

# **WESTERN SYDNEY** UNIVERSITY



## **Deep Learning: Enhancing the Security of Software-Defined Networks**

Ahmed Dawoud

M.Sc. Software Development & M.Sc. Information and Communication Technologies

A thesis submitted for the degree of

Doctor of Philosophy

School of Computing, Engineering and Mathematics

Western Sydney University

June 2019

## Abstract

Software-defined networking (SDN) is a communication paradigm that promotes network flexibility and programmability by separating the control plane from the data plane. SDN consolidates the logic of network devices into a single entity known as the controller. SDN raises significant security challenges related to its architecture and associated characteristics such as programmability and centralisation. Notably, security flaws pose a risk to controller integrity, confidentiality and availability.

The SDN model introduces separation of the forwarding and control planes. It detaches the control logic from switching and routing devices, forming a central plane or network controller that facilitates communications between applications and devices. The architecture enhances network resilience, simplifies management procedures and supports network policy enforcement. However, it is vulnerable to new attack vectors that can target the controller. Current security solutions rely on traditional measures such as firewalls or intrusion detection systems (IDS). An IDS can use two different approaches: signature-based or anomaly-based detection. The signature-based approach is incapable of detecting zero-day attacks, while anomaly-based detection has high false-positive and false-negative alarm rates. Inaccuracies related to false-positive attacks may have significant consequences, specifically from threats that target the controller. Thus, improving the accuracy of the IDS will enhance controller security and, subsequently, SDN security.

A centralised network entity that controls the entire network is a primary target for intruders. The controller is located at a central point between the applications and the data plane and has two interfaces for plane communications, known as northbound and southbound, respectively. Communications between the controller, the application and data planes are prone to various types of attacks, such as eavesdropping and tampering. The controller software is vulnerable to attacks such as buffer and stack overflow, which enable remote code execution that can result in attackers taking control of the entire network. Additionally, traditional network attacks are more destructive.

This thesis introduces a threat detection approach aimed at improving the accuracy and efficiency of the IDS, which is essential for controller security. To evaluate the

effectiveness of the proposed framework, an empirical study of SDN controller security was conducted to identify, formalise and quantify security concerns related to SDN architecture. The study explored the threats related to SDN architecture, specifically threats originating from the existence of the control plane.

The framework comprises two stages, involving the use of deep learning (DL) algorithms and clustering algorithms, respectively. DL algorithms were used to reduce the dimensionality of inputs, which were forwarded to clustering algorithms in the second stage. Features were compressed to a single value, simplifying and improving the performance of the clustering algorithm. Rather than using the output of the neural network, the framework presented a unique technique for dimensionality reduction that used a single value—reconstruction error—for the entire input record. The use of a DL algorithm in the pre-training stage contributed to solving the problem of dimensionality related to  $k$ -means clustering. Using unsupervised algorithms facilitated the discovery of new attacks.

Further, this study compares generative energy-based models (restricted Boltzmann machines) with non-probabilistic models (autoencoders). The study implements TensorFlow in four scenarios. Simulation results were statistically analysed using a confusion matrix, which was evaluated and compared with similar related works.

The proposed framework, which was adapted from existing similar approaches, resulted in promising outcomes and may provide a robust prospect for deployment in modern threat detection systems in SDN. The framework was implemented using TensorFlow and was benchmarked to the KDD99 dataset. Simulation results showed that the use of the DL algorithm to reduce dimensionality significantly improved detection accuracy and reduced false-positive and false-negative alarm rates. Extensive simulation studies on benchmark tasks demonstrated that the proposed framework consistently outperforms all competing approaches. This improvement is a further step towards the development of a reliable IDS to enhance the security of SDN controllers.

## Declaration of Original Work

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university and that to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in the text.

Signe



\_\_28\_\_ / \_\_11\_\_ / \_\_2019

## **Acknowledgements**

First and foremost I want to express my gratitude and appreciations to my supervisor Dr. Seyed Shahrestani for his guidance, support, and motivation during my prolonged PhD journey. I appreciate all his contributions of time, ideas and expertise to make this work possible. I would like to express my special appreciation and thanks to Dr. Chun Raun, for her patient listening and vital comments. Also, it is inevitable to express my thanks to graduate research school administrative staff for their continuous and prompt support.

During the long and intense years of my studies and research I lost people who shared me the dreams, I would like to honour their memory in this work, to my father and my uncle Ibrahim. A special thanks to my beloved wife Sara, and my sons Hamza and Mohamed without them I would finish this work a year and half earlier.

# Table of Contents

<b>Abstract</b> .....	<b>1</b>
<b>Declaration of Original Work</b> .....	<b>3</b>
<b>Acknowledgements</b> .....	<b>4</b>
<b>Table of Contents</b> .....	<b>5</b>
<b>List of Figures</b> .....	<b>8</b>
<b>List of Tables</b> .....	<b>10</b>
<b>List of Abbreviations</b> .....	<b>11</b>
<b>Publications</b> .....	<b>13</b>
<b>Chapter 1: Introduction</b> .....	<b>15</b>
1.1 Introduction .....	15
1.2 Motivation .....	17
1.3 Research Methodology.....	19
1.4 Research Questions and Scope.....	23
1.5 Contributions.....	23
1.6 Thesis Structure.....	25
<b>Chapter 2: Software-Defined Networks</b> .....	<b>28</b>
2.1 Introduction .....	28
2.2 Software-Defined Network Architecture .....	30
2.2.1 Software-Defined Network Controller.....	31
2.2.2 Controller Anatomy: OpenDaylight.....	33
2.2.3 OpenFlow Protocol .....	35
2.3 Software-Defined Network Applications.....	39
2.3.1 Network Traffic Engineering .....	39
2.3.2 Network Monitoring .....	40
2.3.3 Software-Defined Networks for Virtualisation.....	41
2.4 Software-Defined Networks in Various Networking Environments .....	43
2.4.1 Software-Defined Networks for the Internet of Things .....	43
2.4.2 Software-Defined Networks for Cloud and Data Centres.....	44
2.4.3 Software-Defined Networks for Wireless Networks and 5G.....	46
2.5 Software-Defined Network Challenges .....	47
2.5.1 Software-Define Network Model Architecture Drawbacks.....	48
2.5.2 Controller Challenges .....	48
2.5.3 Data Plane Challenges .....	49
2.5.4 Application Plane Challenges .....	49
2.6 Summary .....	50
<b>Chapter 3: Software-Defined Network Security Analysis</b> .....	<b>52</b>
3.1 Introduction .....	53
3.2 Software-Defined Networks for Security.....	54
3.2.1 Policy Enforcement.....	54
3.2.2 Security Policy Verification.....	56
3.2.3 Intrusion Detection.....	56
3.2.4 Threat Response .....	58

3.2.5 Security Tools .....	58
3.3 Security Limitations of Software-Defined Networks .....	61
3.4 Software-Defined Network Security Analysis .....	63
3.4.1 STRIDE .....	65
3.4.2 Attack Trees .....	68
3.4.2.1 Spoofing .....	69
3.4.2.2 Denial-of-service attacks .....	69
3.4.2.3 Escalation of privilege .....	70
3.4.2.4 Information disclosure .....	70
3.5 Simulation .....	71
3.5.1 Simulation Environment .....	71
3.5.1.1 Simulation execution .....	72
3.5.2 Discussion .....	73
3.5.2.1 Information disclosure attacks .....	73
3.5.2.2 Denial-of-service attacks .....	74
3.5.2.3 Spoofing .....	74
3.5.2.4 Tampering .....	75
3.6 Software-Defined Network Security vs. Traditional Networks .....	75
3.7 Summary .....	76
<b>Chapter 4: Threat Detection Framework: A Deep Learning Approach .....</b>	<b>77</b>
4.1 Introduction .....	77
4.2 Framework Components .....	81
4.3 Framework Workflow and Algorithms .....	83
4.4 Framework Design Principles .....	85
4.4.1 Dimensionality Reduction and Anomaly Detection .....	85
4.4.2 Decision Boundaries .....	87
4.4.3 Resolving Clustering and the Curse of Dimensionality .....	88
4.4.4 Network Traffic Features and Deep Learning .....	88
4.4.5 Number of Hidden Layers and Neurons .....	89
4.4.6 Application Considerations .....	90
4.5 The Framework Process .....	91
4.6 Framework for Software-Defined Networks .....	95
4.7 Framework Features .....	97
4.8 Summary .....	98
<b>Chapter 5: Simulation Studies .....</b>	<b>100</b>
5.1 Simulation Overview .....	100
5.1.1 Simulation Scenarios .....	101
5.2 Simulation Tools: TensorFlow and SciKit .....	102
5.3 Dataset .....	102
5.4 Scenario Implementation .....	107
5.4.1 Scenarios 1 and 2 .....	107
5.4.2 Training .....	110
5.4.3 Scenarios 3 and 4 .....	113
5.5 Conclusion .....	120
<b>Chapter 6: Results, Analysis and Evaluation .....</b>	<b>121</b>
6.1 Introduction .....	121
6.2 Results .....	122
6.3 Analysis .....	131
6.4 Design Principles in Action .....	136

6.5 Evaluation .....	139
6.6 Summary .....	144
<b>Chapter 7: Conclusion .....</b>	<b>146</b>
<b>References .....</b>	<b>151</b>



## List of Figures

Figure 1.1. SDN architecture [4].	16
Figure 1.2. OpenDaylight Defense4All application [19].	19
Figure 1.3. Research methodology.	22
Figure 2.1. SDN architecture.	31
Figure 2.2. OpenDayLight controller architecture [19].	34
Figure 2.3. OpenFlow switch architecture [49].	36
Figure 2.4. OpenFlow entry field [49].	36
Figure 2.5. OpenFlow flowchart [49].	38
Figure 2.6. OpenDaylight OpenStack application support [19].	45
Figure 3.1. Network policy enforcing middleware device locations.	55
Figure 3.2. Controller security threats.	63
Figure 3.3. Controller high-level dataflow diagram.	65
Figure 3.4. Detailed dataflow diagram.	66
Figure 3.5. Simulation environment.	73
Figure 3.6. Network throughput before and after denial-of-service attack.	74
Figure 4.1. Proposed detection system architecture.	82
Figure 4.2. Detection system flowchart.	83
Figure 4.3. Applying deep learning as a pre-step for support vector machine classification.	86
Figure 4.4. Generic autoencoder architecture.	87
Figure 4.5. Projecting inputs to a new higher dimension.	89
Figure 4.6. Deployment of intrusion detection systems in software-defined network architecture.	96
Figure 4.7. Deployment of intrusion detection systems in traditional networks.	97
Figure 5.1. Simulation scenarios flowchart.	101
Figure 5.3. Restricted Boltzmann machine neuron activation.	114
Figure 5.4. Reconstruction phase.	115
Figure 6.1. Optimisation using stochastic gradient descent and Adam optimiser.	123
Figure 6.2. Reconstruction error distributions for autoencoder and restricted Boltamann machine.	125
Figure 6.3. k-means graphical representation for clusters.	126

Figure 6.4. Graphical representation of mean shift clusters.....	127
Figure 6.5. k-means cluster distribution.....	128
Figure 6.6. Confusion matrix graphical table. ....	133
Figure 6.7. Autoencoder and k-means (400 samples).....	134
Figure 6.8. Autoencoder and k-means (800 samples).....	134
Figure 6.9. Autoencoder and k-means (1300 samples).....	134
Figure 6.10. Autoencoder and mean shift (400 samples).....	134
Figure 6.11. Autoencoder and mean shift (800 samples).....	134
Figure 6.12. Autoencoder and mean shift (1300 samples).....	134
Figure 6.13. Restricted Boltzmann machine and k-means (400 samples).....	135
Figure 6.14. Restricted Boltzmann machine and k-means (800 samples).....	135
Figure 6.15. Restricted Boltzmann machine and k-means (1300 samples).....	135
Figure 6.16. Restricted Boltzmann machine and mean shift (400 samples).....	135
Figure 6.17. Restricted Boltzmann machine and mean shift (800 samples).....	135
Figure 6.18. Restricted Boltzmann machine and mean shift (1300 samples).....	135
Figure 6.19 Reconstruction errors used for forming clusters.....	136
Figure 6.20 Accuracy without increasing the number of neurons at the first layer.....	138
Figure 6.21. Reconstruction error distribution without increasing dimensionality in the first layer. ....	138
Figure 6.22. Using the encoder from deep autoencoders with a classifier. ....	140
Figure 6.23. Autoencoders vs. non-symmetric deep autoencoders.....	141
Figure 6.24. Using the encoder output for classification. ....	142
Figure 6.25. Proposed classification system based on sparse autoencoder. ....	143

## List of Tables

Table 1.1. Anomaly-Based vs. Signature-Based Intrusion Detection Systems .....	18
Table 3.1. SDN Security Tools Survey .....	60
Table 3.2. STRIDE Graphical Components.....	64
Table 3.3. Controller Threats .....	68
Table 4.1. Deep Learning for Network Anomaly Detection.....	80
Table 4.2. Algorithm for Anomaly Detection .....	84
Table 5.1. KDD99 Input Features .....	103
Table 5.2. KDD99 Dataset Statistics.....	104
Table 6.1. k-Means Cluster Contents .....	126
Table 6.2. Mean Shift Clusters.....	127
Table 6.3. Autoencoder Simulation Scenario Results Summary .....	129
Table 6.4. Restricted Boltzmann Machine Simulation Scenario Results Summary .....	130
Table 6.5. Confusion Matrix Statistics.....	131
Table 6.6. Confusion Matrix for Autoencoder and Restricted Boltzmann Machine + k-Means.....	132
Table 6.7. Confusion Matrix for Autoencoder and Restricted Boltzmann Machine +Mean Shift.....	133
Table 6.8. Related Work Performance .....	139
Table 6.9. Performance of NDAE vs. DBN and proposed system in this research. ....	142

## List of Abbreviations

ARP	Address Resolution Protocol
CD	Contrastive divergence
DAE	Deep autoencoder
DBN	Deep belief network
DDoS	Distributed denial-of-service
DFD	Dataflow diagram
DL	Deep learning
DNS	Domain name system
DoS	Denial-of-service
FDR	False-discovery rate
FNR	False-negative rate
FPR	False-positive rate
FN	False negative
FP	False positive
IDPS	Intrusion detection and prevention systems
IDS	Intrusion detection systems
IETF	Internet engineering task force
IoT	Internet of Things
IP	Internet Protocol
KPCA	Kernel principal component analysis
NDAE	Non-symmetric deep autoencoder
NetConf	Network Configuration
NOS	Network operating system
NPV	Negative predictive value
OVSDB	Open vSwitch database
PCA	Principal components analysis
PPV	Positive predictive value
RBM	Restricted Boltzmann machine
REST	Representation state transfer
SAL	Service abstraction layer
SANE	Secure Architecture for Networked Enterprise

SDN	Software-defined networking
SGD	Stochastic gradient descent
SNMP	Simple network management protocol
STRIDE	Spoofing, tampering, repudiation, information disclosure, denial of service, privilege escalation
SVM	Support vector machine
TCP	Transmission Control Protocol
TN	True negative
TP	True positive
USDL	Unsupervised deep learning algorithms

## Publications

The results and outcomes of part of the research works contained in this thesis have been published in the following papers.

A. Dawoud, S. Shahristani and C. Raun, ‘Unsupervised deep learning for software defined networks anomalies detection’, in *Transactions on Computational Collective Intelligence XXXIII. Lecture Notes in Computer Science*, N. T. Nguyen, R. Kowalczyk and F. Xhafa, Eds., Berlin, Heidelberg, Germany: Springer, 2019, pp.

A. Dawoud, S. Shahristani and C. Raun, ‘Deep learning and software-defined networks: Towards secure IoT architecture’, *IoT*, vol. 3–4, no. 201810, pp. 82–89, Sep. 2018, doi: 10.1016/j.iot.2018.09.003.

A. Dawoud, S. Shahristani and C. Raun, ‘Dimensionality reduction for network anomalies detection: A deep learning approach’, in *Web, Artif. Intell. Netw. Appl., WAINA-2019*, L. Barolli, M. Takizawa, F. Xhafa and T. Enokido, Eds. 2018, pp. 957–965.

A. Dawoud, S. Shahristani and C. Raun, ‘Deep learning for network anomalies detection’, in *2018 Int. Conf. Mach. Learn. Data Eng.*, Sydney, Australia, 3–7 Dec. 2018, pp. 149–153.

A. Dawoud, S. Shahristani and C. Raun, ‘A deep learning framework to enhance software defined networks security’, in *32nd Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Krakow, Poland, 16–18 May 2018, pp. 709–714.

A. Dawoud, S. Shahristani and C. Raun, ‘Software-defined network’s controller security: Empirical study’, in *Int. Conf. Inf Tech Appl (ICITA)*, Sydney, Australia, 1–4 Jul. 2017. [Online]. Available: <http://www.icita.org/2017/abstracts/au-dawoud.htm>. Accessed: 24 Jun. 2019.

---

A. Dawoud, S. Shahrstani and C. Raun, ‘Software-defined network security: Breaks and obstacles’, in *Networks of the Future: Architecture, Technologies, and Implementations*, M. Elkhodr, Q. F. Hassan and S. Shahrstani, Eds., Boca Raton, FL: CRC Press, 2018, pp.

---

# Chapter 1: Introduction

## 1.1 Introduction

Networking is the enabling technology for an enormous number of communication applications, including the internet, the Internet of Things (IoT) and cloud computing. The growth of applications has necessitated expansion of data communication infrastructure. Therefore, additional flexibility in management and interoperability is required. However, traditional networks are based on a rigid architecture that does not fully satisfy the requirements of emerging technologies [1]-[3]. Software-defined networking (SDN) is a novel networking model that provides the features required for supporting emerging networking technologies [1], [4]. Figure 1.1 depicts the architecture of SDN. The SDN model proposes the separation of the forwarding and control planes by aggregating and abstracting a device's logic into a new central entity called the network controller [4], [5].

The controller concept is analogous to operating systems, which are responsible for interactions between applications and devices. The existence of controller entity boosts programmability, enabling developers to code applications for many purposes, including network management, load balancing and network monitoring. The architecture enhances network resilience, simplifies management procedures and supports network policy enforcement [1], [6]. Additionally, the new features of SDN enhance security by facilitating several security measures such as threat detection and prevention [7].

However, SDN architecture design suffers from significant security flaws [1], [8]. Paradoxically, the characteristics of SDN that make it a promising substitute for conventional networks also present security sever challenges. A centralised network entity that has control over the entire network is a valuable target for network intruders. The controller is located at a central point between applications and the data planes, with both northbound and southbound communications being vulnerable to various types of attacks [8]. The controller software is prone to vulnerabilities such as buffer and stack overflow. Hence, providing security measures to protect the controller itself is crucial to fully unleash the capabilities of the new model.



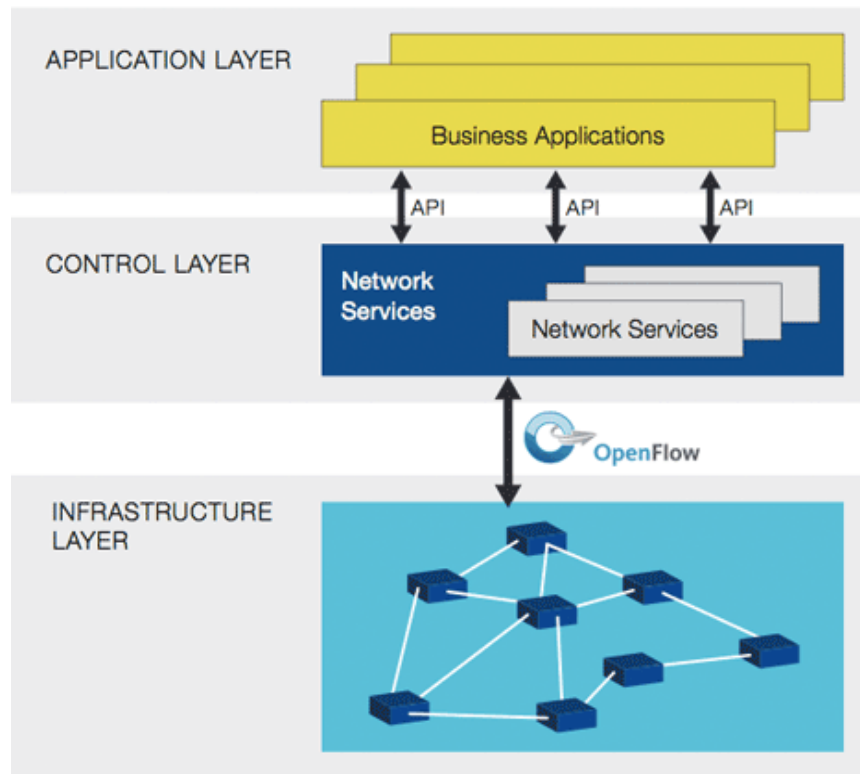


Figure 1.1. SDN architecture [4].

For decades, conventional networks have employed firewalls and intrusion detection systems (IDS) as the standard security solutions to deter and mitigate various network threats. However, innovative solutions are required for the unprecedented security threats emerging from recent advances in internetworking, such as the IoT, SDN and grid computing.

IDSs adopt either signature-based or anomaly-based detection approaches to identify threats [9]. Signature-based detection is limited in its ability to recognise attacks that do not exist in the IDS threats profile. However, anomaly-based detection is more problematic than signature-based detection because of its precision deficiencies [10].

Anomaly-based detection techniques are classified into two categories: statistical and machine learning [11], [12]. The latter uses algorithms such as support vector machines (SVMs) neural networks and principal components analysis (PCA), all of which fail to provide high detection accuracy [11], [13]. Recent achievements in machine learning, advances explicitly in training deep learning (DL) neural networks are promising [14], [15]; however, few studies have investigated the applicability of DL for detecting network anomalies.

DL neural network architecture is multi-layered, with hidden layers between the input and output layers. The first layer represents network input (data features) and the final layer represents network output. Even though DL neural networks have long existed [16], they have been unable to train the network for various reasons, including the vanishing gradient descent in backpropagation, unsatisfactory generalisation and the need for intensive computation power.

In 2006, a pre-training step using restricted Boltzmann machines (RBM) [14] advanced DL, leading to the development of innovative algorithms such as linear rectifier units rather than sigmoid functions and dropout to solve generalisation problems [15]. These algorithms can be divided into two categories: supervised and unsupervised machine learning.

## **1.2 Motivations**

Controller security is crucial for the security of the entire SDN architecture [7], [8]. Several security solutions have been proposed for securing the controller, including standard security measures such as firewalls and IDS. However, controller security remains a significant concern, curbing the potential of SDN capabilities [1], [7], [8], [17], [18].

IDSs have been deployed in traditional networks to enhance network security for decades [9], [11]. Both IDS approaches—signature-based and anomaly-based detection—have limitations [11]. While signature-based detection is unable to detect zero-day attacks, anomaly detection can theoretically detect unprecedented threats. However, it suffers from low detection accuracy. The limitations of anomaly detection have significant consequences for the deployment of SDN. For instance, in the traditional network, damage resulting from an attack affects only a set of network nodes with limited consequences, while in an SDN network, a compromised controller may lead to the collapse of the entire network. Traditional network-distributed architecture can tolerate a margin of IDS inaccuracy (e.g. a false negative). However, the cost of a false-negative alarm in SDN may be catastrophic to the network, particularly from attacks targeting the controller. Hence, improvements to current IDS approaches are essential to boost SDN controller security.

Table 1.1 provides a comparison of the two conventional detection methods, with each method having its advantages and disadvantages. A significant drawback of the signature-based method is its inability to detect new attacks because of its reliance on a database of known threats. In contrast, anomaly-based detection has a higher false-positive alarm rate because of accuracy limitations in the underlying detection algorithms, leading to the possible detection of threats in the absence of malicious activity.

*Table 1.1. Anomaly-Based vs. Signature-Based Intrusion Detection Systems*

	<b>Anomaly-based</b>	<b>Signature-based</b>
Performance	Medium	High
Protection against zero-day attacks	High	Low
False-positive alarms	High	Low
False negatives	High	Medium
Configuration	Low	High

Anomaly-based detection systems utilise various techniques, such as statistical and machine learning. Recent advances in machine learning have led to the need to evaluate new machine learning algorithms in network anomaly-based detection.

Several intrusion detection applications have been developed to detect malicious activities in SDN networks. For example, the OpenDaylight (ODL) controller uses the Defense4All application to detect and mitigate distributed denial-of-service (DDoS) attacks [19], [20]. Figure 1.2 depicts the deployment of Defense4All application at the ODL. However, the application does not protect the controller itself; rather, it deploys a set of rules to protect the network at its edges. In the event of malicious activity, the Defense4All application requests network information from the controller and acts via its attack mitigation module. Security limitations of this application include the following:

- The application must first communicate with the controller to gather statistics and raw data used by the IDS to decide whether an activity is malicious. Consequently, the controller is exposed to the threat prior to the decision being made.

- The controller’s location in the architecture makes it vulnerable to new types of attacks that require novel mechanisms, such as those that ensure the security of communications between the controller and the IDS.
- Controller software may be prone to traditional software vulnerabilities, which require advanced detection techniques such as deep packet inspection.

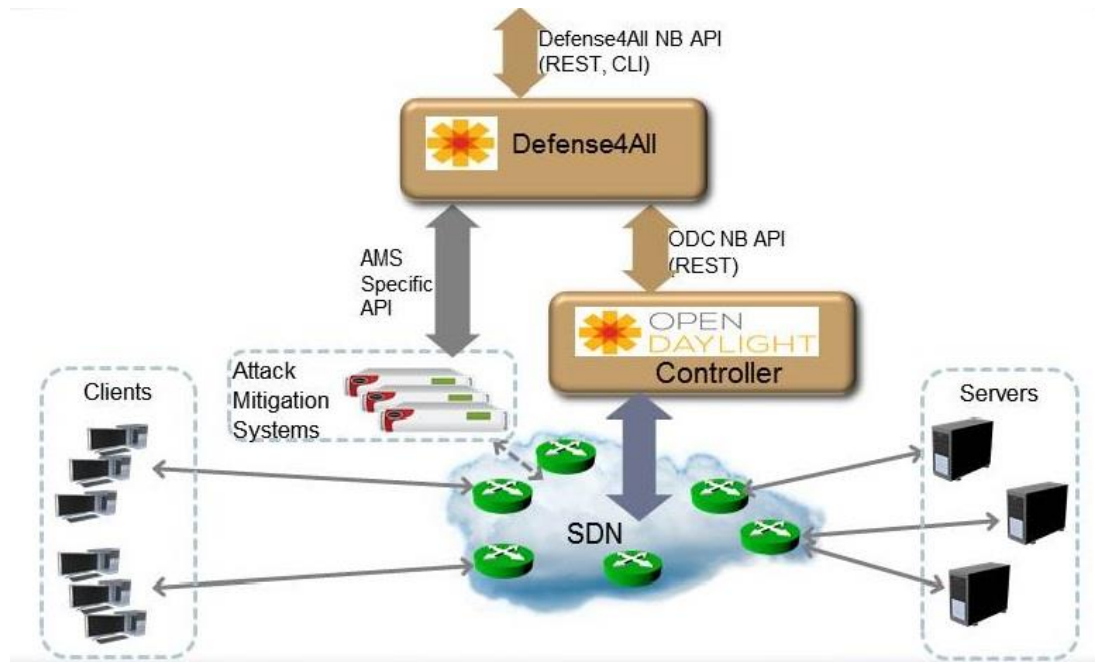


Figure 1.2. OpenDaylight Defense4All application [19].

### 1.3 Research Methodology

The research goal is to improve the efficiency of IDS capabilities for threat detection to enhance the security of the SDN controller and reduce concerns related to security threats to SDN architecture. The research was divided into three phases: security threat analysis, IDS design and implementation and performance analysis and evaluation. Figure 1.3 illustrates the research stages. In the first phase—SDN security analysis—the new networking architecture is introduced. The controller is a significant point of attraction for new threats. Identifying the threat list is a starting point for proposing appropriate security solutions. In [8], a threat vector for SDN is defined and seven threats are listed, with three being exclusive to the SDN model. Notably, SDN-specific threats originate from the controller:

- Attacks on control plane communications
- Attacks on controller vulnerabilities

- Attacks on the controller originating from the application layer.

The first stage of the study involved the analysis of mentioned threats. There are two main approaches to security analysis: system-oriented and attack-oriented. For research comprehensiveness and consistency, a method from each approach was selected.

STRIDE is a system-oriented threat modelling method [21] that models dataflow diagrams (DFDs) of the system under analysis. The main elements of the model are data flow, data stores, processors, interactors and trust boundaries. DFD components were examined against the set of attacks specified by STRIDE: spoofing, tampering, repudiation, information gathering, denial of service and elevation of privilege.

Attack trees are a formal, attack-oriented approach to defining possible attacks against the system [22]. The attack tree starts with a root node denoting the attack goal, and various tree leaves specify the means of reaching the node. Logical AND/OR operators are used to aggregating the leaves. The attack tree analysis is supported by tools such as Security and Isograph. The goal of the second stage was to simulate attacks as proof of concept to provide a deeper understanding of threats.

Additionally, the analysis phase provided a simulation of various attacks derived from the previous analysis phase. The simulation was conducted using an SDN simulator integrated with a real controller (ODL). The main goal of this stage was to provide a more profound proof of concept of various threats and their impacts on network assets.

In the solution domain, IDSs are used as security solutions in traditional networks; however, their limitations need to be addressed in SDN deployment. In the second phase, the detection system was designed and implemented. The new architecture and challenges brought by SDN have increased the need to investigate various architectures of IDSs in the SDN controller. The goal of this phase was to propose a framework that delivers highly accurate detection functionalities to protect the controller. This phase was divided into three stages. The first stage provided the theoretical design principals for the framework, main components and decision boundaries. Six design principles based on the formalised definition of the problem were identified. In the second stage, an implementation based on the design principles was constructed as a proof of concept. The final stage in this phase involved the training and execution of the models. The final phase was a simulation and evaluation of the proposed framework with a focus on

performance metrics. During the simulations, the dataset was normalised and fed to the system before results were collected, recorded and analysed. For the analysis, confusion matrices, a common technique for classification and clustering analyses, were used to compare results with related works. In the final step, similar works were evaluated, and their contributions were highlighted.

