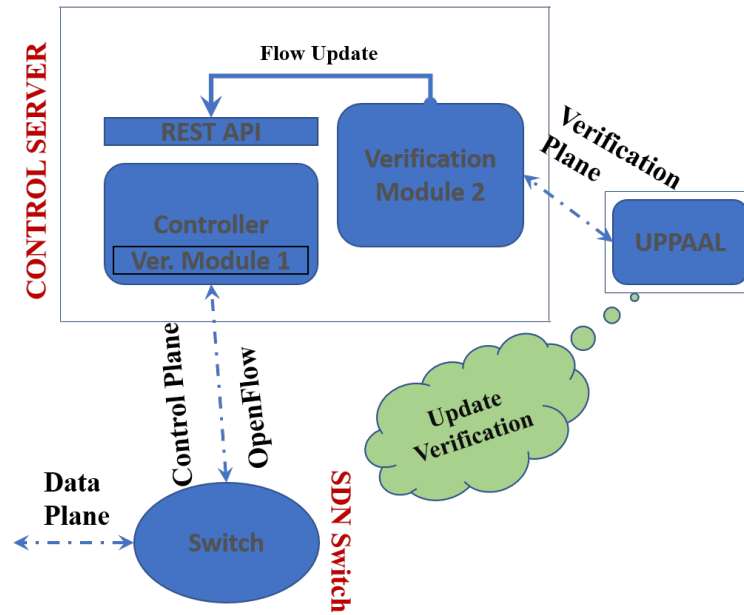


Figure 4.7: Overall Verification Scheme

The overall procedure is described in the following steps:

At the modules level

- a. The first verification module is inserted between the controller's modules and the SDN switches and it is responsible for receiving all flow insertion and modification requests and sending them for verification.
- b. When a single flow is verified, it forwards the flow down the control plane to the destination switches.
- c. If the flow is not verified, the module drops it and sends an alert back to the origin.
- d. The second verification module accepts the "to be verified" flows from the first module through a REST API. Then, this module forwards the flows to the verification tool through the verification plane.
- e. To construct an SDN topology description using UPPAAL we need to translate the data into UML where each switch is represented as a box containing

an ID, flow table, and physical characteristics. Each flow table entry has an input port, matching header entry, lifetime, and output port. Different tools were considered for this purpose including the cross-platform diagram editor DIA.

At the UPPAAL level

a. The verification tool listens for incoming updates and changes its view of the network accordingly. After that, the graph is queried using the preconfigured properties.

b. The verification graph is composed of state machines representing the switches and another graph representing the controller's different states.

c. Each node is guarding a matching rule on the incoming packet port and changes certain system variable when this rule is invoked.

d. If all the properties are verified, a notification will be sent to the verification module. In this case, the new topologies are saved. If an inconsistency flag is captured, the module is also notified and new changes to the topologies are discarded and the old view is restored.

Our proposed scheme is designed in such a way that it can detect an inconsistency or verify a flow in real-time using minimal controller resources and with minimum configuration. Not only does it verify the consistency of the inserted rules, but it can also track the physical topology changes, again benefiting from the controller's view, and query them against unexpected or unwanted behaviors.

4.3.2 Testing and Simulation

We implemented and tested each module separately to verify its correctness; more work is in progress to reach a complete dynamic system. The setup was done initially using a single Linux (Ubuntu 14.04.2) virtual machine using VirtualBox (3.2.10). We used Mininet as a network emulator, and POX and OVS version

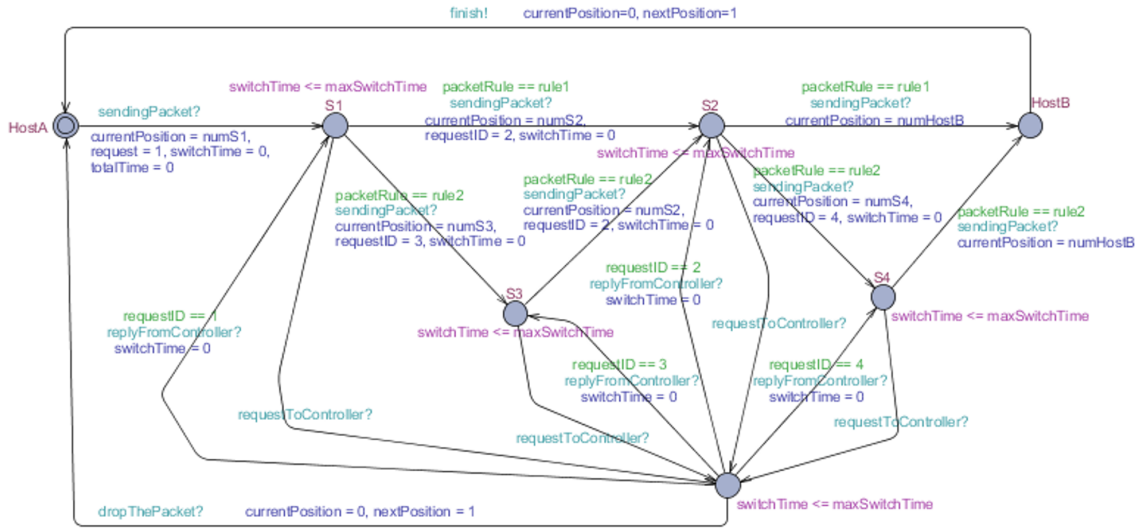


Figure 4.8: SDN Topology State Machine in UPPAAL

2.4.0 as the controller and software switch, respectively.

At the verification tool level, we implemented the SDN topologies in UPPAAL (shown in the Figure) and defined all system parameters and characteristics. Our proposed test topology is composed of two hosts, four switches and a single controller. Each node in the topology is identified by a `currentPosition` and guarding a rule that controls the packet's next position. Each time a `packetRule` is to be checked, a certain path is followed and the set of predefined rules are checked each time for any violation. The switch in UPPAAL (shown in the Figure) has two states: an idle state where the switch is in listening mode and a state that is reached when a packet is forwarded and to the idle state after that. When a single packet is forwarded, the `TopologytotalTime` variable is incremented. The time delay for a packet to exit switch A and reach switch B is defined by `sendingTime`.

The verification properties are written in UPPAAL code as follows:

- a. Loop detection: "A \square Topology.totalTime $j=15$ ". In our case we are testing the topology against a maximum of 15 counts before a loop flag is raised.

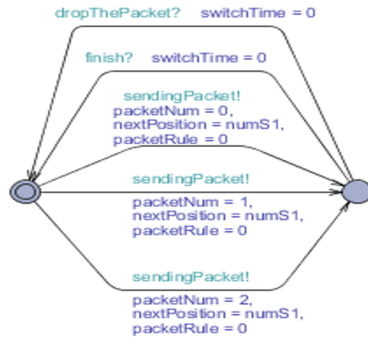


Figure 4.9: Switch States in UPPAAL

b. Deadlock (will the destination be reached): “ $A \not\models \text{Topology.HostB} \text{ imply } (\text{packetRule} == 1 \text{ or } \text{packetRule} == 2) \text{ imply } ((\text{packetRule} == 1 \text{ imply } \text{Topology.totalTime} \leq 9) \text{ or } (\text{packetRule} == 2 \text{ imply } \text{Topology.totalTime} \leq 15))$ ”. Deadlocks are tested according to the PacketRule being invoked. If a packet doesn’t reach a destination after a specific number of counts then a deadlock flag is raised.

c. Waypointing (security domains): $A \models (\text{Topology.S3} \text{ imply } \text{packetRule} == 2) \text{ and } (\text{Topology.S4} \text{ imply } \text{packetRule} == 2)$. If a certain rule leads to a packet crossing a security domain edge switch then a security flag is raised.

d. Update delay time verification (delay between controller & switch B) vs. (delay between switch A and switch B): “ $A \models (\text{Topology.S3} \text{ imply } \text{packetRule} == 2) \text{ and } (\text{Topology.S4} \text{ imply } \text{packetRule} == 2) \text{ and } \text{Delay} \leq \text{sendingTime}$ ”. Here this property is verifying that it is always true that an update from the controller can reach a specific switch before a packet can reach the same switch from the upstream switch. This specific property is mainly uses with sensitive security related data, that when installed all packets should be processed according to these rules starting from the gateway of the network. This property is topology related and a network simulator could be used to estimate the switch/controller delay times for the verification tool to test.

Figure 4.10 shows the verification process in an SDN topology view. Each switch has its own flow-table with a different set of matching rules. As a new rule is requested to be installed on a specific switch, this triggers the verification system to test this rule. UPPAAL works on installing the rule on its view of the network and queries the network as follows:

- a. The verification starts by injecting a packet from Host A destined to Host B carrying a packet P1 header PH=1. The packet is sent to S1 which checks if a rule matching P1 exists. If so, the packet is forwarded to S2 according to the output of the matching rule. In return S2 runs the rule matching mechanism similar to S1. Then, it forwards P1 to S4 which applies the same mechanism and sends the packet to Host B. In this first scenario, the verification tool will return verified since none of the rules were violated.

- b. In scenario two, a certain controller module requests to install a rule on S1 to forward the same packet PH=1 to S3 instead of S2 and this rule having higher priority than the previous one. After the request for verification reaches the verification tool, it will install the new rule over its state machine network view and begins the testing procedure. Packet P2 is injected from Host A destined to Host B and sent to S1. S1 matches PH=1 and this time sends it to S3. S3 undergoes rule matching and sends P2 to S2. After that, S2 will forward the Packet to S4. Each time a packet reaches a certain switch, the installed consistency rules are being checked. As soon as the packet reaches S4, a flag will be raised detecting a security breach since in our case we have defined a rule stating that no packet can pass through S3 and S4 at the same time.

Different tests were done using different set of rules leading the packet to take different paths from Host A to Host B. One test was done on a topology path Host A to S1 to S2 to Host B where a manual flow request was sent to be verified,

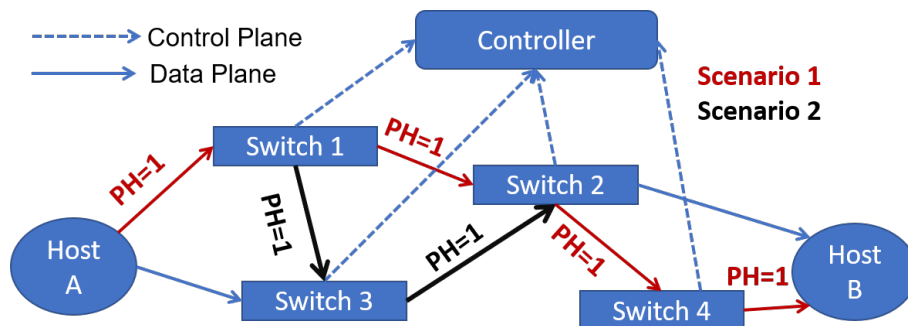


Figure 4.10: SDN Topology View of the Verification Tool

which forwards incoming packets from S1 to S2 back to S1. While testing this new flow, a loop inconsistency was flagged.

Another test was done on our topology where we manually deleted from S2 the rule that forwards incoming packets to Host B. Upon testing this case, the packet wasn't received at Host B and rule matching was done for a certain time, so the system identified that the packet is not traversing the network anymore and a deadlock violation occurred. Other violations were also captured throughout our tests done on the verification tool UPPAAL.

4.4 ARS Architecture

We have progressed a long way since the security and verification planes. Our work have advanced throughout five years of research and development in this area. Thus, the next few chapters are a summary of our final goal.

The ARS architecture consists of two main components: The ARS Security module and the ARS Consistency module. It adopted multiple features that both modules use in their data collection and processing techniques. Our architecture is based on multiple AI layers that are structured according to each module and the environment in hand. We argued that using dumb switches in a network

facing the world of developing attacks and threats is putting the security of SDN and programmable networking a step behind. Thus we discussed introducing an ARS agent, that uses low processing to achieve the necessary intelligence a switch needs to collaborate with its controller safely and face the different network threats. The details of the ARS features and architecture specific to each module are discussed and analyzed in the following chapters.

Chapter 5

ARS Security System

A resilient network takes its strength from the robustness of its security system. We will discuss throughout this chapter the different features and techniques that builds up our proposed security system.

The major impact and significance of AI-based techniques in the different domains of networking lead us to focus our work towards enhancing network security and resilience through the integration of AI techniques and programmable networks such as SDN. As such, we are trying to investigate the different AI techniques for our double layer security system and find the best combination for each. Another significant aspect in our work is handling the data efficiently in order to provide our system with the necessary data while keeping the overhead and processing load as low as possible. Our contribution is summarized as follows:

1. The proposal of a General solution for Network resiliency in a Software-based network.
2. Such a design could be adopted in different networking and communications environment regardless of the AI technique being used.

3. The proposal of a multi-layer AI-based security solution adopting new techniques with higher efficiency.
4. The proposal of an efficient technique to provide our system with the necessary data with low processing overhead.
5. Implementation and testing of the proposed designs.

5.1 System Overview

Our investigation will be carried out in three phases:

1. Edge feature extraction to ensure real-time detection, less overhead and computations, robustness, scalability and to avoid redirecting traffic to the controller to minimize the security risks at the controller level. We shall consider an edge node to be any relevant node in the network, which is specified by the environment and the case at hand. Feature extraction includes only the necessary features that are forwarded as a vector to the next destination, every time cycle.
2. A multi-layer detection technique to enable anomaly detection in one phase and attack specification in the second. This technique would allow us to detect anomalies and unknown attacks faster and at a lower processing rate.
3. Feature marking: this is where the extracted features are being marked by the ingress points. This new feature will be included in the training and decision making so that any small change in the randomness or global flow of the network would trigger a change in the output of the AI system. Feature marking would increase the efficiency, especially when dealing with DDoS and also related sequence of events that could possibly lead to an attack.

5.2 AI for Network Security

5.2.1 Applying Artificial Neural Networks for anomaly detection

Different solutions have been investigated prior to our work. The main obstacles found in these solutions was the load that the processing node should handle and the excessive traffic overhead, which might be due to packet cloning, port mirroring, or even header extraction. These duplicated data being uploaded to the controller, open the door for vulnerabilities and security issues. Another drawback of the prior work is the lack of robustness and scalability.

A new technique needs to be established in order to build a solid ground for a general security solution. Thus, we propose applying feature extraction at the data plane level, where selected input nodes in the network are assigned the task of extracting the necessary features from the packets that are passing through, and forwarding a single vector, each time cycle, to the next destination. At this point, two techniques will be proposed: 1) enabling the input nodes to send a single vector to the remote management node (i.e. the controller in SDN networks) each time cycle, containing the necessary information about the extracted features. 2) overlaying the AI technique used (e.g. artificial neural networks) over the control plane such that each node in the network handles a part of the processing load, in other words, each node handles the work of a single or multiple layers. The input layer neurons would extract the features in real-time and forward the extracted data as a vector to the next hidden layer, which performs the necessary processing and then, after passing through multiple hidden layers, the output vectors are sent to the output layer for decision making.

The extraction phase ensures low overhead and real-time extraction while

tagging the extracted features according to the input gateway (i.e. from where they entered the network). This feature marking will help in analyzing the traffic flow and studying its randomness. This process enables the security module to detect distributed attacks such as DDoS and multi-stage attacks where a sequence of events would trigger a specific attack alert.

The first technique uses the remote management processing power to apply the AI system after receiving the necessary input data from the network. Such a solution provides faster processing and quick mitigation after the detection stage.

The second technique applies an independent overlay solution that is managed by the controller; it can act independently throughout the feature extraction and the detection stage. Only a minimal processing load is given to each node in the network and no traffic is being duplicated or forwarded, which classifies this process as a safe environment on any programmable network.

Both techniques were studied and tested for advantages and disadvantages and then compared in different environments. The goal is to use these general techniques and modify the top AI methods so that they can be applied in the proposed context for anomaly detection and attack identification and mitigation.

5.2.2 Designing a two-step process for anomaly detection and attack identification

As discussed earlier, we intend to design a security solution as a two-step process. We aim to identify anomalies within the network traffic, which may be triggered by a network architectural change, a network administrative change, or a security issue. As investigated in the literature, anomaly detection requires less processing and fewer features to be tested, and in our case, it would be more efficient since

not only the traffic flow is being studied but also its randomness. As such, we propose to apply the anomaly detection as an initial stage, followed by the attack identification process, only if an anomaly trigger is set.

Other advantages of such an architecture is the ability to detect unknown attacks. This is possible when a trigger is set in the first stage, and the second stage is not able to identify the attack type. At this point, an independent layer with new techniques would be used to teach the system of the new attack type to be able to label such flows in the future according to its features. Another advantage is the fact that there is no need to retrain the whole system in case of new attacks; we only need to train the second layer offline while the first layer remains working online.

5.2.3 Designing an Ensemble of AI techniques

Reaching a satisfactory solution requires integrating the intelligence of different multipurpose AI techniques to create a self-learning system that is capable of dealing with unexpected situations on its own. The literature and our tests showed that different techniques tend to perform better in different environments, under different circumstances, and for different goals. We aim to reach a suitable solution that strikes a good balance among our goals. Different sets of AI techniques would be used for different environment, thus providing robustness and effectiveness for our system.

5.3 ARS Architecture OverView

The two proposed architectures are based on either: Distributed Extraction, Centralized Processing, and Centralized management ($D \sim C^2$), or Distributed

Extraction, Distributed processing, and Centralized management ($D^2 \sim C$).

Figure 5.1 shows an overview of the $D \sim C^2$ design where, on one hand, the Artificial Intelligence Resiliency System (ARS) agent resides on the edge nodes for feature extraction and marking. On the other hand, the ARS security and consistency solution sits on the remote management side, which also handles the management and monitoring part. Figure 5.2 shows the $D^2 \sim C$ design with the ARS AI agent residing on the edge nodes for feature extraction and marking, also handling the processing of part of the AI-based technique and exchanging relevant data. In this design, the remote management handles the monitoring and management part, including optimizing the distribution of the processing and exchange tasks in the overlay network.

5.3.1 The $D \sim C^2$ architecture can be described as follows:

1. The remote management handles the control and monitoring part including choosing the edge nodes.
 2. The edge nodes handle the distributed extraction and marking.
 3. The remote management handles the centralized processing.

5.3.2 The $D^2 \sim C$ architecture can be described as follows:

1. The remote management handles the control and management part including choosing the edge nodes and identifying the ARS AI agents that will handle the distributed processing.
 2. The edge nodes will handle the distributed extraction and marking.
 3. The distributed processing is in the form of an AI-based overlay network (example neural network) as illustrated in Figure 2. Each node would handle

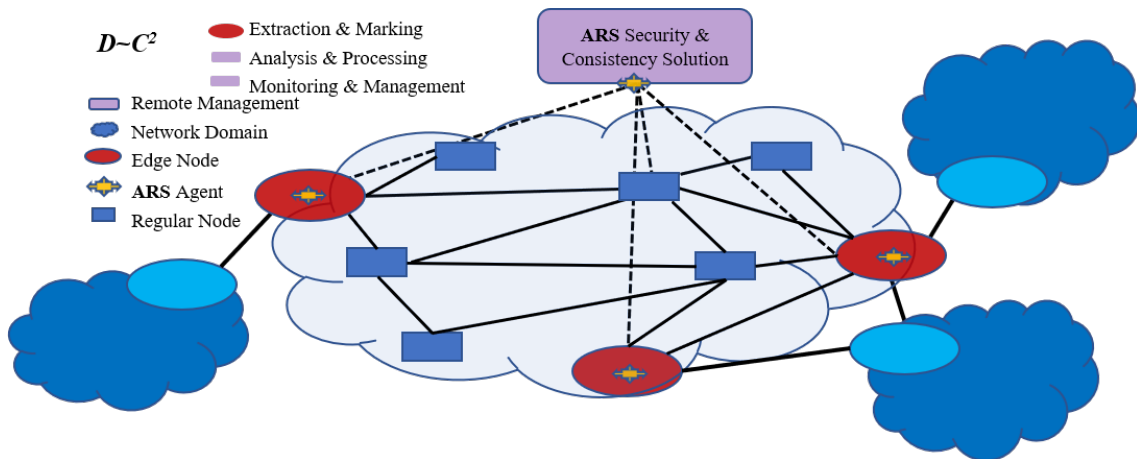


Figure 5.1: Proposed $D^2 \sim C$ Architecture

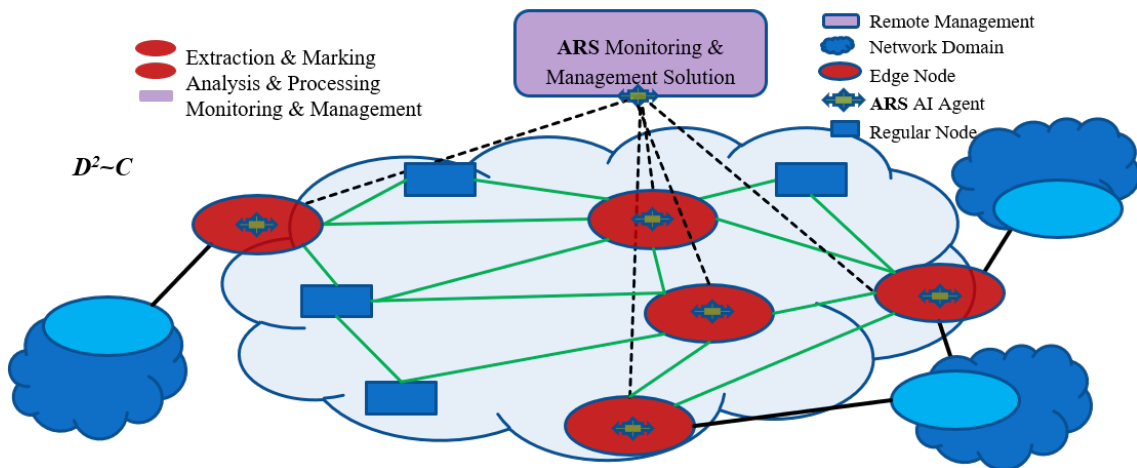


Figure 5.2: Proposed $D^2 \sim C$ Architecture

a portion (eg.layer) of the processing that may be the work of a single layer or more depending on the environment and different parameters.