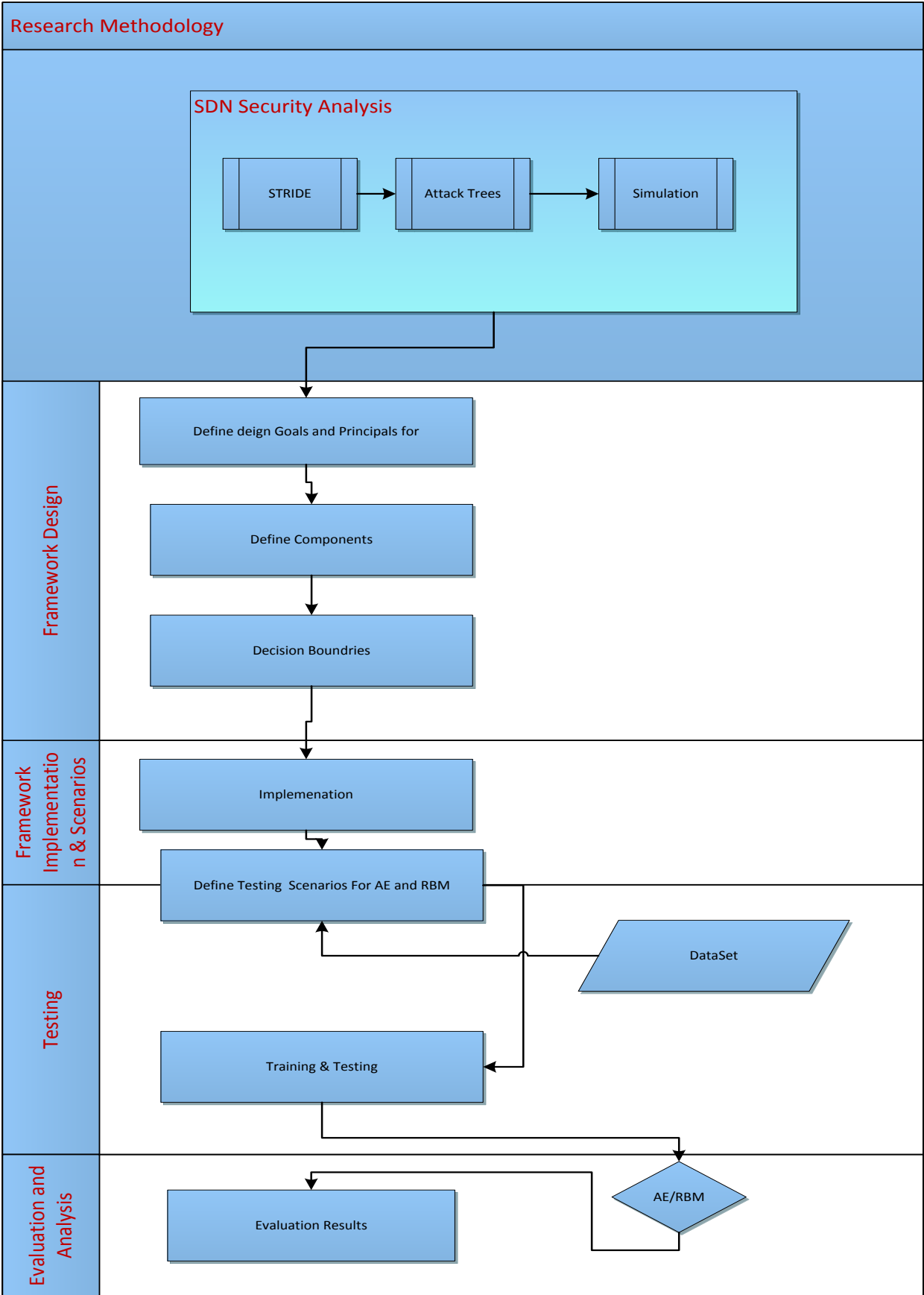


Figure 1.3. Research methodology.

## 1.4 Research Questions and Scope

This research is aimed at enhancing SDN controller security. The research objectives gave rise to the following questions:

The primary research question is: How may an efficient threat detection framework be designed that will improve the detection accuracy of IDS and, hence, the security of SDN?

The following sub-questions were derived from the primary research question:

- What are the main security threats to the controller? How are these threats different from those in traditional networking models?
- How is it possible to improve the performance of IDS to provide reliable security measures to the SDN controller?
- How can recent advances in machine learning help improve the detection accuracy of IDSs?
- How can unsupervised DL be used in network anomaly detection?
- How can dimensionality reduction be achieved using unsupervised DL?
- What is the best deployment of the proposed framework in SDN architecture?

## 1.5 Contributions

This thesis enhances the security of the SDN model by improving the efficiency of current security solutions. This thesis proposes a novel detection framework based on unsupervised DL algorithms for threat detection.

The study explored the potential of DL for revealing network threats by utilising unsupervised DL algorithms. The research examined the ability of DL to detect anomalies through evaluation of generative energy-based models (RBMs) and non-probabilistic algorithms (autoencoders). Following this, an in-depth analysis of DL algorithms was conducted, with results showing promising detection accuracy.

This thesis provides an empirical analysis of SDN controller security to identify, formalise and quantify security concerns related to the new model. The study explored threats related to SDN architecture, specifically those originating from the controller



plane. The study analysed controller security over three stages. The first stage defined potential threats based on a review of the literature. The second stage demonstrated and modelled threats using a STRIDE analysis. Additionally, an in-depth attacks-oriented analysis was developed using several attack trees. The third stage introduced an experiment to reveal threats and consequences. The study provides a comprehensive understanding of the problem by specifying the security flaws of SDN.

The framework consisted of two phases: a DL algorithm and a clustering algorithm using either  $k$ -means or mean shift clustering. The DL algorithm represented the pre-training phase, which simplified the input to the clustering algorithm. The framework employed dimensionality reduction of the input data, compressing the dimensions of the input data to a single value to simplify and improve the performance of the clustering algorithm in the second phase.

The study improved the performance of the  $k$ -means algorithm. The  $k$ -means relies on calculating the distance between different samples—an increase in the number of samples results in a dramatic decrease in distance between them. Hence, the use of the DL algorithm in the pre-training phase reduced the problem of dimensionality related to  $k$ -means. The framework solved this problem by reducing the number of inputs based on critical procedures in the autoencoder and RBM, generating more straightforward inputs to the  $k$ -means.

The framework was based on two unsupervised algorithms, which have the ability to find patterns in data with no previous labelling, enabling the detection of zero-day attacks. The framework presents a unique method of dimensionality reduction. Instead of using the output of the neural network from either the RBM or the autoencoder, the framework used a single value—the difference between the input and the output—for the entire input record.

The proposed framework design was implemented using Tensorflow [23]. Accordingly, a simulation of several scenarios was conducted using the KDD99 network dataset [24]. Following various executions over several testing cycles, the data were collected and statistically analysed. A systematic analysis was conducted using confusion matrices to evaluate results against other related works. The simulation showed a significant

accuracy of  $\approx 99\%$  from the integration of the autoencoder with the  $k$ -means clustering algorithm.

## 1.6 Thesis Structure

This thesis is organised as follows: Chapter 2 presents the background, applications, networking environments, solutions and challenges of SDN. The first section introduces the limitations of the current networking model and the motivation for a novel model to handle such limitations. Section 2 introduces the model architecture, the three planes of the model and, given that this thesis focuses on security flaws related to the controller, a broader discussion of the control plane. This section provides a comprehensive anatomical view of one of the most renowned SDN controllers, ODL, which was also used in the security attack simulation. Additionally, this section discusses the OpenFlow protocol, which is the dominant southbound protocol in SDN architecture. Section 3 explores three applications of SDN, including traffic engineering, network virtualisation and network monitoring and measurements. It discusses the current challenges of each of these applications and how characteristics of SDN such as centralisation and global view are expected to overcome the limitations of traditional networks in such applications. Additionally, this section presents several software solutions for each category. Section 4 introduces new emerging networking environments in which SDN integration enhances communications and solves problems such as management complexity and network abstraction. This section discusses the integration of SDN in IoT, cloud computing, data centres and wireless networks, including cellular and fifth generation (5G) networks. For each technology, we discuss the challenges and contributions of SDN. Additionally, we provide examples of SDN platforms and application solutions for each area. Section 5 presents the current challenges and limitations of SDN, which are categorised into four classes: architecture design, application plane, control plane and data plane.

Chapter 3 presents an in-depth analysis and simulation of SDN security threats, a significant flaw in SDN architecture. The chapter outlines SDN security issues, including structural security flaws and how the SDN model may be used to enhance security. The first section presents several security applications for SDN, including policy enforcement and verification, threat detection and response. Additionally, it provides a survey of SDN security tools. The next section discusses the security flaws

of the SDN model, specifically controller threats. Section 4 presents an analysis of SDN security using two approaches—STRIDE and attack trees—to identify security threats and show how they may be executed. Section 5 describes the experimental study conducted to demonstrate attacks against the SDN controller.

Chapter 4 presents the solution domain. The thesis focuses on enhancing the security of SDN through intrusion detection. This chapter introduces the solution methodology, which is machine learning. The first section discusses network intrusion detection techniques and compares different approaches such as signature-based and anomaly-based detection. The second section explores anomaly-based detection methods, including statistical and machine learning. Section 3 introduces the DL algorithms, focusing on unsupervised DL algorithms, autoencoders and RBMs. Section 4 discusses the opportunities for using DL for anomaly detection.

Chapter 5 presents the proposed detection framework. The first section introduces the components of the framework, which consists of two phases. The first phase uses an unsupervised DL algorithm, and the second phase uses a simple clustering algorithm. Section 2 depicts the framework workflow and the framework algorithm in pseudocode. Additionally, a detailed description of the different steps is included. Section 3 outlines the five design principles of the framework, which include dimensionality reduction, decision boundaries, clustering and curse of dimensionality, network traffic features, the number of hidden layers and neurons and the assumptions required for framework applicability. Section 4 describes the framework in action—the theoretical background for the algorithms. Section 5 describes the integration of the framework in an SDN model. Section 6 discusses the advantages of the framework.

Chapter 6 provides the implementation, simulation and evaluation of the proposed framework. The first section is an overview of the simulation. The second section proposes four different scenarios for the simulation based on different algorithms in the first and second phases of the framework. The third section presents an analysis and rationale for the dataset, and the different software libraries and tools used in the simulation. The fourth section presents the implementations of the framework, including the implementation of two unsupervised DL algorithms—an autoencoder and an RBM—and two clustering algorithms— $k$ -means and mean shift. Following implementation, the section also demonstrates how the system was executed, including

training and testing. The fifth section shows the results of several executions on the dataset. Additionally, this section provides an in-depth analysis of the results using confusion matrices. Section 6 presents an evaluation of the framework results compared with other similar proposed frameworks.

Chapter 7 concludes the thesis, briefly describing the problem of the SDN controller security flaws, the research contribution, which mainly focuses on frameworks for anomaly-based detection in SDN networks, and a proof of concept implementation for the framework towards solving the problem. It highlights the thesis contributions, lists research limitations and suggests directions for future work.

## **Chapter 2: Software-Defined Networks**

SDN has introduced a revolutionary communications model through the decoupling of the control and forwarding planes and the relocation of the network logic to a new layer known as the network controller. Features of this model include centralisation and network programmability, which pave the way for various networking solutions and innovations. As an emerging technology, SDN provides several opportunities and challenges. This chapter discusses the novel networking model of SDN, including its major technological drivers, motivations, components and challenges.

The primary purpose of this chapter is to present a comprehensive review of SDN, including its design, models and characteristics, its role as enabling technology in several environments and its applications and limitations.

The chapter explores SDN architecture and discusses the three planes of the SDN model: the application, controller and physical planes. It mainly focuses on the responsibilities and essential services offered by the controller, a new plane introduced by SDN, and discusses several issues related to it, including scalability, availability and interoperability. Given that the controller is critical for SDN security, the chapter provides a more in-depth anatomical view of its components and its various roles, using ODL as the model, and discusses the applications of SDN in areas such as traffic engineering, network monitoring and virtualisation. For each application domain, the chapter provides an in-depth discussion of the enabling features of SDN that help to solve current problems with conventional networks.

Additionally, SDN deployment in various networking environments is introduced and SDN challenges for each plane are highlighted. SDN security challenges are discussed in the following chapter.

### **2.1 Introduction**

The traditional data communication model is composed of three planes: management, control and data planes. The management plane provides services to monitor and configure the network, while the control plane generates the data required to establish forwarding tables on physical devices. Subsequently, the forwarding plane directs

packets to ingress and egress ports based on the tables. In the traditional network model, both the control and forwarding planes are tightly coupled to the same device (e.g. a switch or a router). This model is efficient from a performance perspective. However, as the complexity of networks has increased, the need to adopt a new architecture has emerged [1].

Network management comprises various activities, including management of faults, configuration, performance, security, inventory and accounting. Each network includes several interoperating devices, each having its own configuration firmware, from different vendors. To perform management activities or to add or remove devices in the network, the network administrator must obtain different software packages or make changes to various devices, which increases the complexity of management [1], [3], [25], [26]. For complex networking environments such as data centres, management activities become even more complicated. A single misconfiguration can lead to unexpected policy violations [2].

Additionally, given the rigid structure of the network, scaling the network vertically (by increasing the capacity of current resources) or horizontally (by adding new resources) is a complicated procedure. This may be addressed using the process of abstraction. For example, if firmware installed on different devices is abstracted to a single software, this will facilitate integration and configuration of network devices. Hence, the concept of separating the logic from the hardware is key to tackling the rigid and static structure of traditional networks. The evolution of SDN is similar to that of distributed and personal computing.

The SDN model consists of three planes known as the forwarding, control and application planes. SDN architecture separates the control plane from the forwarding plane introducing an independent plane known as the controller or the network operating system (NOS) The forwarding plane comprises devices such as switches, routers and middleboxes, which switch data flow but do not have the logic required to populate the forwarding tables [5],[27], [28]. The network intelligence resides in the controller, which abstracts devices and provides services such as network state and topology information. Additionally, the controller provides a northbound application program interface (API) to communicate with applications and a southbound API to

communicate with forwarding devices. OpenFlow is the dominant southbound interface used in SDN [29].

The application plane lies on the top of the SDN model stack. Programmability is a fundamental concept of the SDN paradigm in which applications communicate with physical devices. Programmability provides opportunities for innovation for an enormous number of network applications, including monitoring, traffic engineering, security and cloud applications [30], [31]. Centralisation is a distinctive feature of the SDN architecture, providing a global view of the entire network and facilitating management and monitoring processes. Additionally, it reduces errors in configuration and deployment of network policies. Centralisation also improves flexibility—for instance, a pool of devices from various vendors may be deployed and abstracted in the same network [32].

## **2.2 Software-Defined Network Architecture**

Conventional networks are divided into three planes, namely the management, control and forwarding layers. The management plane provides services to monitor and configure the network. The control plane generates the data required to establish forwarding tables, which, in turn, are used by the forwarding plane to direct packets to ingress and egress ports. In traditional network models, both the control and forward planes are tightly coupled within a single device (e.g. switch or router). This model is efficient from a performance perspective. However, as the complexity of networks has increased, the need to develop a new architecture has emerged.

Figure 2.1 shows the three layers of SDN. The essence of SDN architecture is the separation of the control and forwarding planes. The separation draws the device's logic (software), leaving the network devices as forwarding devices only. These devices do not have the capability to decide on forwarding requirements.

The network control plane is an independent entity known as the network controller or NOS. The forwarding layer, on the other hand, comprises of network devices such as switches, routers and middleboxes, which do not have their own logic. Network intelligence resides in the controller or NOS, which abstracts the devices and provides services such as network state and topology information services. Additionally, the controller provides a northbound API to communicate with the application layer and a

southbound API to communicate with the forwarding layer devices. OpenFlow is the dominant southbound protocol in the SDN model [1], [3].

The application layer, which lies at the top of the SDN stack, introduces network programmability—the ability to communicate with the network’s underlying devices—which is a fundamental concept in SDN. Programmability provides opportunities for network innovation for an enormous number of network applications, including network monitoring, traffic engineering, security and cloud applications.

Centralised control enables a global view of the network, which facilitates management and monitoring processes. Additionally, it reduces errors in configuration and deployment of network policies and improves flexibility—for instance; a pool of devices from various vendors may be deployed and abstracted within the same network.

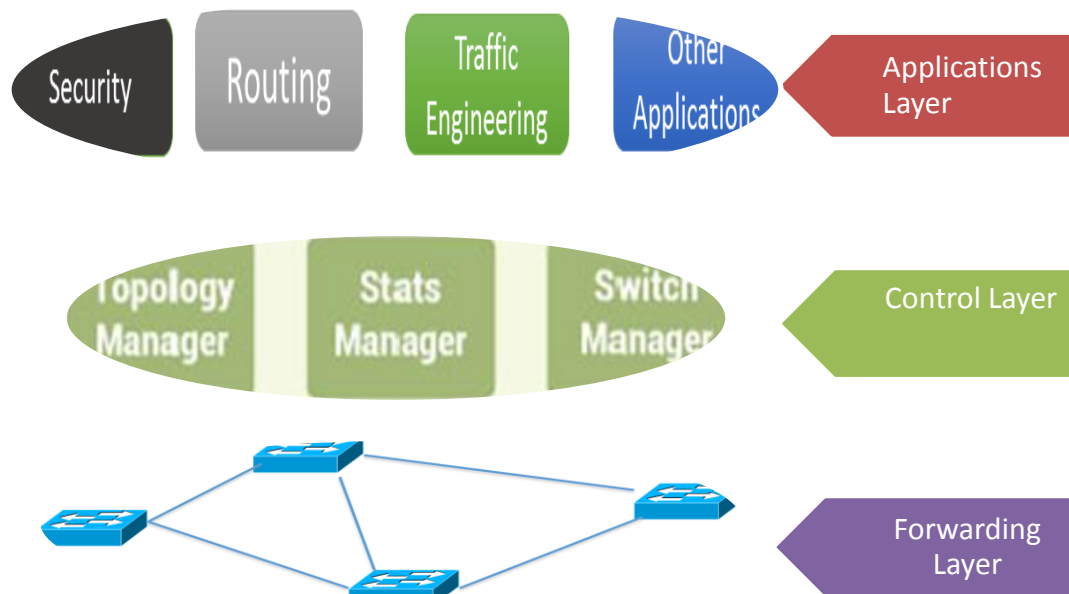


Figure 2.1. SDN architecture.

### 2.2.1 Software-Defined Network Controller

The controller, or the NOS, abstract devices and provides the resources required to program low-level forwarding devices. The controller provides services such as network state and topology information. Additionally, the controller provides a northbound API, which facilitates communication with applications, and a southbound API, which provides accessibility to forwarding devices. OpenFlow is the de facto SDN southbound protocol [4], [27]. The application plane resides at the top of the SDN model stack.



Network programmability is a privilege primarily achieved by the SDN model in which applications in the top plane can access physical devices through the controller. Programmability facilitates and accelerates innovation of an enormous number of network applications, including monitoring, traffic engineering, security and cloud applications. Centralisation is an essential characteristic of the SDN architecture. The central entity is the controller, which provides a global view of the entire network and facilitates management and policy enforcement. Additionally, it decreases faults in configuration and deployment of network policies. Centralisation enhances network resilience and interoperability—for example, multiple devices from various industries may be integrated and abstracted in one network.

The SDN controller consists of the following elements:

- Basic network services: These are the core functions of SDN controllers and include topology, device events, status managers, shortest path forwarding and underlying security mechanisms.
- Service abstraction layer (SAL): Orchestrates the southbound API (e.g. plug-in management).
- Southbound API: Typically, the southbound API refers to the OpenFlow protocol. However, SDN supports the integration of various protocols, such as Forwarding and Control Element Separation and Open vSwitch Database Management Protocol, in the southbound API.
- East/westbound API: Connects controllers within the distributed architecture.
- Northbound API: Facilitates communication between applications and lower devices via the controller.

Traditional network hardware has been managed by proprietary software such as the Cisco Internetworking Operating System. The core component of the control plane is the NOS, which provides the basic functionality for applications to access and manage devices in the physical plane. Similar to other generic operating systems (e.g. Windows and Linux), the NOS provides mechanisms to manage and abstract hardware resources.

Based on its architecture, the NOS can be classified into two categories—centralised or distributed. In a centralised architecture, the NOS is installed on a single computing device. While this is efficient from a performance perspective, it has limitations in

scalability and availability (a single point of failure) [1], [33]. Trema [34], Ryu [35], Floodlight [36], Meridian [37] and Beacon [38] are all classified as centralised NOSs, with support for multithreading and concurrency to achieve high throughputs.

In a distributed deployment, the NOS is installed onto several nodes to support scalability and high availability requirements for large data centres or large networks. These nodes can be in a single cluster or distributed over several clusters that are physically separated. Clusters or nodes are designed on the basis of peer-to-peer or hierarchal architectures [1]. Distributed architecture improves fault tolerance and high availability—for example, if there is a failure or security breach in a portion of the NOS, network administrators have more options for recovery (e.g. isolation). Several controllers, including Onix [39] and ONOS [40], adopt distributed architecture.

However, a distributed architecture is also related to issues such as consistency and latency. Given that the controller is distributed over several nodes, each node must retain the latest data view (e.g. network topology or switch status). Additionally, nodes or clusters must communicate across the network, causing latency [7].

### **2.2.2 Controller Anatomy: OpenDaylight**

ODL is a modular open-source controller project under the Linux Foundation. It has wide support from the industry, including Cisco, IBM, Microsoft and Huawei, and more than 1,000 developers. Figure 2 shows ODL's Lithium version controller components. The controller layer provides basic network services such as network topology, network status and switch manager.

Representational state transfer (REST) API represents a northbound API to facilitate communication between the controller and the uppermost layers. REST API uses non-persistent connections. Southbound APIs include OpenFlow and protocol plug-ins that interface with devices. The controller implements core services, including topology, statistics and switch management, host tracking and Address Resolution Protocol (ARP) handling. Further, the controller provides services for standard protocols. The SAL allows the controller to support various protocols such as OpenFlow, Simple Network Management Protocol and Border Gateway Protocol in the southbound API.

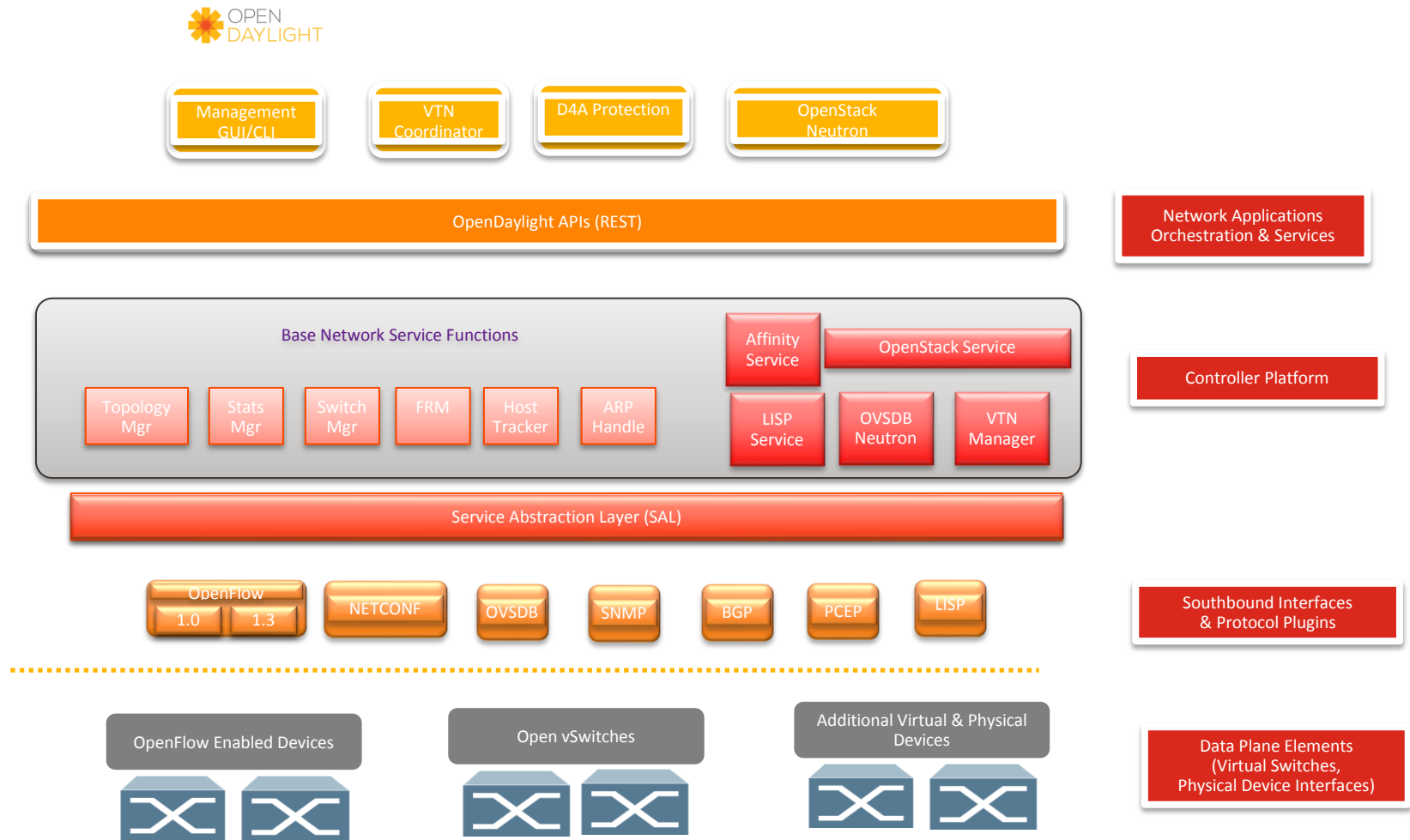


Figure 2.2. OpenDayLight controller architecture [19].

ODL provides a plug-in to support the Open vSwitch Database Management Protocol [41], which is a configuration management protocol designated to the SDN virtual switch (vSwitch) [42]. Additionally, ODL provides a network configuration (NETCONF) plug-in to support configuration installation and deletion on devices in the forwarding plane [43]. ODL supports standard routing and network management protocols such as the Border Gateway Protocol (BGP) [44] and the Simple Network Management Protocol (SNMP) [45] and provides a plug-in for the Path Computation Element Protocol [46] and the Locator ID Separation Protocol [47]. For virtualisation support, ODL offers Virtual Tenant Manager at both the control and application planes.

### **2.2.3 OpenFlow Protocol**

OpenFlow is the de facto southbound interface protocol for SDN. It facilitates communications between the controller and forwarding devices at the lower plane. OpenFlow evolved from the Stanford projects Secure Architecture for Networked Enterprise (SANE) and Ethane [48]. SANE was developed as a single layer responsible for governing connectivity and access control as a centralised entity to provide network security.

OpenFlow inherits the concept of forwarding tables from traditional network protocols such as Ethernet. However, its flow-based approach means that sequences of packets belonging to the same flow are subject to the same rules and decisions. These rules are installed in the forwarding tables, which are handled by controllers installed on devices. OpenFlow allows bidirectional communications between devices and the controller, meaning that devices can notify or refer to the controller for specific decisions [49].

Figure 2.3 shows the main components of an OpenFlow switch. The controller communicates with the switch via the control channel to manage one or more flow tables [49].

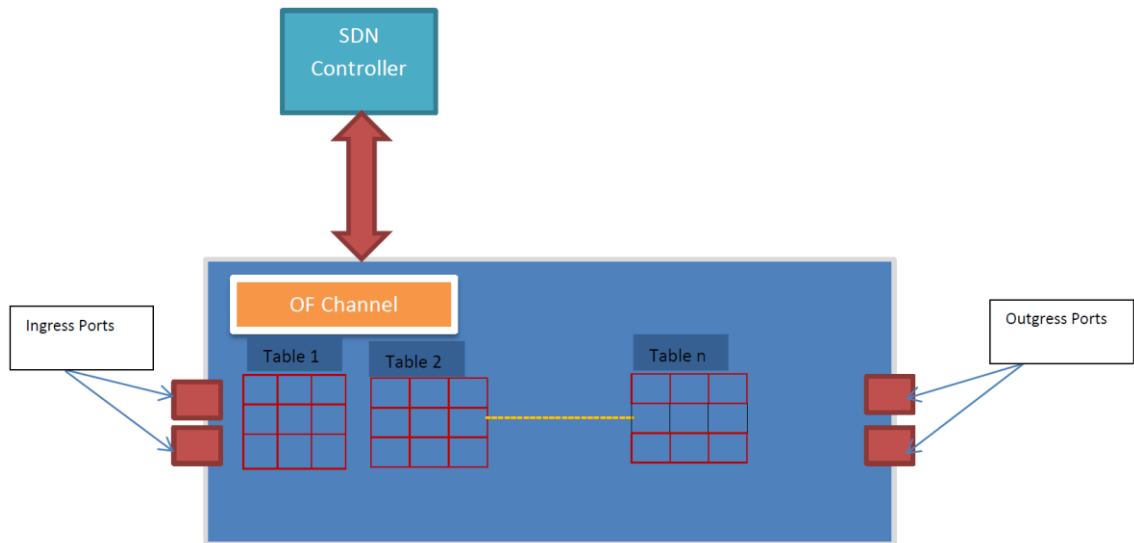


Figure 2.3. OpenFlow switch architecture [49].

Each switch in the SDN network is configured by the Internet Protocol (IP) address and a port number to communicate with the controller. Communications are secured over transport layer security channels. The first message sent by the controller is a feature request, which collects the device’s configuration, such as its physical address and available ports.

Each table contains several records known as flow entries, which are accessed by the controller. Figure 2.4 depicts the flow entry fields. The entry is used to match against the incoming packet headers to execute an action on the packets.



Figure 2.4. OpenFlow entry field [49].

Flow entry fields contain the following elements:

- Match fields: used to match against packets and include the ingress port, Ethernet source address, destination address and type, virtual local area network and priority, IP source address, a destination address, protocol and quality of service and Internet Control Message Protocol type and code
- Priority: matches precedence of the flow entry

- Counters: contain statistics on the packets and flow, e.g. per-flow counters, received packets, received bytes and duration seconds and nanoseconds
- Instructions: actions applicable to the packet, e.g. Forward, Enqueue, Drop and Modify-Field
- Timeouts: expire time for the flow in the switch
- Cookies: thresholds implemented by the controller to filter statistics and modify flows
- Flags: decide how a sequence of packets is processed.

The controller uses two approaches, proactive and reactive, to install rules in switches in the flow table. In a proactive installation, the controller adds the rules in advance (before the packets reaching the switch). In the reactive mode, there is no match for the packet initially, but the device forwards the packet to the controller, which then adds the appropriate rule in the flow table.

OpenFlow supports three types of messages:

- Controller-to-switch messages, which are initiated by the controller:
  - Features: In request/reply mode, the controller sends a feature request message to the switch to inquire about the identity and capabilities of the switch and the switch replies with a feature reply message.
  - Configuration: The controller sets and queries the switch configurations and the switch responds to the query, sending the required information to the controller.
  - Modify-State manages installed rules on the switch flow table and configures switch ports.
  - Read-State collects statistics from the switch.
  - Send-Packet sends the packet out through a specific port on the switch.
  - Barrier: Used for message verification.
- Asynchronous messages are initiated by the switch:
  - Packet-in encapsulates a packet to send to the controller either because no predefined rule exists, or while the rule exists, its associated action is forwarded to the controller.
  - Flow-Removed notifies the controller of an entry removal from the flow table.

- Port-status notifies the controller if the port status has changed.
- Error informs the controller of fault occurrences on the switch.
- Symmetric messages are initiated by both the controller and the switch:
  - Hello messages initiate the session.
  - Echo request/replay messages are similar to ping in the Internet Control Message Protocol.
  - Vendor: customised messages sent by the vendor.

In OpenFlow protocol specifications, the controller is responsible for modifying the forwarding tables in SDN devices. The flowchart in Figure 2.5 depicts the OpenFlow process of incoming packets. Upon the arrival of a new packet, the switch searches for a matching forwarding entry in the forwarding table. If a record matches the packet fields, a predefined action will be executed. OpenFlow allows a set of measures to be taken, including to drop, forward or modify the packet. If no match occurs, the switch forwards the packet to the controller to conduct computations according to the policy issued by the application layer.

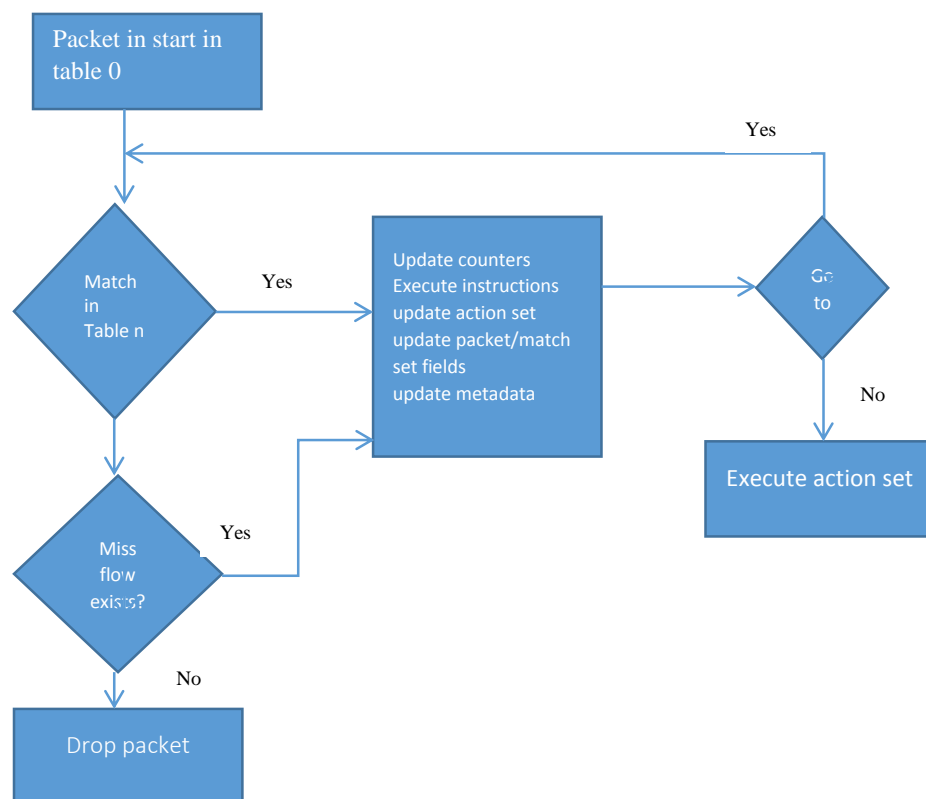


Figure 2.5. OpenFlow flowchart [49].

## **2.3 Software-Defined Network Applications**

A primary concept in the SDN model is network device programmability, which boosts applications developed for various purposes. Applications can be classified according to their purposes, such as network traffic engineering, network monitoring and measurements, Virtualisation, and network security. The following three subsections cover the first three classes, while security is covered in Chapter 3.

### **2.3.1 Network Traffic Engineering**

The primary objective of traffic engineering is performance optimisation in networking through achieving a set of objectives, including reducing congestion, end-to-end delays, power consumption and packet loss, maximising the quality of service and optimising load balance and resource utilisation. The static architecture of traditional networks is a significant challenge for traffic engineering tasks [49]. Characteristics of SDN, such as resilience, programming, centralisation and network function virtualisation capabilities, promise the facilitation of traffic engineering solutions.

Multiprotocol label switching is a common traffic engineering routing mechanism to forward data from one device to the next based on short labels rather than long network addresses, reducing the time of table lookups [50]. However, multiprotocol label switching suffers from limitations.

Traffic splitting is a common mechanism to reduce network traffic congestion. There are two approaches to traffic splitting: packet-based and flow-based. Packet-based splitting can result in packet reordering that may be overheard at the other end of the connection, especially in Transmission Control Protocol (TCP) sessions, resulting in congestion at the destination. In flow-based splitting, decisions are made by forwarding devices; however, these decisions may not be optimal because they are based on local parameters, rather a global network view [51]. Hence, SDN provides a solution because it provides a global view of the network. Additionally, SDN can improve optimal path computations because it provides a logically centralised view of the network. Databases used by traffic engineering mechanisms must present a real-time view of the network—in traditional architecture, device states are scattered throughout the network, but in SDN, the controller has mechanisms to update the database in real-time for all devices in the network.



Pythia is a traffic engineering system for data centres that utilises the SDN–ODL controller model. Hadoop MapReduce is a big data analytics tool to analyse and refine control of data centres networks [52]. QNOX is an extension of the NOX controller that promotes quality of service enforcement; its authors claim it improves resource discovery, route computations and fault notifications [53]. Aster\*X is an application based on the NOX controller for web server load balancing—it uses the controller to harvest the node states and control paths using OpenFlow to facilitate network reconfiguration by allowing administrators to control capacity [54]. ElasticTree is an energy consumption optimiser for data centres based on NOX—it tunes active devices in real-time according to traffic loads [55].

### **2.3.2 Network Monitoring**

Network monitoring and measurement are essential mechanisms for network operators and administrators, while awareness of device status and network behaviour is critical for making decisions regarding network management, quality of service, threat response and traffic engineering [56]. The network monitoring process includes five stages [57]:

- Measurements and collection of data from network devices over predefined time frequencies: Measurements are classified as active or passive. In active measurements, network agents probe devices for return of data, while in passive measurements, agents act only as receivers of data sent by the network nodes.
- Pre-processing: Data collected from different nodes are aggregated and normalised.
- Transmission: Raw harvested data are transferred from the data sink (e.g. the management information base) to the node responsible for analytics. Simple Network Management Protocol is a widely used protocol to transfer data.
- Analysis: Different algorithms are applied to the data to identify specific patterns and big data algorithms and tools are used for data analytics.
- Visualisation: Presents results in formats that are easily understandable and may be quickly absorbed by network administrators for making decisions.

SDN can improve the subprocess of collection and transmission based on its architectural attributes such as centralisation and programmability. In traditional networks, network agents collect data from network nodes periodically—this approach

is rigid and inefficient in terms of consistency, performance and resource optimisation. In contrast, SDN involves a central entity with a global view that can intelligently decide which data from which devices need to be collected. For data transmission, instead of using a classic management information base, SDN offers a flexible development in new data structures according to requirements.

Procera is a framework based on SDN that allows network administrators to annotate policies applicable to responding to specific network events [58]. Its policy is written in high-level functional programming language and compiled to a set of forwarding rules at the underlying nodes in the physical plane.

OpenSample is a platform to reduce sampling latency based on the Floodlight controller [59]. It uses a modified flow standard for packet export. Its authors claim it reduces latency from 1–5 seconds to 100 ms. OpenNetMon is a module integrated into the POX controller that monitors flow metrics related to packet loss, throughput and latency [60]. It probes flow source and destination devices periodically in cases where poll time slots are subject to changes.

### **2.3.3 Software-Defined Networks for Virtualisation**

Network virtualisation enables and maximises resource sharing between several isolated networks running in their own containers [61]. Network virtualisation solutions efficiently increase hardware utilisation and reduce expenditure and operational costs. Virtualisation is an essential service technology in data centres and cloud computing infrastructure, allowing tenants to acquire networking services according to their requirements [62].

Virtualisation in the traditional networking model faces two challenges: network topology and addressing. Various networking environments require different network topologies. Additionally, addressing schemes such as IP versions 4 and 6 are related to physical devices.

SDN abstraction capabilities facilitate virtualisation by adding an intermediate layer, which is analogous to middleware hypervisors in computing virtualisations. The new layer acts as a proxy between the NOS and physical devices. The purpose of the layer is

to seamlessly encapsulate the process required for sharing resources and isolating tasks [63].

The hypervisor, or virtual machine monitor, is responsible for monitoring various virtual networks and allocates required resources such as link capacity [64], [65].

Hypervisors have three abstraction attributes:

- Device abstraction: Similar to other computing devices, virtualisation targets the central processing unit and related storage. This is mainly used for flow table resources.
- Physical link abstraction: This focuses on virtualisation of physical connections, available link capacity and buffers at both ends of the link.
- Topology abstraction: The hypervisor uses the abstraction of devices and links to implement the required network topology.

An essential virtualisation attribute is isolation. In SDN-based virtualisation, isolation must be done on three levels [65], [66]:

- Addressing isolation: Each slice of the network or virtual network flow spaces, which represent a subset of the entire available flow, must be separated from each other. Additionally, consistency problems such as generalisation, correlation and shadowing issues may exist in the access control list.
- Data plane isolation: Device central processing units, associated storage and physical links should be isolated for each tenant.
- Control plane: Each slice must have its own controller.

FlowN is a NOX-based SDN distributed hypervisor that adopts the concept of containerisation in which the entire network is running on a single controller, with each tenant having a standalone slice [67]. AutoSlice is a proposed virtualisation layer that focuses on the automation of the slicing process itself [68]. Slices Isolator handles problems related to virtual network isolation and performance and flexibility trade-offs [69]. It offers different levels of isolation from which network operators can choose according to their performance and isolation requirements.

## **2.4 Software-Defined Networks in Various Networking Environments**

SDN can replace the traditional networking model in several networking environments, including IoT, data centres, cloud computing and wireless networks. The following sections discuss these various environments, the challenges of current traditional networks and how SDN can mitigate these limitations, providing some sample implementations.

### **2.4.1 Software-Defined Networks for the Internet of Things**

IoT introduces new challenges to the conventional communication model. IoT network characteristics such as object heterogeneity and scalability require revolutionary solutions. Currently, there is no universal architecture for IoT. However, several architectures have been proposed based on SDN. SDN introduces network programmability and centralisation, which facilitate network abstraction, simplify network management and ease evolution. The proposed framework in chapter 4 with the SDN integration can be utilised as a novel communication architecture for IoT. SDN enhances network resilience and scalability, which are essential in large-scale IoT deployments such as smart cities.

IoT expands the capability of the internet by connecting smart objects such as grid health and environmental devices. Advancements in wireless communication, embedded systems and sensor technologies have accelerated the adoption of the IoT model in several domains. However, higher connectivity increases the risk of privacy and security threats.

IoT introduces three challenges: first, the heterogeneous composition of the network; second, the adoption of widely distributed architecture, specifically in applications such as smart cities and smart grids; and third, the introduction of new protocols to handle specific issues related to power and computation limitations of network sensors [70]-[73].

The IoT threat vector has been extended with new attacks, including object cloning, firmware replacement and extraction of security parameters. Several studies have proposed an SDN-based architecture to enhance the security of IoT. Some studies have considered a domain-based architecture in which the network includes multiple domains

[74], [75]. The separation of domains enhances the availability of the network. However, a robust performance analysis has not been conducted. Bhunia and Gurusamy [76] propose a detection system based on SDN for denial-of-service (DoS) attacks on IoT, with the authors claiming they achieved a precision of around 98%. Chakrabarty *et al.* [78] propose an SDN-based IoT architecture called Black SDN, which secures payload and metadata through encryption. However, routing suffers complications as the source and destination data in the header are also encrypted. Jararweh *et al.* [79] focused on IoT management aspects by proposing a comprehensive SDN-based architecture—SDIoT—to enhance IoT management by enhancing the forwarding, storing and securing of data generated from IoT objects.

#### **2.4.2 Software-Defined Networks for Cloud and Data Centres**

Cloud computing is a model of Internet-based computing that represents an integrated platform of network hardware and software that provide specific internet services on a pay-per-use basis. Cloud computing provides three levels of service: software as a service, platform as a service and infrastructure as a service. The top level, software as a service, provides software on demand—examples include email software such as Microsoft Office 365. Platform as a service offers platforms used by application developers, while infrastructure as a service, the lowest level, offers the most basic services such as virtual machines and virtual networks.

The cloud computing paradigm considers two characteristics: elasticity and dynamic reconfiguration. The cloud platform operates in several data centres, including Amazon EC2 and Microsoft Azure, and this environment contains an enormous number of networking devices, servers and dense existence for virtualisation services. The complex structure of these data centres and the vast number of internetworking devices and servers raise issues related to scalability and performance. As discussed previously, the rigid structure of the traditional network creates a challenge for cloud computing platforms. The giant leader Google built B4, an SDN-based wide area network connecting Google data centres around the globe [83].

SDN characteristics such as centralisation, programmability, a global view of the network and, most importantly, virtualisation capabilities, allows SDN to be an enabler technology for data centres and cloud computing platforms [1]. Based on various SDN-

based clouds computing architecture, the authors concluded that an abstract architecture consists of three layers mapped to SDN model planes as follows:

- Cloud manager application receives requests for resource allocations and provides services for management, monitoring and performance optimisation. The ODL controller—discussed in section 2.1.1—allows the integration of cloud manager software such as OpenStack [19].
- Controllers similar to SDN architecture provide basic NOS services. Figure 2.6 shows the ODL controller support OpenStack at the controller plane with the OpenStack service module.
- The physical plane includes the network resources to be provisioned by the cloud manager.

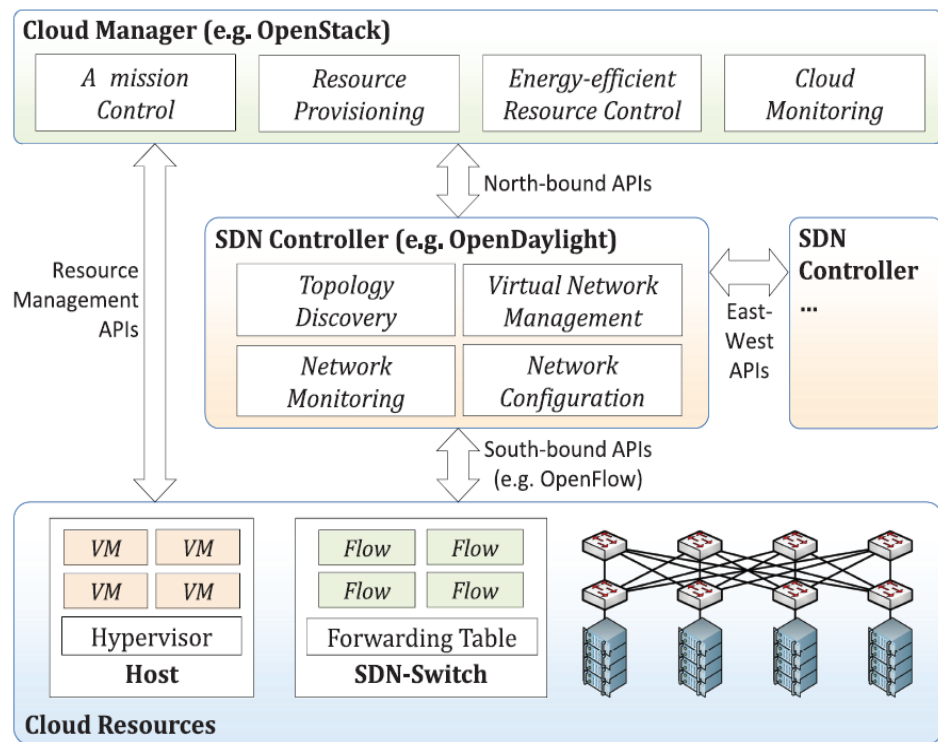


Figure 2.6. OpenDaylight OpenStack application support [19].

CloudNaas is a NOX-based cloud networking platform that supports infrastructure as a service cloud for virtual network creation and isolation [84]. Meridian is an IBM cloud platform for creating and managing virtual network topologies according to workload [37]. Meridian can be integrated with OpenStack and IBM SmartCloud [85].

### 2.4.3 Software-Defined Networks for Wireless Networks and 5G

Wireless networks can be classified into four main classes: cellular, wireless sensors and wireless mesh networks [86], [87]. Cellular networks integrate a combination of technologies such as 4G, Long-Term Evolution, various standards of wi-fi and Worldwide Interoperability for Microwave Access. This combination requires transparent, soft, hands-off, efficient resource management. Cellular networks are composed of two major components, a core network and a radio access network. The core network is the basis of connectivity, providing access to mobile stations or end users. The core network provides connectivity between different radio access networks, managing services such as hands-off, roaming and quality of service. In wireless sensor networks, the major challenges are related to limited computation and power resources in the sensors—these challenges were covered in the previous section on IoT. Wireless mesh networks or ad hoc wireless networks involve the connection of devices without infrastructures such as access points, with routing on a hope-to-hope basis. The routing mechanism and absence of a central node cause interference and negatively affect performance [86].

Fundamentally, the wireless network faces challenges such as interference and frequency management that do not exist in wired networks. Additionally, security threats originating from the medium used by wireless networks, which is not constrained by wires or optical fibres as in wired networks, are mounting. The medium imposes the need for new solutions because techniques such as collision detection are not applicable in wireless networks; hence, to avoid collisions in advance, a solution for collision avoidance emerges. Solutions for sharing frequency bands, such as various types of multiplexing (e.g. time-division multiplexing and frequency-division multiplexing), create additional problems such as hidden and exposed nodes, which require solutions such as Request to Send and Clear to Send, increasing complexity and affecting performance.

SDN characteristics such as resilience and centralisation offer opportunities to tackle issues such as power and frequency changes or network handovers in the dynamically changing environment of wireless networks. Additionally, wireless networks are heterogeneous, and the concept of abstraction in SDN, by supporting different devices from different vendors, is key to handling the problem of heterogeneity in wireless

networks [1]. In cellular network resource utilisation, optimisation is essential, particularly in high-density areas, and network designers adopt various techniques to allow more users in the same cell to use the frequency efficiently. One of these techniques is cell splitting in which a cell is divided into smaller cells, with each sub-cell having its own base station with lower transmission power to avoid interference from adjacent cells. The technique has its drawbacks, including an increased number of cells, which increases the probability of interference and complexity in management [88]-[90].

For resource allocation of radio access in cellular networks, SoftRAN provides an abstraction for base stations. At the control plane, the abstraction is conducted in three dimensions: time, frequency and space [91]. The SoftRAN control plane is responsible for operations such as hands-off and transmission power controls for each base station to avoid interference. In the core network, Softcell is an SDN-based application that resolves the complexity and delay associated with the resources allocation in the core network, allowing the core network to access the data plane in the radio access network and have a global view of the entire network to support routing through middleboxes installed on switches [92].

## **2.5 Software-Defined Network Challenges**

Despite the opportunities introduced by the novel model, SDN faces various challenges, raising questions regarding its suitability as a singular model. In this section, we categorise those challenges into four classes:

- Architectural challenges related to design, which affect the non-functional requirements of the model as a unit
- Controller challenges, such as distributed controller design
- Data plane challenges, such as switch design and interoperability
- Application plane challenges.

Security is also a major challenge of the SDN model. Given that this thesis focuses on the security of SDN, we discuss this flaw in the next chapter.



### **2.5.1 Software-Define Network Model Architecture Drawbacks**

Traditional networking models enclose the control and data planes within the same device. Required communications between both planes are almost simultaneous. In SDN, the controller and data planes communicate over an OpenFlow communication transport layer security channel. Communication and its associated encryption and decryption processes cause latency. Additionally, latency increases in distributed controller architecture in which controllers use east/westbound channels to communicate and synchronise the global view of the network. In conventional networks, the control plane is distributed in case of failure for various reasons, including security breaches. Affected devices will be out of service, but other devices will still be able to operate, enhancing the availability of network services. Centralisation of the control plane creates a single point of failure if the controller is out of service. Subsequently, all devices at the data plane will also fail [1], [7].

### **2.5.2 Controller Challenges**

In high-density networking environments such as data centres, a single controller model is impractical because large data centres, such as Google B4, are geographically scattered over different locations and have high availability and throughput requirements. Hence, a scalable distributed design is more practical. Distributed architecture may be hierarchal or peer-to-peer. Controller scalability faces two challenges: latency in controller communications and management of the backend database by the controllers [1]. [40], [92].

Controller scalability by integrating different controllers is another challenge for SDN deployment. The controller comprises software that is coded in a specific programming language. Languages such as C++ support performance over portability. Java offers excellent portability, but its performance is affected by the two-step encoding by the compiler and the interpreter. The programming language will affect controller interaction with the applications plane (northbound communication) and intercommunications between controllers in distributed controller architecture (east/westbound communication). Solutions focus on two approaches—general network policy programming language and API. Pyretic was an early attempt to abstract applications in which the network administrator or programmers could build a modular

application from already existing modules (similar to the concept of programming language packages). However, given its weak performance, it was not industrially applicable [94]. API in the controller scenario facilitates east/westbound communication between controllers. The Internet Engineering Task Force (IETF) has proposed an SDN interface protocol for inter-SDN controller communications. However, these steps are still far from meeting practical interoperability requirements [95].

### **2.5.3 Data Plane Challenges**

Traditional networks have existed for decades, with industry and governments investing heavily in its infrastructure. The transition from this model to SDN should consider interoperability between the two models. Another challenge at the data plane is device heterogeneity, with vendors providing different switches with a wide range of inconsistencies in performance, features and compliance with protocol specifications [1], [7]. One solution for design inconsistency problems in SDN-compatible devices, offered by tinyNBI, is the provision of a basic API [95]. The authors extracted five fundamental abstractions and provided a low-level API, which can be used for higher-level abstractions regardless of the OpenFlow version or switch design. Additionally, the SDN-promoting organisation Open Networking Foundation have founded a specialised group, the Forwarding Abstractions Working Group [96], which is working to deliver new standards for network forwarding targets. The main goal of the group is to enhance and enforce OpenFlow standards on forwarding devices.

### **2.5.4 Application Plane Challenges**

SDN applications require a high level of abstractions. Traditional programming languages offer a low level of abstractions (even when comparing scripting languages to more level programming languages such as C and C++). The purpose of SDN applications is to annotate network policy, which requires a high level of abstraction that is closer to formal specification notations. Application authentication and access control to the services offered by the controller is an essential step to secure the SDN. Additionally, application isolation should be done at two levels—first, applications should be isolated from each other, and second, the control plane should be isolated from the application plane [8].

## 2.6 Summary

This chapter provided the background of the SDN model, which introduced the separation of the control and data planes. The chapter focused on SDN architecture, which consists of three planes: application, control and data planes. Given that the controller is the most critical element for model security, the chapter provided an in-depth examination of this plane. Main components of the control plane were discussed in detail and, subsequently, an anatomical view of one of the most renowned SDN controllers, ODL, was applied. Additionally, the chapter introduced and discussed the dominant southbound protocol, OpenFlow, which is responsible for communications between the controller and the networking devices at the forwarding plane. It is essential to understand how the protocol is integrated with the controller.

The following two sections discussed the applications of SDN and the environments in which SDN can provide fundamental solutions. SDN applications include networking traffic engineering, network monitoring and virtualisation solutions. Several emerging technologies can benefit from SDN architecture, including IoT, clouds, data centres and wireless technologies, including 5G cellular networks. For each of these environments, the basic concepts, challenges and solutions offered by SDN was discussed.

The global view of SDN enhances decision-making in network traffic engineering. Additionally, it provides an efficient routing path computation supported by the centralised controller. Network monitoring applications such as OpenNetMon provide efficient mechanisms for measuring statistics related to network throughput and packet loss. Another notable success of SDN is its virtualisation ability, which is supported by device abstraction and hypervisor layers implemented at the control planes.

This chapter introduced several challenges and ongoing research in SDN networking. These challenges were classified into four groups: challenges related to architecture design and those related to the application, control and data planes. The majority of these challenges are related to programmability and centralisation of SDN. For example, centralisation introduced new challenges for the controller architecture—questions about performance, scalability, flexibility and security in both centralised and distributed controller architectures were raised. Network programmability allows applications to access networking devices, raising concerns related to authentication,

authorisation and accounting. Additionally, the model inherited challenges, including those related to security, from traditional IP networking. Security is a significant challenge—the following chapters will focus on security challenges and solutions.

## Chapter 3: Software-Defined Network Security Analysis

This chapter discusses the security of SDN. The primary goal of this chapter is to provide a broad and inclusive understanding of security in the SDN model. It investigates the controller's security flaws and how these threats differ from threats in traditional networks. We identify threat attributes and their consequences on the network assets. A comprehensive understanding of attacks will improve the efficiency of countermeasures. The security analysis is conducted in three stages. First, a STRIDE (spoofing, tampering, repudiation, information gathering, and denial of service and elevation of privilege) analysis is conducted to identify possible threats from the design perspective. The second analysis identifies attacks using attack trees. The third analysis simulates attacks to identify practical consequences and recommended measures to address threats.

The chapter is organised into five sections. The first section is an introduction to both the opportunities and deficiencies in the security of SDN, providing an in-depth view of SDN security and security limitations.

Sections 2 and 3 present a review of SDN security from the current literature. The second section discusses how SDN improves network security through its wide range of security applications that enable the enforcement of security policies and monitor and detect threats.

The third section investigates the security deficiencies of the SDN model. This section provides an analytical view of SDN-related threats. Analytics were carried out using STRIDE and attack trees. STRIDE is a system-oriented threat modelling method that models DFDs of the system under analysis. The main elements of the model are data flow, data stores, processors, interactors and trust boundaries. Thereafter, DFD components were examined against a set of attacks specified by the STRIDE list (spoofing, tampering, repudiation, information gathering, denial of service and elevation of privilege). An attack tree is a formal, attack-oriented approach to identify possible attacks against the system. The attack tree begins with a root node that represents an attack goal, with many tree branches specifying methods to reach the node. Logical

AND/OR operators were used to aggregate leaves. The attack tree analysis was supported by various tools such as Isograph.

The fourth section discusses the simulation of several attacks identified in the previous section on an SDN network. The simulation was applied to an SDN network using Mininet [97] and ODL SDN controller, followed by the use of several exploits to launch the attacks. The final section concludes the chapter.

### **3.1 Introduction**

Data communication architecture has remained stable for decades. As the pace of technology has accelerated, there is a need to adopt a new model to reduce the complexity and inflexibility of traditional networks. Pillar technologies of SDN, such as central network control, programmability and network virtualisation, have been researched for decades [1]. OpenFlow introduces the concept of separating the control and forward planes and represents a novel communication architecture.

Centralisation and programmability offered by the SDN model are critical attributes utilised by developers to implement new security applications for various purposes such as monitoring and threat response. Despite the significant advantages offered by the new SDN architecture, including flexibility, programmability and centralisation, the model introduces unprecedented security threats.

Security is a primary concern of the new model. The SDN controller is a crucial layer in the network. A single point orchestrating the entire network may be utilised to enhance network security; however, paradoxically, this centralised architecture is more vulnerable to attacks. The controller is an attractive target for attackers because it is accessible from applications in the higher plane and dominates physical devices at the lower plane.

This study examines threats related to SDN architecture, specifically those related to the controller plane. There are two approaches for carrying out security analysis, namely system-oriented and attack-oriented approaches. For research comprehensiveness and consistency, a method from each approach was chosen. The security analysis was conducted in three stages. First, a STRIDE analysis was developed to identify possible threats in the SDN architecture design model. Second, various possible attacks using

attack remodelling were described. Third, attacks were practically simulated for demonstration and proof of concept.

The purpose of the study was to provide an inclusive realisation of threats emerging from the introduction of the control layer. This chapter investigates threat attributes, types of threats in traditional networks and the SDN model and consequences of threats. An understanding of attacks will improve the efficiency of countermeasures.

## **3.2 Software-Defined Networks for Security**

The rigorous and inflexible architecture of the traditional communication network has hindered its innovation [98], [99]. Multiple attempts have been made to adopt a flexible network model with separate control and forward planes. Separation has introduced programmability and centralisation features, which have been harnessed to enhance the security of the network. This section explores SDN applications that enhance network security, such as those that enforce and verify network policies and detect and mitigate threats.

### **3.2.1 Policy Enforcement**

Network policy is a set of configurations, rules and constraints that govern network operations (e.g. network access, incident handling and communications isolation) [100]. The architecture imposes policy enforcement through network middleboxes. Middleboxes are devices deployed to manipulate network traffic for specific purposes such as inspection, threat detection and access control.

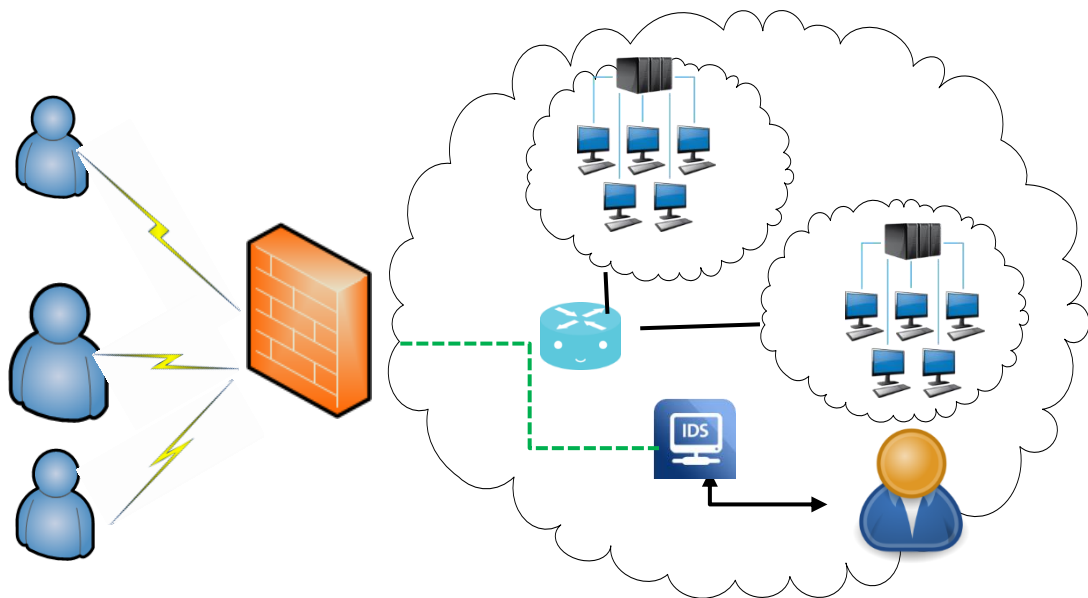
Traditionally, two approaches have been used to enforce network policy, either by deploying middleboxes between endpoints in network paths or by attaching them to middle switches. Given that both options necessitate rigorous deployment, they lack flexibility [101].

SDN architecture offers two advantages that are not available in traditional networks:

- Complete network coverage: Network policy is enforced at switching devices by installing flow rules. In conventional networks, middleboxes such as firewalls and intrusion prevention systems are located at specific points in the network, typically at network entry points such as demilitarised zones, either on or off

network paths. Both deployments are inflexible and incomprehensive. Figure 3.1 depicts a firewall dedicated to external traffic where coverage of internal traffic is limited. The IDS provides protection for specific subnets. In SDN networks, programmable switches are distributed over multiple locations in the network. This architecture avoids single point failure and enforces policy inside the network between endpoints.

- Centralisation facilitates policy deployment and configuration. This is in contrast to middlebox configurations in existing networks in which network administrators implement and deploy the policy from a single point rather than configuring appliances explicitly.



*Figure 3.1. Network policy enforcing middleware device locations.*

Historically, the OpenFlow protocol evolved as a successor of the Ethane project [102]. The purpose of Ethane was to define a network policy and enforce it at the switches. Ethane was an instantiation of SANE [48]. The domain controller based on network policy calculates the flow table entries installed on the switches. Given that the project requires custom switches, network upgrading was expensive. Integration of Ethane networks with current networks did not provide holistic policy enforcement where there was a probability of traffic passing through other non-Ethane custom switches.



SDN features reintroduced a policy enforcement method known as active security [103].

This concept includes five phases of adaptable network policy:

- Initial configuration of the infrastructure
- Sense: the controller responsible for collecting data from the network
- Adjusting the configuration as the controller updates the policy according to the network status
- Forensics: the controller gathers information related to attacks
- Respond: the controller initiates a reconnaissance and counter-reaction.

### **3.2.2 Security Policy Verification**

As the complexity of networks has escalated, there has been the need to ensure and verify the attached security policies. The conflict between policies or even between rules in the same policy may lead to network exposure.

FlowGuard is an SDN-based framework to detect firewall policy violations. Upon the update of the network status, FlowGuard will dynamically analyse the path space to detect firewall rule conflicts [104]. Flover is another SDN security policy verifier [105] based on checking systems and was built on the NOX controller to provide formal verification of security policies. Flover transforms flow table rules into a binary tree diagram and applies formal methods to detect rule violations.