4. The remote node will handle the monitoring of the entire process along the way.

Figure 5.3 shows a simple neural network architecture that include 3 inputs, 1 hidden layers, and a single output layer. This scenario is overlaid over our $D^2 \sim C$ architecture (Figure 5.4) using 3 edge and 2 internal nodes such that each edge ARS AI agent will play the roles of an input node and internal agent for

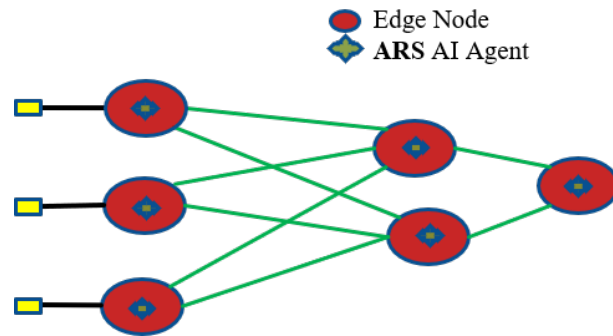


Figure 5.3: Simple Neural Network Architecture

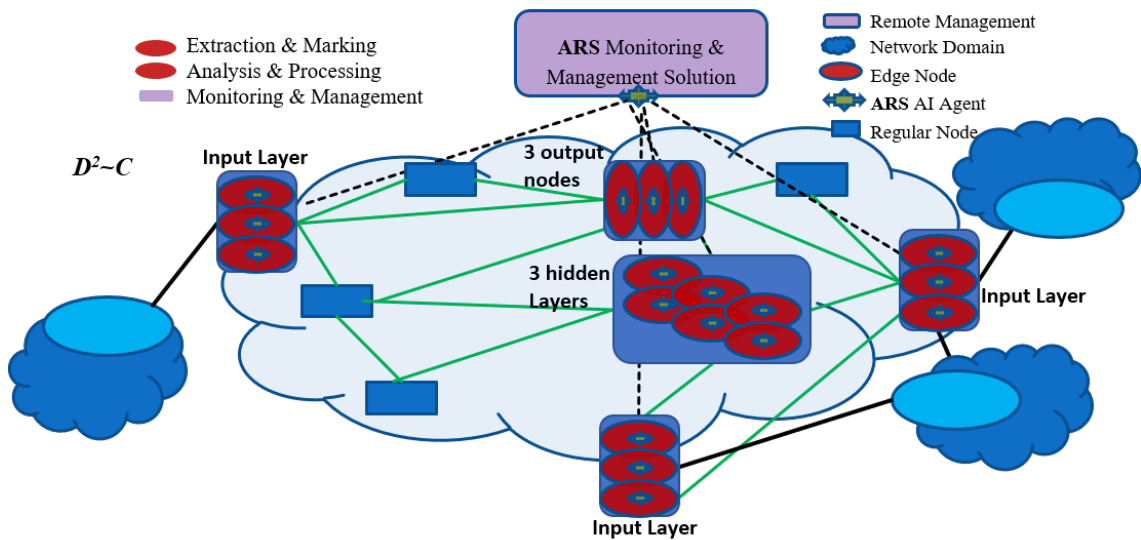


Figure 5.4: Distributed NN Overlay Architecture

the hidden layer, and another one for the output layer. The distribution would vary according to the available resources. Thus we managed to overlay 3 NN networks over our SDN network using 5 nodes. Such technique allows scalability such that we are able to implement a large Neural Network with multiple layers and multiple neurons in each layer over a small network. The number of NN overlays depend on the number of edge nodes in the network. The distributed NN overlay is clearly illustrated in figure 5.4.

5.4 ARS System discussion

The ARS system aims to benefit from the advantages from both centralized and distributed security techniques. As discussed, ARS runs in two modes. The first mode integrated a double layer of defense such as:

Distributing NN overlays to secure the ingress points of the network, thus securing the network from both external attacks on the network and leaking specific information from certain servers in the network through these points. This layer of security handles detecting anomalies in the network. We defined anomalies as unfamiliar traffic flows that were not taught to the NNs during the idle phase (training phase of the network). As traffic is entering the network the extraction phase is processing the packets in real time. The extracted features are structured in vectors till the next time cycle. In this mode the vectors are processed as input features at the edge agents before being forwarded to the designated internal agent for the hidden layer processing. After that the results are forwarded to the next internal agents for the output processing. The output is either disregarded if the traffic is normal or pushed to the controller if the output is abnormal.

At this stage, and as an abnormal output is triggered, a predefined packet is sent to the edge agent carrying the vector ID responsible for the alarm. As a pre-mitigation technique, the edge agents will install a predefined rule to block the corresponding source-destination IP pair. The same packet is sent from the output agent to the controller to sync the matching table of the corresponding edge switch.

Functioning in a centralized fashion is the next layer of this mode, which starts after an alarm is triggered. After the pre-mitigation process is complete,

the edge agent will send the corresponding vectors to the controller. Before this stage, the AI system at the controller level trained to detect specific types of attack, that may vary from one environment to another. Our chosen set is discussed in the final chapter. So, as the set of vectors reach the controller to the security module, they are treated as input vectors for the AI system. After that the system will try to identify the attack type. If the identification stage is successful, the security module can either keep the pre-mitigation rules or insert new ones from a predefined set. Else, if the AI system could not identify the attack with a high confidence score; such that the attack has similar features to multiple attack types but not identical to any; then the system will treat it as an unknown attack and add its symptoms to the database. At this point the administrator could intervene, such that the new behavior could be a new normal behavior that was recently introduced to the network. If this is the case the administrator can label it as normal, else it would be given an ID and added as a new attack. In either case the new features will be trained offline to the system and the new weights will be aggregated with the old ones to include the new behavior. During this phase the network will stay secured by the first layer, while the second layer will only become offline during the aggregation phase at a specific time chosen by the administrator or chosen randomly from an interval for more security.

The second mode of our system is advised for environments with less processing power at the switch side, or networks that require security systems with a large AI processing requirements that are over the tolerance of the network devices. Other reasons could be considered also. This mode works over the same ARS architecture till the point where the features are extracted at the edge node and structures in vectors. At this point, and after a time cycle ends the vectors are

uploaded to the controller for the necessary security processing. In this mode both AI layers are centralized at the controller level. Same as the previous mode, the first AI layer is responsible for anomaly detection and the second for the attack specification stage. The second layer is only functional after an alarm is triggered from the first layer, and undergo the same data exchange as the first mode. As does the first mode, the second mode benefits from the double layer technique to detect unknown or new behaviors (attacks) in the network. Both modes, and due to the common ARS architecture in general and the edge feature extraction and vector structure in specific, ensure data privacy along with security in a near realtime detection system. All the necessary tests and results are discussed in the final chapter.

Chapter 6

ARS Consistency System

We strongly argue that the necessity of network verification and consistency establishment equals the importance of enforcing security measures in a vision for a resilient network. As attackers and attack strategies are developing, their focus deviated from volume-based attacks to low-profile specific target attacks that have more impact on the network.

Our goal to enhance network resiliency drove us to direct the influence of AI-based systems towards the network consistency domain. Therefore, we researched the diverse AI techniques alongside an efficient data management system. We focused on preserving data privacy while supplying our system with the required information while maintaining the processing load and overhead as low as possible. Our proposals and contribution are summarized as follow: 1. An AI-based multi-layer consistency solution adopting new techniques with higher efficiency. 2. An effective technique aiming to provide our systems with the required data while preserving data privacy and maintaining an acceptable processing overhead. The proposed designs were implemented and tested, as discussed later, with the results presented in the following chapters. Our proposed system relies on a hy-

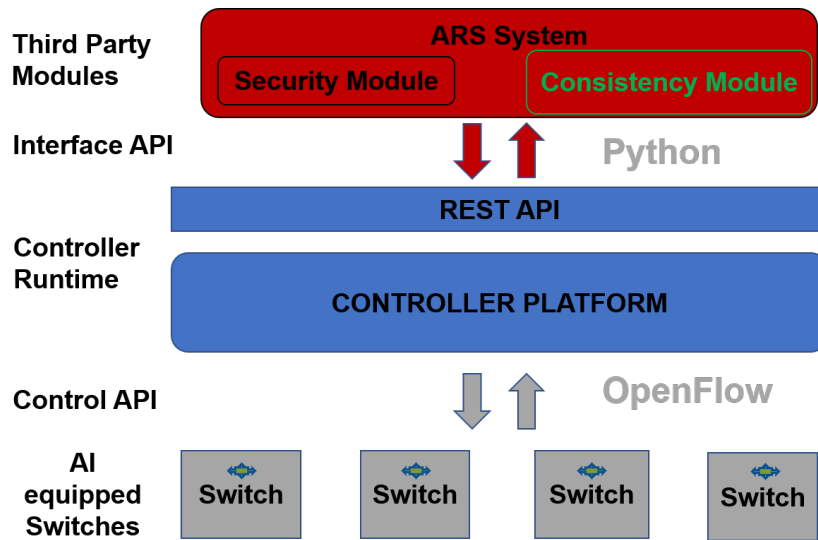


Figure 6.1: System architecture Distribution

brid architecture that includes both centralized processing at the controller level and distributed processing at the nodes level over the data plane.

6.1 General ARS Architecture Overview

The proposed architecture is based on:

Distributed Extraction, Centralized Processing, and Centralized management (D C2).

The ARS agent resides on the network nodes for feature extraction. The ARS consistency module resides on the controller level, where the monitoring and management processing is handled as shown in Figure 2.

The D C2 architecture can be described as follows: a) The controller handles the monitoring and management of the network including the ARS agents. b) The nodes handle the distributed extraction and data hashing phase, including vector assembly and forwarding. c) The controller performs the centralized verification processing and analysis.

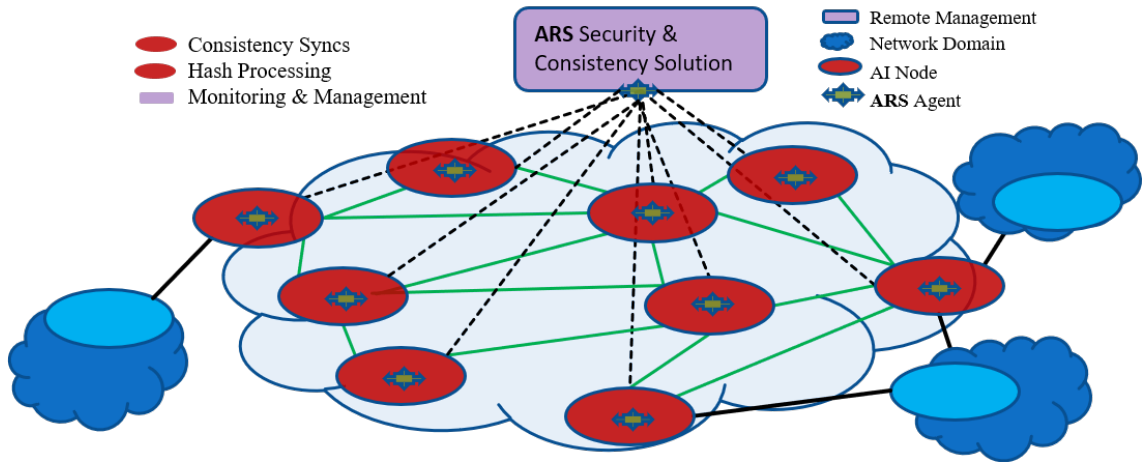


Figure 6.2: System Consistency Establishment Architecture

6.2 Research Investigation Phases

Our research towards an efficient consistency establishment system focused on overcoming the main security issues in the techniques that were proposed in the literature. These issues mainly concern privacy due do sensitive data being exchanged for processing in a remote location, or the lack of authentication of the output of such checks. In addition to the processing overhead and latency associated with these solutions.

Our research investigations focused on three phases:

1. Distributed feature extraction to preserve data privacy and limit traffic uploaded to the controller, minimizing security risks. We are considering distributed extraction from the node's flow table itself. The extracted features will undergo a hash process, and the output is then forwarded in a vector to the destination, every time cycle, tc.

2. The multi-layer verification technique provides consistency verification using two layers: the information vectors from the data plane are uploaded to the controller to be verified against the controllers view of the network using a fast

processing technique. Any inconsistency in the first stage would trigger a second layer to try to identify the class of the flow-based attack that triggered this event. A third layer is available in case of detecting an unknown consistency attacks.

3. Feature granularity allows us to relate any inconsistency triggered by the output of the AI system to a specific node in the network and in particular to the exact flow in that node, in addition to relating the attack to specific features being altered in that corresponding flow.

6.3 Proposed Consistency System

Inspired by the brain-awareness human consistency techniques, that is also part of the total human security system, we are proposing a network consistency establishment system that relies on the controller's complete view of the network to check for any misbehavior in the network nodes. Such technique would increase our confidence regarding the correctness of the network nodes, before starting an external network security analysis.

For the controller to check for inconsistencies every time cycle, t_c , it should be able to know what each node should be doing at every time t . The consistency module at the controller would enable the controller to save an updated image of the matching (flow) table of each node. Before any consistency check, the module should make sure that no updates are being injected at this time. At t_c , the module would probe the security agent at the node level for a hash of the extracted features from the local matching (flow) table. This process ensures the security and privacy of this sensitive data not being exchanged through the network. At the same t_c , the module would calculate its own local hash outputs. Since no updates are done at this time, the output of each network node (actual

behavior) should match the output of its corresponding image at the controller (intended behavior).

Each agent will send a single vector belonging to a single node containing the hashed features each tc. The extracted features from each node are divided into three classes:

1. Constant features: these are features that do not change with time: Source Ip address, Destination Ip address, Source Mac address, Destination Mac address, Input port, Action.

2. Time features: these are features that show the age of each flow and the duration since the last update: Duration, hard_age.

3. Statistical features: these are features that are not extracted directly from the flow table but calculated by the ARS agent during a tc. Features such as the update rate of each flow during a tc and a flow overlap check on each flow. These features are: Update_rate, Check_overlap.

We have enabled our ARS agent to extract the necessary features from the local flow table and keep count of the flow updates each tc. Another task was to query the OVS node using the Check_overlap command, which was added to OpenFlow 1.4 [81]. A third and important task was to perform a hash function on the three classes of extracted features. Such a technique preserves both security and privacy, since no sensitive data is being exchanged in the clear throughout the network. The goal is to provide the system with the necessary data while keeping a good balance between a secure hash and the required processing overhead. Furthermore, to secure the node from physical attacks and to prevent an attacker from generating his own fake hashes, we propose to pre-install a key in a tamper proof device on each node. This device will be accessed explicitly by our ARS agent and a set of these keys would be also stored on the controller to complete

the consistency checks.

At this stage, the ARS agent is able to feed the hash function with a string input of the features at each predefined time cycle, t_c , and extracting a fixed output and forwarding it to the controller. At the same time, the controller would calculate the same hash output using each key and the same features of each node. Next, the consistency module will consider each flow of each node in the network as a set of three binary matrices representing the three class features described earlier. Here, the module would compare each class matrix with its corresponding actual class from the network nodes in order to prepare the input data for our AI system.

Another key feature in our technique is applying a flow-based verification, such that any small inconsistency or attack on a single flow in any network node would trigger a clear alarm in the output of the AI system due to the double layer of hash and compare techniques applied on the extracted features of each flow. The data preprocessing stage is shown figure 6.3 and detailed in the following chapters.

6.4 Consistency system analysis

Our research towards an efficient consistency establishment system focused on overcoming the main security issues in the techniques that were proposed in the literature. These issues mainly concern privacy due do sensitive data being exchanged for processing in a remote location, or the lack of authentication of the output of such checks. In addition to processing overhead and some latency introduced by these solutions.

We have relied on the existing structure of our security system, in particular

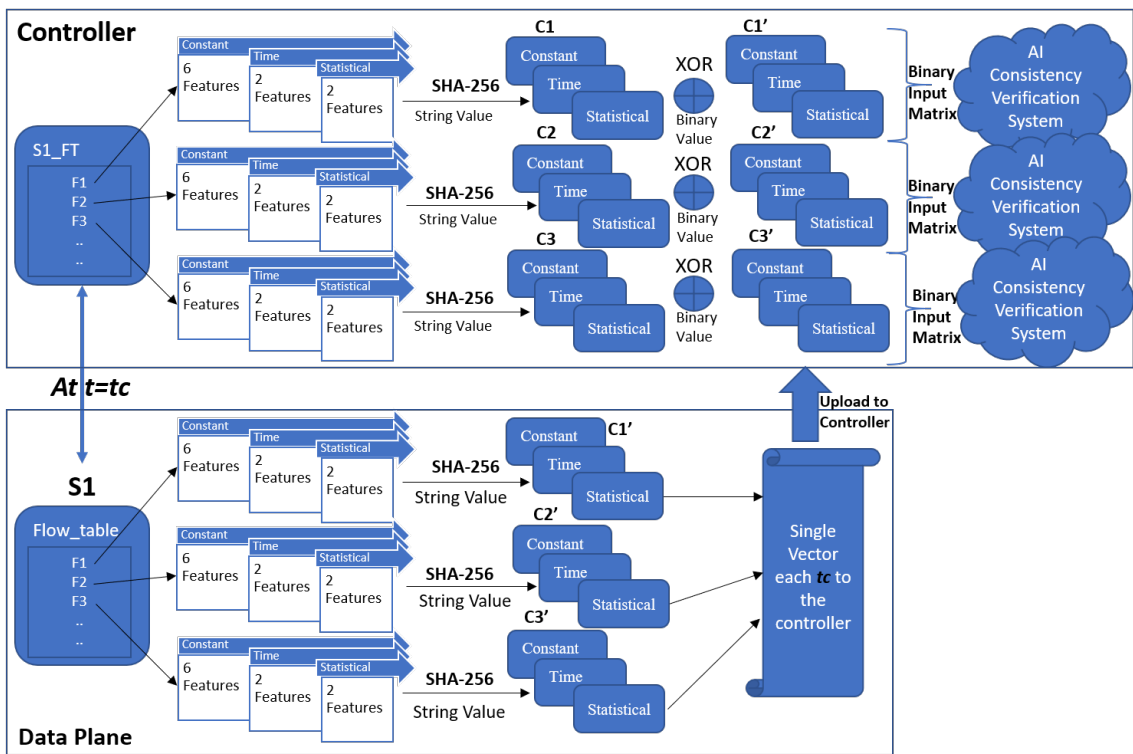


Figure 6.3: Data extraction and processing phases

our ARS agent, and equipped it with the ability to perform an authenticated hash function of the local flow table. In order to ensure both security, authentication, and also preserve privacy, we have chosen the HMAC-SHA2 to perform this task. Such a choice provides a good balance between a secure hash and the required processing overhead. The SHA-2 (SHA-256) function accepts variable size inputs and produces a fixed 256 bits (32 bytes) size output, which can be carried out in our security vector to the controller, in order to minimize the required traffic overhead. Another important feature of HMAC is providing authentication of the delivered hash, therefore protecting against replay and man in the middle attacks. This feature was possible through generating two key pads from the original key for both the authentication and hashing processes. Finally, to secure the node from physical attacks and to prevent an attacker from generating her own fake hashes, we propose to pre-install each key in a tamper proof device on each node. This device will be accessed explicitly by our ARS agent and a set of these keys would be also stored on the controller to complete the consistency checks.