### **3.2.3 Intrusion Detection**

Intrusion detection and prevention systems (IDPSs) are software or hardware systems dedicated to observing the network for security breaches. Standard IDPS processes comprise three stages: collection of data from the network, analysis and execution of actions in case of threat detection. There are three major data analysis methods for detection of breaches: signature-based detection, anomaly-based detection and specification-based detection [11]. Signature-based detection is used when a system has a database of predefined violation signatures and matches that signature against network activity signatures. Anomaly analysis is used when the system identifies abnormal activities. Normal activities are identified in a baseline profile, which the system develops in a learning phase. In stateful protocol analysis, a predefined pattern of protocol behaviour is established and a comparison between network activities and the

expected behaviour defined by protocols raises the alarm in the case of profile violation. A combination of methods is used to maximise IDPS performance. A study compared various detection methods proposed in [11]. Each method has its advantages and disadvantages—a significant weakness of signature-based detection is its inability to detect new attacks, while anomaly-based detection has a higher false alarm rate. The majority of commercial implementations use a hybrid approach.

Fundamentally, network-based IDPSs have a packet or flow-capturing module [12]. The capturing engine sniffs packets or flows for specific features. Feature selection relies on the threats targeted by the IDPS.

From the perspective of SDN, current research [10]-[12] has focused on packet and traffic measurements such as traffic engineering, load balancing, monitoring and security. Network central view and programmability provide the necessary assets to develop a robust packet/flow inspection system. SDN consists of three layers, namely applications, controller and forwarding devices. The controller has the capacity to communicate with devices through southbound protocols such as OpenFlow. OpenFlow provides the API to poll devices for traffic statistics. Traffic data are aggregated to the controller, which, in turn, communicates with the application layer through the OpenFlow interface.

The architecture of anomaly-based detection based on SDN has been proposed in [106]. The framework distributed a DoS attack detector based on flow inspection. The system has been implemented on the NOX controller. OpenSketch is a notable example of SDN traffic measurement architecture [107]. The platform provides a library to customise measurements to meet specific tasks and sets measurements to detect anomaly behaviour. A comprehensive view of the system, which is the essence of SDN architecture, is a significant feature. It reinforces the design of the robust data collection module in IDSs.

Studies have used the architecture of the anomaly-based detection method in SDN [106]. Concentration on anomaly detection based on the SDN is supported by the controllability of traffic. However, there is a need to adopt other detection methods in SDN to exploit its capabilities.

Specifically, SDN architecture can contribute to the enhancement of detection analysis techniques. Features such as scalability and ease of configuration in the case of anomaly detection can be improved by exploiting the centralised architecture of SDN. Developing a central analysis module may reduce the overhead on the monitored system, leading to improved performance.

Several intrusion detection applications have been developed to detect malicious activities in SDN networks. For example, Defense4All is an application in the ODL controller to detect and mitigate DDoS attacks.

### **3.2.4 Threat Response**

The SDN controller has a consistent real-time view of the entire network. Detecting attacks in real-time is essential for establishing an active response system. SDN is a flow-based rather than a destination-based networking model. Traffic control is a crucial feature of the response module. For example, on the assumption of threat existence, network middleboxes forward traffic to virtual appliances or honeypots for further investigation or forensic processes. Additionally, SDN programmability allows applications—particularly IDP applications—to communicate with forwarding devices. The flexibility of the architecture facilitates response mechanisms. For instance, if a section of the network is compromised, the response module isolates infected devices to mitigate the risks.

### **3.2.5 Security Tools**

In this section, several SDN security solutions are surveyed. Table 3.1 shows a survey of different SDN-based security tools. These tools are classified into two categories: security enhancers or SDN security resolutions. Security enhancement tools aim to improve network security by utilising SDN features, while SDN security resolutions are tools to improve the security of the SDN itself. Additionally, the table indicates the layers the solution covers.

FRESCO is a security composition framework that focuses on anomaly detection and mitigation [108]. Netfuse is an example of a solution that addresses security flaws in the SDN architecture [109], protecting the network from DoS attacks. However, there is a

significant shortcoming in the research related to improving the security of the SDN itself.

The majority of the survey tools focus on using SDN to enhance security, more specifically for policy enforcement solutions. The resilience of the SDN architecture effectively supports the adoption of policy execution and verification applications. MAPPER, FlowTags, SIMPLE and OpenSafe are examples of SDN solutions for policy enforcement [110]–[113]. CloudWatcher [114] controls network flows to guarantee network security, with devices inspecting each flow. Veriflow inspects and verifies flow rules in real-time to ensure integrity [115].

Table 3.1. SDN Security Tools Survey

Security Solution	Solution Domain		Layer			Description
	Security Enhancer	SDN Security Resolution	App	Control	Forward	
FRESCO [108]		X		X		Security services composition framework
LiveSec [116]	X		X	X	X	Security policy enforcement
Netfuse [109]		X		X	X	Protection against traffic overload externally (DDoS) or internally
SDN RTBH [113]	X			X	X	DoS mitigation
MAPPER [110]	X		X	X		Policy enforcement
FlowTags [111]	X		X	X	X	Policy enforcement and verification
SIMPLE [112]	X		X	X	X	Policy enforcement
OpenSafe [117]	X		X	X		Policy verification
CloudWatcher [114]	X		X	X		Ensures network packets are inspected
Fortnox [118]		X		X	X	Prioritises flow rules to eliminate inconsistencies
Flover [105]	X		X	X	X	Rule verification
VeriFlow [115]	X		X		X	Verifies and debugs flow rules
OpenFlow-RHM [119]	X			X	X	Mutates hosts as a response to threat existence
OrchSec [120]	X		X	X	X	Security application development framework
FlowNac [121]	X			X	X	Flow-based access control
PermOF [122]		X		X	X	Fine-grained permission and isolation system for SDN apps

### 3.3 Security Limitations of Software-Defined Networks

Despite the many voices preaching the promising future of SDN, various challenges prevent the broad adoption of the new model. Contradictorily, the primary advantages of the new architecture are the origins of its weaknesses. Performance, scalability, resilience and security are the main issues to tackle in the context of current research on SDN [1], [7], [8].

In contrast to traditional networks, SDN has performance trade-offs. A tightly coupled data and control plane in a single processing device is performance oriented. In SDN process flow, devices refer to the controller to perform logical decisions. The delegation of logical processes causes latency and negatively affects the throughput of devices [7]. The current stream of research is focused on improvements to hardware such as processing chips [1].

Essential questions about SDN scalability are raised. The network controller is responsible for logically updating forwarding tables in the connected device pool. In real-world networks, the controller is responsible for processing a large number of messages sent from forwarding devices. This raises the question regarding the number of nodes a controller should support. In this study, the network was scaled by adding more controllers to manipulate issues such as consistency. The term ‘consistency’ is essential in the SDN network because the controller, or a set of controllers, should maintain the same view of the network. HyperFlow [123] provides a solution for updating the network state by propagating events that affect the network state.

A single point of control is equivalent to a single point of failure. This configuration is a significant threat to network resilience and fault tolerance. SDN resilience remains an open question [1]. A distributed controller has been proposed to improve SDN flexibility [7].

Security threats are critical challenges in traditional networks and are escalated in SDN networks. The new architecture has brought additional challenges that did not exist in traditional networks. In particular, threats target the control layer [8]. The following sections highlight the security concerns of the controller and the standard southbound protocol, OpenFlow.

Security breaches are significant challenges in traditional data communications systems. Security challenges are escalated in SDN networks because the architecture introduces additional concerns that did not exist in traditional networks. OpenFlow's security analysis study [124] revealed several attacks derived from the SDN-prevalent protocol, such as DoS attacks on flow tables and control channels. Conflicts in application privileges propagate to flow rules. Fortnox provides role-based authentication and security policy enforcement [101]. Fortnox conducts a real-time rule conflict analysis to reveal rule contradictions [118]. Several intrusion detection applications have been developed to detect malicious activities in SDN networks. Defense4All is an application that detects and mitigates DDoS in the ODL controller [20]. However, the application does not protect the controller itself; rather, it deploys a set of rules to protect the network at its perimeters. The Defense4All application requests network information from the controller. On detection of malicious activities, the application executes mitigation actions according to its attack response module. A conventional technique to protect the controller is to deploy a distributed controller platform. However, significant concerns regarding distributed architecture have emerged, including network performance trade-offs. Given that multiple controllers exchange information for orchestrated network control, this exchange process results in latency. Additionally, there are concerns related to data consistency and synchronisation at each control point [1], [7]. In communications between the controller and switching devices, data exchanged over communication channels need to be ciphered because TCP connections are exposed to various threats. Transport Layer Security encryption will provide standard security measures to mitigate man-in-the-middle attacks. FRESCO is a security composition framework focused on anomaly detection and mitigation [108]. Netfuse is an example of a solution that addresses security flaws in SDN architecture [109], protecting the network from DoS attacks. However, there is a significant shortcoming in research related to improving the security of the SDN itself, particularly from threats attacking the controller [1], [7], [8], [124]. SDN security flaws are an important ongoing research topic. Several papers have studied challenges related to SDN architecture. Kreutz *et al.* [8] reveal seven threats associated with SDN architecture. Figure 3.2 depicts three threats directly related to the controller.

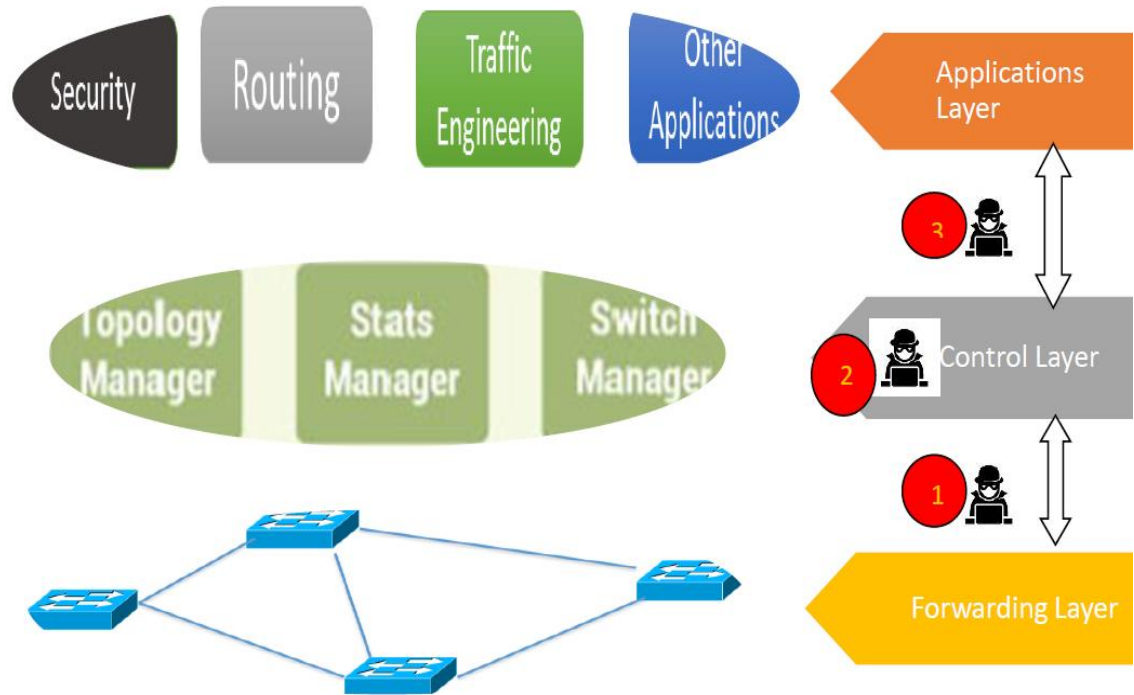


Figure 3.2. Controller security threats.

Ruffy *et al.* [125] used STRIDE to analyse SDN security. The study identified security deficiencies in generic SDN design, such as spoofing in SDN networks caused by unauthenticated access to one of the network elements. The authors offer potential countermeasures, suggesting that attacks may be resolved by enforcing contemporary authentication procedures. In another recent paper, the authors focused on controller and forwarding plan security [126]. The study conducted an analysis process based on Petri nets and attack trees; however, the study was limited in its scope of attacks. The study presented in this thesis focuses on the analysis process of the controller and augments the analysis with an experiment that covers several threats:

1. Attacks on communications between the controller and data plane devices
2. Attacks on controller vulnerabilities
3. Attacks on the controller originating from untrustworthy applications that communicate with the controller.







### 3.4 Software-Defined Network Security Analysis

The first step involved defining critical security objectives, such as system availability and dependability. At this level, the objective was to outline the system characteristics.



The second step involved specifying system components, data flows and trust boundaries. The third step involved dissecting the system using a DFD. The diagram consisted of the elements shown in Table 3.2.

*Table 3.2. STRIDE Graphical Components*

Entity	Details	Graphical Representation	Attacks
External entity	Represents the entities that interact with the system under the modelling		Spoofing identity
Process	System nodes that perform actions on the data flow in the system		Tampering with data
Multiprocess	A process composed of a subprocess		Repudiation
Data stores	Where the data are kept (e.g. database tables)		Information disclosure
Data flow	Data movements in the system		Denial of service
Privilege boundaries	Represent the change in the level of trust		Escalation of privilege

We conducted the analysis on three levels. First, we developed a STRIDE analysis to define possible threats from the design perspective. Second, we described various attacks using attack trees. Third, we simulated possible attacks to identify actual consequences and recommend measures to address threats.

At this point, it is essential to clarify the terms risk, threat, vulnerability and attack. The threat is the harm that can occur to a system asset. System assets are a broad range of resources that vary from devices to information. Threats occur when an intruder carries

out an attack by exploiting a weakness in the system, referred to as vulnerability. Risk is the intersection between threat, vulnerability and consequence. Threat modelling is the process of identifying and evaluating threats. Risk analysis is the identification and assessment of risk severity. Risk management is concerned with risk mitigation and elimination through the adoption of countermeasures.

### 3.4.1 STRIDE

The analysis was conducted using two approaches: a system-oriented approach, which focuses on system components, and an attack-oriented approach. The first stage employed STRIDE (spoofing, tampering, repudiation, information gathering, denial of service and elevation of privilege) as the system-oriented analysis method [21]. Threat modelling included five steps. The first step was to define the critical security objectives, such as system availability or dependability. At this level, the objective was to define system characteristics. The second step was to specify system components, data flows and trust boundaries. The third step was to dissect the system using a DFD. The fourth step examined the DFD elements against the STRIDE attacks; for instance, whether a particular data store element was exposed to information disclosure. The final step was to identify vulnerabilities to threats. Microsoft’s threat modelling tool was used to automate the process of analysis [127]. Figure 3.3 shows a primary DFD of the controller. Through multiple iterations of analysis, we identified a list of potential threats against the controller.

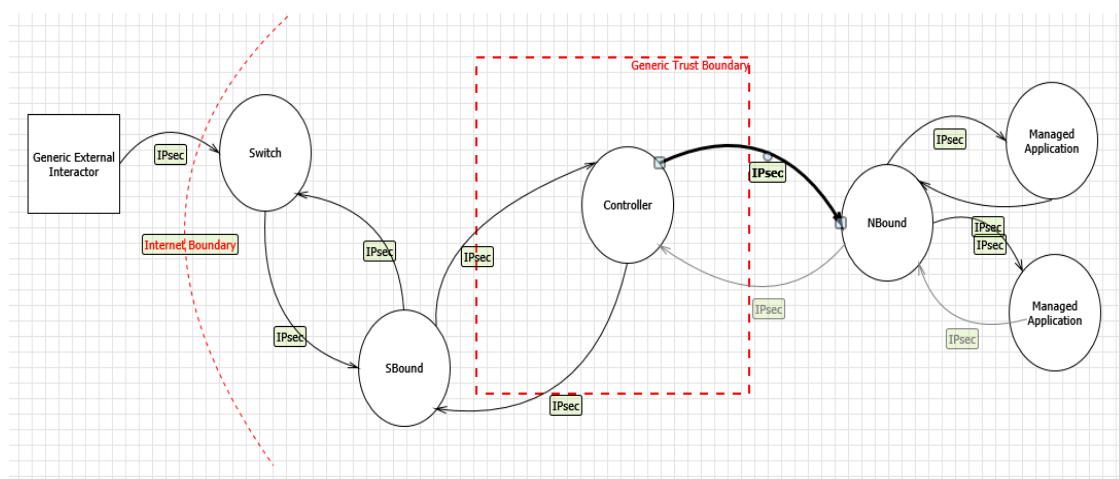


Figure 3.3. Controller high-level dataflow diagram.

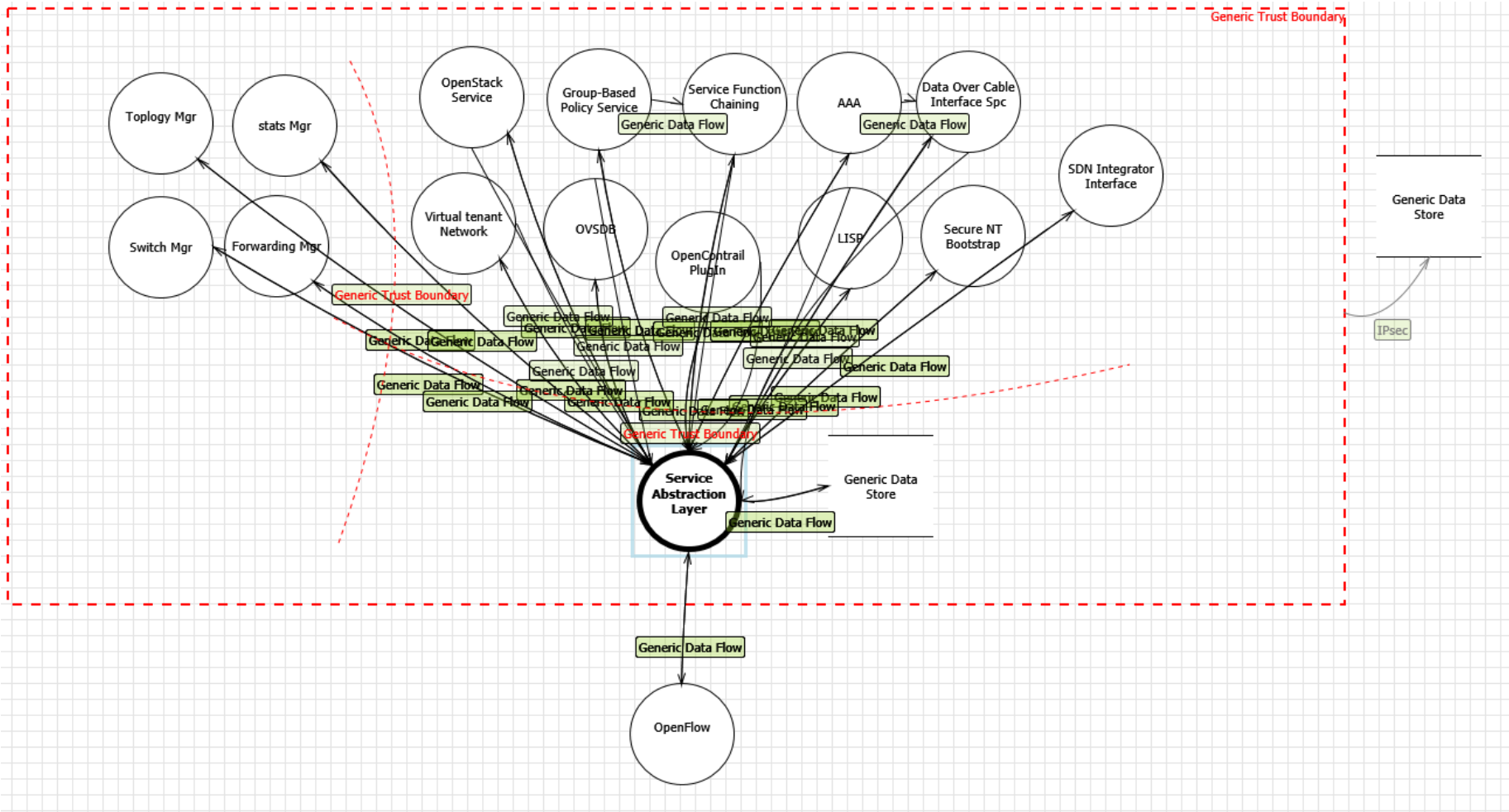


Figure 3.4. Detailed dataflow diagram

Figure 3.4. shows the threats against each component in the controller. Services and plug-ins are exposed to privilege escalation by changing code flow execution through exploiting vulnerabilities such as buffer overflow, heap overflow and string formation attacks. Additionally, services and plug-ins are threatened by DoS attacks by sending requests that exceed the available computation resources. The SAL is exposed to privilege elevation and DoS attacks. Additionally, SAL has a buffer that might be compromised by information disclosure attacks, repudiation attacks and data tampering attacks via changing bits. The controller has a data log file that is also at risk of information disclosure, tampering and repudiation threats. Data flow between the SAL and controller services is vulnerable to information disclosure via sniffing and spoofing attacks. Likewise, data flow between the SAL and OpenFlow is exposed to spoofing and disclosure attacks.

Table 3.3. Controller Threats

	<b>S</b>	<b>T</b>	<b>R</b>	<b>I</b>	<b>D</b>	<b>E</b>
Service abstraction layer (SAL)					X	X
SAL–OpenFlow flow	X			X		
SAL–Other services flow	X			X		
State manager service					X	X
Topology manager					X	X
Switch manager					X	X
Forwarding manager					X	X
OpenStack service					X	X
Virtual tenant network					X	X
Open vSwitch Database Management Protocol					X	X
Group-based policy service					X	X
OpenContrail plug-in					X	X
Authentication, authorisation and accounting					X	X
Service function chaining					X	X
LISB					X	X
Data over cable interface					X	X
Secure NT bootstrap					X	X
SDN integrator interface					X	X
Controller services and plug-ins—SAL flow	X			X		
SAL buffer		X	X	X		
Controller DB		X	X	X		

### 3.4.2 Attack Trees

In the previous section, we identified the threat vectors that may be possible risks to the controller. The current step is to formally describe the execution of threats—or attacks—from the intruder perspective. Attack trees are a semi-formal representation of attacks as a tree data structure. The root of the tree represents the attacker’s ultimate goal, while various nodes attached to the root represent the techniques used to reach the target [22], [128]. Operators OR and AND specify the logical relationships between the

tree branches to form the attack. For example in the spoofing attack below the root spoofing authentication requires all the conditions in 1.1, and 1.2 to be executed to launch the attack. In attack 2 either 2.1 or 2.2 will be sufficient to execute the attack.

#### *3.4.2.1 Spoofing*

Spoofing attacks hack identity, which is not exclusive to individuals. Identity also includes machine identity (IP or media access control addresses), processes running on a host and file spoofing. The controller components are exposed to various spoofing threats. The attack tree is shown below:

---

**Goal: Spoofing access to the controller (OR)**

---

1. Spoofing authentication (AND)
    - 1.1. Spoofing the username (OR)
      - 1.1.1. Social engineering (OR)
      - 1.1.2. Brute force attacks
    - 1.2. Spoofing the password (OR)
      - 1.2.1. Social engineering
      - 1.2.2. Brute force attacks
  2. Spoofing the source address (OR)
    - 2.1. Spoofing the media access control address
    - 2.2. Spoofing the Internet Protocol address
- 

#### *3.4.2.2 Denial-of-service attacks*

DoS attacks exhaust system resources such as bandwidth, memory and storage. Typically, the SDN controller adds a processing overhead to network resources. Thus, the severity of DoS attacks is a significant concern in SDN networks. The attack tree is shown below:

---

**Goal: Attack controller availability by exhausting system resources (OR)**

---

1. Attack controller services (OR)
    - 1.1. Flood requests to controller services such as authentication, authorisation and accounting
  2. Attack controller database
  3. Flood service abstraction layer buffer
  4. Congest network bandwidth
- 

#### *3.4.2.3 Escalation of privilege*

These attacks are based on program flaws, employing techniques such as fuzzing, static analysis and reverse engineering to reveal coding deficiencies. Intruders without privileges can exploit vulnerabilities to execute remote code; subsequently, in the post-exploitation phase, they can escalate their privileges to break into the system entirely. The SDN controller software is likely to have coding flaws, which may be exploited for privilege escalation attacks. The attack tree is shown below:

---

**Goal: Exploit system vulnerabilities to escalate privilege (OR)**

---

1. Exploit vulnerabilities in controller services (OR)
    - 1.1. Fuzzing controller inputs
    - 1.2. Analysis of controller code (OR)
      - 1.2.1. Static analysis
      - 1.2.2. Dynamic code analysis
    - 1.3. Reverse engineering the controller
  2. Escalate standard user privileges granted by the administrator to higher privileges
    - 2.1. Migrating process
- 

#### *3.4.2.4 Information disclosure*

In information disclosure threats, system information such as files, file names and databases are exposed to unauthorised entities. An important example of information disclosure is Structured Query Language injection attacks. The SDN controller

exchanges data with applications and physical planes. These data are accessible through ARP table poisoning. The attack tree is shown below:

---

**Goal: Hacking controller data (OR)**

---

1. Communication sniffing (OR)
    - 1.1. Address Resolution Protocol poisoning
    - 1.2. Domain name system spoofing
    - 1.3. Internet Protocol spoofing
  2. Access controller machine
    - 2.1. Vulnerability exploitation
    - 2.2. Physical access
      - 2.2.1. Bypass authentication
        - 2.2.1.1. Social engineering
        - 2.2.1.2. Brute force
  3. Web attacks (controller web interaction) (OR)
    - 3.1. Structured Query Language injection
      - 3.1.1. Fuzzing
    - 3.2. Cross-site scripting attacks
- 

### **3.5 Simulation**

The purpose of the simulation was to demonstrate the threats identified in the STRIDE and attack tree stages that are relevant to the SDN controller. Possible attacks were executed against a functional controller to deduce real-time statistics and consequences.

#### **3.5.1 Simulation Environment**

A Mininet simulator [97] was used to emulate an SDN network. The system under test consisted of a set of hosts connected via OpenFlow switches in a tree topology. The network utilised an external ODL controller. The hostile machine was a Kali Linux machine connected to the same subnet to which the controller was connected [129]. The Kali machine used a toolkit to demonstrate various attacks.



ODL is an open-source SDN controller project backed by industry leaders such as Microsoft, Cisco, Juniper and Ericsson and is aimed at accelerating the adoption of SDN networks [19]. This study adopted ODL as the empirical controller for the following reasons:

- ODL is sufficiently close to the standard architecture of the SDN controller. ODL is a distributed controller that demonstrates availability features and supports east/westbound APIs. The majority of controllers propose OpenFlow as the only southbound API; ODL goes beyond this concept by providing an SAL that supports the coexistence of various protocols at the southbound API.
- ODL provides Defense4All as an intrusion detection system. This may be used to demonstrate deficiencies of the current detection approach in contrast to the approach proposed by other studies.
- Other security features include security logging and auditing, authentication and authorisation services and secure control plane communication.

#### *3.5.1.1 Simulation execution*

A Mininet simulator was used to implement the SDN network using a tree topology with three switches and three hosts for each switch. The simulator used a real-time external ODL controller. The command used was:

```
sudo mn-topo tree,depth=2,fanout=3-controller remote
```

Figure 3.5 shows the simulation network from the perspective of the controller. The network consisted of four OpenFlow switches, with each switch connected to the hosts. All switches were connected to the ODL controller (installed on an Ubuntu machine). In the same network, there was a hostile machine (Kali). In our simulation, we assumed that the attacker already had access to the network. The intruder had several ways of hacking the network perimeter, such as by exploiting host vulnerabilities and social engineering.

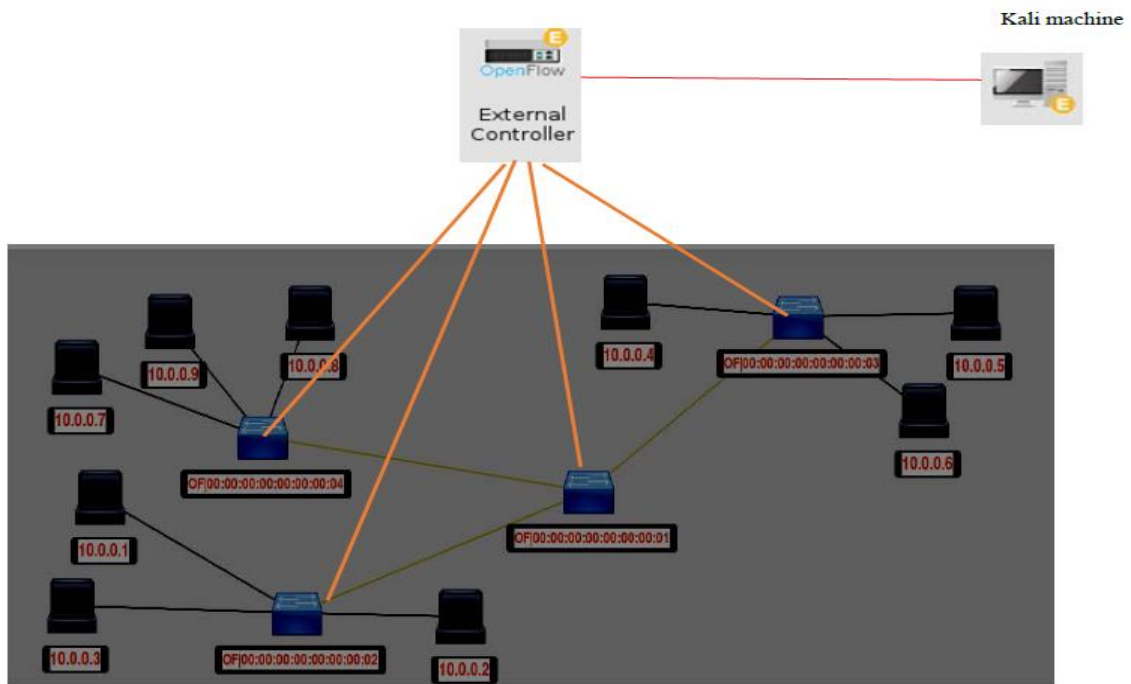


Figure 3.5. Simulation environment.

### 3.5.2 Discussion

In this section, we demonstrate how controller vector attacks may be executed and their consequences on network resources.

#### 3.5.2.1 Information disclosure attacks

Revealing information to unauthorised entities is a crucial issue for SDN controller security. In standard scenarios, the centralisation of SDN information in a single entity is a significant advantage of SDN. However, in the attack scenario, a data-intensive single point is a target for intruders. In this simulation, the attacker launched reconnaissance attacks to discover service availability by checking the OpenFlow port 6633. A port scanning technique using Nmap can reveal all services available on a network [130]. The SDN centralisation paradox spares the attacker from enumerating the entire network. Far-reaching information disclosure attacks can occur when intruders gain access to the controller using publicly available exploits. In this simulation, the attacker used an exploit developed on a remote file inclusion vulnerability [131]. A Python script exploit was downloaded to the controller flow table. A closer inspection of the flows shows that the attacker can map the entire network, list nodes, services and access control lists.

### 3.5.2.2 Denial-of-service attacks

The centralisation concept is a significant design flaw in SDN architecture. In DoS, the attacker floods the controller with an enormous number of requests, eventually exhausting resources and causing the controller to collapse. The consequences of DoS attacks on SDN networks are significant because the entire network turns into a ‘body with no brain’. In this simulation, the Kali Linux machine flooded the controller using a Python script of-flood. Figure 3.6 depicts the network throughput upon executing the flooding attack. The vertical axis represents the throughput of the controller (Ubuntu machine) in bytes, while the horizontal axis shows the controller time domain as the hostile Kali machine begins to flood the controller. The network throughput was between 21:25:55 and 21:26:10, which is low, given there were few communications between the controller and network devices. After 21:26:10, the attacker flooded the controller with requests, escalating throughput and eventually exhausting and plunging the controller. Subsequently, the entire network was brought down. Hence, despite centralisation being a key feature in the SDN model, it can also bring down the controller—request flooding may result in complete network failure.

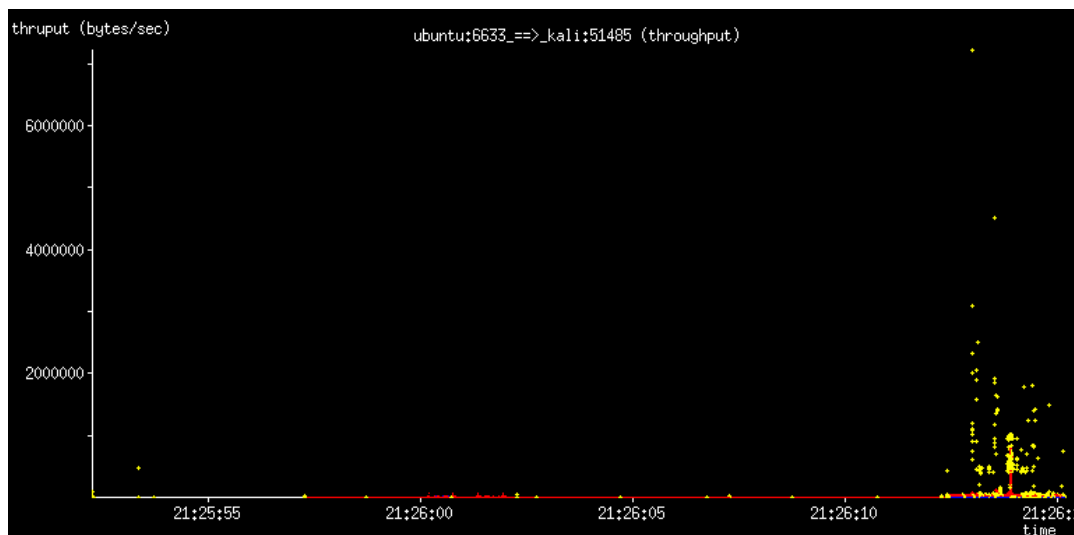


Figure 3.6. Network throughput before and after a denial-of-service attack.

### 3.5.2.3 Spoofing

Traditionally, intruders utilise tools to poison the ARP table or to spoof an IP address to carry out man-in-the-middle attacks. Once again, a feature of the SDN network can become a significant disadvantage from a security perspective. Rather than adding

hardware to the network, programmability saves costs and increases network flexibility by replacing devices with a few lines of code. From an adversary's perspective, programmability is also a great advantage. In this simulation, a Python script was used to impersonate a switch in the network [131].

#### *3.5.2.4 Tampering*

Alteration of data represents a serious penetration of the system. Additionally, compromising the entire network flow from a single point is a critical issue in SDN architecture. The attacker used a script to modify, drop and add entire rules.

### **3.6 Software-Defined Network Security vs. Traditional Networks**

The security analysis and simulation models both revealed the same threats. Similar attack vectors can be launched against SDN networks. However, the architecture of SDN exposes the network to more critical consequences. In conventional networks, if the attacker is successful in executing a remote code against an application or vulnerable operating system, the consequences are limited to a single machine. The intruder would need to escalate privileges or use the victim machine as a pivot to break into other machines or subnets. In SDN networks, the controller comprises software similar to that of any other program; therefore, as demonstrated in the simulation, it is vulnerable to attacks. In traditional systems, software and hardware are integrated separately within various network devices (e.g. routers, switches and middleboxes); hence, in the case of attacks, the damage will be isolated or can be quarantined. In the SDN network, isolation or reduction of damage is more complex, particularly when the controller itself is under attack. Intruders have the advantage of centralisation, meaning the speed and domain of attacks can be accelerated. Therefore, the SDN model increases security challenges.

In the following chapters, we propose a threat detection framework. Given that both SDN and traditional networks are vulnerable to similar attack vectors, the framework addresses traditional network attacks; however, its deployment considers the new architecture. In deploying the framework, we recommend integration of the detection system at the control layer to provide essential protection to the controller itself. By securing the controller, we overcome a significant security issue with SDN networks.

### 3.7 Summary

In this chapter, we provided an in-depth study of controller security. The SDN model introduces unprecedented security challenges, specifically threats related to the controller. While the controller is a crucial entity in the architecture, it is also a valuable target for intruders. We analysed threats using two methods—STRIDE and attack trees—before carrying out an experiment to demonstrate various attacks. The experiment provided examples of vulnerabilities in the architecture. We demonstrated several attacks and the significant consequences of penetrating the controller. DoS attacks led to the failure of the entire network. Network programmability facilitated spoofing attacks because intruders could impersonate devices through coding. The effects of tampering attacks were more significant as the attacker could take control of the entire network by crafting data flow tables through a central point. Additionally, taking over the controller exposed the entire system's information.

SDN architecture paves the way for network innovation and reduces the complexity of traditional networks. However, controller security is a significant issue in the SDN model. Enhancing the security of the controller will increase the opportunity for SDN becoming the dominant networking model. In next chapters, we will focus on implementing an intelligent module to protect the controller itself.

## **Chapter 4: Threat Detection Framework: A Deep Learning Approach**

This chapter introduces the anomaly-based detection framework. The proposed approach utilises unsupervised DL to reduce dimensionality before applying simple (with respect to computation resources and calculations) clustering to the digested data. The framework consisted of two phases. The first phase used an unsupervised DL neural network and the second phase employed a simple clustering algorithm.

DL algorithms may be used for different purposes. For example, an autoencoder may be used for data reconstruction and dimensionality reduction. Therefore, framework design principles were derived from the goals and objectives of the proposed system before being implemented. This chapter provides a theoretical proposal for the integration of the framework in the SDN model and presents the proposed detection framework.

The first section discusses DL algorithms in anomaly-based detection. The second section introduces the components of the framework, which consisted of two phases: an unsupervised DL algorithm and a simple clustering algorithm. The third section depicts the framework workflow and the framework algorithm in pseudocode. Additionally, a detailed description of the different steps is included. The fourth section outlines the design principles adopted for building the framework, including dimensionality reduction, decision boundaries, clustering and curse of dimensionality, network traffic features, the number of hidden layers and neurons and the assumptions required for framework applicability. The fifth section describes the framework in action and the theoretical background of the algorithms. The sixth section depicts the integration of the framework in an SDN model. The seventh section explores the advantages of the framework.

### **4.1 Introduction**

Several machine learning algorithms have been used for network anomaly detection [132]. A typical fundamental deficiency is a poor accuracy, which has made the approach industrially inapplicable. The proposed framework presented in this chapter shows improvement in detection accuracy. The framework adopted semi-unsupervised

algorithms for novel detection to tackle the rapid developments in cybersecurity attacks. The framework used the more elegant technique of unsupervised DL, which dramatically reduces features from the first phase. The framework was designed based on a set of principles in which the design goals were to reduce computations and enhance accuracy.

Following advances in neural nets, DL has been successfully applied in various domains. For object recognition, Hinton *et al.* [14] used a deep belief network for Modified National Institute of Standards and Technology's dataset image recognition, scoring a 1.25% error rate compared with the next lowest error rate of 1.4% achieved by an SVM. In an ImageNet challenge (2012), the convolutional neural network succeeded in reducing the error rate from 26.2% to 16.4% in a dataset containing about 14 m labelled images and 1,000 classes [15]. Speech recognition and signal processing are some of the remarkable application domains for DL. Traditionally, researchers used Gaussian mixture models and hidden Markov models in speech recognition applications. Mohamed *et al.* [133] reduced the phoneme error rate from 26% to 20.7% using deep belief networks.

DL is a set of nonlinear algorithms for multi-layered models. DL algorithms may be supervised or unsupervised. In supervised learning, the training dataset contains input data and data labels. The algorithm learns to predict  $p(y/x)$  where  $x$  and  $y$  are the inputs and outputs, respectively [134]-[136]. This approach is suitable for classification and regression tasks. In unsupervised learning, only an unlabelled dataset is available. Unsupervised DL algorithms aim to learn the probability distribution of a specific dataset. Unsupervised applications include clustering, dimensionality reduction and noise removal. For network anomaly detection, we believe the unsupervised approach has the following advantages: first, unsupervised learning can detect internal representation of the dataset, which conforms to online detection. Second, unsupervised algorithms will theoretically discover unprecedented threats. The automatic discovery of features improves the probability of detecting new attacks in the context of network anomaly detection. Third, we can use unsupervised learning as a pre-training phase before using supervised or reinforcement learning to enhance detection accuracy.

An autoencoder is a neural network consisting of two phases:

- An encoder, which is a deterministic mapping function ( $f_\theta$ ) that transforms an input vector ( $x$ ) into a hidden representation ( $y$ ):
  - $\theta=[W]$ , where  $W$  is the weight matrix and  $b$  is bias
  - $f_\theta(x) \approx x'$
- A decoder, which reconstructs the hidden representation ( $y$ ) to the reconstructed input ( $x'$ ) via  $g_\theta$ .

The autoencoder measures the reconstruction error between  $x'$  (reconstructed) and the input ( $x$ ) to minimise this error (information loss):

$$J(W) = \sum \|x_n - x'_n\| \quad (1)$$

Where  $J(W)$  is the cost function to minimise the cost.

$$\text{Arg min } (J(W))_{\{w,w',b,b'\}} \quad (2)$$

where  $w$  and  $b$  are encoder weights and biases, respectively, and  $w'$  and  $b'$  are weights and biases, respectively, for the decoder.

Various functions, such as squared error, may be used as cost functions. For cost function optimisation, several options, including stochastic gradient descent (SGD) and adaptive moment estimation (Adam) optimiser, are available.

RBMs are energy-based models in which each feature configuration is assigned scalar energy [135]. The learning process updates the energy function to ensure the shape has desirable properties. The probability distribution of energy function is shown in Formula (3):

$$p(x) = \frac{e^{-E(x)}}{Z} \quad (3)$$

where  $Z$  is the partitioning function, defined in (4):

$$Z = \sum_x e^{-E(x)} \quad (4)$$

Boltzmann machine's energy function is defined in (5):

$$E(x) = -x^T W x - b^T x \quad (5)$$



where  $W$  is weight matrix and  $b$  is the bias parameter.

To enhance the RBM, hidden units are introduced. RBMs are a type of Boltzmann machine with restrictions on connections between visible-visible and hidden-hidden units. The energy function of RBMs is represented by (6):

$$E(v, h) = -b'v - c'h - h'Wv \quad (6)$$

where  $b'$  and  $c'$  are the biases for visible and hidden units, respectively, and  $W$  is the weight of connections between hidden and visible units.

Table 4.1 lists several research papers utilising the deep learning algorithms for anomalies detection. The table shows the used dataset and other algorithms used to compare the performance of the deep learning algorithm. The used algorithms are unsupervised, different variations of autoencoders, and RBM.

*Table 4.1. Deep Learning for Anomaly Detection*

Research	DL Algorithm		Classic Machine Learning Algorithms				Dataset
	AE	RBM	SVM	PCA	KPCA	Other	
[137]	✓			✓	✓		Images
[138]		✓	✓	✓	✓		KDD99, USENET, Thyroid
[139]		✓					KDD99 and bot data
[140]	✓						Generated traffic
[141]		✓					KDD99
[142]	✓			✓	✓		Lorenz, sat-A
[143]	✓						KDD99

*Note:* AE: autoencoder; RBM: restricted Boltzmann machine; SVM: support vector machine; PCA: principal component analysis; KPCA: kernel PCA.

A comprehensive study evaluated seven unsupervised machine learning algorithms [144], benchmarking the KDD99 dataset, and concluded that all algorithms performed poorly in detecting remote to local attacks, while SVM and Y-means performed well over the other techniques in detecting user to root attacks. Further, C-means delivered

the most unsatisfactory results in almost all experiments. Lastly, fuzzy clustering was not suitable for distinguishing normal and abnormal data in intrusion detection.

The concept of dimensionality reduction refers to projecting highly dimensioned data onto a lower subspace without a significant loss of data meaning. Additionally, in lower dimensional data, the discrimination between normal and abnormal classes is evident [145]. PCA is a dimensionality reduction algorithm that learns linear relationships:

$$f(x) = W_{(x)}^T + b \quad (7)$$

where  $x$  is input and  $x \in R^{(d_x)}$ .

Kernel PCA is a nonlinear version of basic PCA [145]. To represent nonlinear relationships, a kernel function is used to map the data to higher dimensions before using PCA to reduce dimensionality. Autoencoder algorithms imply dimensionality reduction because they convert data into new representations that keep most of their significant features (encoder) before executing a reconstruction phase (decoder). Various studies compared autoencoders as a dimensionality reduction algorithm with PCA and its nonlinear extension, kernel PCA. One experiment that compared PCA, kernel PCA, autoencoder and demonising autoencoder found that the autoencoder and demonising autoencoder performed significantly better. The study applied the four algorithms to an artificial dataset and two real datasets.

## 4.2 Framework Components

Figure 4.1 depicts the main components of the proposed detection system. The framework was based on unsupervised deep neural networks. Two types of Unified Service Description Language (USDL) algorithms were used: autoencoders and RBMs. As shown in Figure 4.1, the first layer was a deep neural network, the purpose of which was to reduce input features. Outcomes were then fed as inputs into the subsequent algorithm.

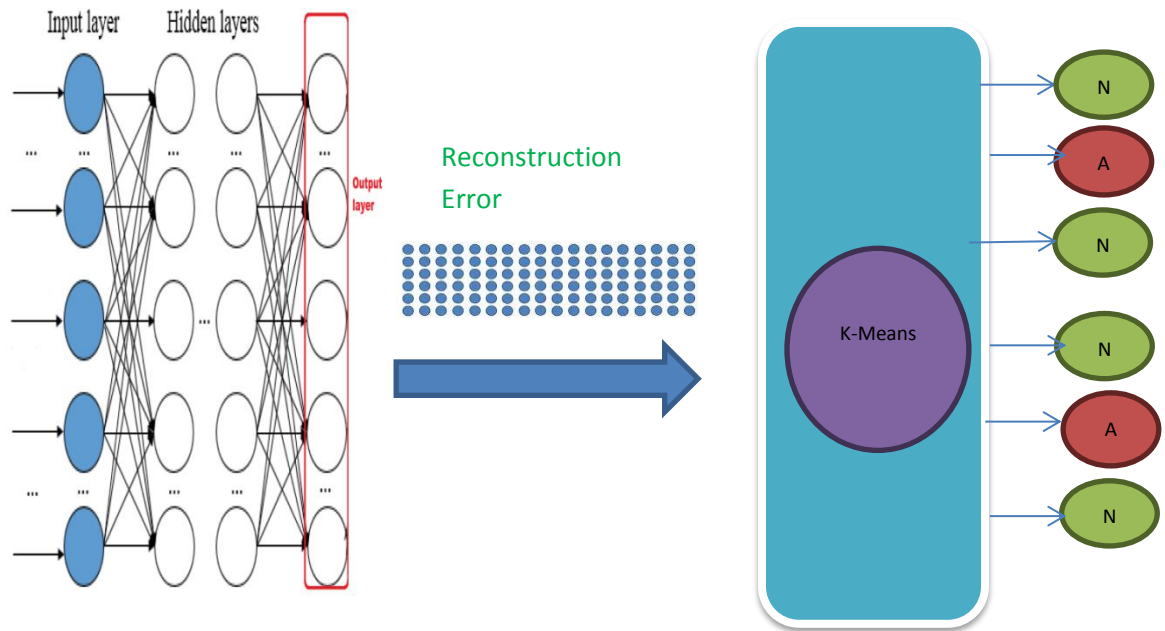


Figure 4.1. Proposed detection system architecture.

Network threats are continually expanding; thus, from a security perspective, an IDS must be capable of detecting attacks that have not been previously seen. This concept is theoretically achievable using unsupervised learning. In contrast, supervised algorithms—traditional algorithms or those based on DL—must have previously been exposed to samples and classes to classify new records. The approach presented here adopted a new method of employing USDL that did not use the direct output for further processing by the second phase. Instead, it calculated the difference between inputs and outputs. This dramatically reduced the number of features that had to be passed to the second phase to a single value rather than a vector of values.

Reconstruction errors were passed to the second phase for clustering. Clustering is the classification of samples into different groups or, more precisely, the partitioning of a dataset into subsets (clusters) so that the data in each subset ideally share common characteristics. Measuring the distance between samples and the predefined cluster centre is a common approach used in clustering. The algorithm produces a set of clusters, with each cluster containing reconstruction errors that represent normal or abnormal records. This is eventually translated as normal and abnormal clusters by linking reconstruction errors back to their input records.

### 4.3 Framework Workflow and Algorithms

Figure 4.2 depicts the framework flowchart; the first step in the workflow was to conduct data normalisation. The data were collected from network traffic. Each record represented a packet, with a typical packet containing two types of information, numeric or string. The initial step in normalisation was to convert non-numeric values to numbers that could be handled by the DL algorithm. Once the data were normalised, they were traversed through the autoencoder or RBM, which produced the corresponding output (using previously learned weights and biases). The difference between the input and output was calculated and then passed to the second layer for clustering.

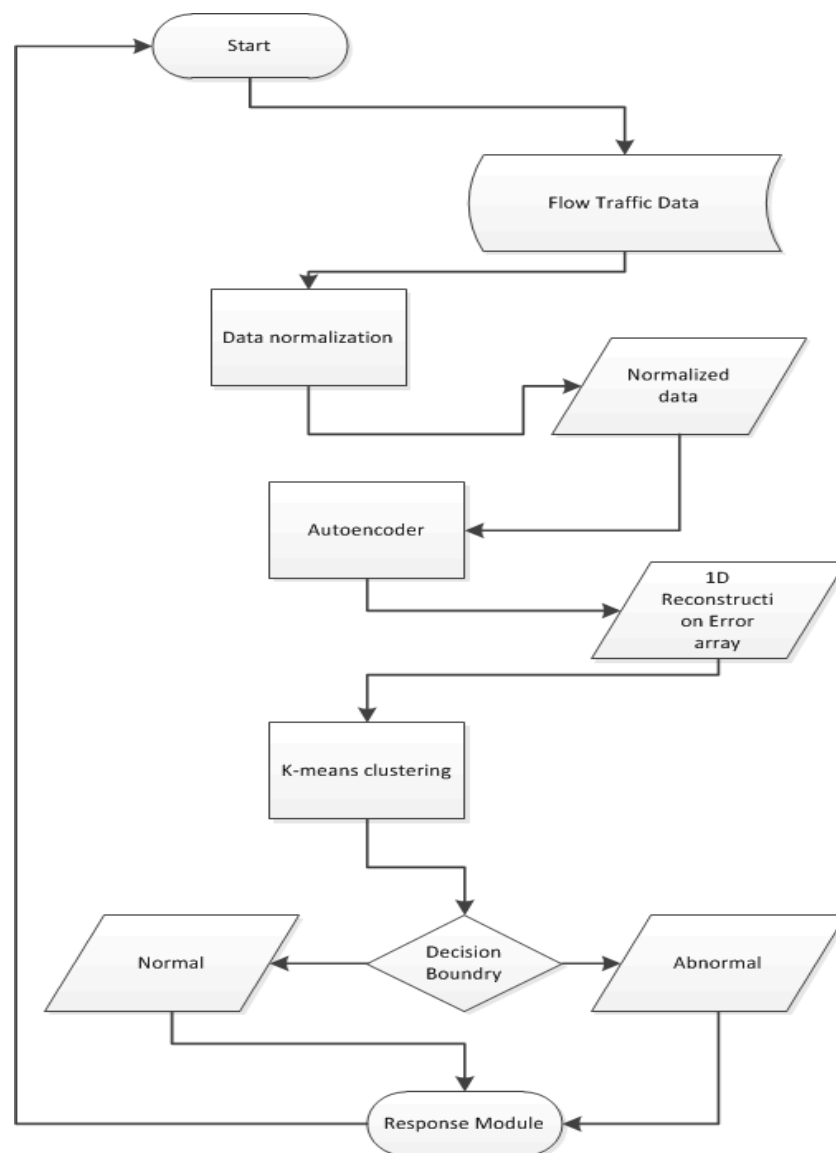


Figure 4.2. Detection system flowchart.

The final action was to place the record into one of the clusters (normal or abnormal).

The pseudocode in Table 4.2 provides an elaborated view of the proposed system functionality. The algorithm processed D vector with 41 elements (number of entries in each packet used for the simulation), taking  $T_k$  records during the training phase. The final output of the system is set of clusters  $C_0 \dots C_n$ , with each C marked as normal or abnormal. In the execution mode, the system initiated the model—such as weights and biases—during the training step. Building the model involved identifying patterns in the data, which was done through discovering the appropriate weights and biases through several sweeps of the data (sliced into batches). Internal functions for calculating and minimising loss were used during the model-building process.

*Table 4.2. Algorithm for Anomaly Detection*

<p><b>Data:</b> Network traffic records (continuous and digit values) (D).</p> <p><b>Input:</b> <math>t \in T_0 \dots T_k</math> where T is <math>1 \times 41</math> tensors and <math>k =</math> no. of traffic records in the normalised dataset DNT generated by scaling <math>k</math> samples of the D.</p> <p><b>Output:</b> A set of clusters <math>C_0 \dots C_n</math>, where each C is normal or abnormal.</p> <p><b>Procedure:</b></p> <p><b>Training:</b></p> <p>Let EP be the number of epochs.</p> <p>Let <math>s =</math> batch size and no. of batches (BN) = DNT/s.</p> <p>Let <math>i, z = 0</math>.</p> <p>While (<math>i &lt; EP</math>) do:</p> <p>For (<math>z = 0; z &lt; BN; z++</math>):</p> <p style="padding-left: 2em;">For each t:</p> <p style="padding-left: 4em;">Pass t through the RBM/autoencoder network.</p> <p style="padding-left: 4em;">Calculate weights and biases after reconstruction.</p> <p style="padding-left: 4em;">Update RBM weights (W) and biases (b)</p> <p>EP++</p> <p>Return RBM/autoencoder trained the model with updated weights and biases.</p> <p><b>Testing:</b></p> <p>For each <math>t_s \in T_s</math> where <math>t_s \notin T</math></p> <p style="padding-left: 2em;">Pass <math>t_s</math> through the autoencoder/RBM network</p> <p style="padding-left: 2em;">Calc reconstruction error (reconstruction error) tensor</p> <p>Pass <math>t_s</math> to k-means.</p> <p>Initialisation: set K seed points randomly.</p> <p>Assign each sample to the cluster of the nearest seed point measured with a predefined distance metric.</p> <p>Calc. new centroids of the clusters of the current cluster.</p> <p>Go back to Step 2), stop when no more new assignment.</p> <p>Return: <math>C_0 \dots C_n</math> of reconstruction error</p>
--

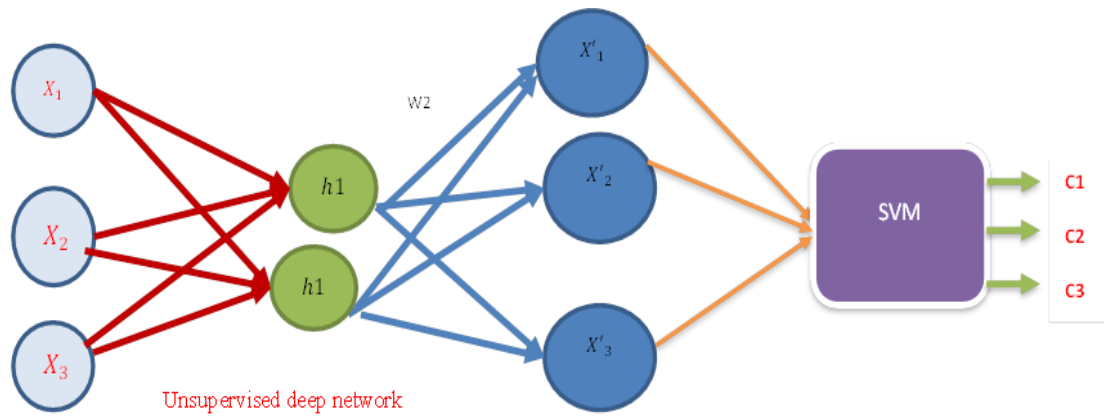
Once the model was established, it was evaluated in the testing process. The testing dataset ( $T_s$ ) was a set of samples that did not belong to the training dataset ( $T_k$ ). Subsequently, the outputs of  $T_s$  were used to calculate the reconstruction errors, with a vector containing a single value for each  $t_s \in T_s$ . The vector was passed to the unsupervised algorithm. The algorithm in the second phase was not computationally expensive. The output of the second stage was a set of normal and abnormal clusters.

## **4.4 Framework Design Principles**

This section introduces six principles upon which the framework was proposed. The first principle is a form of dimensionality reduction in which a new space (features) was generated from the inputs as a pre-step to isolating abnormalities. The second principle is the separation between a normal and an abnormal (decision). The third principle is resolving the dimensionality reduction in clustering. The fourth is related to the low number of the features in the network packets. The fifth one relates to deciding on the number of hidden layers. The last principle is a consideration for the issues related to the application domain.

### **4.4.1 Dimensionality Reduction and Anomaly Detection**

Applying the unsupervised algorithm as a pre-training step is a common practice to enhance accuracy in many frameworks. For example, Figure 4.3 shows a classification framework in which a pre-training unsupervised step was included before outputs were passed to the SVM, which performed the classification. However, this comes at the cost of additional computation resources such as memory, processing and time.



*Figure 4.3. Applying deep learning as a pre-step for support vector machine classification.*

One of the goals of the framework was to reduce the additional cost of computation simply—rather than adding a series of inputs, possibly hundreds or thousands of features, to the second phase, a single value was passed.

The first stage of the proposed framework was based on the idea of dimensionality reduction. The concept of dimensionality reduction refers to projecting highly dimensioned data onto a lower subspace without significant loss of data meaning. Additionally, in lower dimensional data, the discrimination between normal and abnormal classes is evident. PCA is a dimensionality reduction algorithm that can learn linear relationships, while kernel PCA is a nonlinear version of the basic PCA. To learn nonlinear relationships, a kernel function is used to map the data to higher dimensions before PCA is used to reduce dimensionality.

Figure 4.4 shows the underlying architecture of autoencoders; autoencoder algorithms imply dimensionality reduction as they convert data into a new representation that keeps most significant features (encoder) before executing a reconstruction phase (decoder). Various studies have compared autoencoders with PCA and its nonlinear extension, kernel PCA, as a dimensionality reduction algorithm.

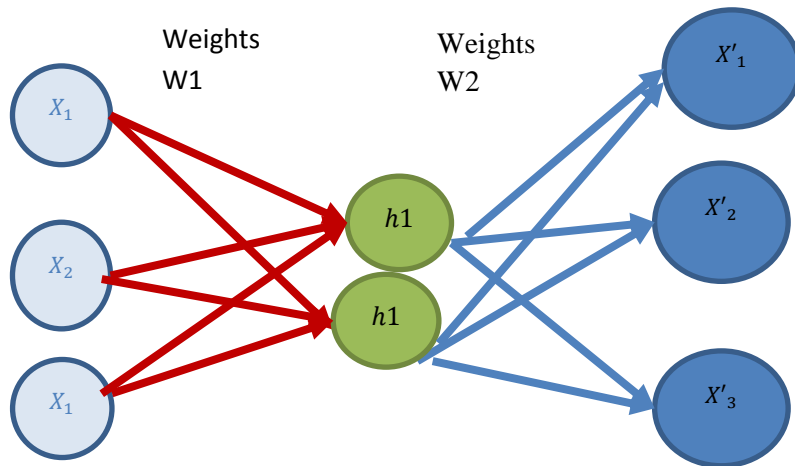


Figure 4.4. Generic autoencoder architecture.

*Principle 1: A new space with lower dimensionality, in which normal and abnormal samples can be separated, may be generated from the original space. A smaller space with reduced dimensions is equivalent to less memory and better performance, such as classification. Models of smaller spaces consume less memory and runtime.*

#### 4.4.2 Decision Boundaries

To identify anomalies using dimensionality reduction techniques, the data sample was projected onto the correlation structure deduced by the algorithm. Records with significant reconstruction errors—relative to a predefined threshold—were marked as anomalies.

A similar approach is theoretically applicable by passing the data through a trained model before defining a reconstruction error threshold to isolate anomalies. This was achievable using a single regression algorithm in the second phase.

However, the experiments showed that reconstruction errors were not linearly separable using a regression algorithm or a simple threshold. Hence, the framework provided a nonlinear algorithm to cluster reconstruction errors.



### 4.4.3 Resolving Clustering and the Curse of Dimensionality

The framework employed a clustering algorithm in the second phase.  $k$ -means or mean shift were used to group reconstruction errors.

*Reconstruction errors belonging to normal instances will be correlated enough to occur in the same cluster or clusters, while the same principle is valid for abnormal instances.*

A substantial reason for selecting a clustering algorithm in the second phase was to maintain the ability of self-learning. If no prior knowledge is required, or at least no complete knowledge in semi-supervised deployment, this feature conforms to zero-day attack detection. From a different perspective, the framework aimed to improve the detection accuracy of clustering algorithms using a pre-phase of data processing. This technique takes advantages of both worlds, such as dimensionality reduction using DL and the ability to detect new attacks from clustering. Additionally,  $k$ -means is a fast and computationally efficient algorithm. However, the curse of dimensionality has a significant impact on  $k$ -means.

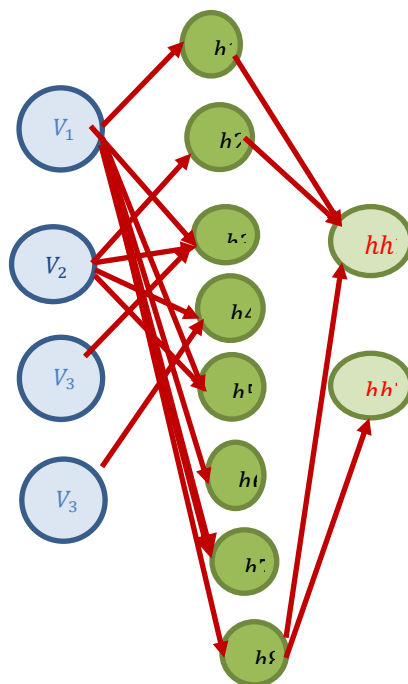
$k$ -means is a standard clustering algorithm that iteratively partitions training datasets to learn a partition of the given dataset to produce a set of clusters. The clustering is produced by minimising the sum of the squared distance to its representative object in each cluster. As the number of features increases, the distance between any two points in the dataset converges. Increasing dimensionality increases sparsity. To revoke the curse, dimensions must be reduced—maintaining fewer features results in more efficient clustering. Hence, the role of the DL phase was to reduce each record to a single value.

### 4.4.4 Network Traffic Features and Deep Learning

DL is a data-striving algorithm that excels when there are large volumes of data. In cases with limited data samples or features, the model will suffer from overfitting. In the context of the networking domain, the packet features are limited. Additionally, many of the features are off. To overcome these limitations, the system follows a similar approach to that of the kernel PCA trick. The basic idea of the kernel trick lies in Vapnik-Chervonenkis's theory:

*Projecting the input data to higher dimensions enables greater clustering power. Increasing the number of neurons in the first hidden layers allows the separation of samples.*

The concept is similar to zooming into the data so that they become more separable. The framework presented in this thesis adopted this concept. Figure 4.6 depicts the idea of increasing the dimension of the input data. In the first hidden layer, the number of the neurons was doubled before being dramatically reduced in the following layers. The experiment showed improved accuracy (discussed in Chapter 6).



*Figure 4.5. Projecting inputs to a new higher dimension.*

#### **4.4.5 Number of Hidden Layers and Neurons**

To decide on the number of the hidden layers, it was necessary first to identify why more hidden layers were needed. The answer lies in the basic purpose of the neural network— approximation. Feed-forward neural networks are capable of approximate continuous functions on a specific dataset. Theoretically, a neural network with a single hidden layer can be used to approximate any continued function.