At this stage, the ARS agent is able to feed the HMAC-SHA2 function with a binary input of the flow table at each predefined time cycle, tc' , and extracting a 32-byte output and forwarding it in our security vector to the controller. At the same time, the controller would calculate the same hash output using each key and the flow table image of each node. After this stage, the consistency module would compare each hash with its corresponding actual hash from the network nodes in order to check for any inconsistencies between them. The security module will be informed of the output of each check to act accordingly.

Chapter 7

ARS AI Optimization

7.1 AI Optimization

In the machine learning based security community we should have more work with probabilistic models and uncertainty, for their importance in decision making, especially when dealing sensitive or classified networks or data. This probabilistic view of machine learning offers confidence bounds for data analysis and decision making, information that a security network administrator for example would rely on to analyze his/her data, or, for a more sensitive issue, an autonomous car would use to decide whether to brake or not. In analyzing data or making decisions, it is often necessary to be able to tell whether a model is certain about its output, being able to ask “maybe I need to use more diverse data? or change the model? or perhaps be careful when making a decision?”. Such questions are of fundamental concern in machine learning.

When using deep learning models, we generally only have point estimates of parameters and predictions at hand. The use of such models forces us to sacrifice our tools for answering the questions above, potentially leading to situations

where we can't tell whether a model is making sensible predictions or just guessing at random. Most deep learning models are often viewed as deterministic functions, and as a result viewed as operating in a very different setting to the probabilistic models which possess uncertainty information. Perhaps for this reason it is quite surprising to see how close modern deep learning is to probabilistic modeling. In fact, we shall see that we can get uncertainty information from existing deep learning models for free—without changing a thing. The main goal of this chapter is to work on such practical tools to reason about uncertainty in deep learning.

AI optimization starts from Evaluating and calibrating an AI model after testing it. Evaluating a model is just as important as creating the model in the first place. Calibration is a post-processing technique to improve error distribution of a predictive model. The evaluation of machine learning (ML) models is a crucial step before deployment. It is essential to assess how well a model will behave for every single case. In many real applications, along with mean error of the model, it is also important to know how this error is distributed and how well probability estimations are made. Many current ML techniques are good in overall results but have a bad distribution assessment of the error.

From the scientific context, the primary goal of ML methods is to build a hypothesis (model) from a given data set. After the learning process, the quality of the hypothesis must be evaluated as precisely as possible.

Our ARS system implements both Binary classifiers, such as the anomaly detection techniques where the output is each normal or abnormal, and probabilistic techniques, such as the attack identification techniques where we need the output to be a more probabilistic vote between the given attack classes, rather than a point estimate. A summary of such classifiers is discussed in the following

sections.

7.1.1 Binary classifiers

When dealing with two class classification problems we can always label one class as a positive and the other one as a negative class. The test set consists of P positive and N negative examples. A classifier assigns a class to each of them, but some of the assignments are wrong. To assess the classification results we count the number of true positive (TP), true negative (TN), false positive (FP) (actually negative, but classified as positive) and false negative (FN) (actually positive, but classified as negative).

- $TP + FN = P$ and

- $TN + FP = N$

The classifier assigned $TP + FP$ examples to the positive class and $TN + FN$ examples to the negative class. Let us define a few well-known and widely used measures:

- $FPrate = FP / N$

- $TPrate = TP / P = Recall$

- $Yrate = (TP + FP) / (P + N)$

- $Precision = TP / (TP + FP)$

- $Accuracy = (TP + TN) / (P + N)$.

Precision and Accuracy are often used to measure the classification quality of binary classifiers.

7.1.2 Probabilistic classifiers

A probabilistic classifier is a function $f : X \rightarrow [0, 1]$ that maps each example x to a real number $f(x)$. Normally, a threshold t is selected for which the examples where $f(x) \geq t$ are considered positive and the others are considered negative. This implies that each pair of a probabilistic classifier and threshold t defines a binary classifier. Measures defined in the section above can therefore also be used for probabilistic classifiers, but they are always a function of the threshold t . Note that $TP(t)$ and $FP(t)$ are always monotonic descending functions. For a finite example set, they are step-wise, not continuous. By varying t we get a family of binary classifiers.

A summary of some calibration techniques is discussed in the following sections:

7.1.3 Class Calibration (CC)

CC is the degree of approximation of the true class distribution with the estimated class distribution. The standard way to calibrate a model in this way is by changing the threshold that determines when the model predicts “A” or “B”, making this threshold stricter with class “A” and milder with class “B” to balance the proportion.

7.1.4 Probabilistic Calibration (PC)

PC is a classifier which accompanies each prediction with a probability estimation. If we predict that we are 99% sure, and if we are only right 50% of the time, this is not calibrated because our estimation was too optimistic. Similarly, if we predict that we are only 60

7.2 General Discussion

Neural Networks are commonly used in classification and decision tasks. In this chapter, we focus on the problem of the confidence of their results. We will present an overview of the existing confidence measures and finally discuss a simple measure which combines the benefits of the probabilistic interpretation of network outputs and the estimation of the quality of the model. Our test results done on our modules are discussed and show that the simplest measure behaves often better than more sophisticated ones.

Unlike for classification problems, where machine learning models usually return the probability for each class, regression models typically return only the predicted value. In order to calculate the model's confidence, we need to re-engineer our models to return a set of (differing) predictions each time we perform inference. We can then use the distribution of these predictions to calculate the model's confidence intervals. we'll show you how we can do this for any neural network, including those you've already trained. Our implementation will be in Keras — a popular library for prototyping deep learning models.

7.2.1 Dropout

A very popular method of regularization of neural networks is dropout. This idea is actually very simple - every unit of our neural network (except those belonging to the output layer) is given the probability p of being temporarily ignored in calculations. Hyper parameter p is called dropout rate and very often its default value is set to 0.5. Then, in each iteration, we randomly select the neurons that we drop according to the assigned probability. As a result, each time we work with a smaller neural network. The figure below shows an example of a neural

network subjected to a dropout.

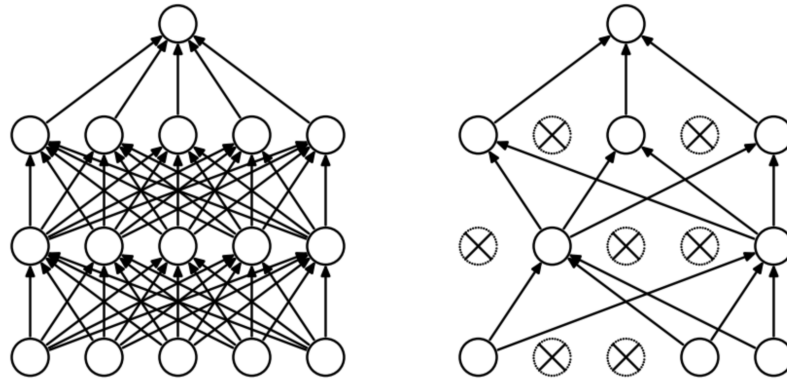


Figure 7.1: Applying dropout to a simple Neural Network (Hidden Layers)

The effectiveness of this method is quite surprising and counter-intuitive. Let us look at this problem from the perspective of a single neuron. Since in each iteration, any input value can be randomly eliminated, the neuron tries to balance the risk and not to favour any of the features. As a result, the values in the weight matrix become more evenly distributed. The model wants to avoid a situation in which the solution it proposes, will no longer make sense, because it no longer has information flowing from an inactive feature.

To turn our single-valued regression model into one capable of returning multiple (different) predictions, we will re-purpose a technique we usually use only during training — dropout. When we apply dropout we “turn off” a randomly chosen fraction of the units of the model (or certain layers of the model). When training this helps prevent overfitting, by reducing co-adaptation between units and so forcing all units to generalize well to unseen data.

By applying dropout when performing inference, we are therefore sampling one of these “thinned” networks to generate our predictions. By sampling enough times, we can build up a distribution of predictions from our single trained model, and use this distribution to calculate the confidence intervals of the original (un-

“thinned”) model’s predictions.

Choosing the right dropout

In the beginning of our tests we set dropout = 0.5, however we have no idea if this is the correct amount of dropout to use. If the dropout is too large then the predictions generated will be very diverse, and so the confidence intervals estimated from them will be too large. Conversely if the dropout is too small then the predictions generated will be too similar, and so the confidence intervals will be too small. We can judge the suitability of the dropout by looking at the distribution of its predictions around the median. For a dropout of 0.5 if the distribution of predictions is much broader than the errors, so the amount of dropout needs to be reduced. An example is shown in the figure 7.1:

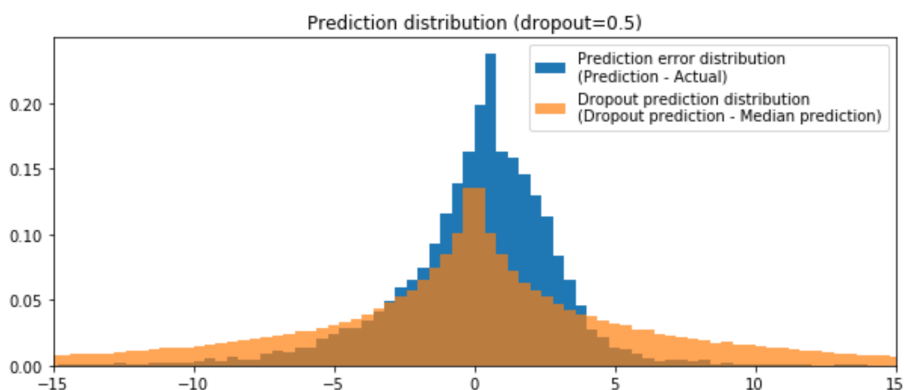


Figure 7.2: dropout interval distribution at d=0.5

To determine the optimal dropout to use, we can look at the percentage of actual values which fall within each calculated confidence interval. For the optimal dropout we would expect 10% of actual values to fall within the 10% confidence interval, 20% within the 20% and so on. To choose the optimal dropout value, we calculate the percentage of actual values within the various predicted confidence intervals for a range of different dropout values.

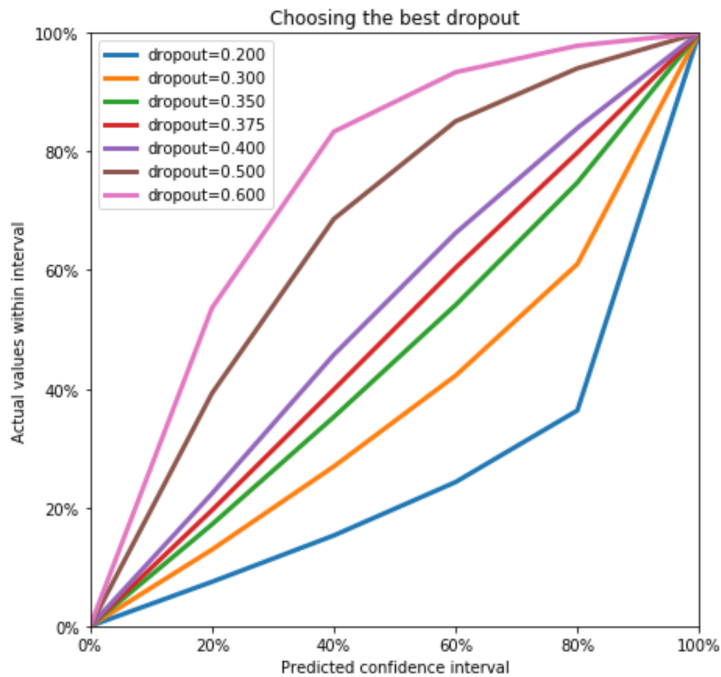


Figure 7.3: dropout interval plot for different dropout values

When the dropout is too large the confidence intervals are overestimated: for a dropout of 0.5, 40% of actual values fall within the 20% confidence interval. When the dropout is too small the confidence intervals are underestimated: for a dropout of 0.2, 20% of actual values fall within the 50% confidence interval. However, when the dropout value is just right, the confidence interval matches the distribution of actual values almost perfectly, in this case for a dropout value of 0.375.

Plotting the distribution of predictions for the optimal dropout of 0.375, we see it matches the prediction error pretty well:

Working with dropouts in testing mode

We have modified a function in Keras (`dropout_predict`) to take the configuration and weights from our pretrained model, and use them to create a new model with

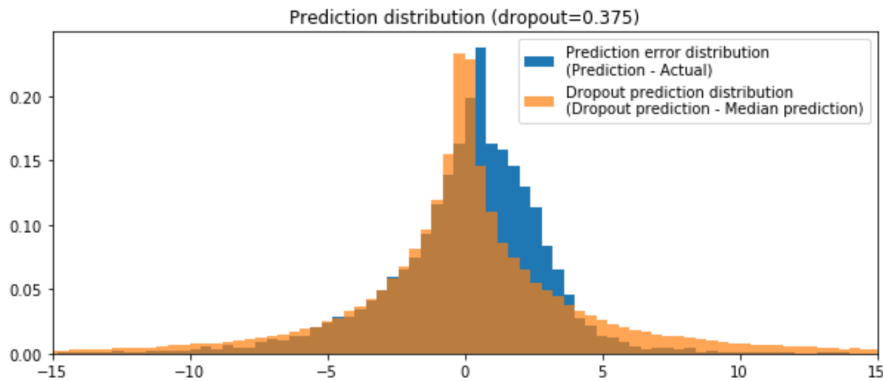


Figure 7.4: dropout interval distribution at $d=0.375$

the specified amount of dropout applied to all layers. This is done by looking for the layers which contain dropout and setting the dropout to the desired value.

However, Keras turns off dropout by default when performing inference, so we cannot simply use this new model to generate our predictions. Instead we have to trick Keras into thinking we are still training the model, and so still using the dropout. This is done by setting the learning_phase to 1. Therefore, we create a separate predict function (`predict_with_dropout`), which takes both the model inputs and learning phase, and returns the model outputs.

Creating confidence intervals

We predict with dropout 20 times, giving us 20 different predictions for each sample in the input data. With more predictions the confidence interval estimates will become more accurate, however the prediction process will last longer. The use of 20 predictions therefore seems a fair compromise. From these predictions it is then trivial to calculate the upper and lower limits for a given confidence interval.

Such a technique can be done offline on all our AI NN models (NN, DNN, CNN). Therefore, we created an optimization module that works on evaluating

and calibrating the AI system through retesting the modules using the incoming data in dropout mode to calculate our confidence measures and then compare the results with the original ones and logs them for future calibrations.

After our test we have seen that without enabling our third layer module, that is now responsible for unknown attack detection and confidence estimation, different types of new attacks were classified as one of the four attack classes. This provides us with a conclusion that high accuracy rates in an AI system doesn't always mean that we are dealing with a good system regarding security or any other domain. We divided our module into three parts. The first would analyze the distribution of the confidence vector in order to classify new attacks. The second would retrain the system offline with the data of the new attack and integrate the new weights. The third part is responsible for calculating all dropout estimates and logging all confidence vectors with two class estimates less than a threshold for future administrative analysis and interventions.

Our system was also tested using dropouts after being trained using both the presented datasets. Both the DNN network of the second security layer (attack specification), and the CNN network of the second consistency layer (consistency class specification) were optimized using the dropout technique implemented in the third layer of each system. The dropout technique was further optimized through celebrating the parameters that best fit our test network and data. The parameters calibrated were the precision τ (10, 20, 50), dropout probability p (0.3 0.5 0.7), test repetitions (10 13 16 20). The tests were made to find a suitable compromise between the best accuracy, converged confidence interval and processing overhead. First the tests were repeated fixing p and varying τ to increase the output accuracy in the testing stage. The best accuracy was obtained with $p=0.5$ and $\tau=20$ for the security system and $p=0.6$ and $\tau=20$ for the consistency

system. Regarding the processing overhead (we inserted a new security attack that contains feature from both a DoS attack and a port scanning attack) (we also inserted a new consistency attack containing features from both DoU class and the deadlock class). A compromise was done between the overhead from the (10 13 16 20) repetitions and the convergence of the output probabilities between the two related classes in each test (security and consistency). Regarding the probabilities in the confidence interval of the security test, they varied between 15% and converged at 7%, after 13 repetitions, difference between the two classes. Regarding the consistency test the probabilities varied between 12% and 5%, after 16 repetitions. These tests will control the threshold decision that will be set to check each confidence interval for decision confidence, such that if the probabilities of two classes are less than a threshold then a warning will be set showing that the confidence of this decision is low. In the cases of the normal class being one of these classes then an alert will be set providing a probability of having a false positive or true negative.

7.2.2 Overfitting

Thanks to a huge number of parameters (thousands and sometimes even millions) neural networks have a lot of freedom and can fit a variety of complex datasets. This unique ability has allowed them to take over many areas in which it has been difficult to make any progress in the ‘traditional’ machine learning era. Sometimes, however, their greatest advantage becomes a potential weakness. Lack of control over the learning process of our model may lead to overfitting - situation when our neural network is so closely fitted to the training set that it is difficult to generalize and make predictions for new data. Understanding the origins of this problem and ways of preventing it from happening, is essential for

a successful design of NN.

In practice, detecting that our model is overfitting is difficult. It's not uncommon that our trained model is already in production and then we start to realize that something is wrong. In fact, it is only by confronting new data that you can make sure that everything is working properly. However, during the training we should try to reproduce the real conditions as much as possible. For this reason, it is good practice to divide our dataset into three parts - training set, dev set (also known as cross-validation or hold-out) and test set. Our model learns by seeing only the first of these parts. Hold-out is used to track our progress and draw conclusions to optimise the model. While, we use a test set at the end of the training process to evaluate the performance of our model. Using completely new data allows us to get an unbiased opinion on how well our algorithm works.

It is very important to make sure that your cross-validation and test set come from the same distribution as well as that they accurately reflect data that we expect to receive in the future. Only then we can be sure that the decisions we make during the learning process bring us closer to a better solution. I know what you are thinking about. . . “How should I divide my dataset?” Until recently, one of the most frequently recommended splits was 60/20/20, but in the era of big data, when our dataset can count millions of entries, those fixed proportions are no longer appropriate. In short, everything depends on the size of the dataset we work with. If we have millions of entries at our disposal, perhaps it would be better idea to divide them in 98/1/1 ratio. Our dev and test sets should be simply large enough to give us high confidence in the performance of our model. Recommended methods of dividing the dataset according to its size are illustrated in figure 7.5.

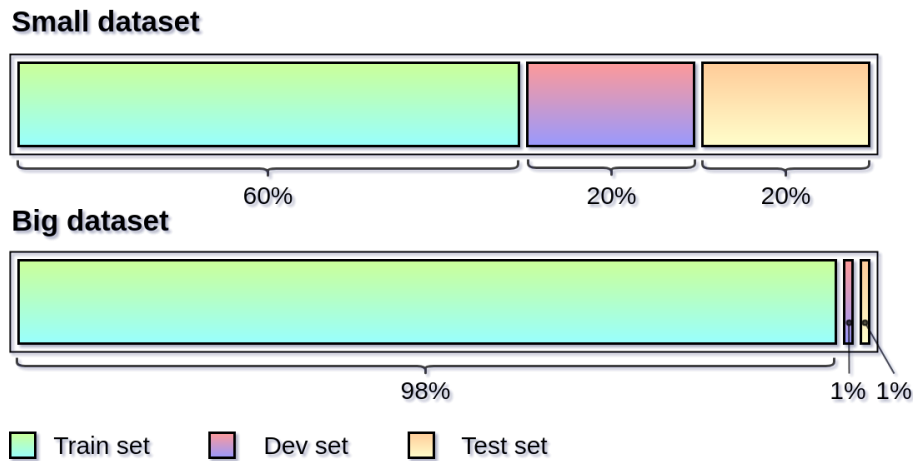


Figure 7.5: Recommended method of dividing the data set

Ways to prevent Overfitting

There are many methods that can help when our neural network has a high variance. Some of them, such as obtaining more data, are quite universal and work well every time. Others, such as regularization, require a lot of finesse and experience. Imposing too many restrictions on our NN may compromise its ability to learn effectively. We will present some of the most popular methods of reducing overfitting and discuss the reasons they work. Dropouts is one of the methods used, in the training phase, to prevent overfitting.

L1 and L2 Regularizations One of the first methods we should try when we need to reduce overfitting is regularization. It involves adding an extra element to the loss function, which punishes our model for being too complex or, in simple words, for using too high values in the weight matrix. This way we try to limit its flexibility, but also encourage it to build solutions based on multiple features. Two popular versions of this method are L1 - Least Absolute Deviations (LAD) and L2 - Least Square Errors (LS). Equations describing these regularizations are given below.

In most cases the use of L1 is preferable, because it reduces the weight values of less important features to zero, very often eliminating them completely from the calculations. In a way, it is a built-in mechanism for automatic feature selection. Moreover, L2 does not perform very well on datasets with a large number of outliers. The use of value squares results in the model minimizing the impact of outliers at the expense of more popular examples.

$$J_{L1}(W, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \lambda \|W\|_1 \quad \|W\|_1 = \sum_{j=1}^{n_X} |W_j| \quad (7.1)$$

$$J_{L2}(W, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \lambda \|W\|_2 \quad \|W\|_2 = \sum_{j=1}^{n_X} W_j^2 \quad (7.2)$$

Lambda factor and its effect In the previously mentioned formulas for regularization in both versions of L1 and L2, the hyper-parameter λ was introduced — also called regularization rate. When choosing its value we try to hit the threshold between simplicity of our model and fitting it to the training data. Increasing the λ value also increases the regularization effect.

Early Stopping The graph below shows the change in accuracy values calculated on the test and cross-validation sets during subsequent iterations of learning process. We see right away that the model we get at the end is not the best we could have possibly create. To be honest, it is much worse than what we have had after 150 epochs. Why not interrupt the learning process before the model starts overfitting? This observation inspired one of the popular overfitting reduction method, namely early stopping.

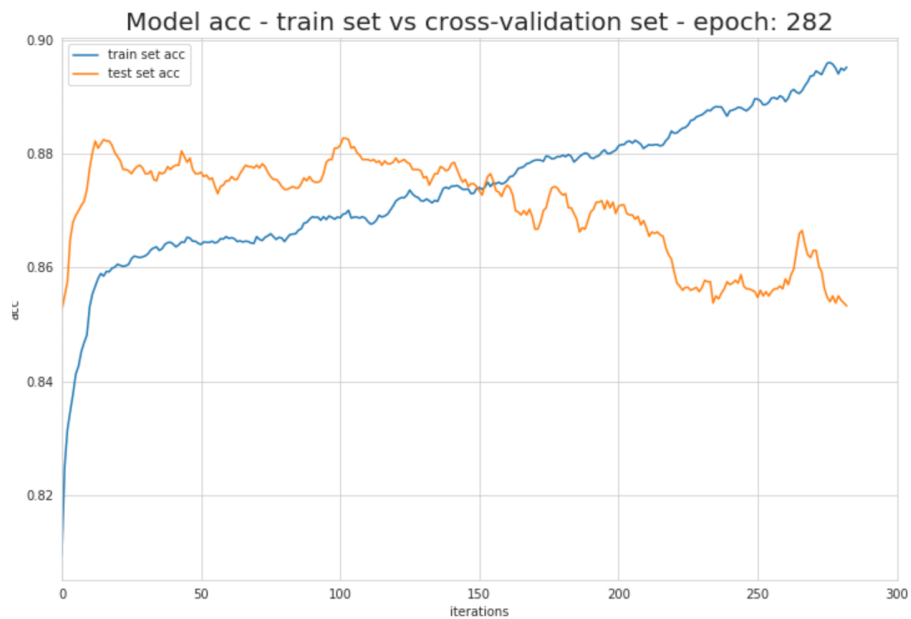


Figure 7.6: Change of accuracy values in subsequent epochs during neural network learning

In practice, it is very convenient to sample our model every few iterations and check how well it works with our validation set. Every model that performs better than all the previous models is saved. We also set a limit, i.e. the maximum number of iterations during which no progress will be recorded. When this value is exceeded, the learning is stopped. Although early stopping allows for a significant improvement in the performance of our model, in practice, its application greatly complicates the process of optimization of our model. It is simply difficult to combine with other regular techniques.

7.3 ARS Optimization using stochastic regularization techniques (SRT)

One requirement of tools for AI uncertainty estimations would be to scale well to large data, and scale well to complex models (such as CNNs and RNNs). Much more important perhaps, it would be impractical to change existing model architectures that have been well studied, and it is often impractical to work with complex and cumbersome techniques which are difficult to explain to non-experts. Existing approaches to obtain model confidence often do not scale to complex models or large amounts of data, and require us to develop new models for existing tasks for which we already have well performing tools. We will thus concentrate on the integration of practical techniques to obtain model confidence in deep learning, techniques which are also well rooted within the theoretical foundations of probability theory and Bayesian modeling. Specifically, we will make use of stochastic regularization techniques (SRTs). Stochastic regularization techniques are techniques used to regularize deep learning models through the injection of stochastic noise into the model. SRTs are developed techniques for model regularization that have been tremendously successful within deep learning, and are used in almost all modern deep learning models. These techniques adapt the model output stochastically as a way of model regularization (hence the name stochastic regularization). This results in the loss becoming a random quantity, which is optimized using tools from the stochastic non-convex optimization literature. Popular SRTs include dropout, multiplicative Gaussian noise (MGN, also referred to as Gaussian dropout), dropConnect, and other recent techniques.

As we will see below, we can take almost any trained network, and given some input x^* obtain a predictive mean $E[y^*]$ (the expected model output given

our input), and predictive variance $\text{Var}[y^*]$ (how much the model is confident in its prediction). To obtain these, we simulate a network output with input x^* , treating the SRT as if we were using the model during training (i.e. obtain a random output through a stochastic forward pass). We repeat this process several times (for T repetitions), sampling outputs $\{y^*_1(x^*), \dots, y^*_T(x^*)\}$. As will be explained below, these are empirical samples from an approximate predictive distribution. We can get an empirical estimator for the predictive mean of our approximate predictive distribution as well as the predictive variance (our uncertainty) from these samples:

$$\mathbb{E}[y^*] \approx \frac{1}{T} \sum_{t=1}^T \hat{y}_t^*(x^*) \quad (7.3)$$

$$\text{Var}[y^*] \approx \tau^{-1} \mathbf{I}_D + \frac{1}{T} \sum_{t=1}^T \hat{y}_t^*(x^*)^T \hat{y}_t^*(x^*) - \mathbb{E}[y^*]^T \mathbb{E}[y^*] \quad (7.4)$$

The previous equations result in uncertainty estimates which are practical with large models and big data, and that can be applied in image based models, sequence based models, and many different settings such as reinforcement learning and active learning.

7.3.1 Classification

We did some standalone tests to proof our arguments. In order to assess our model's confidence in classification we tested our CNN network trained on the MNIST dataset [303]. We trained the CNN model with dropout applied before the last fully connected inner-product layer (the usual way dropout is used in

CNNs). We used dropout probability of 0.5. We trained the model for 10^6 iterations.

The model was evaluated using a continuously rotated image of the digit 1 (shown on the X axis in the figure 7.7). We scatter 100 stochastic forward passes of the softmax input (the output from the last fully connected layer (left side)), as well as of the softmax output for each of the top classes (right side). The plots show the softmax input value and softmax output value for the 3 digits with the largest values for each corresponding input. When the softmax input for a class is larger than that of all other classes (class 1 for the first 5 images, class 5 for the next 2 images, and class 7 for the rest), the model predicts the corresponding class. For the 12 images, the model predicts classes [1 1 1 1 1 5 5 7 7 7 7 7]. Looking at the softmax input values, if the range of high uncertainty of a class is far from that of other classes (for example the left most image) then the input is classified with high confidence. On the other hand, if the range of high uncertainty intersects that of other classes (such as in the case of the middle input image), then even though the softmax output can be arbitrarily high (as far as 1 if the mean is far from the means of the other classes), the softmax output uncertainty can be as large as the entire space. This signifies the model's uncertainty in its softmax output value. In this scenario it would not be reasonable to use argmax to return class 5 for the middle image when its uncertainty is so high. We will use such a technique in our security models to return alarms of uncertainty, if any. Therefore, our third layer model would intervene to declare the input vector as a new attacks or even to calculate the weight variation to increase the precision of our system.

We also show that by using dropout's uncertainty we can obtain a considerable improvement in predictive log-likelihood and root mean square error (RMSE)

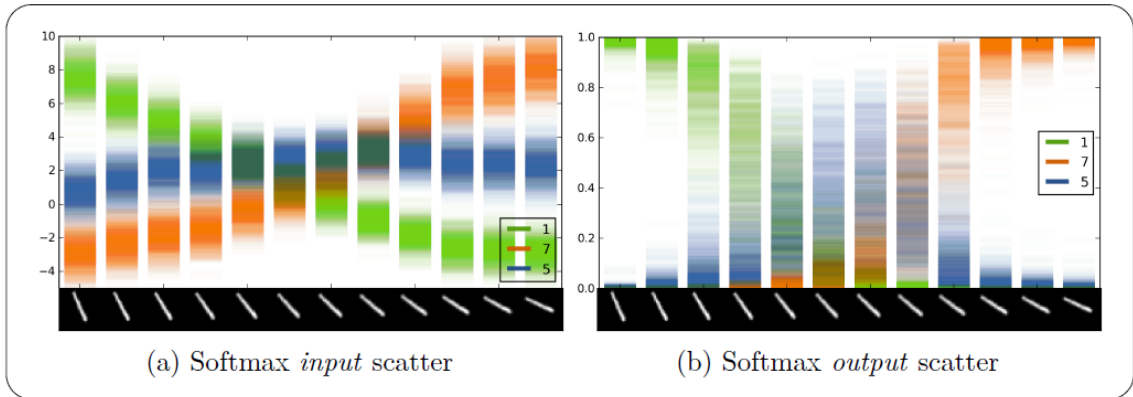


Figure 7.7: Graphical analysis of the top 3 classifications of the CNN model for each input

compared to the original tests. Predictive log-likelihood captures how well a model fits the data, with larger values indicating better model fit. Uncertainty quality can be determined from this quantity as well. We need to define a prior length-scale, and find an optimal model precision parameter τ which will allow us to evaluate the predictive log-likelihood. We used Bayesian optimization (BO) to find the optimal τ , and set the prior length-scale to 10^2 . Note that this is a standard dropout NN, where the prior length-scale l and model precision τ are simply used to define the model’s weight decay through the following equation.

$$\tau = \frac{(1 - p)l_i^2}{2N\lambda_i} \quad (7.5)$$

7.3.2 Uncertainty estimate discussion

We will discuss some measurements to get good predictive uncertainty estimates. First, it seems that “over-parametrised” models result in better uncertainty estimates than smaller models. Models with a large number of parameters can capture a larger class of functions, leading to more ways of explaining the data, and as a result larger uncertainty estimates further from the data. Similar be-

havior was noted with regard to model size. In the dropout case, this conforms with the observation that better RMSE can be obtained when a large number of parameters is used (larger than when dropout is not used as discussed). The dropout probability is important as well, with larger models requiring a larger dropout probability: varying the dropout probability p (through either grid-search or Bayesian optimization) we have that large models (large K) push p towards 0.5, since the weight of the entropy w.r.t. p is scaled by K . For a fixed model size K , smaller probabilities p result in decreasing predictive uncertainty. Further, short model length-scale results in more erratic functions drawn from the posterior hence higher uncertainty values. Intuitively, high model precision (large τ) and large amounts of data (large N) give the expected log likelihood a higher weight than the prior KL, resulting in models that can fit the data well but might overfit. On the other hand, long prior length-scale (large l) gives the prior KL a higher weight than the expected log likelihood, resulting in heavily regularized models that might not fit the data as well. The prior length-scale, model precision, and dropout probability can be optimized using Bayesian optimization and cross validation over test log likelihood. Finally, it seems that model structure affects predictive uncertainty considerably. Many existing models were designed and developed in order to obtain good RMSE, but the same model structure might not be ideal to get good uncertainty estimates. Adapting model structure to result in good uncertainty estimates as well as RMSE might be helpful in improving test log likelihood.

Chapter 8

System Simulation and results

In this chapter we will discuss all simulations, implementations, and different scenario results of both our security and consistency systems. Different system architectures were implemented and tested. Different AI techniques trained with multiple data-sets representing the multiple layers of our system were implemented and tested. Finally, we have implemented and tested our proposed optimization module to reinforce our AI systems to further enhance the confidence in our systems results.

8.1 System Security Simulation and Results

Since our solution considers AI-based techniques, the first stage of our work consists of finding multiple, efficient, ensemble of AI techniques that achieves a high accuracy rate. These techniques are to be, afterwards, integrated as part of our architecture.

8.1.1 System Datasets

The first stage of working with AI is choosing the data. We considered two datasets. The first being the benchmark NSL-KDD dataset, a modified real dataset proposed to solve a number of the existing problems of the older KDD'99 data set mentioned in [304][305]. The second being the Intrusion Detection Evaluation Dataset (CICIDS2017) [50], a more recent dataset published by the Canadian Institute for Cybersecurity (CIC), providing a more reliable dataset that covers the variety of recent attacks [306].

The NSL-KDD dataset contains around 150,000 records including normal traffic as well as anomaly traffic categorized into 4 attack classes: 1. Denial of Service (DoS), e.g. SYN flood. 2. User to Root (U2R), unauthorized access to local super-user (root) privileges, e.g., various “buffer overflow” attacks. 3. Remote to Local (R2L), unauthorized access from a remote machine, e.g. guessing password. 4. Probing (Probe), surveillance and other probing, e.g., port scanning. 5. Normal.

Table 8.1 shows some attacks that fall under each class in the dataset. For our training and testing stages we divided the dataset such that 60% extracted for training and 40% between testing and validation as presented in Table 5.

Table 8.1: Attack Names Included in Attack Categories

Attack class	Attack name
DoS	Smurf, Land, Pod, Teardrop, Neptune, Back
U2R	Perl, buffer_overflow, Rootket, Loadmodule
R2L	Ftp_write, Gess_pass, Imap, Multihope, phf, spy
Probe	Ipsweep, nmap, portsweep

Table 8.2: Numbers of Class Records in NSL-KDD Dataset

Dataset	Normal	DoS	U2R	R2L	Probe	Total
Training	67,343	45,927	993	54	11,656	125,973
Testing	9,711	7,458	2,421	533	2,421	22,544

Our tests are performed on the following 41 features that are included in the literature for network security. The features are classified as follows:

- 9 basic features extracted from individual tcp connections (e.g. duration, wrong_fragment)
- 9 features extracted from a 2-second time window (e.g. number of connections to the same host and percentage of syn_error)
- 10 features extracted from a window of 100 connections (e.g. count for destination host and percentage of rej_error)
- 13 features extracted within a connection suggested by domain knowledge (e.g. number of shell_prompt and number of failed login_attempt)

The CICIDS2017 dataset contains 350,000 records including normal traffic as well as anomaly traffic classified as attack types rather than attack classes, which leads to a more accurate result regarding our second security layer that aims towards identifying specific attack types. The attack types include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, SQL Injection, Port scanning, Infiltration, Botnet and DDoS. In addition, the data set offers 80 network features, and traffic is categorized according to time stamps, which allows us to extract more statistical features that would enhance the reliability of the system.

Table 8.3 presents a sample of attack types and the extracted set of features for each attack detection process.

8.1.2 System Simulation

The setup was done using a single Linux (Ubuntu 14.04.2) server with MATLAB 2017 and Python installed. Initially, tests were done using MATLAB and then, Python was used to shift to a more realistic environment for controllers to work with. In the first study cases, we used Principle Component Analysis (PCA)

Table 8.3: Attack Types and Related Features

Attack Type	Features
Benign	B.Packet Len Min
	Subflow F.Bytes
	Total Len F.Packets
	F.Packet Len Mean
DoS GoldenEye	B.Packet Len Std
	Flow IAT Min
	Fwd IAT Min
	Flow IAT Mean
Heartbleed	B.Packet Len Std
	Subflow F.Bytes
	Flow Duration
	Total Len F.Packets
DoS Hulk	B.Packet Len Std
	B.Packet Len Std
	Flow Duration
	Flow IAT Std
DoS Slowhttp	Flow Duration
	Active Min
	Active Mean
	Flow IAT Std
DoS slowloris	Flow Duration
	F.IAT Min
	B.IAT Mean
	F.IAT Mean
SSH-Patator	Init Win F.Bytes
	Subflow F.Bytes
	Total Len F.Packets
	ACK Flag Count
FTP-Patator	Init Win F.Bytes
	F.PSH Flags
	SYN Flag Count
	F.Packets/s
Web Attack	Init Win F.Bytes
	Subflow F.Bytes
	Init Win B.Bytes
	Total Len F.Packets

Table 8.4: Attack Types and Related Features Continued

Attack Type	Features
Infiltration	Subflow F.Bytes
	Total Len F.Packets
	Flow Duration
	Active Mean
Bot	Subflow F.Bytes
	Total Len F.Packets
	F.Packet Len Mean
	B.Packets/s
PortScan	Init Win F.Bytes
	B.Packets/s
	PSH Flag Count
DDoS	B.Packet Len Std
	Avg Packet Size
	Flow Duration
	Flow IAT Std

for feature reduction. These tests implement PCA to reduce the entire set of features into 4 features to be used as input for the different techniques. Each technique is being tuned for the finest possible parameters to give the highest possible accuracy for our test environment. Tests are done on both balanced and unbalanced data. We are using unbalanced data in order to train our system with a realistic traffic environment. The goal is to study the response of these techniques in such environments.

The next step on our list was the edge node feature extraction. We assigned an agent residing next to each edge node. The agent was programmed to analyze specific headers of each incoming packet and extract certain features. The extraction was limited to only the participating interfaces. The agents are also responsible for relating the extracted features to an individual connection in a statistical manner. The future goal is to reach an efficient statistically-based set of features that is resilient against attacker’s interventions and manipulations.

This technique helps our system to focus on the general functionality and randomness of the network rather than on a specific entry point. Overall, allowing better detection of unknown distributed behavior.

At the end, all agents would form a vector of extracted features and send it securely to the ARS module at the controller. The feature marking techniques allows a single server to handle and track the work of multiple clients. If any further processing is required on the uploaded set of features, it would be handled by the ARS module before preparing the features for the AI model.

8.1.3 System Training

Generally, every training algorithm for BP neural network includes two phases. The first one is the forwarding phase during which the output of each layer is calculated successively. The second one is the backward phase during which the error is transited backward to fix the weights of all connections. After the forwarding phase, an error function, i.e. the objective function, can be obtained. Different training algorithms have their own schemes to utilize the error function in the backward phase.

The BP net training algorithm used in our system is RPROP [304]. It is a resilient backpropagation approach. It converges faster than the gradient-descent [307] approach. In the used scheme, the objective function in batch learning mode is:

$$E = \frac{1}{2} \sum_p \sum_q (y_{p,q} - d_{p,q})^2 \quad (8.1)$$

Where: - p refers to the pth output.

- q refers to the q th training sample.
- $y_{p,q}$ is the real output.
- dp,q is the expected output.

In gradient-descent, the size of weight update is directly proportional to the learning rate η and the size of partial derivative dE/dW . The authors of [307] believe that the benefit of carefully adapted learning rate can be upset by the unforeseeable behavior of the partial derivative. Hence, in resilient backpropagation, the derivative only determines the direction of weight update, δw . [307] introduces an update-value δij . Its update follows the following rule:

$$\Delta_{ij}(t) = \begin{cases} \min(\eta^+ \cdot \Delta_{ij}, \Delta_{max}), & \text{if } \frac{\partial E(t)}{\partial \omega_{ij}(t)} \cdot \frac{\partial E(t-1)}{\partial \omega_{ij}(t)} > 0 \\ \max(\eta^- \cdot \Delta_{ij}, \Delta_{min}), & \text{if } \frac{\partial E(t)}{\partial \omega_{ij}(t)} \cdot \frac{\partial E(t-1)}{\partial \omega_{ij}(t)} < 0 \\ \Delta_{ij}(t-1), & \end{cases}$$

where $0 \leq \eta^- \leq 1 \leq \eta^+$.