This raises the question of whether more hidden layers are required. A neural network with more hidden layers (i.e. more structure) can understand the structure of the dataset. In the scope of this research, the main focus was on IP packet fields, enabling the network to find a more complex structure in the entire packet rather than in individual elements only.

In practice, there is no clear recommendation for deciding on the number of hidden layers. However, implementation was derived from two factors:

- A comparison of the problem with other typical domains such as the Modified National Institute of Standards and Technology database, the dataset for handwritten digits . Images are  $28 \times 28 = 784$  pixels to be translated to 784 neurons at the input layer. With two hidden layers, the accuracy is reasonable.
- Experiments and trials: During the implementation, many combinations of different layers and neurons were tested.
- Generalisation is the way in which the model may be generalised to new samples (not included in the training phase). The built model should avoid overfitting and underfitting. Overfitting occurs when the model is trained perfectly (by an increasing number of hidden layers). Underfitting occurs when the model has limited generalisation because of inadequate training where essential features are not detected.

#### **4.4.6 Application Considerations**

The framework adopted a semi-supervised approach. In supervised detection, the model is trained on labelled instances in which each record is labelled as normal or abnormal before entering the operation mode, which is expected to recognise unlabelled instances. The supervised approach suffers two limitations: first, if the system experiences instances that do not occur in training samples, it will fail to predict them. In network anomaly detection, new attacks (samples) persistently emerge. Second, the training sample is usually imbalanced because attacks are less frequent in network traffic, which negatively affects the quality of the generated model during the training. In the unsupervised approach, the model can identify the structure of the data. For anomaly detection, this approach assumes that the frequency of anomalies is lower than those corresponding to normal behaviour. This assumption is likely to affect detection

accuracy. For example, in DoS attacks, the attacker floods the system with an enormous number of requests; in such cases, the frequency of abnormalities may surpass regular traffic records.

The framework adopted a semi-supervised approach in which the system was trained using a typical dataset to build the model. Subsequently, during the testing (or operation) mode, abnormal samples could be discriminated by the model. If the framework found an unprecedented pattern, it would classify it as an abnormality. This approach provided the following advantages:

- Theoretically, it was capable of deciding on unprecedented attacks.
- In the training phase, there was no need for a balanced dataset; however, the model tolerated abnormalities in the dataset.

#### 4.5 The Framework Process

An autoencoder is a neural network consisting of two phases. An encoder is a deterministic mapping function ( $f_\theta$ ) that transforms an input vector ( $x$ ) into a hidden representation ( $y$ ):

$$f_\theta(x) \approx x'$$

$$\theta = \{\mathbf{W}, b\}, \text{ where } \mathbf{W} \text{ is the weight matrix and, } b \text{ is bias} \quad (8)$$

A decoder reconstructs the hidden representation ( $y$ ) to the reconstructed input ( $x'$ ) via  $g_\theta$ .

The autoencoder measures the reconstruction error between  $x'$  (reconstructed) and the input ( $x$ ) to minimise this error (information loss):

$$J(W) = \sum ||x_n - x'_n || \quad (9)$$

where  $J(W)$  is the cost function to minimise the cost.

$$\text{Arg min } (J(W))_{\{w,w',b,b'\}} \quad (10)$$

where  $w$  and  $b$  are encoder weights and biases, respectively, and  $w'$  and  $b'$  are weights and biases, respectively, for the decoder.

Various functions such as mean squared error may be used as cost functions. For cost optimisation, several options, including SGD and Adam optimiser, are available.

RBM is an energy-based model in which each feature configuration is assigned a scalar energy [10]. The learning process updates the energy function to ensure the shape has desirable properties. The probability distribution of the energy function is shown in Formula (11):

$$p(x) = e^{(-E(x))/Z} \quad (11)$$

where  $Z$  is the partitioning function, defined in (12):

$$Z = \sum_v e^{-E(x)} \quad (12)$$

Boltzmann machine's energy function is defined in (13):

$$E(x) = -x^T W x - b^T x \quad (13)$$

where  $W$  is weight matrix and  $b$  is the bias parameter.

To enhance the RBM, hidden units were introduced. RBMs are a type of Boltzmann machine with restrictions on connections between visible-visible and hidden-hidden units. The energy function of RBMs is represented by (14):

$$E(v,h) = -b^v v - c^h h - h^T W v \quad (14)$$

where  $b^v$  and  $c^h$  are the biases for visible and hidden units, respectively, and  $W$  is the weight of connections between hidden and visible units.

The autoencoder was implemented with an input layer equal to the length of the input vector. Several layers were added to the encoder. Weight matrices were defined to construct connections between each layer and its subsequent layer. For the decoder, the output of the encoder (final hidden layer) was the input for the decoder, followed by a series of hidden layers with associated weight matrices. Data were passed to the framework as a single dimensional vector in which each element in the array represented a single feature. The input data were sliced into batches. For each input vector, the encoder used the activation function to define neuron status. Functions such

as sigmoid, tanh or ReLu are operable by autoencoders. The activation function was applied to each layer. The sigmoid function used the weight matrix, input vector and bias vector as inputs, with outputs in the sigmoid function being either 0 or 1 for each neuron. The encoder used the same activation function to reconstruct the input from the final layer in the encoder. The main goal of the autoencoder was to rebuild a similar version of the input with minimal error, where the error does not equal zero. Otherwise, the model would suffer a generalisation problem. After reconstruction, the model measured the distance between the input and the output. Several functions were utilised to find the distance or reconstruction error. For example, mean squared error calculated the summation of squared difference for each neuron in the input and its corresponding neuron in the output, divided by the number of neurons in one vector.

The model used an optimiser to adapt weights and biases to reduce reconstruction errors. For example, the Adam optimiser combined RMSProp and SGD to store an exponentially decaying average of past squared gradients and an exponentially decaying average of past gradients.

Through several epochs, the model reconstructed and optimised until the network had been established (experimentally). The next step was to pass the reconstruction error to the second phase. The reconstruction errors were stored in a one-dimensional array and were input to a clustering algorithm such as *k*-means. *k*-means initialised the cluster centres randomly before allocating each reconstruction error to the nearest cluster. The distance was measured using specific functions such as Euclidean distance. Then, it calculated the new centre for each cluster by minimising the sum of the squared distance of its elements.

A similar reconstruction procedure was conducted for RBMs; however, RBMs involved different steps following the reconstruction.

Activations were combined with individual weights and biases. Results were passed to the visible layer. The RBM reconstructed data by making several forward and backward passes between the visible and hidden layers. Samples from probabilistic tensor selected the input in the reconstruction phase. The same weight matrix and visible layer biases were used for the sigmoid function. The output produced was a reconstruction, which approximated the original input.

RBM as an energy-based probabilistic model defines a probability distribution as:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \mathbf{P}(\mathbf{v}, \mathbf{h}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}{Z} \text{ where } Z = \sum_{\mathbf{v}} e^{-F(\mathbf{v})} \quad (15)$$

where  $F(\mathbf{v})$  is the free energy function =  $-\log \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \mathbf{P}(\mathbf{v}, \mathbf{h}) = \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v}, \mathbf{h})}}{\sum_{\mathbf{v}} e^{-\log \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}} \quad (16)$$

An energy-based model can be learned by performing SGD on the empirical negative log-likelihood of the training data, where the log-likelihood and the loss function are:

$$L(\theta, V) = \frac{1}{N} \sum_{v^i \in V} \log p(v^i) \quad \text{and} \quad l(\theta, V) = -L(\theta, V) \quad (17)$$

Then, the data negative log-likelihood gradient has the following form:

$$-\frac{d \log p(\mathbf{v})}{d\theta} = \frac{dF(\mathbf{v})}{d\theta} - \sum_{\tilde{\mathbf{v}}} p(\tilde{\mathbf{v}}) \frac{dF(\tilde{\mathbf{v}})}{d\theta} \quad (18)$$

$$-\frac{d \log p(\mathbf{v})}{d\theta} = \frac{dF(\mathbf{v})}{d\theta} - \frac{1}{|N|} \sum_{\tilde{\mathbf{v}} \in N} p(\tilde{\mathbf{v}}) \frac{dF(\tilde{\mathbf{v}})}{d\theta} \quad (19)$$

where  $\mathbf{v}'$  is a sample of  $N$ .

To minimise loss, we must maximise the product of probabilities assigned to the training set  $dF(\mathbf{v})/d\theta$ :

$$P(\mathbf{v}) = -b'v - c'h - h'Wv \text{ or } F(\mathbf{v}) = -b'v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)} \quad (20)$$

where  $E(\mathbf{v}, \mathbf{h}) = -b'v - c'h - h'Wv$  and  $b'$  and  $c$ , are biases.

From (19) and (20):

$$-d \log(p(\mathbf{v})) / (dW_{ij}) = E_{\mathbf{v}} [p(h_i | \mathbf{v}) \cdot v_j] - v_j \cdot \sigma(W_i \cdot v^{(i)} + c_i) \quad (21)$$

Formula (21) defines the loss function as the average negative log-likelihood, with the objective being to minimise it. To achieve this, we needed the partial derivative of this function with respect to all its parameters. From Formula (21), optimisation or minimising loss depended on adjusting the weights ( $W$ ) and biases ( $C$ ). SGDs were used to find the optimal  $W$  tensor.

The derivation had two terms. The first term is the positive term  $E_v [p(h_i | v) \cdot v_j]$ , which depended on the data ( $v$ ) and increased the probability of the inputs. The second was the negative term  $-v_j^{(i)} \cdot \sigma(W_i \cdot v^{(i)} + c_i)$ , which depended on the model and decreased the probability of the output generated by the model.

The positive phase increased the probability of training data.

The negative phase decreased the probability of samples generated by the model.

The negative phase was difficult to compute; therefore, we used a method known as contrastive divergence (CD) to approximate it. It was designed in such a way that the direction of the gradient estimate was somewhat accurate, even when the size was not (in real-world models, more accurate techniques such as CD-k or PCD are used to train RBMs). During the calculation of CD, we used Gibbs sampling to sample from our model distribution.

CD is a matrix of values that were computed and used to adjust the values of the  $W$  matrix. Changing  $W$  incrementally led to the training of  $W$  values. Subsequently, at each step (or epoch),  $W$  was updated to the new value  $W'$  using the following equation:

$$W' = W + \alpha \cdot CD \quad (22)$$

Here,  $\alpha$  is some small step rate, also known as the ‘learning rate’.

## 4.6 Framework for Software-Defined Networks

This section discusses the integration of the proposed detection system in the SDN model. The actual integration is beyond the scope of this research. Multiple intrusion detection applications have been developed to detect malicious activities in SDN networks. For example, the ODL controller uses the Defense4All application to detect and mitigate DDoS attacks. However, the application does not protect the controller itself; rather, it deploys a set of rules to protect the network at its edges. In the event of malicious activity, the Defense4All application requests network information from the controller and acts through its attack mitigation module. Security limitations of this application include the following:



- The application must first communicate with the controller to gather statistics and raw data used by the IDS to decide whether an activity is malicious. Consequently, the controller is exposed to the threat before the decision is made.
- The controller's location in the architecture makes it vulnerable to new types of attacks that require novel mechanisms. For example, mechanisms to ensure security in communications between the controller and the IDS should be present.
- Controller software may be prone to traditional software vulnerabilities, which require advanced detection techniques such as deep packet inspection.

Figure 4.6 shows the deployment of the proposed system. Integrating the IDS as an extension of the controller plane provided the following advantages:

- Centralisation: Figure 4.6 shows the deployment of classical IDS dispersing over the network, where it protects a network portion or set of them. The proposed architecture takes advantage of the centralisation feature of SDN, in which the proposed IDS has a global view of the entire network. This deployment protects higher, lower and control planes. Compared to the deployment of IDS in conventional networks shown in figure 4.7, the proposed deployment offer global view and centralisation, which boost the performance.

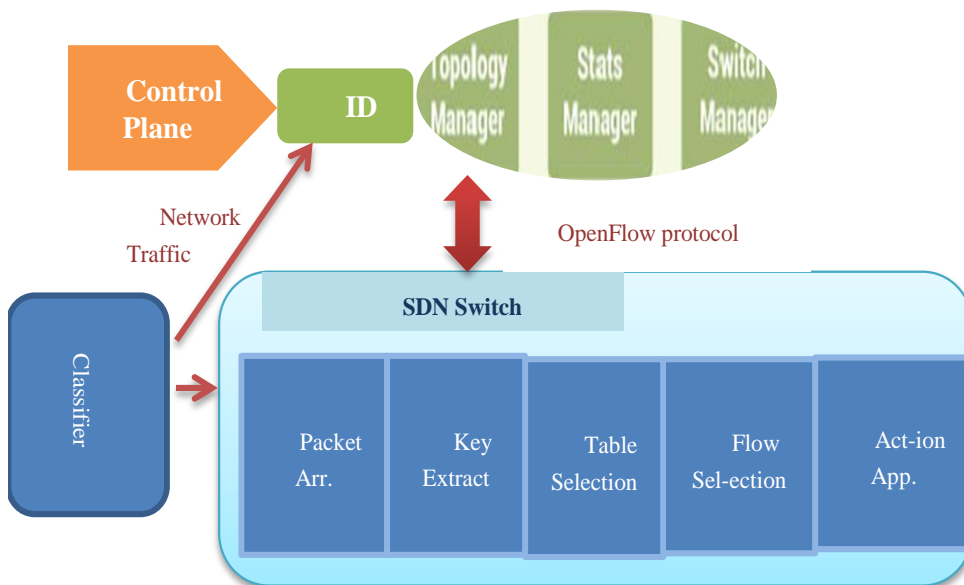
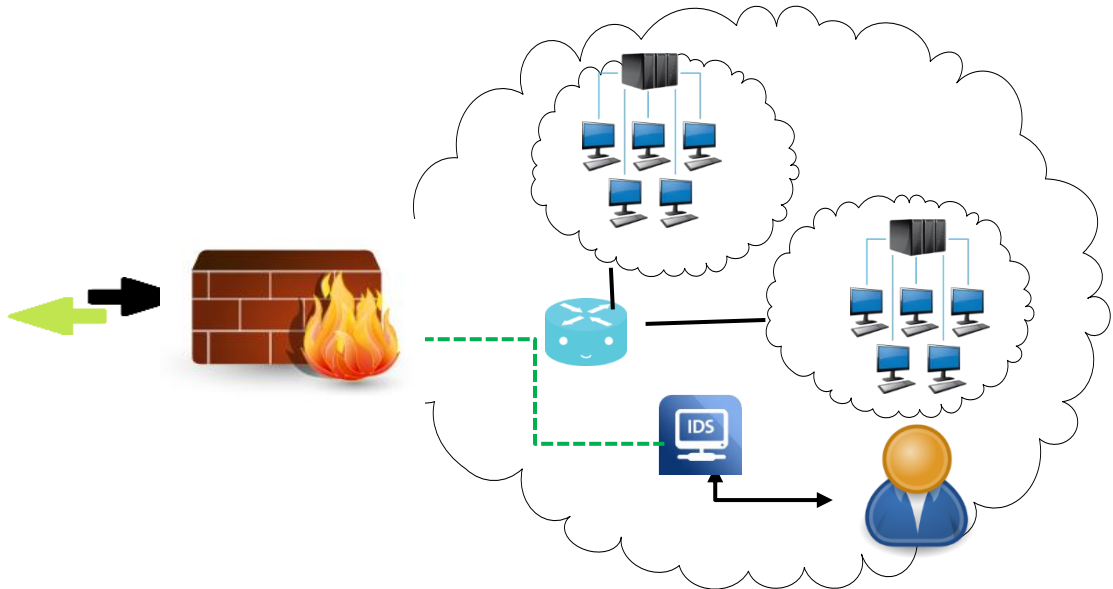


Figure 4.6. Deployment of intrusion detection systems in software-defined network architecture.

- In extensibility scenarios, where the IDS is responsible for attacks, its positioning at the control plane mitigates threat propagation.
- The deployment protects the controller itself because the IDS is deployed as a plug-in that works with the controller, rather than as a component managed by the controller.



*Figure 4.7. Deployment of intrusion detection systems in traditional networks.*

For instance, ODL provides a model-driven service abstraction layer through which new functions can be added to the controller.

## 4.7 Framework Features

We proposed a framework based on DL for attack detection in network traffic. We investigated various aspects of applying DL for network anomaly detection. This research focused on unsupervised algorithms because they have the potential to detect novel attacks. We provided a comparative study of autoencoders and RBMs. Given that it is not possible to use USDL as a standalone for anomaly detection, we adapted algorithms for anomaly detection purposes. The proposed detection framework consisted of two phases: the first phase was based on unsupervised DL algorithms, while in the second phase, the outputs were forwarded to a simple clustering machine learning algorithm. Two unsupervised DL algorithms were used to demonstrate prediction accuracy. Therefore, the framework provided the following advantages:

- Proposed and implemented a threat detection framework: The framework was applicable to different networking models, including conventional networks, SDN and IoT.
- The framework provided a method of solving the problem of classical clustering algorithms such as  $k$ -means, which performs poorly in high-dimensional data. The use of unsupervised DL as a step to reduce dimensionality dramatically enhanced the accuracy of  $k$ -means.
- The framework adopted the reconstruction errors produced from the DL algorithms as a boundary decision for anomaly detection instead of applying the unsupervised algorithms as a pre-training step only. However, the decision did not rely on a simple regression procedure; a clustering approach was adopted as reconstruction errors are not linearly separable.
- Compared the accuracy of two major unsupervised DL algorithms: RBMs and autoencoders. The analysis shows the framework achieved an accuracy of over 99% with the integration of the autoencoder and the  $k$ -means.

## 4.8 Summary

This chapter introduced a framework for network anomaly detection. The framework employed USDL in the first phase and  $k$ -means or mean a shift in the second phase. Several related works have used deep learning to reveal anomalies in network traffic. This chapter provided a theoretical foundation for the framework implementation. This foundation defined six design principals. The design principals consider the requirements for the network anomalies identification. The small number of features in network traffic packets represents one of the obstacles for deep learning as it shines with massive data. To tackle this problem, an approach similar to kernel trick is used with autoencoders, where the inputs are projected on a higher dimension. Also, this chapter has introduced some new criteria, in place of the threshold, to distinguish the normal from the abnormal, where it is not accurate to consider the absolute reconstruction error for that distinction.

The chapter has also discussed the ways for deciding the number of hidden layers and the number of neurons at each layer. The limitations of the inputs derived from the number of features in each packet imposes represents a challenge to use deep learning algorithms.

The chapter discussed the potential of applying DL as a pre-training phase to reduce dimensionality. Dimensionality reduction is an essential step to improve the detection accuracy of network anomaly detection. The research focused on unsupervised algorithms because they are more likely to detect new threats. The study focused on autoencoders. The chapter presented a set of principles used in the design process, including dimensionality reduction and the use of reconstruction errors for the decision. Additionally, the chapter discussed the integration of the framework into the SDN networking model.

## Chapter 5: Simulation Studies

This chapter provides the implementation of the framework proposed in the previous chapter. The implementation included four scenarios—for each scenario, the first phase used a DL algorithm, either an autoencoder or an RBM, and the second phase used a clustering algorithm, either  $k$ -means or mean shift. This chapter introduces the required tools and libraries and provides an in-depth discussion of the dataset, feature selection and normalisation procedures. The chapter presents the critical code snippets required for the framework and how the code is related to workflow and algorithms provided in Chapter 4. Additionally, a detailed description is provided for training, testing and tracking the data lifecycle during the execution.

The chapter is divided into five sections. The first section introduces the simulation goals and generic descriptions of the simulation procedures. The second section outlines the various simulation scenarios. The third section presents the DL tools and various Python modules used for coding. The fourth section is an in-depth discussion of the dataset and its limitations and rationale for using it in the simulation. The fifth section, which is divided into two subsections representing each of the scenarios, maps the framework design to implementation. The final section provides a chapter summary and conclusion.

### 5.1 Simulation Overview

The proposed anomaly detection framework was based on unsupervised DL. The framework utilised USDL in a semi-supervised mode in which labelled normal traffic was passed through the framework in the training phase. During the testing phase, both normal and abnormal traffic was passed through the framework—because the framework could detect normal traffic, we could classify the other samples as abnormal.

The simulation aimed to:

- implement the proposed detection framework using a state-of-the-art DL library, Google TensorFlow
- implement an autoencoder and an RBM in the context of networking anomaly detection

- experimentally evaluate the application of USDL algorithms for network anomaly detection
- compare the accuracy of autoencoders and RBMs in different scenarios.

### 5.1.1 Simulation Scenarios

The simulation was conducted in two main scenarios based on autoencoders and RBMs, respectively. In each scenario, there were two sub-scenarios. The main scenarios involved the implementation of the algorithms for autoencoders and RBMs. The sub-scenarios used two different simple classical algorithms, *k*-means and mean shift algorithms, at the second phase for clustering. The scenarios occurred as follows:

1. Autoencoder phase followed by *k*-means clustering
2. Autoencoder phase followed by mean shift clustering
3. RBM phase followed by *k*-means clustering
4. RBM phase followed by mean shift clustering.

The purpose of using different clustering algorithms in the second phase was to ensure the accuracy of the DL algorithm results. Figure 5.1 depicts the flowchart for the first scenario involving autoencoder and *k*-means. The other scenarios had the same flow.

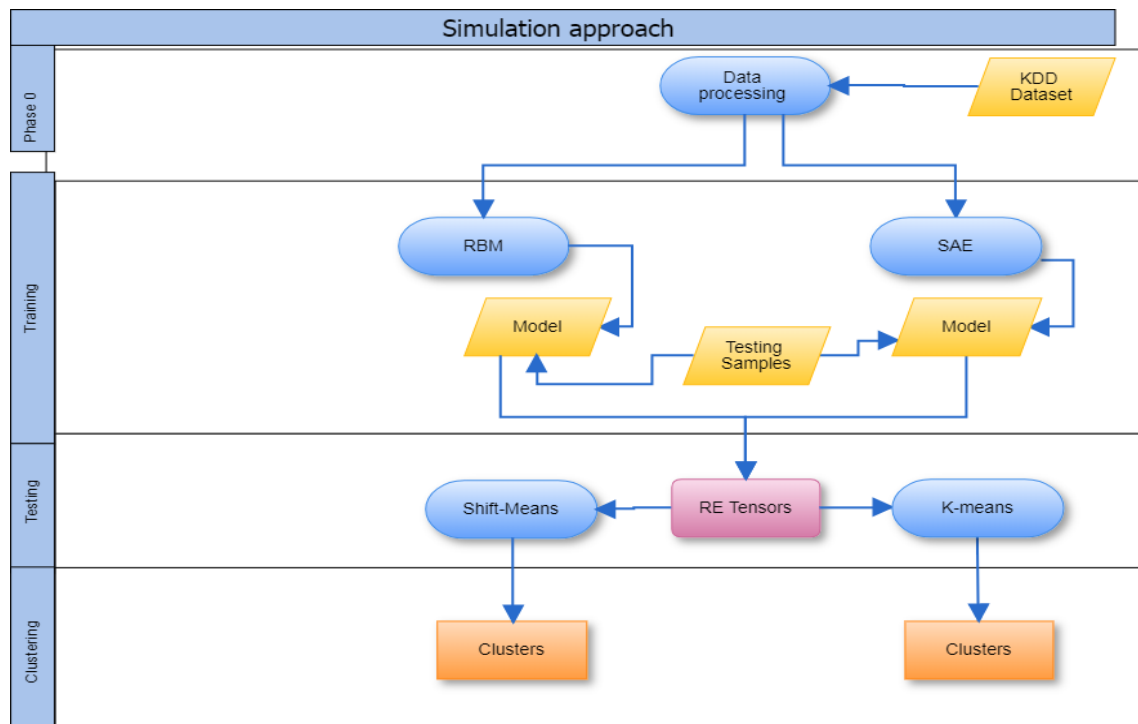


Figure 5.1. Simulation scenarios flowchart.

Each of the scenarios included the following process:

- Data normalisation: converted the dataset to a specific numerical format processable by the USDL algorithm
- Application of USDL: combined the training and testing phases as well as several sub-processes depending on the type of the algorithm; however, the necessary process included calculating weights, outputs, loss and optimisation
- Clustering of the reconstruction error (loss): included the application of a simple clustering procedure to a one-dimensional array of inputs.

## 5.2 Simulation Tools: TensorFlow and SciKit

TensorFlow was developed by Google’s Brain research team as an open-source framework for machine learning research and industrial applications [23]. It focuses on current trends in machine learning, specifically DL. TensorFlow takes its name from neural networks operations—any neural network consists of creating tensors (multidimensional arrays) for input, weights, biases and output. Computations are done in a graph model—graphs consist of nodes (operations for example activation functions and optimisation) and edges (data for example inputs and biases). In this simulation, we used TensorFlow for two reasons:

- TensorFlow, based on GitHub statistics and Stack Overflow, is the most widely used framework in DL [146].
- TensorFlow provides basic support for both algorithms: autoencoders and RBMs.

In all simulation scenarios, DL algorithms were developed using TensorFlow libraries. SciKit is the source Python library for data mining and analysis [147]. SciKit was used to implement the clustering algorithms (i.e.  $k$ -means and mean shift).

## 5.3 Dataset

KDD99 is the most widely used dataset in machine learning and intrusion detection. The dataset represents real collected network traffic data. The dataset includes 4,898,431 traffic records for training and 311,029 records for testing [148].

Table 5.1. KDD99 Input Features

Feature Name	Description	Type
Duration	Length (number of seconds) of the connection	Continuous
Protocol type	Type of protocol, e.g. TCP, UDP, etc.	Discrete
Service	Network service on the destination, e.g., HTTP, telnet, etc.	Discrete
src_bytes	Number of data bytes from source to destination	Continuous
dst_bytes	Number of data bytes from destination to source	Continuous
flag	Normal or error status of the connection	Discrete
land	1 if connection is from/to the same host/port; 0 otherwise	Discrete
wrong_fragment	Number of 'wrong' fragments	Continuous
urgent	Number of urgent packets	Continuous
hot	Number of 'hot' indicators	Continuous
num_failed_logins	Number of failed login attempts	Continuous
logged_in	1 if successfully logged in; 0 otherwise	Discrete
num_compromised	Number of 'compromised' conditions	Continuous
root_shell	1 if root shell is obtained; 0 otherwise	Discrete
su_attempted	1 if 'su root' command attempted; 0 otherwise	Discrete
num_root	Number of 'roots' accessed	Continuous
num_file_creations	Number of file creation operations	Continuous
num_shells	Number of shell prompts	Continuous
num_access_files	Number of operations on access control files	Continuous
num_outbound_cmds	Number of outbound commands in an FTP session	Continuous
is_hot_login	1 if the login belongs to the 'hot' list; 0 otherwise	Discrete
is_guest_login	1 if the login is a 'guest' login; 0 otherwise	Discrete
Count	Note: The following features refer to these same-host connections	Continuous
serror_rate	% of connections that have 'SYN' errors	Continuous
rerror_rate	% of connections that have 'REJ' errors	Continuous
same_srv_rate	% Of connections to the same service	Continuous
diff_srv_rate	% Of connections to different services	Continuous
srv_count	Number of connections to the same service as the current connection in the past two seconds	Continuous
srv_serror_rate	% of connections that have 'SYN' errors	Continuous
srv_rerror_rate	% of connections that have 'REJ' errors	Continuous
srv_diff_host_rate	% Of connections to different hosts	Continuous



The dataset contained four types of attacks:

- DoS attacks: Attackers exhaust target resources, such as computations and memory, by flooding the target host with an enormous number of requests; the victim host denies legitimate requests.
- User to root attacks: Privilege escalation attacks in which the user obtains access (usually legitimate), then escalates access to the root role, where the attacker has full access to the compromised system.
- Remote to local attacks: The attacker exploits application/system vulnerabilities to gain access to the system.
- Probing attacks: Reconnaissance attacks in which the intruder gathers information about the system, such as open ports, operating systems and various versions of protocols and applications.

*Table 5.2. KDD99 Dataset Statistics*

<b>Class</b>	<b>Training set</b>
Normal	97,278
Probe	41,102
Denial-of-service	3,883,370
Remote to local	1,126
User to root	5,252
Total	4,898,431

KDD99 is extensively used in intrusion detection research. However, it has been heavily criticised [148]. One of the significant issues associated with KDD99 is data redundancy. During the training, records were selected randomly, and testing samples were selected manually from different locations within the file to avoid redundancy. Additionally, redundancy is one of the causes of highly correlated data problems, which is discussed in the following chapter. However, the use of KDD99 was not avoidable in this research because most related work has been benchmarked to this dataset.

A sample data record is shown below:

*0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal*

Records were labelled as either normal or abnormal—these labels were used for result validation, specifically to identify the number of normal and abnormal records in each identified cluster.