If $\delta E/\delta w_{ij}$ retains its sign, it can be increased by η^+ to speed up the convergence. If the partial derivative $\delta E/\delta w_{ij}$ changes its sign, it means that the last update of ω_{ij} is too large and the algorithm jumped over a minimum of the error surface. So, the update-value Δ_{ij} of the corresponding weight w_{ij} has to be decreased with a factor η^- . Next, comes the renew of weight-update. If the partial derivative does not change its sign, the update of δw follows the following rule:

$$\Delta\omega_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & \text{if } \frac{\partial E(t)}{\partial \omega_{ij}(t)} > 0 \\ +\Delta_{ij}(t), & \text{if } \frac{\partial E(t)}{\partial \omega_{ij}(t)} < 0 \\ 0, & \text{else} \end{cases}$$

If the partial derivative changes its sign, a method called backtracking [307] would be used. The update of δw follows the rule:

$$\Delta\omega_{ij}(t) = \Delta\omega_{ij}(t - 1) \quad (8.2)$$

Then, set $\delta E(t)/\delta w_{ij}(t) = 0$. Thus, if the step of the algorithm is too big to miss a minimum in error surface, it will go back to the previous position.

At last, the weight can be renewed by the following rule:

$$\omega_{ij}(t + 1) = \omega_{ij}(t) + \Delta\omega_{ij}(t) \quad (8.3)$$

$w(t+1)$ is the new weight after the $(t+1)$ th iteration, $w(t)$ is the old weight before the $(t+1)$ th iteration.

8.1.4 Tests and Results

The following section presents a comparison between the different techniques tested for both stages on both datasets, using balanced and unbalanced data,

BD and UBD, respectively. The results in Tables 8.5 and 8.6 show that random forest achieves better results as compared to the other techniques tested at this stage, followed by the neural network based technique. Figures 7 and 8 represent the random forest test confusion matrix (CM) for both unbalanced (UBD) and balanced (BD) data, respectively. We can notice an increase in the precision of the DNN from the first dataset to the second. This reflects the strength of deep AI techniques when handling different attack types and as the dataset increases in size. Note that although the Random forest technique showed the higher precision, yet the DNN showed faster processing with competitive precision. This sheds light on part of our future work were our system should be equipped with an algorithm aimed for chosen the best ensemble of AI techniques for the first and second security layers for a prestored database according to different network parameters and conditions.

Another test scenario that was done on the full system till this point aimed to simulate both the security and consistency systems working together. The simulation was based on Mininet [308], an SDN network emulator, and miniedit [309], an extension to Mininet for graphical network topologies, were we constructed a fully connected tree network topology that consists of 20 OVS switches and 14 virtual hosts. We manipulated the OVS code in order to connect two physical servers on two different OVS switches as shown in the figure.

The purpose of this test was to launch different attacks at the same time from each external physical server on the SDN controller. For this task we assigned the edge nodes being the directly connected OVS switches as shown in the figure.

The tests were based on the D C2 architecture, were the edge nodes are responsible for extracting the specified features and uploading them to the controller during each time cycle. Since the network converged and no frequent updates

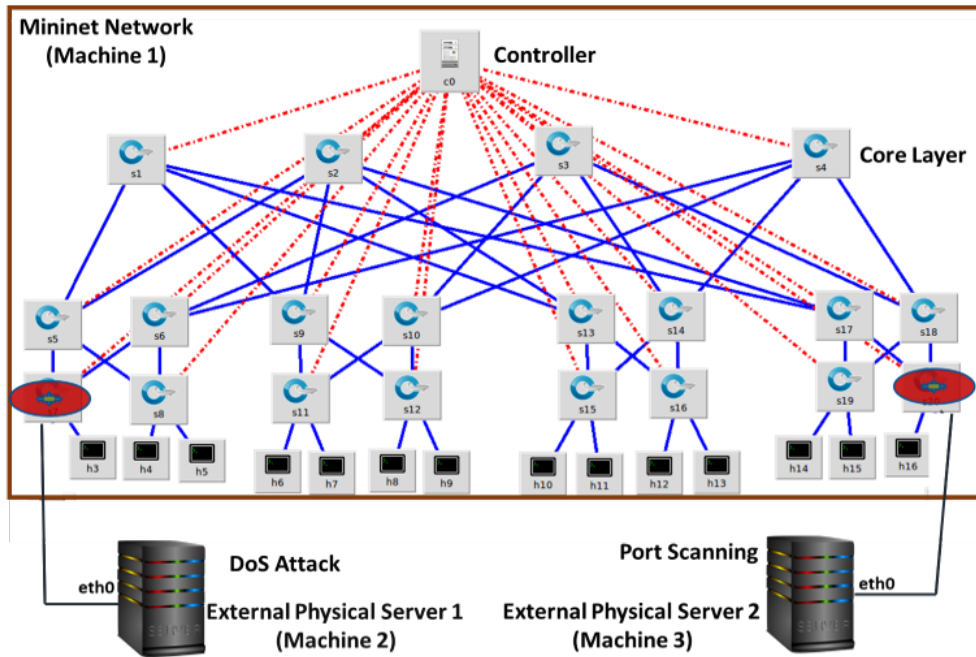


Figure 8.1: Attack/Detection Test Scenario

were needed, we activated the consistency client within the same time cycle as the security client such that the consistency client would include the hash output in the same vector sent each time cycle to the controller. The following tables show the different AI techniques tested for both layer while trained by one of two datasets.

Table 8.5: Test Results Summary Based on NSL-KDD

Tech.	Anomaly Detection (BD)	Anomaly Detection (UBD)	Attack identification (BD)	Attack identification (UBD)
Random Forest	99.2%	99.3%	98.3%	99.61%
SVM	96.74%	96.93%	90.64%	93.6%
KNN	94.5%	96.4%	90.19%	90.5%
DT	95.76%	96.4%	89.5%	90.89%
MLP	97.5%	97.0%	94.3%	92.73%
BPNN	98.7%	98.6%	77.2%	75.8%
DNN	96.11%	96.13%	95.03%	93.67%

Table 8.6: Test Results Summary Based on CICIDS2017

Tech.	Anomaly Detection (BD)	Anomaly Detection (UBD)	Attack identification (BD)	Attack identification (UBD)
Random Forest	98.1%	98.0%	97.3%	97.6%
SVM	95.4%	95.5%	89.1%	90.1%
KNN	94.0%	94.4%	88.2%	89.5%
DT	93.6%	94.0%	87.2%	88.0%
MLP	96.5%	96.0%	93.7%	92.1%
BPNN	97.9%	97.6%	77.9%	75.9%
DNN	97.8%	97.35%	97.11%	96.70%

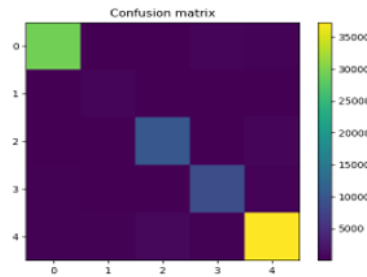


Figure 8.2: RandomForest Confusion Matrix for Unbalanced Data

After the results on the initial tests were shown, another set of tests were done on both the RF technique for the anomaly detection layer, and DNN for the attack identification process. Both systems were trained by the CICIDS2017 dataset. After time t_0 we initiated both a DoS attack from server-1 and port scanning on the controller from server-2. It took around 6 secs from t_0 for the

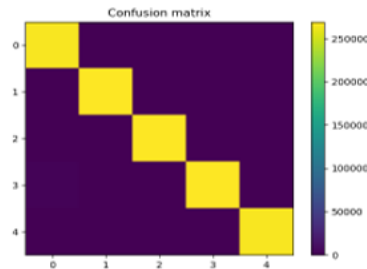


Figure 8.3: RandomForest Confusion Matrix for Balanced Data

first layer to detect a change in the network traffic; at this point the second layer was activated due to the first alert, and the abnormal traffic were injected in the second system. It took around 3 secs from $t_0 + 6$ for the second layer to identify both attacks. Even though the second layer requires more processing time, but as the results show, it took less time. This is due to the double layered technique since we are injecting the second system with a subset of the traffic that were flagged by the first system. Therefore, resulting in a faster and more efficient detection process.

We added an extra security feature to our system: during an attack, the security module is able to block the attack source from the directly connected edge node. This process was possible by injecting a new rule on the edge nodes forcing them to block all traffic matching the attacker's source-IP. Through monitoring the attacker's traffic on the egress ports of the edge node, we can see the specific traffic being blocked as illustrated in Figure 8.4.

Figure 8.4 shows how the DoS attack intensity (packet numbers) increases exponentially throughout the attack period while the port scanning traffic is constant. The attack started at $t = 0$ secs and at $t = 6$ secs, the anomaly detection flagged the incoming traffic before the attack identification system detected the attack type at $t = 9$ secs and injected the attack mitigation rule to block the attack before $t = 10$ secs.

After we have tested the centralized mode of our work, we started our tests on the distributed NN overlay security layer. We selected the Neural Network AI technique for this stage due to its high accuracy results in the anomaly detection tests discussed earlier. Also due to its architecture that made it possible to consider such a new distribution technique.

The security module in this case with run an algorithm to select the best

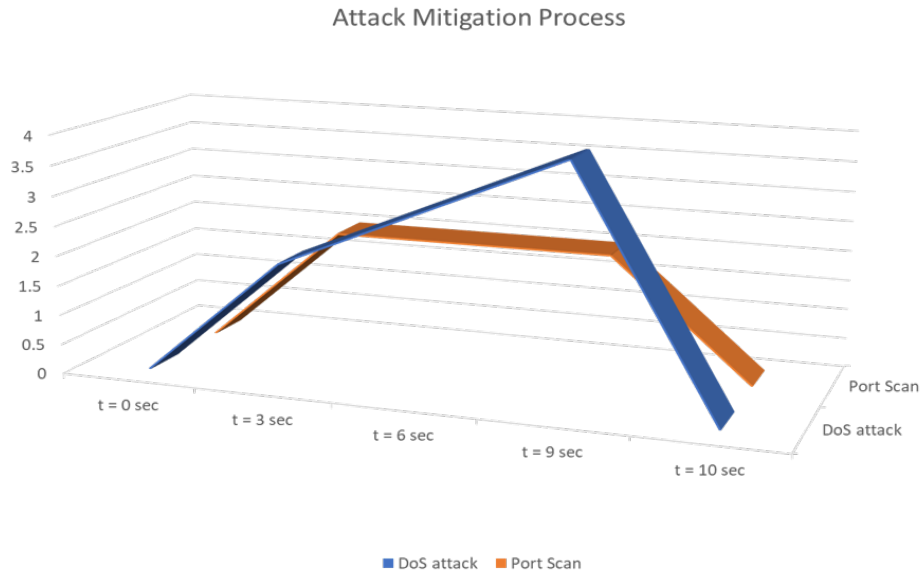


Figure 8.4: Attack Mitigation Process

suitable candidates to participate in the distribution process. The algorithm can query the network for the number of physical connections of each node. If the network is already running, the security module would query the ARS agent for the CPU load percentage (e.g `iostat -c`). The other parameter should be inserted by the administrator, which is the processing capability = CPU Speed (Mhz) x Number of CPUs x Ram (Ghz). The node with the highest parameters would be chosen as candidates. The edge node are excluded from this procedure since they are included and adge nodes by default. The number of participating nodes from the chosen candidates depends on the number of NN to be overlaid, which also depends on the number of input points on the network. Another feature was implemented was a database for manual selection by the administrator, which would override this stage.

After the participating nodes are set, the ARS agent is enabled to play its new role. The controller would send a packet to each ARS agent to let it know wich part of its NN script to enable (hidden or output), along with other parameters

(number of layers, number of neurons in each layer, and the number of NN network to participate with). After this stage, each edge agent will be informed of the ID of the participating internal agents in its own NN. Each edge agent will then send a packet to the corresponding internal agents to be linked as their next hop in the AI processing phase. At this point all the NN networks are set, each protecting a specific entry point of the network.

We tested the distributed system for abnormal traffic using the same test done on the first layer of the centralized security mode. The test was done on the network of figure 8.1, with 2 edge nodes. To protect the 2 ingress points we create 2 NN overlays. The 2 edge nodes played the role of the input layers and (s5,s8) along with (s18,s19) played the 2 hidden layers. while s6 and s17 were the output layers. The same NN architecture was tested for the centralized test. After the overlay was set, the weights were injected in the network. The weights are calculated after training the same NN network offline using the 60% of the CICIDS2017 dataset. We then injected the remaining 40% of the dataset as traffic in the network from both external servers. The overall results of the 2 NNs was around **94.8%** accuracy. The same latency test was done as the centralized mode, where a DDoS attack was launched from one server into s7 and a port scan attack was launched into s20. The attack started at t=0 secs and at t=4 secs the alarm was triggered detecting an abnormal traffic. The detection latency will vary between the centralized and distributed security layer depending on the network itself and the node capabilities. The higher the capabilities the more a node can handle hidden layer, thus less processing time. On the other hand the network congestion and the distance between the controller and the edge nodes play a role in the latency of the centralized mode.

8.2 System Consistency Simulation and Results

This section discusses all the necessary simulations and results that provided our consistency system with the proof of concept and effectiveness. The following sections includes our data collection phase, which presents the flow-based attack classes and data structure. Followed by the data preprocessing phase, which also discusses the first consistency verification layer till the point where our system is able to flag any inconsistent flow. Next, we present the classification results, which include the second layer of the system where the AI module is able to identify the class for each inconsistency flagged by the first layer. Also, a third consistency layer is discussed, which is responsible for detecting unknown consistency attacks in case an attack is triggered and not classified. The first simulations where done on our dataset, followed by real time consistency check test scenarios presented in the following sections.

8.2.1 Data collection

For the purpose of this work, we have generated our own real traffic to be able to extract the necessary data from our simulated SDN network represented in the figure 8.6. The network consists of 9 nodes with 3 border nodes, 3 core nodes, and 3 edge nodes.

After the first traffic flow have ended and the network have converged, the data extraction stage starts. The flow table of each node contained around 100 flows. At the same time t , the controller was keeping a database of each update being inserted on each node for future checks.

At time t_c , the first extraction took place from each node. This process was repeated for 100 traffic flows with each traffic flow = t_c giving a total of $100t_c$

data collection duration. Hence, the total data collected consists of 100 instances of each node with each instance containing around 100 flows.

During each traffic flow, we performed multiple consistency-based attacks on multiple flows on each node in the network. Our data was classified into 5 classes as shown in the following Table.

Table 8.7: Consistency Attack Classes

Class	Type	Description
1	Consistent	Flow is consistent
2	Traffic redirect	Traffic changed its original path (e.g. Action=New destination IP)
3	Dead Lock	Traffic is deviated to a blocked destination (e.g. Action=Virtual Port)
4	Loop Insertion	Traffic have entered a network loo (e.g. Action=InPort)
5	DoU	Denial of Update (e.g. an overlap flow is added by an attacker with higher priority but different action, each time a specific update is inserted)

8.2.2 Data Preprocessing

After collecting all necessary data from the network nodes and preparing the controller's data, the hashlib python library [310] is used to perform the chosen SHA-2 [311] hash function. The SHA-2 (SHA-256) function accepts variable size inputs and produces a fixed 256 bits (32 bytes) size output, which can be carried out in our vector to the controller, in order to minimize the required traffic overhead. At this point, the hash function is applied to the three feature classes. The features pertaining to each class are concatenated in one string and thus, the hash is applied to a string resulting in a 32-byte hexadecimal output.

After hashing the node and the controller data, the hexadecimal output ob-

tained for each features class is transformed to bits. Consequently, after concatenating the binary outputs of the three feature classes, we obtain a binary vector containing $16*16*3$ elements for each flow entry. An XOR operation is performed between two corresponding vectors of each flow entry. The first vector is the one obtained based on the controller data and the second vector is the one obtained based on the node's flow table. As a result, we obtain a $16*16*3$ vector.

At this point, a simple check is sufficient to verify the consistency of each flow in each matching table in the network. Hence, if the obtained vector contains all zeros, then the corresponding flow entry is the same as the one at the controller, thus it can be considered as consistent. Else, this means that some unauthorized modifications were invoked at the node level.

In addition to flagging inconsistent flows, we have enabled our first consistency verification layer to visually represent the output data for more clarity. In this context, the visualization of the obtained XOR-ed data, as illustrated in the figure 8.5, shows clearly how our technique, with low processing of bits detection, helps in the differentiation between consistent and non-consistent flows. Thus, to visualize these vectors, we transform them to RGB images of size $16*16$, with samples of the resulting images shown in the figure. It can be noticed that the inconsistent flows present different RGB colors while the consistent flows present black images. An important parameter extracted from these images, aside from being colored or not, is the color itself. A RED image implies that the altered flow belongs to the first class of features (constant class), while the GREEN color implies that the altered flow belongs to the second class (time class), final the BLUE color represents the third class (statistical class).

Once an inconsistency has been detected, a second layer in the system will be triggered. This layer will take as input the vector of XOR-ed data that was

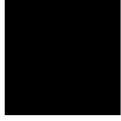

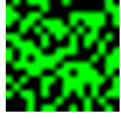

	
Consistent	Non-consistent
	
Non-consistent	Non-consistent

Figure 8.5: Consistency Results Visual Data Representation

responsible for triggering the inconsistency in order to specify the class of attack that may have caused this issue. Furthermore, knowing the attack class and the color of the image of this specific flow allows us to identify the features that were modified within that corresponding flow.

Our human brain has a distinguished technique of learning new attacks (viruses and others) through learning their symptoms. These attacks, that could affect our body on their first attempt, would be recognized by our immunity system in later attacks. Thus it could be treated or mitigated according to similar symptoms in the future. Inspired for this technique and benefiting from what AI has to offer in the area on know and unknown attacks, we have modified our system to detect unknown consistency attacks. A third layer was implemented with the purpose of handling attacks that were flagged by the first layer but failed to be recognized by the second. The third layer would detect such an attempt an unknown new attack and integrate it with the set of altered featured to be taught, offline, to our AI system. The new weights are integrated and updated to the current system at a specific time chosen by the administrator as a network idle time.

8.2.3 Classification Results

After the data preprocessing stage, we obtain our labeled data of the XOR-ed output between the nodes and controller. The obtained $16*16*3$ vectors are passed to the classifier. The data was randomly over-sampled for balancing. In our test, we compared Different deep learning architectures: LeNet5, AlexNet, ConvNet, GoogleNet, ResNet, RNN, and DNN. Cross-validation was applied with 4 folds. At each fold, the different architectures were trained with 50 epochs and 50 as batch size. Moreover, the data was split into 60% for training, 20% for validation, and 20% for testing. Moreover, at each fold, the following performance measures were recorded and at the end, the average over the 4 folds was computed:

$$\text{Accuracy: } (TP+TN) / (TP+TN+FP+FN)$$

$$\text{Precision: } TP / (TP + FP)$$

$$\text{Recall: } TP / (TP + FN)$$

$$\text{F1-score: } 2*TP / (2*TP + FP + FN)$$

Where TP, the number of correctly classified instances to pertain to class X; TN, the number of correctly classified instances not pertaining to X; FP, the number of instances erroneously classified to pertain to X; and FN, the number of instances erroneously classified as not pertaining to X.

The experiments were run on a Linux machine having an Intel® Core™ i7-3630QM CPU @ 2.40GHZ x 8 (8 cores). 16 GB of memory were available. Operating system was Ubuntu 14.04 LTS-64 bit.

The comparison results, presented in the following table, show that the ConvNet architecture gives the best results with 99.39% as accuracy, precision, recall and f1-score. In fact, ConvNet was able to differentiate between different attack classes even though the images resulting for the same attack, may be caused by altering features from different classes and thus, resulting in images of dif-

ferent colors. As such, ConvNet has shown its ability to recognize the patterns contained in the XOR-ed data of each type of attack.

Table 8.8: Deep Learning Consistency Verification Results

	Accuracy	Precision	Recall	F1-score
AlexNet	83.02%	85.5%	82.93%	82.79%
ConvNet	99.39%	99.39%	99.39%	99.39%
LENet5	72.9%	77.09%	72.9%	69.72%
GoogleNet	69.65%	58.48%	69.43%	62.45%
ResNet	95.96%	96.27%	95.98%	95.97%
RNN	83.56%	89.1%	83.53%	83.62%
DNN	83.78%	84.27%	83.69%	83.66%

8.2.4 System Consistency Tests

The following section describes a real-time consistency verification test of our system. The setup consists of a Linux (Ubuntu 16.04) server with Python-3 installed. The simulation was based on Mininet [308], an SDN network emulator, and mini-edit [309], an extension to Mininet for graphical network topologies, where we constructed a fully connected tree network topology that consists of 20 OVS switches and 14 virtual hosts. We manipulated the OVS code in order to connect two physical servers on two different OVS switches as shown in the figure.

The purpose of this test was to launch different configuration-based attacks from the two external servers on different switches in the network. At this point, our AI system is fully trained offline as discussed earlier. After we attacked the flow table of 4 switches (s5, s12, s15, s17) in the network (12 flows in each switch), we wait for $t_c = 10$ sec, as programmed, for the consistency module to start the next check. The altered 12 flows in each switch are based on multiples flow-based attacks belonging to different attack classes. We attacked each switch with two

types of classes as shown in Table.

Table 8.9: Switch and Corresponding Attack Classes Scenario

Switch	Attack Class 1	Attack Class 2
S5	Traffic Redirect (Redirecting traffic for sniffing purposes)	Traffic DeadLock (Blocking all Controller packets)
S12	Traffic Loop Insertion (Inserting loops through matching egress to ingress port)	DoU of Security Updates (Denial of update of any new flow in the security flow)
S15	Traffic Redirect (Redirecting traffic towards controller)	Traffic Loop Insertion (Inserting loops through matching egress to ingress port)
S17	Traffic DeadLock (Blocking all incoming packets exiting a specific port)	Traffic Redirect (Redirecting traffic randomly to disrupt the functionality on the network)

The test was extended to include an unknown attack through altering a random set of features. This attack was done on S4. This set was not taught or included in any of the previously mentioned attacks. Such a test would show us the precious and effectiveness of the third consistency check layer.

We have chosen the best three deep learning AI techniques to be tested for this second scenario. The techniques are CNN, DNN, RNN.

The test started with feature extraction, followed by the first layer of consistency verification after the first vector of hashed features was upload to the controller at $t_c = 10$ sec. After the first layer, the system was able to detect the inconsistencies found in the targeted 4 switches while all other 16 switches returned a consistent result. A sample of the graphical representation of inconsistencies, shown in figure 8.7, show a comparison between flows of switch S1 and S5.

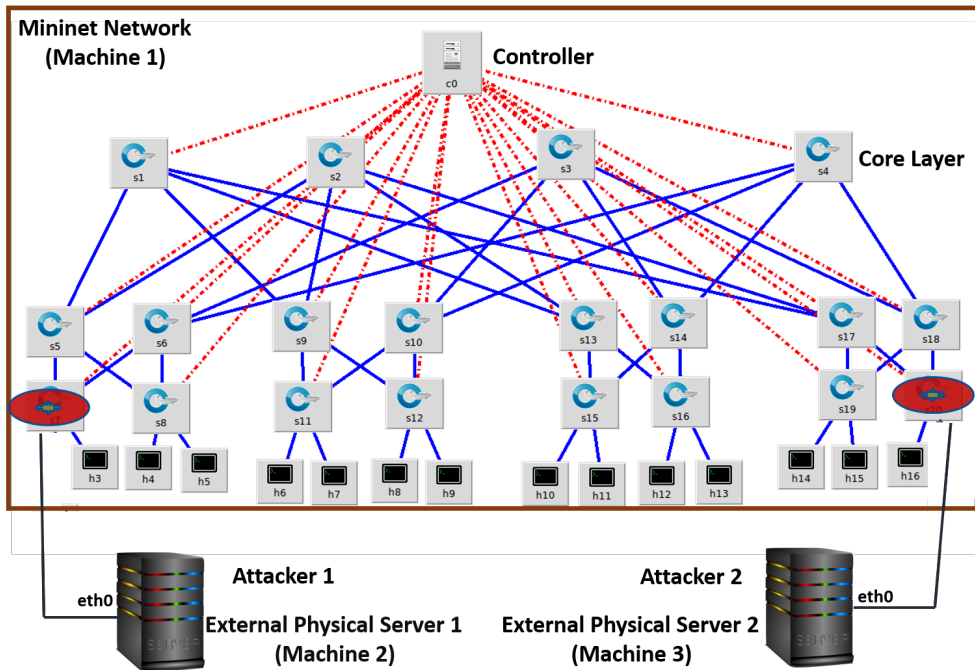


Figure 8.6: Consistency Verification Test Scenario

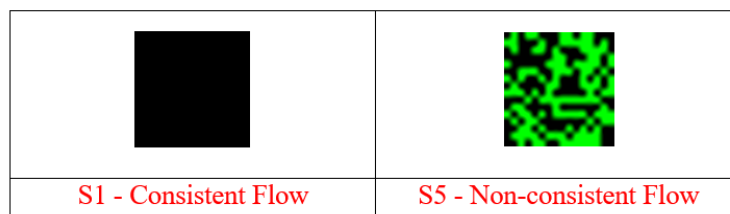


Figure 8.7: S1-S5 Graphical Consistency Comparison

As the work of the first layer is finished and an inconsistency is flagged, the second layer starts taking as input the same binary matrices of the 4 switches that flagged the inconsistency in the first layer. The results of the 3 chosen deep learning techniques are shown in the following table.

Table 8.10: Consistency Test Scenario Results

	Accuracy	Precision	Recall	F1-score
ConvNet	96%	96%	93%	90%
RNN	96%	83.4%	78.3%	78.5%
DNN	95.5%	93%	81.6%	77.5%

At the end of the test we checked the database of the third layer that included the new attack with its related set of altered features. Also we checked the resulting new weights that resulted for the new training process. In order to verify these results, we rerun the same test, but with including the new attack in S17 instead of the deadlock attack. The new attack was given the name "unknown1" with id=7. The results were the same as the previous one shown in the previous table, and the updated system was able to classify the new attack as a known rather than unknown attack.

The final results show that CNN have also given the highest accuracy and precision values and was able to classify the attack class even in real-time scenarios where only few flows were modified (6 flows per attack class).

Other test results are the processing times of the two layers. The attack started at $t = 0$ sec and was finished at $t = 7.5$ sec, followed by feature extraction, hash and vector composition, which was finished at $t_c = 10$ sec. At the controller side, the Processing time of the first verification layer for all switches, including the graphical representations, was 3 sec. Regarding the second verification layer, the processing time of the CNN system was 2.5 sec to verify a single switch in the network (4 switches = 10 sec). Note that only the flagged switches are transferred

to the next layer, and hence, minimizing the processing load and time. Another technique is the ability to perform multiple CNN modules in parallel for each switch to further minimize the processing time.

Chapter 9

Conclusion

We presented a full system analysis including the general architectural design. We discussed modifying and managing AI techniques to design a general system solution to protect our network against different types of attacks based on a combination of both centralized and distributed capabilities with the least possible overhead. Our goal is to deploy a state-of-the-art adaptive security system over a consistency verified network for better network resiliency.

We proposed and implemented a virtual ANN network overlay as a first layer of security. It is from the simple and parallel computational capabilities of neurons in an ANN, that it is possible to distribute the processing of a traditional ANN over the network. The ARS agents played role of different layers in the neural network. Every node/agent contributes a free part of its storage and computational capacities to virtualize one or more NN layer. These agents will connect to each other using logical links to exchange data and results; hence, leading to an ANN-based overlay network. Such a design minimized traffic overhead through enabling independent processing, achieving real-time detection. The second security layer consists of an ensemble of AI techniques centralized at the controller

with higher processing power for attack specification and mitigation.

We presented a new AI-based consistency verification system integrated with our general ARS architectural design. We discussed how adopting distributing data extraction techniques can provide the necessary information to any AI system while keeping a low traffic overhead and preserving privacy.

The consistency solution provides a double layer consistency verification system inspired by the human brain-immunity cooperation system. The first layer works on comparing a hashed version of specific extracted features from the flow table of each network node with its corresponding image at the controller. The comparison is based on a simple XOR between the two hash vectors. We equipped our system with a graphical representation of the consistency results.

In case of any inconsistency in any node, a second AI-based layer is triggered to identify the attack class that triggered these inconsistencies on all the flagged switches. Our results show that by adopting a double layer technique, we can perform faster checks, and in-depth classification only when necessary, minimizing processing time and overhead.

At the end of our research we have proof of concept of our work after presenting our ARS system through both implementation and test results. We have proven that with both multiple layers of security and consistency constructed in efficient techniques, we can reach a well resilient network. Our work has provided network consistency with real-time protection, while preserving privacy and minimizing traffic overhead.

Chapter 10

Future Work

The future work regarding our thesis can be divided into two sections: The future work that can further develop our work, and how our work can contribute in other domains.

As part of our system's future work we propose a physical hardware implementation in real networks, especially involving our distributed security overlay. Another important feature would be a graphical interface providing graphical processing and data analysis. A third advancement would be enabling protection from leak of sensitive information from specific nodes or servers in the network to the outside. Along with enabling protection against attacks from host connected on the network LAN. A final approach would be further system optimization in terms of AI techniques and network parameters.

Future work could be done using our techniques in other domains such as as IoT and VANETs. In addition to applying the same techniques for traffic identification such as applications, and others. Other contributions could be done through applying the similar techniques in large Data centers.

Appendix A

Abbreviations

SDN	Software Define Networking
AI	Artificial Intelligence
ARS	Artificial Intelligence Resiliency System
ML	Machine Learning
NN	Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
DBN	Deep Belief Network
BM	Boltzmann Machine
AE	Auto Encoder
SVM	Support Vector Machine
RF	Random Forest
MPTCP	Multi Path TCP
RMSE	Root Mean Square Error

Bibliography

- [1] O. N. foundation, “software-defined networking: the new norm for networks,” *ONF white paper*, vol. 2, pp. 1–6, 2012.
- [2] OpenNetworkingFoundation, “Principles and practices for securing software defined networks.” 2015.
- [3] arbor network, “Worldwide infrastructure security report,” vol. 12, pp. 2009–09, 2015.
- [4] service provider nfv, “Sdn investments to reach 21b by 2020. available.” 2014.
- [5] S. jain al, “b4: experience with a globally-deployed software defined wan,” *acm sigcomm computer communication review*, vol. 43, pp. 3–14, 2013.
- [6] Cisco, *software-defined networking: why we like it and how we are building on it*. San jose, CA, USA: cisco systems, 2013.
- [7] N. gude al, “Nox: towards an operating system for networks,” *acm sigcomm computer communication review*, vol. 38, pp. 105–110, 2008.
- [8] S. sezer al, “Are we ready for sdn? implementation challenges for software-defined networks,” *IEEE communications magazine*, vol. 51, pp. 36–43, 2013.

- [9] S. A. mehdi, J. khalid, and S. A. khayam, “Revisiting traffic anomaly detection using software defined networking,” *in international workshop on recent advances in intrusion detection pp*, pp. 161–180, 2011.
- [10] C. chung al, “Nice: network intrusion detection and countermeasure selection in virtual network systems,” *IEEE transactions on dependable and secure computing*, vol. 10, pp. 198–211, 2013.
- [11] G. gibb, H. zeng, and N. mckeown, “Outsourcing network functionality,” *in proceedings of the first workshop on hot topics in software defined networks pp*, pp. 73–78, 2012.
- [12] G. lu al, “using cpu as a traffic co-processing unit in commodity switches,” *in proceedings of the first workshop on hot topics in software defined networks pp*, pp. 31–36, 2012.
- [13] G. huang al, “dynamic measurement-aware routing in practice,” *IEEE network*, vol. 25, 2011.
- [14] J. R. ballard, I. rae, and A. akella, *extensible and scalable network monitoring using opensafe*. in inm/wren, 2010.
- [15] R. skowrya, S. bahargam, and A. bestavros, “software-defined ids for securing embedded mobile devices,” *IEEE, in high performance extreme computing conference (hpec)*, vol. 2013, pp. 1–7, 2013.
- [16] C. jeong al, “scalable network intrusion detection on virtual sdn environment,” *IEEE 3rd international conference on cloud networking (cloudnet)*, vol. 2014, pp. 264–265, 2014.

- [17] S. hogg, *sdn security attack vectors and sdn hardening*. artikkeli network, 2014.
- [18] O. N. Foundation, “Open networking foundation.” 2017.
- [19] B. S. al, “Onf sdn architecture overview,” *open networking foundation*, vol. 8, p. 12, december 2013.
- [20] A. doria al, “Forwarding and control element separation (forces) protocol specification,” vol. 2010, 2010.
- [21] L. yang al, “Forwarding and control element separation (forces) framework,” vol. 2004, 2004.
- [22] T. lakshman al, “The softrouter architecture,” in *ProC. acm sigcomm workshop on hot topics in networking*, 2004.
- [23] A. rodriguez-natal al, *software defined networking extensions for the locator/id separation protocol*. IETF lisp working group internet-draft, 2014.
- [24] H. zheng and X. zhang, *Path computation element to support software-defined transport networks control*. march: internet draft, 2014.
- [25] Floodlight, “Floodlight.” 2017.
- [26] Openstack, “Openstack pike api reference documentation [openstack documentation].” 2017.
- [27] Brocade, “Vyatta brocade documentation.” 2016.
- [28] M. scharf al, “Dynamic vpn optimization by alto guidance,” *second european workshop on software defined networks (ewsdn)*, vol. 2013, pp. 13–18, 2013.

- [29] A. tootoonchian and Y. ganjali, “hyperflow: a distributed control plane for openflow,” in *Proceedings of the 2010 internet network management conference on research on enterprise networking*, pp. 3–3, 2010.
- [30] N. feamster, J. rexford, and E. zegura, “The road to sdn: an intellectual history of programmable networks,” *acm sigcomm computer communication review*, vol. 44, pp. 87–98, 2014.
- [31] D. J. wetherall, J. V. guttag, and D. L. tennenhouse, “Ants: a toolkit for building and dynamically deploying network protocols,” *IEEE, open architectures and network programming*, vol. 1998, pp. 117–129, 1998.
- [32] S. Bhattacharjee, K. L. calvert, and E. w. zegura, “An architecture for active networking,” in *high performance networking viianonymous*, pp. 265–279, 1997.
- [33] A. G. al, “A clean slate 4d approach to network control and management,” *acm sigcomm computer communication review*, vol. 35, pp. 41–54, 2005.
- [34] M. C. al, “Sane: a protection architecture for enterprise networks,” in *usenix security symposium pp*, vol. 50, 2006.
- [35] M. C. al, “ethane: taking control of the enterprise,” in *acm sigcomm computer communication review pp*, pp. 1–12, 2007.
- [36] S. Sorensen, *security implications of software-defined networks*. fierce telecom, 2012.
- [37] S. M. Kerner, *Is sdn secure*. enterprise networking planet, 2013.
- [38] N. N. function virtualization, *Network functions virtualization - introductory white paper*. october, 2012.

- [39] N. M. al, “Openflow: enabling innovation in campus networks,” *acm sigcomm computer communication review*, vol. 38, pp. 69–74, 2008.
- [40] E. Charniak, *Introduction to artificial intelligence*. Pearson Education India, 1985.
- [41] P. Kazienko, E. Lughofer, and B. Trawinski, “Hybrid and ensemble methods in machine learning j. ucs special issue,” *J Univers Comput Sci*, vol. 19, no. 4, pp. 457–461, 2013.
- [42] S. Umarani and D. Sharmila, “Predicting application layer ddos attacks using machine learning algorithms,” *International Journal of Computer, control Quantum and information Engineering*, vol. 8, no. 10, 2014.
- [43] Wikipedia, “Artificial neural network.”
- [44] W. Zhanyi, “The applications of deep learning on traffic identification,” tech. rep., Black Hat, 2015.
- [45] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [46] M. Stampar and K. Fertilj, “Artificial intelligence in network intrusion detection,” in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, pp. 1318–1323, IEEE, 2015.
- [47] C. Grosan and A. Abraham, “Intelligent systems—a modern approach, volume 17 of intelligent systems reference library,” 2011.

- [48] S. X. Wu and W. Banzhaf, “The use of computational intelligence in intrusion detection systems: A review,” *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.
- [49] J. Bi, K. Zhang, and X. Cheng, “Intrusion detection based on rbf neural network,” in *Information Engineering and Electronic Commerce, 2009. IEEEC’09. International Symposium on*, pp. 357–360, IEEE, 2009.
- [50] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen, “Som pak: The self-organizing map program package,” *Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science*, 1996.
- [51] Y. Fu, Y. Zhu, and H. Yu, “Study of neural network technologies in intrusion detection systems,” in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom’09. 5th International Conference on*, pp. 1–4, IEEE, 2009.
- [52] J. K. Williams, J. Craig, A. Cotter, and J. K. Wolff, “A hybrid machine learning and fuzzy logic approach to cit diagnostic development,” in *AMS Fifth Conference on Artificial Intelligence Applications to Environmental Science*, 2007.
- [53] K. M. Leung, “k-nearest neighbor algorithm for classification,” *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, 2007.
- [54] A. Ng, “Cs229 lecture notes,” *CS229 Lecture notes*, vol. 1, no. 1, pp. 1–3, 2000.
- [55] H. Mark, “K-means algorithm - gi07/m012 - ucl computer science,” tech. rep., University college London, 2014.

- [56] A. Reynolds, G. Richards, and V. Rayward-Smith, “The application of k-medoids and pam to the clustering of rules,” *Intelligent Data Engineering and Automated Learning–IDEAL 2004*, pp. 173–178, 2004.
- [57] K. P. Murphy, “Naive bayes classifiers,” *University of British Columbia*, 2006.
- [58] F. Xhafa, S. Caballé, A. Abraham, T. Daradoumis, and A. A. J. Perez, *Computational intelligence for technology enhanced learning*, vol. 273. Springer, 2010.
- [59] B. Schölkopf, “The kernel trick for distances,” in *Advances in neural information processing systems*, pp. 301–307, 2001.
- [60] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,” *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.
- [61] Q. V. Le, “Building high-level features using large scale unsupervised learning,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8595–8598, IEEE, 2013.
- [62] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, and J. Dean, “On rectified linear units for speech processing,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 3517–3521, IEEE, 2013.
- [63] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, “Semi-supervised recursive autoencoders for predicting sentiment distribu-

- tions,” in *Proceedings of the conference on empirical methods in natural language processing*, pp. 151–161, Association for Computational Linguistics, 2011.
- [64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [65] Z. Wang, “The applications of deep learning on traffic identification,” *BlackHat USA*, 2015.
- [66] M. E. Aminantoa and K. Kimb, “Deep learning in intrusion detection system: An overview,”
- [67] M. C. dacier al, “Network attack detection and defense–security challenges and opportunities of software-defined networking,” *dagstuhl reports*, vol. 6, pp. 1–28, 2017.
- [68] N. el moussaid, A. toumanari, and M. el azhari, “security analysis as software-defined security for sdn environment,” *fourth international conference on software defined systems (sds)*, vol. 2017, pp. 87–92, 2017.
- [69] M. C. dacier al, “security challenges and opportunities of software defined networking,” *IEEE security & privacy*, vol. 15, pp. 96–100, 2017.
- [70] D. B. rawat and S. R. reddy, “software defined networking architecture, security and energy efficiency: a survey,” *IEEE communications surveys & tutorials*, vol. 19, pp. 325–346, 2017.

- [71] T. wan, A. abdou, and P. C. van oorschot, "A framework and comparative analysis of control plane security of sdn and conventional networks." arxiv preprint, 2017.
- [72] D. dave and A. nagaraju, "A pragmatic analysis of security and integrity in software defined networks," in *Proceedings of international conference on communication and networks*, pp. 733–740, 2017.
- [73] T. dierks, "The transport layer security (tls) protocol version 1.2." 2008.
- [74] R. durner and w. kellerer, *The cost of security in the sdn control plane*. conext student workshop, 2015.
- [75] L. yao al, "Network security analyzing and modeling based on petri net and attack tree for sdn," in *computing, networking and communications (icnc)*, vol. 2016, pp. 1–5, 2016.
- [76] A. L. aliyu, P. bull, and A. abdallah, "A trust management framework for network applications within an sdn environment," *International conference on advanced information networking and applications workshops (waina)*, vol. 2017, no. 31, pp. 93–98, 2017.
- [77] T. dargahi al, *A survey on the security of stateful sdn data planes*. IEEE communications surveys & tutorials, 2017.
- [78] N. dayal al, "Research trends in security and ddos in sdn," *security and communication networks*, vol. 9, pp. 6386–6411, 2016.
- [79] D. kreutz, F. 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*, pp. 55–60, 2013.