The data in its original format was not calculable because the algorithms processed numerical data only; hence, it was subjected to data normalisation processing. Normalisation was conducted in two steps. The first step was to replace discreet fields with continuous fields:

```

pdic = {1:'domain_u', 2:'systat', 3:'tftp_u', 4:'link', 5:'nnsf', 6:'sql_net',7:'netbios_dgm', 8:'courier', 9:'uucp',
10:'ftp_data', 11:'time', 12:'gopher', 13:'mtp', 14:'nntp', 15:'telnet', 16:'finger', 17:'echo',18:'imap4', 19:'pop_2',
20:'other', 21:'netbios_ns', 22:'private', 23:'netstat', 24:'shell',25:'eco_i', 26:'kshell', 27:'domain', 28:'discard',
29:'efs', 30:'tim_i', 31:'ldap', 32:'hostnames', 33:'printer', 34:'supdup', 35:'pm_dump',36:'auth', 37:'IRC',
38:'iso_tsap', 39:'netbios_ssn', 40:'ntp_u', 41:'harvest', 42:'Z39_50', 43:'smtp',44:'pop_3', 45:'aol', 46:'ecr_i',
47:'csnet_ns', 48:'whois', 49:'ftp', 50:'remote_job', 51:'X11', 52:'sunrpc', 53:'urh_i', 54:'vmnet', 55:'http',
56:'urp_i',57:'rje', 58:'login', 59:'ssh', 60:'http_443', 61:'klogin', 62:'uucp_path', 63:'http_8001', 64:'ctf',
65:'daytime', 66:'name', 67:'http_2784', 68:'red_i', 69:'bgp', 70:'exec', 71:'icmp'}
rdic = {v: k for k, v in pdic.items()}
pro.append(tmp[4])
tmp[4] = rdic[tmp[4]]
sdic = {'S2':1, 'finger':2, 'X11':3, 'Z39_50':4, 'exec':5, 'courier':6, 'netstat':7, 'csnet_ns':8, 'ecr_i':9, 'private':10,
'nnsf':11, 'hostnames':12, 'iso_tsap':13, 'ntp_u':14, 'ftp_data':15, 'name':16, 'discard':17, 'uucp_path':18, 'S3':19,
'smtp':20, 'SH':21, 'RSTO50':22, 'ctf':23, 'ldap':24, 'urh_i':25, 'uucp':26, 'shell':27, 'echo':28, 'systat':29,
'http_443':30, 'red_i':31, 'urp_i':32, 'netbios_dgm':33, 'aol':34, 'pm_dump':35, 'RSTO':36, 'whois':37,
'domain_u':38, 'bgp':39, 'time':40, 'netbios_ssn':41, 'tim_i':42, 'other':43, 'pop_2':44, 'OTH':45, 'kshell':46, 'ftp':47,
'link':48, 'imap4':49, 'rje':50, 'sunrpc':51, 'RSTR':52, 'domain':53, 'harvest':54, 'REJ':55, 'supdup':56, 'http_2784':57,
'tftp_u':58, 'http_8001':59, 'SF':60, 'sql_net':61, 'vmnet':62, 'gopher':63, 'http':64, 'S0':65, 'ssh':66, 'IRC':67,
'nntp':68, 'netbios_ns':69, 'remote_job':70, 'S1':71, 'login':72, 'telnet':73, 'mtp':74, 'eco_i':75, 'efs':76, 'klogin':77,
'pop_3':78, 'daytime':79, 'printer':80, 'auth':81}
tmp[5] = sdic[tmp[5]]
pro.append(tmp[5])
tmp.pop()
ntmp = tmp[2:]
or z in range(len(ntmp)):
ntmp[z] = float(ntmp[z])
ntmp.insert(0, str(tmp[0]))
ntmp.insert(1, str(tmp[1]))
st = " ".join(str(x) for x in ntmp)
row.append(st)
print(len(row[2]))
print((row[2:10]))
print(len(row))
with open('c:\\Tsoutfile.txt', mode='wt', encoding='utf-8') as myfile:
for lines in row:
print(lines, file = myfile)
myfile.close()

```

For example, the second field service type was replaced by the following values. A Python script was written to sweep the training and testing data to find all continuous values and replace them with numeric values:

```
1.0 55.0 60.0 215.0 45076.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

The second step was to scale the entire record to values in [0, 1]—for each field, the maximum value in the entire dataset was found, then all fields were divided by the maximum value. The record below is a sample of the final normalised data:

```
0.0 0.0 0.33 0.77 0.85 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.03 0.03 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.1 0.63 1.0 0.0 0.04 0.04 0.0 0.0 0.0
```

The primary reason for scaling the data between 0 and 1 was the choice of activation function. The activation function used in the model implementation was sigmoid, which facilitated the normalisation of the input and improved the experimental results. Normalisation at this stage was performed using another Python script, which swept the entire dataset to find the largest value in each field, then divided the field in the entire dataset by the heights value as show below:

```
maxval2 = [66366.0, 3.0, 71.0, 71.0, 62825648.0, 32317698.0, 1.0, 3.0, 6.0, 233.0, 5.0, 1.0, 942.0, 1.0,
2.0, 1013.0, 100.0, 5.0, 7.0, 0.0, 1.0, 1.0, 511.0, 511.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 255.0, 255.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0]
maxval = maxlist(maxval1,maxval2)
with open('c:\\Troutfile.txt','r') as f2, open('c:\\Troutfileb.txt','w') as f :
for line in f2:
    fieldsx = line.split() # parse the columns
    fields = ieldsx[4:]
    intro = fieldsx[0:4]
    rowdata = [float(i) for i in fields] # convert text to numbers
    if len(data) = 0:
    data = rowdata
    normls = divlist(rowdata,maxval ) # accumulate the results
    rec = intro + normls
    st = " ".join([str(i) for i in rec])
    print(st, file = f)
```

## 5.4 Scenario Implementation

The first phase in the proposed framework included the unsupervised DL algorithm for dimensionality reduction. Autoencoders and RBMs were chosen for two reasons: first, unsupervised DL can be classified as two main categories, non-probabilistic models and probabilistic (generative) models. The two algorithms selected represent both categories. Additionally, both algorithms have been used in many extensions—for example, autoencoder has been used in stacked, sparse and denoising autoencoders, and RBM has been used in deep belief networks and conditional and gated RBMs. As part of the analytical study, a comparison of both algorithms in the domain of network anomaly detection was conducted. In the following subsection, the detailed implementation is demonstrated using TensorFlow and other Python libraries. The demonstration includes model building, activation and optimisation functions.

### 5.4.1 Scenarios 1 and 2

Autoencoders are a neural network with symmetric input and output layers with respect to the number of neurons. Several hidden layers are added between the input and output layers. Scenario 1 was divided into two parts: the first contained the input and number of hidden layers (encoders), and the second contained the output and the same number of the hidden layers in the encoder. The number of neurons in each layer was identical to the number in the encoder. Additionally, the final layer in the encoder was the first used for the decoder.

In this implementation, the encoder included two layers after the input layer:

```
X = tf.placeholder("float") [None, 41])
```

Placeholder is a variable that created a tensor that was subsequently populated, allowing the creation of the model without the actual data. Once the model was built, then the data were inserted into it. X is a TensorFlow tensor variable used to load the 41 features of a single IP packet from the KDD99 dataset; the float defined the valid data types to be inserted:

Encoder:

```
def encoder(x):
```

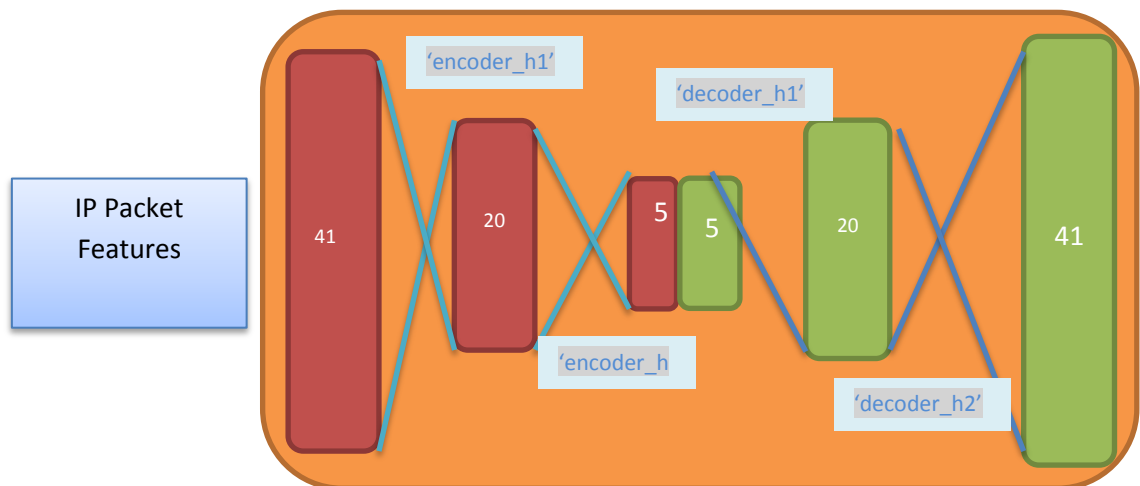
```

layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
                               biases['encoder_b1']))
# Encoder second layer with sigmoid activation #2
layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                               biases['encoder_b2']))
return layer_2

```

The autoencoder consisted of two parts: the encoder, the main goal of which was to find the most important features in the data, and the decoder, the main goal of which was to reconstruct the original data with minimum error. Each phase contained a series of latent layers, which should be symmetric around the middle layer, which is the last in the encoder and the input for the decoder. Figure 5.2 depicts the internal structure of the autoencoder—the red part represents the encoder, while the green represents the decoder.

For the implementation, we deployed two hidden layers, `n_hidden_1` and `n_hidden_2`, for 20 and five neurons, respectively. The following chapter provides a more detailed rationale for the number of neurons in hidden layers, with a comparative analysis of the results of different implementations.



*Figure 5.2. The internal structure of the autoencoder intrusion detection system.*

The second hidden layer contained five neurons, which represented the most significant features in the data.  $W$  represents the weight tensor, which connected the input layer with the first layer and biases for the first layer. In this implementation, there were four

weight tensors linking each layer to the following shown in Figure 5.2: ‘encoder\_h1’, ‘encoder\_b2’, ‘decoder\_h1’ and ‘decoder\_h2’. The dimension of each tensor was calculated by multiplying the number of neurons in each layer; for instance, ‘encoder\_h1’ = 41\*20, with each value representing the connection weight between two corresponding nodes. Similar to weights, bias tensors were created for each layer: ‘encoder\_b1’, ‘encoder\_b2’, ‘decoder\_b1’ and ‘decoder\_b2’.

The encoder used sigmoid as the activation function. Prior to calculating the activation value, the tensors of two layers were multiplied using `tf.matmul()`, biases were added using `tf.add()`, then outputs were activation using `tf.nn.sigmoid()`. Once the output layer was calculated, the cost or information loss was calculated by measuring the difference between the input layer and the output layer: `tf.reduce_mean(tf.square(input,output))`.

```
n_hidden_1 = 20
```

```
n_hidden_2 = 5
```

Two weight variables for the two layers and two bias tensors were present.

```
Weights = {
```

```
    'encoder_h1': tf.Variable(tf.random_normal([n_input, n_hidden_1]))
```

```
    'encoder_h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2]))
```

```
    'decoder_h1': tf.Variable(tf.random_normal([n_hidden_2, n_hidden_1]))
```

```
    'decoder_h2': tf.Variable(tf.random_normal([n_hidden_1, n_input]))
```

```
Biases = {
```

```
    'encoder_b1': tf.Variable(tf.random_normal([n_hidden_1]))
```

```
    'encoder_b2': tf.Variable(tf.random_normal([n_hidden_2]))
```

```
    'decoder_b1': tf.Variable(tf.random_normal([n_hidden_1]))
```

```
    'decoder_b2': tf.Variable(tf.random_normal([n_input]))
```

The decoder phase used Layer 2 from the encoder as the input and reconstructed the output in two layers: the first layer contained 20 neurons and the second (final) layer was the output layer. Hence, weights and biases were updated when the Adam optimiser was used. The Adam optimiser is an efficient version of the SGD optimiser.

The model optimised the cost using `tf.train.AdamOptimizer(1e-1).minimize(cost)`

```
def decoder(x):
```

```
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1'])))
```

```

        biases['decoder_b1']))
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2'])
        biases['decoder_b2']))
    return layer_2
cost = tf.reduce_mean(tf.square(y_true - y_pred))
optimiser = tf.train.AdamOptimizer(1e-1).minimize(cost)

```

### 5.4.2 Training

Once the model was built, the next step was to load the data, divide the data into batches and then run the session. Data loading was done in two steps: the first step was to read the data from the KDD99 dataset file and store it in an array using the following code:

```

reader_train = create_reader(train_file, True, input_dim, 2)
print (type(reader_train))
print ("all")
packet = C.input_variable(input_dim)
packet_label = C.input_variable(2)
batch_size = 400000
viz_input_map = {
    packet: reader_train.streams.features
    packet_label : reader_train.streams.labels_viz
viz_data = reader_train.next_minibatch(batch_size
    input_map = viz_input_map)
flow_data = viz_data[packet].asarray()
flow_type = viz_data[packet_label].asarray()
trax = np.array (flow_data )
Xtrain = []
Ytrain = []
Yhashed = []
for i in trax:
    elem = i[0].tolist()
    Xtrain.append(elem)
Xtrain = np.asarray(Xtrain).astype('float32')

```

Each record of the 400 k element was converted to an array and added to the training data array, Xtrain. The data were divided into batches (total\_batches) before being fed into the model:

```

sess.run([optimizer, cost], feed_dict={X: batch_xs}
for i in range(total_batch):

```

```

batch_xs = Xtrain[i*batch_size:(i*batch_size + batch_size)]
_, c = sess.run([optimizer, cost], feed_dict={X: batch_xs})

```

Once the model was established by calculating optimal weights and biases, a block of the testing data was loaded from KDD99 and passed through the autoencoder:

```

reader_test = create_reader(test_file, True, input_dim, 2)
print (type(reader_test))
print ("all")
packet = C.input_variable(input_dim)
packet_label = C.input_variable(2)
batch_size = 1300
viz_input_map_t = {
    packet: reader_test.streams.features,
    packet_label: reader_test.streams.labels_viz
}
viz_data_t = reader_test.next_minibatch(batch_size, input_map = viz_input_map_t)
flow_data_t = viz_data_t[packet].asarray()
flow_type_t = viz_data_t[packet_label].asarray()
tray = np.array (flow_data_t )
Ytrain = []
Ltst= []
ltest = np.array (flow_type_t )
for i in tray:
    elem = i[0].tolist()
    Ytrain.append(elem)
for i in ltest:
    elem = i[0].tolist()
    Ltst.append(elem)
Ytrain = np.asarray(Ytrain).astype('float32')
Ltst = np.asarray(Ltst).astype('float32')

```

```

encode_decode = sess.run(y_pred, feed_dict={X: Ytrain2})
for i in range(len(encode_decode)):
err2.append(sess.run(tf.reduce_mean(tf.squared_difference(encode_decode[i],
Ytrain2[i]))))

```

The testing samples were inserted in the TensorFlow graph for encoding and decoding: `sess.run(y_pred, feed_dict={X: Ytrain2})`. The test data were reconstructed using the autoencoder, then reconstruction errors were calculated using `tf.squared_difference(encode_decode[i], Ytrain2[i])` for each record in the testing batch. For instance, in the previous code, 1.3 k samples were loaded; by the end of the



processing of the first phase, there were 1.3 k reconstruction errors. These reconstruction errors were forwarded to the next phase. In this scenario, a *k*-means algorithm was implemented using the following code:

```
km = KMeans()
cl = km.fit(npar.reshape(-1,1))
print(cl.labels_)
print (type(cl.labels_))
kcls = [(x,y) for x , y in zip (tst , cl.labels_ )

gcl = list(set(cl.labels_))

print(gcl)
groups = []
for c in gcl:
    clx = [i for i,x in enumerate(cl.labels_) if x == c]
    groups.append(clx)
```

*k*-means is a class in the SciKit library. The fit () function took the reconstruction error array as input and calculated the labels for each sample. Once the *k*-means labels were calculated, a verification code was sent to link the original labels (normal or abnormal) with the labels.

In the second scenario, the *k*-means phase was replaced with mean shift. The mean shift used in this scenario was a flat-based kernel similar to PCA. However, the reconstruction errors were simple vectors where the samples were separable; therefore, using the kernels would have complicated the computation.

```
kclusters = []
for i in groups:
    cla = [list(tst[x]) for x in i ]
    kclusters.append(cla)
for i in kclusters:
    nrm = 0
    abnr=0
    for x in i
    if (x[0] == 1.0) and (x[1] == 0.0):
        nrm= nrm + 1
    otherwise:
        abnr = abnr + 1
    print(" KCluster normal")
    print(" KCluster abnormal")
```

The code below depicts the implementation of the mean shift. The mean shift is a centroid-based algorithm, with the main goal to find the modes in smooth data, to calculate the probability distribution function.

```
Z = np.array(list(zip(err,np.zeros(len(err)))))
ms = MeanShift( bin_seeding=True)
ms.fit(Z)
labels = ms.labels_
cluster_centers = ms.cluster_centers_
labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)
mycls= []
for k in range(n_clusters_):
    my_members = labels == k
    #print ("cluster {0}: {1}".format(k, X[my_members, 0]))
    mycls.append(list(Z[my_members, 0]))
```

### 5.4.3 Scenarios 3 and 4

An RBM network consists of two layers: the input layer, which is visible, and a hidden layer. Each record in the KDD99 dataset has 41 features. Hence, the RBM visible layer has the same number of input neurons.

The hidden layer possesses n neurons. Each hidden unit has a binary state, which we call sn, and it turns either 0 or 1 with a probability that is a sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

where x is the visible neuron (v0), corresponding weight (w) and hidden bias (hb).

```
tf.nn.sigmoid(tf.matmul(v0, W) + hb) # calculating sn for hidden units probabilities of the hidden units.
```

Each neuron in the visible layer (v) also has a bias (vb). W is a matrix representing the weights between the input layer and hidden layer nodes. In the weight matrix, the rows are equal to the visible nodes and the columns are equal to the hidden nodes. Let W be the tensor of 41\*82, where 82 is the number of neurons in the hidden layers:

```
vb = tf.placeholder ("float", [41])           # visible layer biases
hb = tf.placeholder("float", [41])           #Hidden Layer biases
```

```

W = tf.placeholder("float", [41,41])           # weights matrix
v0 = tf.placeholder("float", [None, 41])      # visible layer tensor

```

Inputs were combined with individual weights and bias. Some hidden nodes were activated.

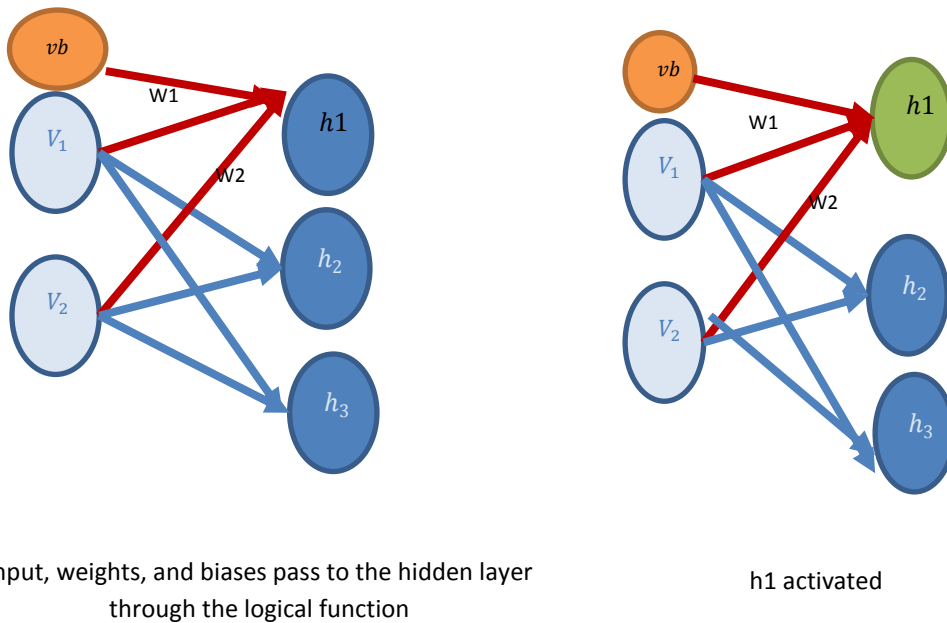


Figure 5.3. Restricted Boltzmann machine neuron activation.

```

_h0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)
h0 = tf.nn.relu(tf.sign(_h0 - tf.random_uniform(tf.shape(_h0))))

```

Figure 5.3 show the activation of RBM model, the activations were combined with individual weights and a bias. Results were passed to the visible layer. The RBM reconstructed data by making several forward and backward passes between the visible and hidden layers as shown in figure 5.4.

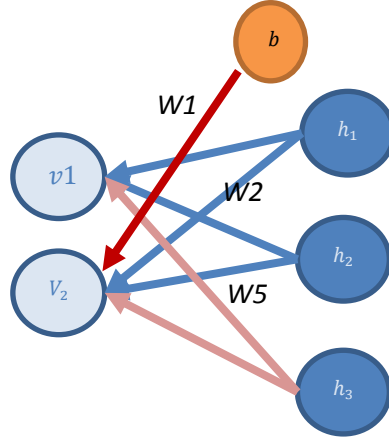


Figure 5.4. Reconstruction phase.

```

_v1 = tf.nn.sigmoid(tf.matmul(h0, tf.transpose(W)) + vb)
v1 = tf.nn.relu(tf.sign(_v1 - tf.random_uniform(tf.shape(_v1)))) #sample_v_given_h
hTensor1 = tf.nn.sigmoid(tf.matmul(v1, W) + hb)

```

Samples from probabilistic tensor were selected as  $h_0$ , which represents the input in the reconstruction phase. The same weight matrix and visible layer biases were used for the sigmoid function. The produced output was a reconstruction, which approximated the original input.

Energy-based probabilistic models define a probability distribution as:

$$p(v) = \frac{e^{-E(v,h)}}{Z} \text{ where } Z = \sum_{v,h} e^{-E(v,h)} \quad (23)$$

$$p(v) = \sum_h P(v, h) = \frac{e^{-E(v,h)}}{Z = \sum_{v,h} e^{-E(v,h)}} \quad (24)$$

An energy-based model can be learned by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data, where the log-likelihood and the loss function are:

$$L(\theta, V) = \frac{1}{N} \sum_{v^i \in V} \log p(v^i) \quad \text{and} \quad l(\theta, V) = -L(\theta, V) \quad (25)$$

Then the data negative log-likelihood gradient has the following form:

$$-\frac{d \log p(v)}{d\theta} = \frac{dP(v)}{d\theta} - \sum_{\tilde{v}} p(\tilde{v}) \frac{dP(\tilde{v})}{d\theta} \quad (26)$$

$$-\frac{d \log p(v)}{d\theta} = \frac{dP(v)}{d\theta} - \frac{1}{|N|} \sum_{\tilde{v} \in N} p(\tilde{v}) \frac{dP(\tilde{v})}{d\theta} \quad (27)$$

where  $v'$  is a sample of  $N$ .

To minimise loss, we must maximise the product of probabilities assigned to the training set  $\frac{dP(v)}{d\theta}$

$$P(v) = -b'v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)} \quad (28)$$

where  $E(v, h) = -b'v - c'h - h'Wv$  and  $b'$  and  $c$  are biases and hidden layers, respectively.

From (27) and (28):

$$-\frac{d \log(p(v))}{dW_{ij}} = E_v[p(h_i|v) \cdot v_j] - v_j^{(i)} \cdot \sigma(W_i \cdot v^{(i)} + c_i) \quad (29)$$

Equation (29) defines the loss function as the average negative log-likelihood and the objective was to minimise it. To achieve this, we needed the partial derivative of this function with respect to its parameters. From Equation (29), optimisation or minimisation of loss depended on adjusting the weights ( $W$ ) and biases ( $C$ ). SGD was used to find the optimal  $W$  tensor.

The derivation has two terms: the positive term  $E_v[p(h_i|v) \cdot v_j]$ , which depends on the data  $V$ , increases the probability of inputs. The second term is a negative term,  $-v_j^{(i)} \cdot \sigma(W_i \cdot v^{(i)} + c_i)$ , which depends on the model and decreases the probability of the output generated by the model.

```

vb = tf.placeholder("float", [41])
hb = tf.placeholder("float", [41])
W = tf.placeholder("float", [41,41])
v0 = tf.placeholder("float", [None, 41])
_h0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb) #probabilities of the hidden units
h0 = tf.nn.sigmoid(tf.sign(_h0 - tf.random_uniform(tf.shape(_h0))))
_v1 = tf.nn.sigmoid(tf.matmul(h0, tf.transpose(W)) + vb)
v1 = tf.nn.sigmoid(tf.sign(_v1 - tf.random_uniform(tf.shape(_v1))))
h1 = tf.nn.sigmoid(tf.matmul(v1, W) + hb)
alpha = 1.0
w_pos_grad = tf.matmul(tf.transpose(v0), h0)
w_neg_grad = tf.matmul(tf.transpose(v1), h1)
CD = (w_pos_grad - w_neg_grad) / tf.to_float(tf.shape(v0)[0])
update_w = W + alpha * CD
update_vb = vb + alpha * tf.reduce_mean(v0 - v1, 0)
update_hb = hb + alpha * tf.reduce_mean(h0 - h1, 0)
err = tf.reduce_mean(tf.square(v0 - v1))
cur_w = np.zeros([41, 41], np.float32)
cur_vb = np.zeros([41], np.float32)
cur_hb = np.zeros([41], np.float32)
prv_w = np.zeros([41, 41], np.float32)
prv_vb = np.zeros([41], np.float32)
prv_hb = np.zeros([41], np.float32)
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
idsys = sess.run(err, feed_dict={v0: Xtrain, W: prv_w, vb: prv_vb, hb: prv_hb})
print (idsys)
epochs = 10
batchsize = 300
weights = []
errors = []
rbf_feature = Nystroem(kernel='rbf', gamma=1, n_components=41, random_state=1)
k = rbf_feature.fit_transform(Xtrain)
#k=Xtrain
for epoch in range(epochs):
    print (epoch)
    for start, end in zip( range(0, 400000, batchsize), range(batchsize, 400000, batchsize)):
        batch = k[start:end]
        print (start)
        cur_w = sess.run(update_w, feed_dict={v0: batch, W: prv_w, vb: prv_vb, hb: prv_hb})
        cur_vb = sess.run(update_vb, feed_dict={v0: batch, W: prv_w, vb: prv_vb, hb: prv_hb})
        cur_hb = sess.run(update_hb, feed_dict={v0: batch, W: prv_w, vb: prv_vb, hb: prv_hb})
        prv_w = cur_w
        prv_vb = cur_vb
        prv_hb = cur_hb
        print (cur_w)
        errors.append(sess.run(err, feed_dict={v0: k, W: cur_w, vb: cur_vb, hb: cur_hb}))
        weights.append(cur_w)
        print (errors[-1])p
    print ('Epoch: %d' % epoch, 'reconstruction error: %f' % errors[-1])

```

The positive phase increased the probability of training data. The negative phase decreased the probability of samples generated by the model.

Given that the negative phase was difficult to compute, the CD was used to approximate it. It was designed in such a way that the direction of the gradient estimate was at least somewhat accurate, even when the size was not (in real-world models, more accurate techniques such as CD-k or PCD are used to train RBMs). During the calculation of CD, we used Gibbs sampling to sample from our model distribution.

CD is a matrix of values that were computed and used to adjust the values of the  $W$  matrix. Changing  $W$  led to the training of  $W$  values. Subsequently, for each step (epoch),  $W$  was updated to a new value ( $W'$ ) using the equation below:

$$W' = W + \alpha * \text{CD} \quad (30)$$

where  $\alpha$  is the learning rate that adjusts the model to respond to the cost.

To compute the CD, a training sample from  $X$  was selected to calculate the probabilities of the hidden units and sample a hidden activation vector ( $h_0$ ) from this probability distribution.

- $h_0 = \text{sigmoid}(X \otimes W + hb)$
- $h_0 = \text{sampleProb}(h_0)$

1. calculate the outer product of  $X$  and  $h_0$  and call this the positive gradient
2.  $w_{\text{pos\_grad}} = X \otimes h_0$
3. From  $h$ , reconstruct  $v_1$ , take a sample of the visible units, then resample the hidden activations  $h_1$  from this (Gibbs sampling step)
4.  $v_1 = \text{sigmoid}(h_0 \otimes \text{transpose}(W) + vb)$
5.  $v_1 = \text{sampleprob}(v_1)$  (Sample  $v$  given  $h$ )
6.  $h_1 = \text{sigmoid}(v_1 \otimes W + hb)$
7. calculate the outer product of  $v_1$  and  $h_1$  and call this the negative gradient
8.  $w_{\text{neg\_grad}} = v_1 \otimes h_1$  (reconstruction 1)
9. Now, CD equals the positive gradient minus the negative gradient
10.  $\text{CD} = (w_{\text{pos\_grad}} - w_{\text{neg\_grad}})$
11. Update the weight to be CD times some learning rate
12.  $W' = W + \alpha * \text{CD}$

13. At the end of the algorithm, the visible nodes will store the value of the sample.

Then, the tensor of probabilities was selected (from a sigmoidal activation) and samples were made from all distributions ( $h_0$ ). Hence, the sampling for the activation vector from the probability distribution of hidden layer values was computed. Samples were used to estimate the negative phase gradient.

The second phase implements the K-means and mean shift algorithms, to cluster the reconstruction errors produced by the RBM model as below.

```
npar= np.array(rcerrs)
km = KMeans(n_clusters=8)
cl= km.fit(npar.reshape(-1,1))
print(cl.labels_)
print (type(cl.labels_))
kcls= [(x,y) for x , y in zip (tst , cl.labels_ )]

gcl = list(set(cl.labels_))

print(gcl)
groups = []
for c in gcl:
    clx= [i for i,x in enumerate(cl.labels_) if x == c]
    groups.append(clx)

kclusters = []
for i in groups:

    cla = [list(tst[x]) for x in i ]
    kclusters.append(cla)

for i in kclusters:
    nrm = 0
    abnr=0
    for x in i :
        if (x[0]== 1.0) and (x[1]== 0.0):

            nrm= nrm + 1

        else:
            abnr = abnr + 1
    print(" KCluster normal")
    print(nrm)
    print(" KCluster abnormal")
    print(abnr)
```



## **5.5 Conclusion**

This chapter has discussed the implementation of the proposed framework. The implementation was conducted for four scenarios. Four algorithms were implemented autoencoder, RBM, k-means, mean shift. Then these algorithms were integrated together, where the first phase is either autoencoder or an RBM, then the second phase is k-means or mean shift. These four scenarios were implemented to ensure the inclusiveness of results. Tensorflow deep learning framework from Google and Scikit Python library were during the development. KDD99 dataset is used during the execution of training and testing steps. In the next chapter, the results are collected, analysed and evaluated.

## Chapter 6: Results, Analysis and Evaluation

DL has been used in several anomaly detection applications, including network anomalies. The framework was trained using a networking traffic dataset before samples of different sizes were passed through the system for testing. Statistical results were presented using a confusion matrix to measure several aspects of the framework, including accuracy and precision. This chapter provides a comparative analysis of results.

To demonstrate the contribution of this thesis, an in-depth analysis of the results is presented. The confusion matrix is a standard statistical tool used to evaluate the performance of machine learning predictors. It was used in this analysis to compare the performance of the framework with similar approaches used by other researchers. An essential component of the analysis was to demonstrate correlations between the theoretical design principles discussed in Chapter 4 and the results summarised in this chapter.

The first section introduces the analysis goals and methods. The second section demonstrates the system in the execution (e.g. training and testing outputs for various implementation scenarios discussed in the previous chapter). Additionally, the results collected during various stages of execution are collected and organised. The third section discusses design principles and how they affected implementation. The fourth section provides a detailed analysis of the results. The fifth section presents an evaluation of the framework results compared with other similar proposed frameworks.

### 6.1 Introduction

DL-based anomaly detection has been explored in several papers on network anomaly detection. A typical approach adopted by researchers is to use unsupervised DL as a pre-processing step to finding patterns in the data before forwarding the output layer from the neural network to a second classification or clustering algorithm. The proposed framework presented in this thesis adopted a similar approach but had different goals, a unique technical implementation and better accuracy and precision metrics. The primary purpose for using DL was to reduce the dimensionality of input data, making the second phase (clustering/classification) more straightforward in terms of computation resources

and time and providing higher accuracy in clustering. In its implementation, the framework simplified the output from the first phase/input to a single value for each network traffic record in the second phase, unlike similar approaches, which have used the output from the DL algorithm or, in some cases, hidden layers. Additionally, during the implementation, the problem of highly correlated data was managed by increasing the features of the first hidden layers, then reducing them in the subsequent layers.

The implementation described in the previous chapter is examined by using several samples of different sizes. Different results were collected during successive execution steps; the output was recorded, then analysed using the confusion matrix. A confusion matrix provided a set of measurement tools such as accuracy, precision and an F1 score. The statistical measurements were applied to data collected in different scenarios described in the previous chapter. There were two primary goals for the analysis process. The first was to compare the performance of the autoencoder and the RBM in network anomaly detection to confirm and validate results, with two different algorithms used in the second phase. Second, analysis statistics were used to compare the framework performance against other related works.

## 6.2 Results