- [80] A. lara, A. kolasani, and B. ramamurthy, “Network innovation using open-flow: a survey,” *IEEE communications surveys & tutorials*, vol. 16, pp. 493–512, 2014.
- [81] O. open networking foundation, “Openflow switch specification version 1.5.1 (protocol version 0x06),” *march*, vol. 26, p. 2015, 2015.
- [82] L. schehlmann, S. abt, and H. baier, “blessing or curse? revisiting security aspects of software-defined networking,” *International conference on network and service management (cnsm)*, vol. 2014, no. 10, pp. 382–387, 2014.
- [83] D. lim, J. kang, and I. joe, “A sdn-based network intrusion detection system to overcome upnp security drawbacks,” in *mobile and wireless technologies 2016anonymous*, pp. 117–126, 2016.
- [84] S. hong al, *Poisoning network visibility in software-defined networks: new attacks and countermeasures*. in ndss, 2015.
- [85] Y. feng al, “Research on the active ddos filtering algorithm based on ip flow,” *icnc’09. fifth international conference on natural computation*, vol. 2009, pp. 628–632, 2009.
- [86] R. braga, E. mota, and A. passito, “Lightweight ddos flooding attack detection using nox/openflow,” *IEEE 35th conference on local computer networks (lcn)*, vol. 2010, pp. 408–415, 2010.
- [87] C. yuhunag al, “A novel design for future on-demand service and security,” *the IEEE international conference on communication technology (icct)*, vol. 2010, no. 12, pp. 385–388, 2010.

- [88] Y. choi, *Implementation of content-oriented networking architecture (cona): a focus on ddos countermeasure*. in proc of 1st european netfpga developers workshop, 2010.
- [89] E. al shaer and S. al haj, “Flowchecker: configuration analysis and verification of federated openflow infrastructures,” *in proceedings of the acm workshop on assurable and usable security configuration*, vol. 3, pp. 37–44, 2010.
- [90] G. yao, J. bi, and P. xiao, “source address validation solution with openflow/nox architecture,” *IEEE international conference on network protocols (icnp)*, vol. 2011, no. 19, pp. 7–12, 2011.
- [91] P. porras al, “A security enforcement kernel for openflow networks,” *in proceedings of the first workshop on hot topics in software defined networks pp*, pp. 121–126, 2012.
- [92] A. K. al, “Veriflow: Verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 467–472, 2012.
- [93] J. H. Jafarian, E. Al-Shaer, and Q. Duan, “Openflow random host mutation: Transparent moving target defense using software defined networking,” *in Proceedings of the First Workshop on Hot Topics in Software Defined Networks pp*, pp. 127–132, 2012.
- [94] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” *in Proceedings of the First Workshop on Hot Topics in Software Defined Networks pp*, pp. 7–12, 2012.
- [95] D. H. al, “Pareto-optimal resilient controller placement in sdn-based core networks,” *in Teletraffic Congress (ITC)*, vol. 2013, no. 25, pp. 1–9, 2013.

- [96] S. Shin, H. Wang, and G. Gu, “A first step toward network security virtualization: from concept to prototype,” *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 2236–2249, 2015.
- [97] S. S. al, *FRESCO: Modular composable security services for software-defined networks*. in Ndss, 2013.
- [98] S. Shin and G. Gu, “Attacking software-defined networks: A first feasibility study,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking pp*, pp. 165–166, 2013.
- [99] K. Benton, L. J. Camp, and C. Small, “Openflow vulnerability assessment,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking pp*, pp. 151–152, 2013.
- [100] S. Scott-Hayward, G. O’Callaghan, and S. Sezer, “Sdn security: A survey,” *2013 IEEE SDN For Future Networks and Services (SDN)*, vol. 4, pp. 1–7, 2013.
- [101] S. N. al, “Enabling secure mobility with openflow,” *2013 IEEE SDN For Future Networks and Services (SDN4NFS)*, vol. 4, pp. 1–5, 2013.
- [102] K. G. al, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” *Computer Networks*, vol. 62, pp. 122–136, 2014.
- [103] S. R. C. al, “Payless: A low cost network monitoring framework for software defined networks,” in *Network Operations and Management Symposium (NOMS)*, vol. 2014, pp. 1–9, 2014.

- [104] S. S. al, “Rosemary: A robust, secure, and high-performance network operating system,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 78–89, 2014.
- [105] S. L. al, “A sdn-oriented ddos blocking scheme for botnet-based attacks,” in *Ubiquitous and Future Networks (ICUFN)*, vol. 2014, pp. 63–68, 2014.
- [106] A. Z. al, “Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions,” in *Network Operations and Management Symposium (NOMS)*, vol. 2014, pp. 1–9, 2014.
- [107] H. H. al, “Flowguard: Building robust firewalls for software-defined networks,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking pp*, pp. 97–102, 2014.
- [108] W. Y. al, “Openflow security threat detection and defense services,” *International Journal of Advanced Networking and Applications*, vol. 6, 2014.
- [109] M. Antikainen, T. Aura, and M. S”arel” a, “Spook in your network: Attacking an sdn with a compromised openflow switch,” in *Nordic Conference on Secure IT Systems*, pp. 229–244, 2014.
- [110] S. Matsumoto, S. Hitz, and A. Perrig, “Fleet: Defending sdns from malicious administrators,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking pp*, pp. 103–108, 2014.
- [111] F. A. Maldonado-Lopez, E. Calle, and Y. Donoso, “Detection and prevention of firewall-rule conflicts on software-defined networking,” in *Reliable Networks Design and Modeling (RNDM)*, vol. 2015, no. 7, pp. 259–265, 2015.

- [112] M. D. al, *SPHINX: Detecting security attacks in software-defined networks*. in *Ndss*, 2015.
- [113] B. W. al, “Ddos attack protection in the era of cloud computing and software-defined networking,” *Computer Networks*, vol. 81, pp. 308–319, 2015.
- [114] J. L. al, “Securing distributed sdn with ibc,” *Seventh International Conference On Ubiquitous and Future Networks (ICUFN)*, vol. 2015, pp. 921–925, 2015.
- [115] S. F. al, “Operetta: An openflow-based remedy to mitigate tcp synflood attacks against web servers,” *Computer Networks*, vol. 92, pp. 89–100, 2015.
- [116] J. W. Kang, S. H. Park, and J. You, “Mynah: Enabling lightweight data plane authentication for sdn controllers,” *International Conference On Computer Communication and Networks (ICCCN)*, vol. 2015, no. 24, pp. 1–6, 2015.
- [117] P. D. al, “Security analysis of software defined networking applications for monitoring and measurement: sflow and bigtap,” in *The 10th International Conference on Future Internet*, pp. 51–56, 2015.
- [118] Y. B.-I. al, “Enforsdn: Network policies enforcement with sdn,” *IFIP/IEEE International Symposium On Integrated Network Management (IM)*, vol. 2015, pp. 80–88, 2015.
- [119] A. Kamisiński and C. Fung, “Flowmon: Detecting malicious switches in software-defined networks,” in *Proceedings of the Workshop on Automated sion Making for Active Cyber Defense*, vol. 2015, pp. 39–45, Dec. 2015.

- [120] S. R. al, "Cindam: Customized information networks for deception and attack mitigation," *IEEE International Conference On Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, vol. 2015, pp. 114–119, 2015.
- [121] X. Wang, M. Chen, and C. Xing, "Sdsnm: A software-defined security networking mechanism to defend against ddos attacks," *Ninth International Conference On Frontier of Computer Science and Technology (FCST)*, vol. 2015, pp. 115–121, 2015.
- [122] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," *International Conference On Computing, Networking and Communications (ICNC)*, vol. 2015, pp. 77–81, 2015.
- [123] H. B. al, "Zombie pc detection and treatment model on software-defined network," in *Computer Science and its Applications Anonymous*, pp. 837–843, 2015.
- [124] S. Singh, R. Khan, and A. Agrawal, "Prevention mechanism for infrastructure based denial-of-service attack over software defined network," *International Conference On Computing, Communication & Automation (IC-CCA)*, vol. 2015, pp. 348–353, 2015.
- [125] S. L. al, "Controller scheduling for continued sdn operation under ddos attacks," *ElectroN. LetT.*, vol. 51, pp. 1259–1261, 2015.
- [126] C. Y. al, "Enabling security functions with sdn: A feasibility study," *Computer Networks*, vol. 85, pp. 19–35, 2015.
- [127] R. Z. al, "Study on authentication protocol of sdn trusted domain," in *Autonomous Decentralized Systems (ISADS)*, vol. 2015, pp. 281–284, 2015.

- [128] R. B. al, "Fingerprinting software-defined networks," *IEEE 23rd International Conference On Network Protocols (ICNP)*, vol. 2015, pp. 453–459, 2015.
- [129] P. V. T. al, "A multi-criteria-based ddos-attack prevention solution using software defined networking," *International Conference On Advanced Technologies for Communications (ATC)*, vol. 2015, pp. 308–313, 2015.
- [130] T. C. al, "An sdn-supported collaborative approach for ddos flooding detection and containment," *IEEE, Military Communications Conference, MILCOM*, vol. 2015, pp. 659–664, 2015.
- [131] M. A. Saleh and A. A. Manaf, "A novel protective framework for defeating denial of service and distributed denial of service attacks," *Scientific World-Journal*, vol. 2015, pp. 230–238, 2015.
- [132] R. S. al, *Towards autonomic DDoS mitigation using software defined networking*. in SENT 2015: NDSS Workshop on Security of Emerging Networking Technologies pp, 2015.
- [133] L. Mutu, R. Saleh, and A. Matrawy, "Improved sdn responsiveness to udp flood attacks," *IEEE Conference On Communications and Network Security (CNS)*, vol. 2015, pp. 715–716, 2015.
- [134] M. Z. Masoud, Y. Jaradat, and I. Jannoud, "On preventing arp poisoning attack utilizing software defined network (sdn) paradigm," *IEEE Jordan Conference On Applied Electrical Engineering and Computing Technologies (AEECT)*, vol. 2015, pp. 1–5, 2015.

- [135] O. F. al, “Sdn based architecture for iot and improvement of the security,” *IEEE 29th International Conference On Advanced Information Networking and Applications Workshops (WAINA)*, vol. 2015, pp. 688–693, 2015.
- [136] M. F. al, “Highly secure communication service architecture using sdn switch,” *10th Asia-Pacific Symposium On Information and Telecommunication Technologies (APSITT)*, vol. 2015, pp. 1–3, 2015.
- [137] G. Garg and R. Garg, “Detecting anomalies efficiently in sdn using adaptive mechanism,” *Fifth International Conference On Advanced Computing & Communication Technologies (ACCT)*, vol. 2015, pp. 367–370, 2015.
- [138] P. Chen and Y. Chen, “Implementation of sdn based network intrusion detection and prevention system,” *International Carnahan Conference On Security Technology (ICCST)*, vol. 2015, pp. 141–146, 2015.
- [139] T. V. Tran and H. Ahn, “A network topology-aware selectively distributed firewall control in sdn,” *International Conference On Information and Communication Technology Convergence (ICTC)*, vol. 2015, pp. 89–94, 2015.
- [140] C. Banse and S. Rangarajan, “A secure northbound interface for sdn applications,” *IEEE, in Trustcom/BigDataSE/ISPA*, vol. 2015, pp. 834–839, 2015.
- [141] M. W. al, “An approach for protecting the openflow switch from the saturation attack,” *4th National Conference on Electrical, Electronics and Computer Engineering*, 2016.

- [142] D. Chalyy, E. Nikitin, and E. J. Antoshina, “A simple information flow security model for software-defined networks,” *17th Conference of Open Innovations Association FRUCT. Yaroslavl, Russia*, pp. 276–282, 2015.
- [143] V. K. Reddy and D. Sreenivasulu, “Software-defined networking with ddos attacks in cloud computing,” *International journal of innovative technologies*, vol. 04, pp. 3779–3783, December 2016.
- [144] J. S. al, “Poster: Ofx: Enabling openflow extensions for switch-level security applications,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1678–1680, 2015.
- [145] W. H. al, “Honeymix: Toward sdn-based intelligent honeynet,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 1–6, 2016.
- [146] R. Costa and F. Ramos, “An sdn-based approach to enhance bgp security.” arXiv Preprint, 2016.
- [147] C. Lee and S. Shin, “Shield: An automated framework for static analysis of sdn applications,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 29–34, 2016.
- [148] T. Park, Y. Kim, and S. Shin, “Unisafe: A union of security actions for software switches,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 13–18, 2016.
- [149] J. Li and T. Wolf, “A one-way proof-of-work protocol to protect controllers in software-defined networks,” *ACM/IEEE Symposium On Architectures*

- for Networking and Communications Systems (ANCS)*, vol. 2016, pp. 123–124, 2016.
- [150] K. Giotis, G. Androulidakis, and V. Maglaris, “A scalable anomaly detection and mitigation architecture for legacy networks via an openflow middlebox,” *Security and Communication Networks*, vol. 13, pp. 1958–1970, 2016.
- [151] A. Lara and B. Ramamurthy, “Opensec: Policy-based security using software-defined networking,” *IEEE Transactions on Network and Service Management*, vol. 13, pp. 30–42, 2016.
- [152] J. H. C. Jr, R. J. Clark, and H. L. O. Iii, “Leveraging sdn to improve the security of dhcp,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 35–38, 2016.
- [153] Y. C. al, “Sd-anti-ddos: Fast and efficient ddos defense in software-defined networks,” *Journal of Network and Computer Applications*, vol. 68, pp. 65–79, 2016.
- [154] Z. Z. al, “Sdn-based double hopping communication against sniffer attack,” *Mathematical Problems in Engineering*, vol. 2016, 2016.
- [155] M. A. Lopez, D. M. F. Mattos, and O. C. M. Duarte, “An elastic intrusion detection system for software networks,” *Annals of Telecommunications*, vol. 71, pp. 595–605, 2016.
- [156] X. Chen and S. Yu, “Cipa: A collaborative intrusion prevention architecture for programmable network and sdn,” *CompuT. SecuR.*, vol. 58, pp. 1–19, 2016.

- [157] X. P. al, “Hogmap: Using sdns to incentivize collaborative security monitoring,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 7–12, 2016.
- [158] G. Garg and R. Garg, “Security of networks using efficient adaptive flow counting for anomaly detection in sdn,” in *Artificial Intelligence and Evolutionary Computations in Engineering Systems* Anonymous, pp. 667–674, 2016.
- [159] Y. Park, S. Chang, and L. M. Krishnamurthy, “Watermarking for detecting freeloader misbehavior in software-defined networks,” *International Conference On Computing, Networking and Communications (ICNC)*, vol. 2016, pp. 1–6, 2016.
- [160] H. Xu, C. Wang, and H. Chen, “An extension approach for threat detection and defense of software-defined networking,” *International Journal of Security and its Applications*, vol. 10, pp. 365–374, 2016.
- [161] S. H. al, *Towards SDN-defined programmable BYOD (bring your own device) security*. in Ndss, 2016.
- [162] J. L. al, “Leveraging software-defined networking for security policy enforcement,” *InF. Sci*, vol. 327, pp. 288–299, 2016.
- [163] M. A. al, “A framework for security enhancement in sdn-based datacenters,” *8th IFIP International Conference On New Technologies, Mobility and Security (NTMS)*, vol. 2016, pp. 1–4, 2016.
- [164] K. K. Karmakar, V. Varadharajan, and U. Tupakula, “On the design and implementation of a security architecture for software defined networks,”

- in *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 671–678, 2016
IEEE 18th International Conference On, 2016.
- [165] F. I. K. al, “Securing software defined network against rogue controllers,” *International Conference on Cyber Security & Digital Forensics*, 2016.
- [166] S. L. al, *DELTA: A security assessment framework for software-defined networks*. in Proceedings of NDSS, 2017.
- [167] M. Monshizadeh, V. Khatri, and R. Kantola, “Detection as a service: An sdn application,” *19th International Conference On Advanced Communication Technology (ICACT)*, vol. 2017, pp. 285–290, 2017.
- [168] W. Lee and N. Kim, “Efficient service chaining framework based on software defined network for domain independent,” *International Journal of Applied Engineering Research*, vol. 12, pp. 2301–2305, 2017.
- [169] A. Shaghghi, M. A. Kaafar, and S. Jha, “Wedgetail: An intrusion prevention system for the data plane of software defined networks,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 849–861, 2017.
- [170] X. Q. al, “A security controller-based software defined security architecture,” *20th Conference On Innovations in Clouds, Internet and Networks (ICIN)*, vol. 2017, pp. 191–195, 2017.
- [171] A. L. Aliyu, P. Bull, and A. Abdallah, “A trust management framework for network applications within an sdn environment,” *1st International Con-*

- ference On Advanced Information Networking and Applications Workshops (WAINA)*, vol. 2017, no. 31, pp. 93–98, 2017.
- [172] R. D. al, “Detecting and mitigating denial of service attacks against the data plane in software defined networks,” *IEEE Conference On Network Softwarization (NetSoft)*, vol. 2017, pp. 1–6, 2017.
- [173] R. S. al, “Adaptive policy-driven attack mitigation in sdn,” *Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures*, vol. 1, 2017.
- [174] K. W. al, “Sdn testbed for validation of cross-layer data-centric security policies,” *International Conference On Military Communications and Information Systems (ICMCIS)*, vol. 2017, pp. 1–6, 2017.
- [175] A. Basit and N. Ahmed, “Path diversity for inter-domain routing security,” *14th International Bhurban Conference On Applied Sciences and Technology (IBCAST)*, vol. 2017, pp. 384–391, 2017.
- [176] M. W. al, “Perm-guard: Authenticating the validity of flow rules in software defined networking,” *Journal of Signal Processing Systems*, vol. 86, pp. 157–173, 2017.
- [177] D. V. Bernardo and B.B. Chua, “Introduction and analysis of sdn and nfv security architecture (sn-seca),” *IEEE 29th International Conference On Advanced Information Networking and Applications (AINA)*, vol. 2015, pp. 796–801, 2015.
- [178] M. Brooks and B. Yang, “A man-in-the-middle attack against openday-light sdn controller,” in *Proceedings of the 4th Annual ACM Conference on Research in Information Technology*, pp. 45–49, 2015.

- [179] T. Alharbi, M. Portmann, and F. Pakzad, “The (in) security of topology discovery in software defined networks,” *IEEE 40th Conference On Local Computer Networks (LCN)*, vol. 2015, pp. 502–505, 2015.
- [180] S. Lee, C. Yoon, and S. Shin, “The smaller, the shrewder: A simple malicious application can kill an entire sdn environment,” in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, vol. 2016, pp. 23–28, 2016.
- [181] B. E. Ujcich, U. Thakore, and W. H. Sanders, “Attain: An attack injection framework for software-defined networking,” *47th Annual IEEE/IFIP International Conference On Dependable Systems and Networks (DSN)*, vol. 2017, pp. 567–578, 2017.
- [182] A. Aseeri, N. Netjinda, and R. Hewett, “Alleviating eavesdropping attacks in software-defined networking data plane,” in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research* pp, 1, 2017.
- [183] T. Nguyen and M. Yoo, “Analysis of link discovery service attacks in sdn controller,” *International Conference On Information Networking (ICOIN)*, vol. 2017, pp. 259–261, 2017.
- [184] R. Kloti, V. Kotronis, and P. Smith, “Openflow: A security analysis,” *21st IEEE International Conference On Network Protocols (ICNP)*, vol. 2013, pp. 1–6, 2013.
- [185] S. Scott-Hayward, G. O’Callaghan, and S. Sezer, “Sdn security: A survey,” *IEEE SDN For Future Networks and Services (SDNFNS)*, vol. 4, pp. 1–7, 2013.

- [186] S. S. al, “Avant-guard: Scalable and vigilant switch flow management in software-defined networks,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 413–424, 2013.
- [187] S. H. al, “Threat modeling-uncover security design flaws using the stride approach,” *MSDN Magazine-Louisville*, pp, pp. 68–75, 2006.
- [188] S. K. al, “Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art,” *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 303–324, 2017.
- [189] S. Veena and R. Manju, “A novel approach to dynamic policy based security in sdn: A survey,” *International Journal of Computer Science and Engineering*, vol. 5, pp. 1566–1573, 2017.
- [190] S. R. C. al, “Payless: A low cost network monitoring framework for software defined networks,” *IEEE, Network Operations and Management Symposium (NOMS)*, vol. 2014, pp. 1–9, 2014.
- [191] A. K. al, “Veriflow: Verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 467–472, 2012.
- [192] Z. Z. al, “Sdn-based double hopping communication against sniffer attack,” *Mathematical Problems in Engineering*, vol. 2016, 2016.
- [193] B. W. al, “Ddos attack protection in the era of cloud computing and software-defined networking,” *Computer Networks*, vol. 81, pp. 308–319, 2015.

- [194] S. L. al, "A sdn-oriented ddos blocking scheme for botnet-based attacks," *Sixth International Conf On Ubiquitous and Future Networks (ICUFN)*, vol. 2014, pp. 63–68, 2014.
- [195] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," *IEEE 35th Conference On Local Computer Networks (LCN)*, vol. 2010, pp. 408–415, 2010.
- [196] Y. Choi, *Implementation of content-oriented networking architecture (CONA): A focus on DDoS countermeasure*. in Proc of 1st European NetFPGA Developers Workshop, 2010.
- [197] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International Workshop on Recent Advances in Intrusion Detection pp*, pp. 161–180, 2011.
- [198] R. B. al, "Fingerprinting software-defined networks," *IEEE 23rd International Conference On Network Protocols (ICNP)*, vol. 2015, pp. 453–459, 2015.
- [199] S. Seeber, L. Stiemert, and G. D. Rodosek, "Towards an sdn-enabled ids environment," *IEEE Conference On Communications and Network Security (CNS)*, vol. 2015, pp. 751–752, 2015.
- [200] D. Jankowski and M. Amanowicz, *Intrusion detection in software defined networks with self-organized maps*. Journal of Telecommunications and Information Technology, 2015.
- [201] N. T. Van, H. Bao, and T. N. Thinh, "An anomaly-based intrusion detection architecture integrated on openflow switch," in *Proceedings of the 6th*

- International Conference on Communication and Network Security*, pp. 99–103, 2016.
- [202] J. A. al, “Dynamic network security protection on cloud computing,” *International Education and Research Journal*, vol. 2, 2016.
- [203] H. H. al, “Flowguard: Building robust firewalls for software-defined networks,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking pp*, pp. 97–102, 2014.
- [204] T. T. Huong and N. H. Thanh, “Software defined networking-based one-packet ddos mitigation architecture,” in *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication pp*, 110, 2017.
- [205] T. V. Phan, N. K. Bao, and M. Park, “Distributed-som: A novel performance bottleneck handler for large-sized software-defined networks under flooding attacks,” *Journal of Network and Computer Applications*, vol. 91, pp. 14–25, 2017.
- [206] J. K. al, “Sdn-based security services using interface to network security functions,” *International Conference On Information and Communication Technology Convergence (ICTC)*, vol. 2015, pp. 526–529, 2015.
- [207] A. AlEroud and I. Alsmadi, “Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach,” *Journal of Network and Computer Applications*, vol. 80, pp. 152–164, 2017.
- [208] M. A. al, “A framework for security enhancement in sdn-based datacenters,” *8th IFIP International Conference On New Technologies, Mobility and Security (NTMS)*, vol. 2016, pp. 1–4, 2016.

- [209] A. G. al, “Five sdn-oriented directions in information security,” *First International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, vol. 2014, pp. 1–4, 2014.
- [210] Y. Chiu and P. Lin, “Rapid detection of disobedient forwarding on compromised openflow switches,” *International Conference On Computing, Networking and Communications (ICNC)*, vol. 2017, pp. 672–677, 2017.
- [211] M. W. al, “An approach for protecting the openflow switch from the saturation attack,” *4th National Conference on Electrical, Electronics and Computer Engineering*, 2016.
- [212] R. S. al, “Flowvisor: A network virtualization layer,” *OpenFlow Switch Consortium, Tech*, vol. 1, p. 132, 2009.
- [213] P. Lin, P. Li, and V. L. Nguyen, “Inferring openflow rules by active probing in software-defined networks,” *19th International Conference On Advanced Communication Technology (ICACT)*, vol. 2017, pp. 415–420, 2017.
- [214] S. Shin, H. Wang, and G. Gu, “A first step toward network security virtualization: from concept to prototype,” *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 2236–2249, 2015.
- [215] K. K. Karmakar, V. Varadharajan, and U. Tupakula, “Mitigating attacks in software defined network (sdn),” *Fourth International Conference On Software Defined Systems (SDS)*, vol. 2017, pp. 112–117, 2017.
- [216] K. W. al, “Livesec: Towards effective security management in large-scale production networks,” *32nd International Conference On Distributed Computing Systems Workshops (ICDCSW)*, vol. 2012, pp. 451–460, 2012.

- [217] A. M. AbdelSalam, A. B. El-Sisi, and V. Reddy, “Mitigating arp spoofing attacks in software-defined networks,” *International Conference on Computer and Computing Technologies*, 2015.
- [218] A. H. al, “Sdn security plane: An architecture for resilient security services,” in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), Germany*, 2016.
- [219] W. Lee and N. Kim, “Security policy scheme for an efficient security architecture in software-defined networking,” *Information*, vol. 8, p. 65, 2017.
- [220] A. H. al, “Sdn verification plane for consistency establishment,” in *The Twenty-First IEEE Symposium on Computers and Communications, Messina, Italy*, 2016.
- [221] L. Barki, A. Shidling, N. Meti, D. Narayan, and M. M. Mulla, “Detection of distributed denial of service attacks in software defined networks,” in *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, pp. 2576–2581, IEEE, 2016.
- [222] A. Akhunzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran, and S. Guizani, “Securing software defined networks: taxonomy, requirements, and open issues,” *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.
- [223] H. Bai, “A survey on artificial intelligence for network routing problems,” *NM, USA: University of New Mexico*, 2007.
- [224] U. Mustafa, M. M. Masud, Z. Trabelsi, T. Wood, and Z. A. Harthi, “Fire-wall performance optimization using data mining techniques,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pp. 934–940, IEEE, 2013.

- [225] D. Mukherjee and S. Acharyya, “Ant colony optimization technique applied in network routing problem,” *International journal of computer applications*, vol. 1, no. 15, pp. 66–73, 2010.
- [226] S. Dotcenko, A. Vladyko, and I. Letenko, “A fuzzy logic-based information security management for software-defined networks,” in *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pp. 167–171, IEEE, 2014.
- [227] J. Mikians, P. Barlet-Ros, J. Sanjuas-Cuxart, and J. Solé-Pareta, “A practical approach to portscan detection in very high-speed links,” in *International Conference on Passive and Active Network Measurement*, pp. 112–121, Springer, 2011.
- [228] M. M. Williamson, “Throttling viruses: Restricting propagation to defeat malicious mobile code,” in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pp. 61–68, IEEE, 2002.
- [229] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, “Shallow and deep networks intrusion detection system: A taxonomy and survey,” *arXiv preprint arXiv:1701.02145*, 2017.
- [230] R. Braga, E. Mota, and A. Passito, “Lightweight ddos flooding attack detection using nox/openflow,” in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 408–415, IEEE, 2010.
- [231] S. Deepa and B. A. Devi, “A survey on artificial intelligence approaches for medical image classification,” *Indian Journal of Science and Technology*, vol. 4, no. 11, pp. 1583–1595, 2011.

- [232] Q. Yan, F. R. Yu, Q. Gong, and J. Li, “Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.
- [233] T. Srinivasan, V. Vijaykumar, and R. Chandrasekar, “A self-organized agent-based architecture for power-aware intrusion detection in wireless ad-hoc networks,” in *Computing & Informatics, 2006. ICOCI’06. International Conference on*, pp. 1–6, IEEE, 2006.
- [234] P. Lichodziejewski, A. N. Zincir-Heywood, and M. I. Heywood, “Dynamic intrusion detection using self-organizing maps,” in *The 14th Annual Canadian Information Technology Security Symposium (CITSS)*, 2002.
- [235] B. C. Rhodes, J. A. Mahaffey, and J. D. Cannady, “Multiple self-organizing maps for intrusion detection,” in *Proceedings of the 23rd national information systems security conference*, pp. 16–19, 2000.
- [236] A. K. Ghosh and A. Schwartzbard, “A study in using neural networks for anomaly and misuse detection.,” in *USENIX security symposium*, vol. 99, p. 12, 1999.
- [237] R. P. Lippmann and R. K. Cunningham, “Improving intrusion detection performance using keyword selection and neural networks,” *Computer Networks*, vol. 34, no. 4, pp. 597–603, 2000.
- [238] W. Li, “Using genetic algorithm for network intrusion detection,” *Proceedings of the United States Department of Energy Cyber Security Group*, vol. 1, pp. 1–8, 2004.

- [239] S. Mukkamala, A. H. Sung, and A. Abraham, “Modeling intrusion detection systems using linear genetic programming approach,” in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 633–642, Springer, 2004.
- [240] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *Computers & Security*, vol. 28, no. 1, pp. 18–28, 2009.
- [241] W. Chimphee, A. H. Abdullah, M. N. M. Sap, S. Srinoy, and S. Chimphee, “Anomaly-based intrusion detection using fuzzy rough clustering,” in *Hybrid Information Technology, 2006. ICHIT’06. International Conference on*, vol. 1, pp. 329–334, IEEE, 2006.
- [242] J. E. Dickerson and J. A. Dickerson, “Fuzzy network profiling for intrusion detection,” in *Fuzzy Information Processing Society, 2000. NAFIPS. 19th International Conference of the North American*, pp. 301–306, IEEE, 2000.
- [243] Z. Ma and A. Kaban, “K-nearest-neighbours with a novel similarity measure for intrusion detection,” in *Computational Intelligence (UKCI), 2013 13th UK Workshop on*, pp. 266–271, IEEE, 2013.
- [244] Y. Liao and V. R. Vemuri, “Use of k-nearest neighbor classifier for intrusion detection,” *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002.
- [245] L. He, C. Xu, and Y. Luo, “vtc: Machine learning based traffic classification as a virtual network function,” in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 53–56, ACM, 2016.

- [246] M. Panda and M. R. Patra, “Network intrusion detection using naive bayes,” *International journal of computer science and network security*, vol. 7, no. 12, pp. 258–263, 2007.
- [247] N. B. Amor, S. Benferhat, and Z. Elouedi, “Naive bayesian networks in intrusion detection systems,” in *Proc. Workshop on Probabilistic Graphical Models for Classification, 14th European Conference on Machine Learning (ECML) and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), 23rd September, in Cavtat–Dubrovnik, Croatia*, p. 11, 2003.
- [248] T. Abbas, A. Bouhoula, and M. Rusinowitch, “Protocol analysis in intrusion detection using decision tree,” in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 1, pp. 404–408, IEEE, 2004.
- [249] Y. Bouzida, F. Cuppens, N. Cuppens-Boulahia, and S. Gombault, “Efficient intrusion detection using principal component analysis,” in *3^{ème} Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, France*, pp. 381–395, 2004.
- [250] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning intrusion detection: supervised or unsupervised?,” *Image Analysis and Processing–ICIAP 2005*, pp. 50–57, 2005.
- [251] L. Khan, M. Awad, and B. Thuraisingham, “A new intrusion detection system using support vector machines and hierarchical clustering,” *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 16, no. 4, pp. 507–521, 2007.

- [252] R.-C. Chen, K.-F. Cheng, Y.-H. Chen, and C.-F. Hsieh, "Using rough set and support vector machine for network intrusion detection system," in *Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on*, pp. 465–470, IEEE, 2009.
- [253] S. A. Mulay, P. Devale, and G. Garje, "Intrusion detection system using support vector machine and decision tree," *International Journal of Computer Applications*, vol. 3, no. 3, pp. 40–43, 2010.
- [254] D. S. Kim, S. M. Lee, and J. S. Park, "Building lightweight intrusion detection system based on random forest," in *International Symposium on Neural Networks*, pp. 224–230, Springer, 2006.
- [255] J. Zhang and M. Zulkernine, "Network intrusion detection using random forests.," in *PST*, 2005.
- [256] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pp. 21–26, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.
- [257] M. E. Aminantoa and K. Kimb, "Deep learning in intrusion detection system: An overview,"
- [258] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Information Networking (ICOIN), 2017 International Conference on*, pp. 712–717, IEEE, 2017.

- [259] N. Gao, L. Gao, Q. Gao, and H. Wang, “An intrusion detection model based on deep belief networks,” in *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, pp. 247–252, IEEE, 2014.
- [260] Q. Niyaz, W. Sun, and A. Y. Javaid, “A deep learning based ddos detection system in software-defined networking (sdn),” *arXiv preprint arXiv:1611.07400*, 2016.
- [261] R. Inc., “Defenseflow – sdn based network ddos, application dos and apt protection,” Tech. Rep. Solution Brief, Radware Inc., 2014.
- [262] T. Uzakgider, C. Cetinkaya, and M. Sayit, “Learning-based approach for layered adaptive video streaming over sdn,” *Computer Networks*, vol. 92, pp. 357–368, 2015.
- [263] M. Latah and L. Toker, “Application of artificial intelligence to software defined networking: A survey,” *Indian Journal of Science and Technology*, vol. 9, no. 44, 2016.
- [264] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, “Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn,” in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp. 27–35, IEEE, 2016.
- [265] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, “Ai 2: training a big data machine to defend,” in *Big Data Security on Cloud (Big-DataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016 IEEE 2nd International Conference on*, pp. 49–54, IEEE, 2016.

- [266] B. Davis, “Leveraging the load balancer to fight ddos,” *SANS GIAC Gold Certification Report*, 2009.
- [267] A. Califano, E. Dincelli, and S. Goel, “Using features of cloud computing to defend smart grid against ddos attacks,” in *10th Annual symposium on information assurance (Asia 15)*, ALBANY, pp. 44–50, 2015.
- [268] C. Chen-Xiao and X. Ya-Bin, “Research on load balance method in sdn,” *International Journal of Grid and Distributed Computing*, vol. 9, no. 1, pp. 25–36, 2016.
- [269] A. M. Ruelas and C. E. Rothenberg, “Implementation of neural switch using openflow as load balancing method in data center,” *Campinas, Brasil: University of Campinas*, 2015.
- [270] L.-D. Chou, Y.-T. Yang, Y.-M. Hong, J.-K. Hu, and B. Jean, *A genetic-based load balancing algorithm in openflow network*, pp. 411–417. Advanced Technologies, Embedded and Multimedia for Human-centric Computing, Springer, 2014.
- [271] R. Blaguer, “Flow embedding algorithms for software defined audio networks,” *Master Thesis ETH*, pp. 1–60, 2014.
- [272] O. Dobrijevic, M. Santl, and M. Matijasevic, “Ant colony optimization for qoe-centric flow routing in software-defined networks,” in *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 274–278, IEEE, 2015.
- [273] A. B. Letaifa, G. Maher, and S. Mouna, “Ml based qoe enhancement in sdn context: Video streaming case,” in *Wireless Communications and Mobile*

- Computing Conference (IWCMC), 2017 13th International*, pp. 103–108, IEEE, 2017.
- [274] S. Kuklinski, J. Wytrebowicz, K. T. Dinh, and E. Tantar, *Application of cognitive techniques to network management and control*, pp. 79–93. EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V, Springer, 2014.
- [275] E. Tantar, M. R. Palattella, T. Avanesov, M. Kantor, and T. Engel, *Cognition: A tool for reinforcing security in software defined networks*, pp. 61–78. EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V, Springer, 2014.
- [276] M. Guidry, “Artificial intelligence in networking: Ant colony optimization,”
- [277] A. Horn, A. Kheradmand, and M. R. Prasad, “Delta-net: Real-time network verification using atoms.,” in *NSDI*, pp. 735–749, 2017.
- [278] S. A. Seshia, D. Sadigh, and S. S. Sastry, “Towards verified artificial intelligence,” *arXiv preprint arXiv:1606.08514*, 2016.
- [279] T. Menzies and C. Pecheur, “Verification and validation and artificial intelligence,” *Advances in computers*, vol. 65, pp. 153–201, 2005.
- [280] W. Liu, R. B. Bobba, S. Mohan, and R. H. Campbell, “Inter-flow consistency: A novel sdn update abstraction for supporting inter-flow constraints,” in *Communications and Network Security (CNS), 2015 IEEE Conference on*, pp. 469–478, IEEE, 2015.

- [281] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, “Towards adaptive state consistency in distributed sdn control plane,” in *2017 IEEE International Conference on Communications (ICC)*, 2017.
- [282] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “Sdn verification plane for consistency establishment,” in *The Twenty-First IEEE Symposium on Computers and Communications*, 2016.
- [283] M. Aslan and A. Matrawy, “Adaptive consistency for distributed sdn controllers,” in *IEEE, 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, 2016.
- [284] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, “Veriflow: Verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.
- [285] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, “Vericon: Towards verifying controller programs in software-defined networks,” in *ACM SIGPLAN Notices*, vol. 49, pp. 282–293, ACM, 2014.
- [286] R. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, “A verification platform for sdn-enabled applications,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pp. 337–342, IEEE, 2014.
- [287] M. Aslan and A. Matrawy, “A clustering-based consistency adaptation strategy for distributed sdn controllers,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018.
- [288] M. Kang, E.-Y. Kang, D.-Y. Hwang, B.-J. Kim, K.-H. Nam, M.-K. Shin, and J.-Y. Choi, “Formal modeling and verification of sdn-openflow,” in

- Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pp. 481–482, IEEE, 2013.
- [289] V. Podymov and U. Popesko, “Uppaal-based software-defined network verification,” in *Tools & Methods of Program Analysis (TMPA), 2013*, pp. 9–14, IEEE, 2013.
- [290] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, “Veriflow: Verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.
- [291] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, “Flow-level state transition as a new switch primitive for sdn,” in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 61–66, ACM, 2014.
- [292] R. Mahajan and R. Wattenhofer, “On consistent updates in software defined networks,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, p. 20, ACM, 2013.
- [293] D. Sethi, S. Narayana, and S. Malik, “Abstractions for model checking sdn controllers,” in *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pp. 145–148, IEEE, 2013.
- [294] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “Sdn security plane: An architecture for resilient security services,” in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, 2016.
- [295] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “Sdn for mptcp: An enhanced architecture for large data transfers in datacenters,” in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2017.

- [296] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “Sdn vanets in 5g: An architecture for resilient security services,” in *2017 Fourth International Conference on Software Defined Systems (SDS)*, pp. 67–74, 2017.
- [297] A. Hussein, A. Kayssi, I. Elhajj, and A. Chehab, “Sdn for quic: an enhanced architecture with improved connection establishment,” pp. 2136–2139, 04 2018.
- [298] A. Hussein, L. Chadad, N. Adalian, A. Chehab, I. H. Elhajj, and A. Kayssi, “Software-defined networking (sdn): the security review,” *Journal of Cyber Security Technology*, vol. 0, no. 0, pp. 1–66, 2019.
- [299] H. Ali, , S. Ola, A. Sarah, E. Imad, C. Ali, and K. Ayman, *SDN and edge computing: key enablers toward the 5G evolution*. Telecommunications, Institution of Engineering and Technology, 2017.
- [300] A. Hussein, I. Elhajj, A. Chehab, and A. Kayssi, “Securing diameter: Comparing tls, dtls, and ipsec,” pp. 1–8, 11 2016.
- [301] A. Hussein, A. Chehab, A. Kayssi, and I. H. Elhajj, “Machine learning for network resilience: The start of a journey,” in *2018 Fifth International Conference on Software Defined Systems (SDS)*, pp. 59–66, April 2018.
- [302] A. Hussein, O. Salman, A. Chehab, I. Elhajj, and A. Kayssi, “Machine learning for network resiliency and consistency,” pp. 146–153, 06 2019.
- [303] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.
- [304] M. Riedmiller and H. Braun, “A direct adaptive method for faster back-propagation learning: The rprop algorithm,” in *Neural Networks, 1993., IEEE International Conference on*, pp. 586–591, IEEE, 1993.

- [305] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pp. 1–6, IEEE, 2009.
- [306] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *th International Conference on Information Systems Security and Privacy (ICISSP)*,, vol. 32, January 2018.
- [307] D. Xu, Z. Li, W. Wu, X. Ding, and D. Qu, “Convergence of gradient descent algorithm for a recurrent neuron,” *Advances in Neural Networks–ISNN 2007*, pp. 117–122, 2007.
- [308] Mininet, “Mininet.”
- [309] Mini-Edit, “Miniedit.”
- [310] “hashlib.”
- [311] Wikipedia, “Sha-2 (secure hash algorithm 2),” 3 February 2019, at 00:36 (UTC).