```
Epoch: 0001 cost= 0.027297018
Epoch: 0002 cost= 0.027274711
Epoch: 0003 cost= 0.027273355
Epoch: 0004 cost= 0.027272990
Epoch: 0005 cost= 0.027274150
Epoch: 0006 cost= 0.027274083
Epoch: 0007 cost= 0.027274333
Epoch: 0008 cost= 0.027273895
Epoch: 0009 cost= 0.027274128
Epoch: 0010 cost= 0.030739360
Epoch: 0011 cost= 0.015888004
Epoch: 0012 cost= 0.012710391
Epoch: 0013 cost= 0.027804116
Epoch: 0014 cost= 0.027811782
Epoch: 0015 cost= 0.027810751
Epoch: 0016 cost= 0.027810829
Epoch: 0017 cost= 0.027407454
Epoch: 0018 cost= 0.027407652
Epoch: 0019 cost= 0.027407601
Epoch: 0020 cost= 0.027407428
```

This section provides the results at various stages of the simulation. During the training phase, the decrease in reconstruction error produced by using optimisers indicated algorithm convergence. Convergence can be shown on two levels, the epoch and the batches. The results provided below are samples of the cost over 20 sweeps of the training samples. Notably, the cost decreased smoothly from the first epoch to the last.

Figure 6.1 shows the performance of two different optimisers: SGD and Adam.

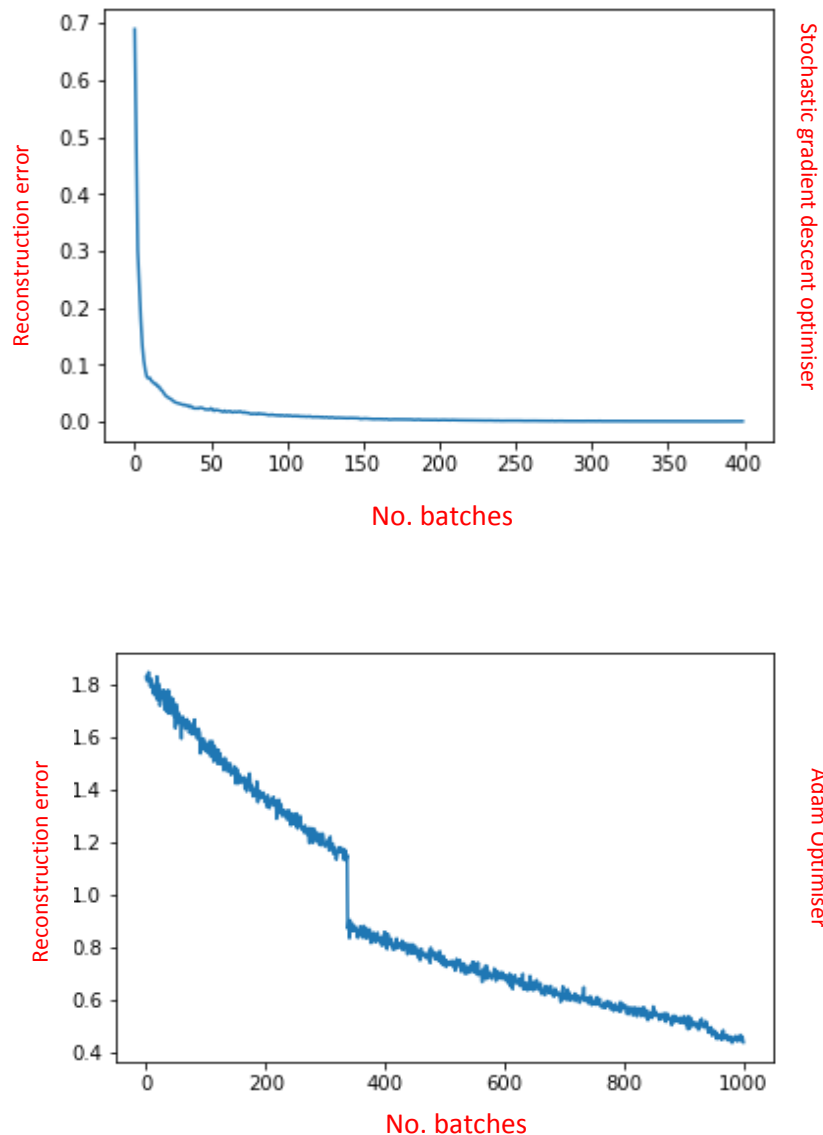


Figure 6.1. Optimisation using stochastic gradient descent and Adam optimiser.

Because the model was constructed using required weights and biases and put through several iterations to optimise the cost of previous diagrams and to confirm the values,

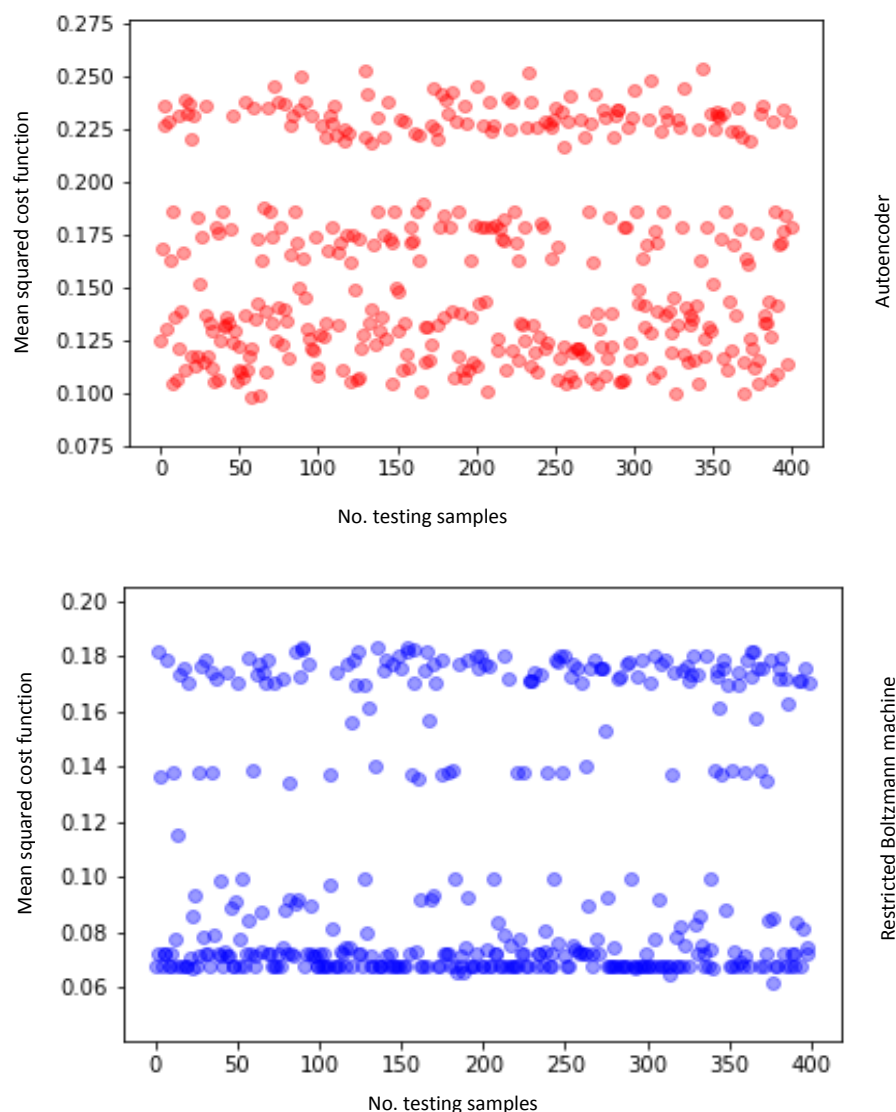
the next step was to subject the system to a testing phase. A sequence of traffic records, shown below, were loaded into the model.

```
|labels 1 0 |features 0.0 0.0 0.33 0.77 0.85 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.01 0.0 0.0 0.03  
|labels 1 0 |features 0.0 0.0 0.33 0.77 0.85 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  
0.0 0.02 0.02 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.01  
|labels 0 1 |features 0.0 0.0 1.0 0.65 0.85 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  
0.0 0.94 0.94 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0  
|labels 0 1 |features 0.0 0.0 1.0 0.65 0.85 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  
0.0 1.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0
```

As discussed in the implementation chapter, the testing data contained normal and abnormal traffic packets for which the records were normalised. Labels were not used by the system during the clustering; rather, they were used to validate and evaluate the accuracy achieved. The model provides a single value representing the reconstruction error for each of the testing records, as shown below.

```
[0.087900057, 0.12927638, 0.0034244901, 0.00090605248, 0.092122331, 0.011742176, 0.12852195,  
0.067880958, 0.15653908, 0.09629482, 0.077442065, 0.08034879, 0.002997661, 0.099123523,  
0.13377792, 0.0036739921, 0.077225119, 0.0017729657, 0.0021743756, 0.00093818858, 0.082389377,  
0.0020744232, 0.082912229, 0.14693747, 0.076985508, 0.10832905, 0.13633011, 0.075034223,  
0.097027674, 0.010716964, 0.082746595, 0.09506125, 0.077529229, 0.091491975, 0.077032238,  
0.1488664, 0.14351323, 0.07653933, 0.084063888, 0.15645327, 0.095991701, 0.093754239,  
0.095752604, 0.094319329, 0.14569174, 0.00071955862, 0.082940266, 0.091592737, 0.076763138,  
0.081645504, 0.075114369, 0.071573846
```

In the testing phase, both the autoencoder and the RBM produced a list of reconstruction errors. Figure 6.2 illustrates the visual distribution pattern of reconstruction errors. The proposed system considered the clustering of these reconstruction errors. Hence, anomalies converged in the same clusters. Of note in the previous distribution is that values were not linearly separable. Hence, a simple regression algorithm was not applicable.



*Figure 6.2. Reconstruction error distributions for autoencoder and restricted Boltamann machine.*

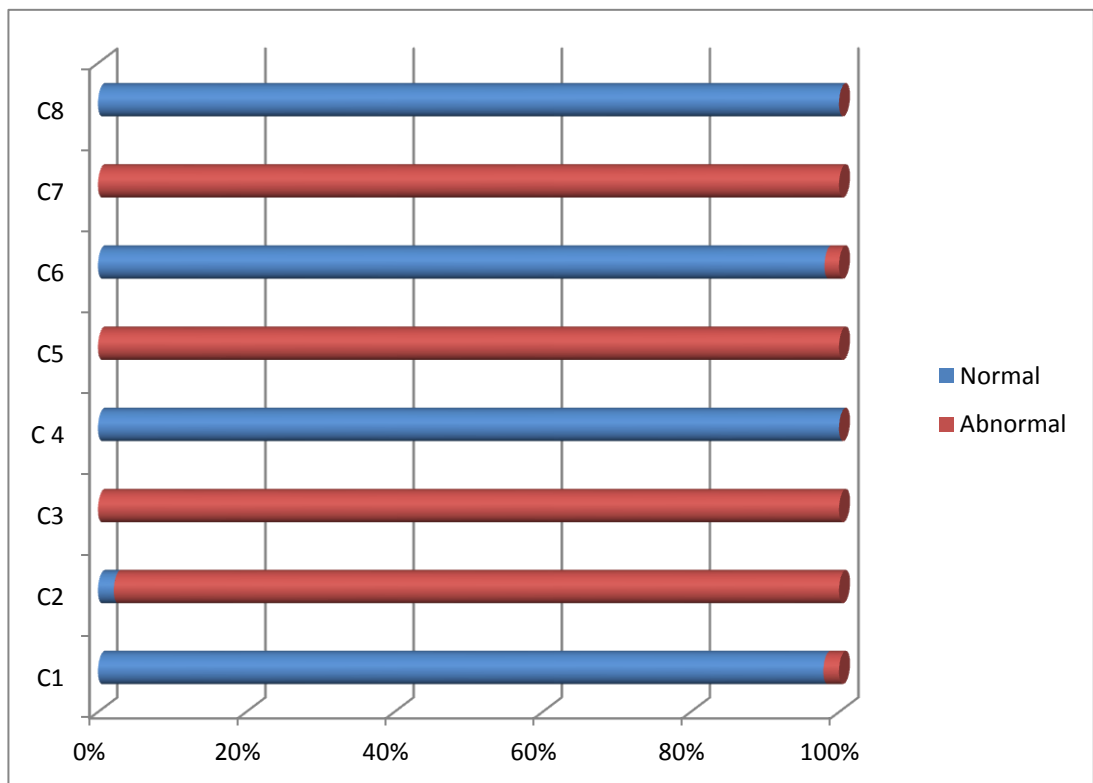
Additionally, given the nonlinearity of the distribution, defining a threshold for definite abnormality was not practically feasible. Hence, the next phase of the detection system was to find patterns automatically in the distribution using the unsupervised approach. The algorithm in the second phase was simple because the required task involved clustering for single value inputs.

$k$ -means algorithms take a vector of reconstruction errors as the input and produce a set of clusters, with each cluster consisting of a set of reconstruction errors. Each value indicates a record that is normal or abnormal. For example, the bar chart in Figure 6.3

below illustrates the identified clusters, with each bar representing a single cluster. In a perfect situation, a cluster will contain either normal or abnormal samples; however, given the relatively small accuracy error—to be discussed in the analysis and evaluation sections—some clusters may contain both but have dominance for one type, marking them as either normal or abnormal. In the tables and diagrams below, the mean shift appears to provide better accuracy; however, in the full analysis, *k*-means was superior.

*Table 6.1. k-Means Cluster Contents*

	Normal	Abnormal
C1	139	3
C2	4	183
C3	0	302
C4	172	0
C5	0	150
C6	100	2
C7	0	61
C8	184	0



*Figure 6.3. k-means graphical representation for clusters.*

Table 6.2. Mean Shift Clusters

	Normal	Abnormal
C1	584	1
C2	0	324
C3	15	335
C4	0	38
C5	0	3

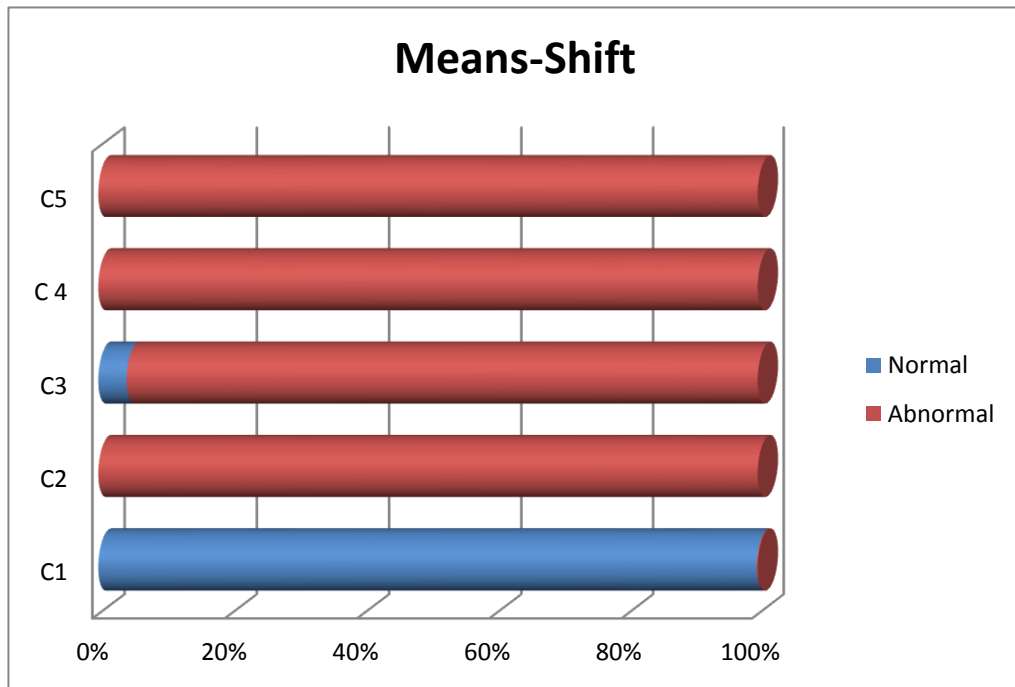


Figure 6.4. Graphical representation of mean shift clusters

Figure 6.5 shows the distribution of clusters produced by  $k$ -means. The vertical axis represents reconstruction errors, while the horizontal axis represents clusters. The distribution demonstrates the nonlinearity of samples where the applicability of thresholds or regression tasks was not applicable.



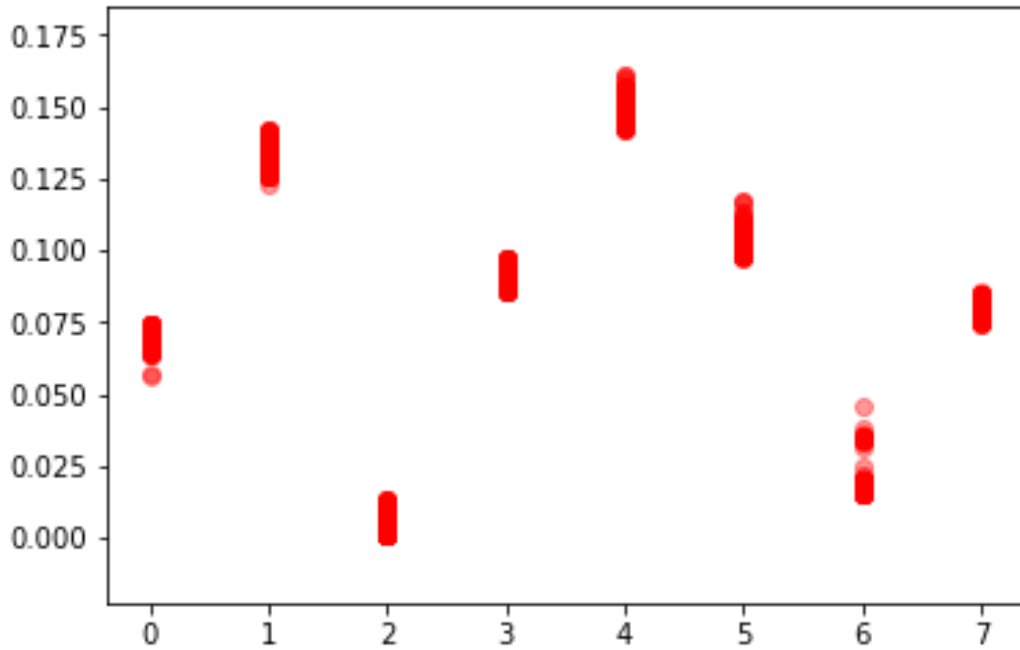


Figure 6.5. *k*-means cluster distribution.

Tables 6.3 and 6.4 summarise the results collected for the autoencoder and RBM with the integration of *k*-means and mean shift for different sample sizes of 1.3 k and 800,400 records. For *k*-means, experimental trials were used to decide on cluster numbers with fewer sampling errors. In contrast, the mean shift did not require a pre-determined number of clusters. Hence, the number of clusters varied for each algorithm and there was more focus on the performance of the algorithm. Additionally, the samples were selected randomly from the KDD99 testing dataset.

Table 6.3. Autoencoder Simulation Scenario Results Summary

		Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8
AE 1,300 <i>k</i> -means	Normal	139	4	0	172	0	100	0	184
	Abnormal	3	183	302	0	150	2	61	0
AE 1,300 mean shift	Normal	584	0	15	0	0			
	Abnormal	1	324	335	38	3			
AE 800 <i>k</i> -means	Normal	54	0	0	112	87	0	127	0
	Abnormal	1	193	92	0	0	89	0	47
AE 800 mean shift	Normal	0	378	2					
	Abnormal	298	1	121					
AE 400 <i>k</i> -means	Normal	0	59	0	28	55	0	0	54
	Abnormal	47	0	72	0	0	43	42	0
AE 400 mean shift	Normal	192	0	4					
	Abnormal	0	114	90					

Table 6.4. Restricted Boltzmann Machine Simulation Scenario Results Summary

		<b>Cluster 1</b>	<b>Cluster 2</b>	<b>Cluster 3</b>	<b>Cluster 4</b>	<b>Cluster 5</b>	<b>Cluster 6</b>	<b>Cluster 7</b>	<b>Cluster 8</b>
RBM 1,300 <i>k</i> -means	Normal	1	27	261	1	1	0	82	130
	Abnormal	413	154	0	0	0	230	0	0
RBM 1,300 mean shift	Normal	0	26	253	115	80	26	1	2
	Abnormal	413	384	0	0	0	0	0	0
RBM 800 <i>k</i> -means	Normal	0	14	1	87	1	159	52	0
	Abnormal	253	120	0	0	0	0	0	113
RBM 800 mean shift	Normal	0	14	154	92	52	1	1	
	Abnormal	253	233	0	0	0	0	0	
RBM 400 <i>k</i> -means	Normal	0	92	25	10	6	0	29	7
	Abnormal	62	0	0	0	119	50	0	0
RBM 400 mean shift	Normal	0	4	87	41	25	4	8	
	Abnormal	119	111	0	0	1	0	0	

Note. RBM: Restricted Boltzmann machine

## 6.3 Analysis

The confusion matrix is a standard statistical tool to measure the performance of machine learning classifiers. To apply the confusion matrix as an evaluation tool, the clustering process is considered a classifier with no prior knowledge about available classes; however, the overall goal of clustering is to classify similar samples together. Hence, the difference between the classifier and the clustering algorithms lies in the starting point in which supervised classifiers have prior knowledge of classes. However, in essence, both classify samples to a specific group (i.e. a cluster or class). Therefore, the framework performed the clustering and normal or abnormal labels arising from the dataset were used in conjunction with the confusion matrix for the analysis. The confusion matrix shows the measurements illustrated in Table 6.5 below.

*Table 6.5. Confusion Matrix Statistics*

<b>Sensitivity (recall)</b> True positive rate (TPR)	$TPR = TP/(TP+FN)$
<b>Specificity</b> Specificity (SPC) or true negative rate (TNR)	$SPC = TN/(FP+TN)$
<b>Precision</b> Precision or positive predictive value (PPV)	$PPV = TP/(TP+FP)$
<b>Negative predictive value</b> Negative predictive value (NPV)	$NPV = TN/(TN+FN)$
<b>False-positive rate</b> Fall-out or false-positive rate (FPR)	$FPR = FP/(FP+TN)$
<b>False-discovery rate</b> False discovery rate (FDR)	$FDR = FP/(FP+TP)$
<b>False-negative rate</b> Miss rate or false-negative rate (FNR)	$FNR = FN/(FN+TP)$
<b>Accuracy</b> Accuracy (ACC)	$ACC = (TP+TN)/(P+N)$
<b>F1 score</b> F1 score (F1)	$F1 = 2TP/(2TP+FP+FN)$

*Note:* True positive (TP): correctly predicts a normal label as normal; true negative (TN): correctly predicts an abnormal label as abnormal; false positive (FP): incorrectly predicts an abnormal label as normal; false negative (FN): incorrectly predicts a normal label as abnormal; precision or positive predictive values (PPV): the proportion of predicted positive cases that were correct; accuracy (ACC): the proportion of total number of correct predictions; F1 score: considers the balance between precision and sensitivity or recall (i.e. the weighted average of sensitivity and precision).

The analysis process considered various metrics so that comparative studies would be comprehensive.

The framework predicted three different samples of different sizes (400, 800 and 1,300). Table 6.6 presents the statistical results for the autoencoder and RBM used in conjunction with  $k$ -means. Accuracy represents how often the framework was correct. The highest accuracy was achieved by the autoencoder using the lowest number of samples (400). Unexpectedly, accuracy declined as the number of samples increased. Additionally, the autoencoder achieved the best accuracy in conjunction with the mean shift algorithm (presented in Table 6.7). The F1 score represents precision (true positive results/total true positives by the framework) and recall (number of true positive results in the total sample). The autoencoder resulted in the highest F1 values, shown in Tables 6.6 and 6.7.

*Table 6.6. Confusion Matrix for Autoencoder and Restricted Boltzmann Machine +  $k$ -Means*

	<b>1,300 <math>k</math>-means</b>		<b>800 <math>k</math>-means</b>		<b>400 <math>k</math>-means</b>	
	<b>AE</b>	<b>RBM</b>	<b>AE</b>	<b>RBM</b>	<b>AE</b>	<b>RBM</b>
Sensitivity	0.9917	1	0.9974	1	1	1
Specificity	0.9943	0.9661	1	0.972	1	0.9747
Precision	0.9933	0.9443	1	0.9554	1	0.9645
NPV	0.9929	1	0.9976	1	1	1
FPR	0.0057	0.0339	0	0.028	0	0.0253
FDR	0.0067	0.0557	0	0.0446	0	0.0355
FNR	0.0083	0	0.0026	0	0	0
Accuracy	0.9931	0.9785	0.9988	0.9825	1	0.985
F1 score	0.9925	0.9714	0.9987	0.9772	1	0.9819

*Note.* AE: autoencoder; RBM: restricted Boltzmann machine; NPV: negative predictive value; FPR: false-positive rate; FDR: false-discovery rate; FNR: false-negative rate.

Table 6.7. Confusion Matrix for Autoencoder and Restricted Boltzmann Machine  
+Mean Shift

	1,300 mean shifts		800 mean shifts		400 mean shifts	
	AE	RBM	AE	RBM	AE	RBM
Sensitivity	0.9932	1	0.9974	1	1	1
Specificity	0.9789	0.9684	0.9952	0.972	0.9808	0.7241
Precision	0.975	0.9483	0.9947	0.9466	0.9796	0.4793
NPV	0.9943	1	0.9976	1	1	1
FPR	0.0211	0.0316	0.0048	0.028	0.0192	0.2759
FDR	0.025	0.0517	0.0053	0.0534	0.0204	0.5207
FNR	0.0068	0	0.0026	0	0	0
Accuracy	0.9854	0.98	0.9963	0.9813	0.99	0.78
F1 score	0.984	0.9735	0.996	0.9725	0.9897	0.648

Note. NPV: Negative predictive value; FPR: false-positive rate; FDR: false-discovery rate; FNR: false-negative rate.

To summarise the performance of the framework, the confusion matrix consists of columns and rows that list the number of testing samples as either predicted or actual ratios. Figure 6.6 provides a general description of the confusion matrix, which has two classes—normal and abnormal. The second phase was conducted to validate results using two different clustering algorithms: *k*-means and mean shift. The confusion matrices for both for different samples sizes are shown in Figures 6.7 to 6.18 below.

		Predicted	
		Normal	Abnormal
Actual	Normal	True Positive TP	False Negative FN
	Abnormal	False Positive FP	True Negative TN

Figure 6.6. Confusion matrix graphical table.

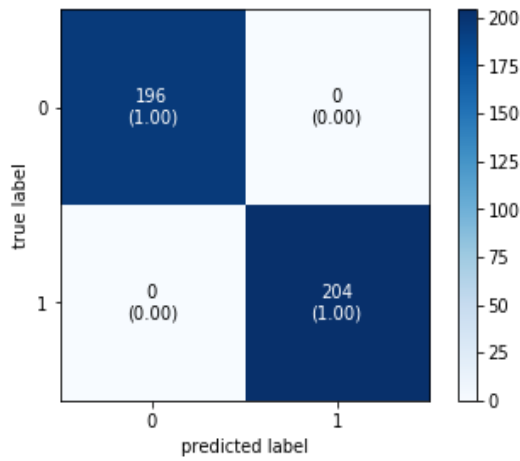


Figure 6.7. Autoencoder and k-means (400 samples)

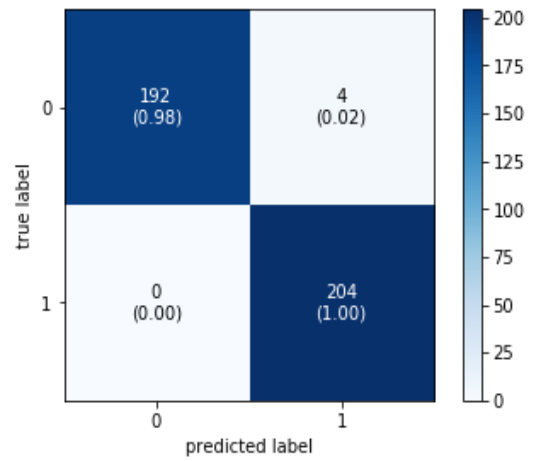


Figure 6.10. Autoencoder and mean shift (400 samples)

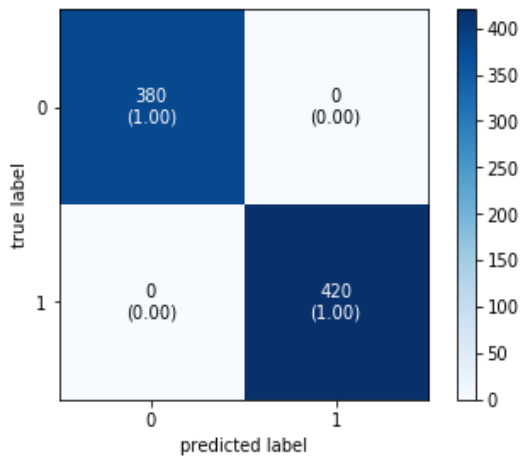


Figure 6.8. Autoencoder and k-means (800 samples)

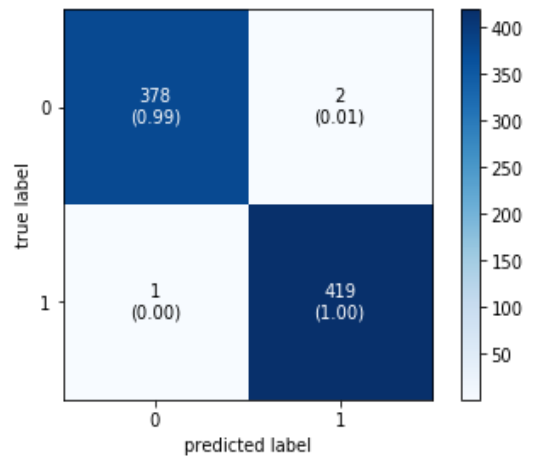


Figure 6.11. Autoencoder and mean shift (800 samples)

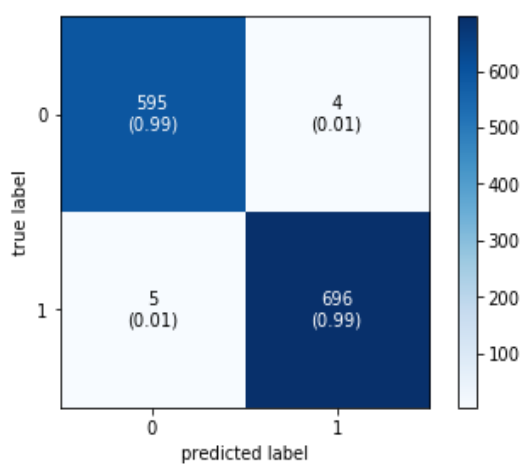


Figure 6.9. Autoencoder and k-means (1300 samples)

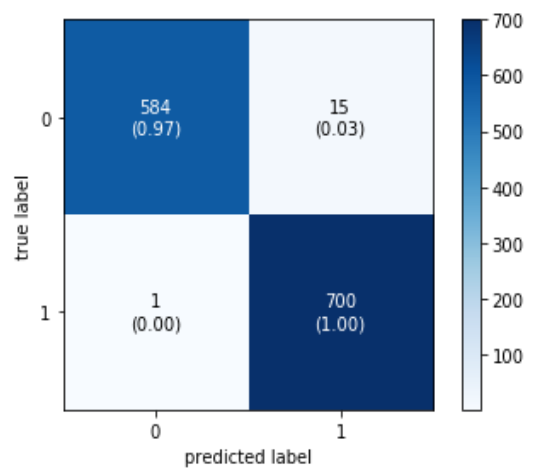


Figure 6.12. Autoencoder and mean shift (1300 samples)

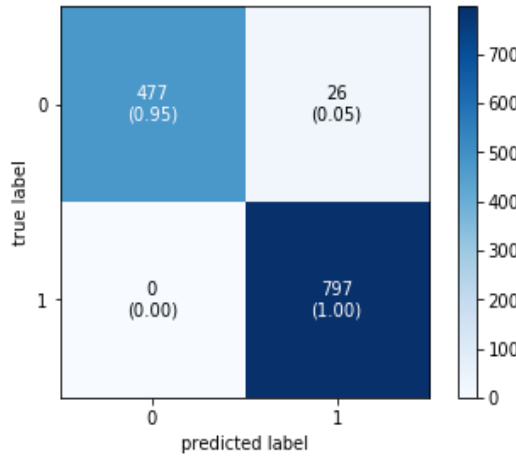


Figure 6.13. Restricted Boltzmann machine and k-means (400 samples)

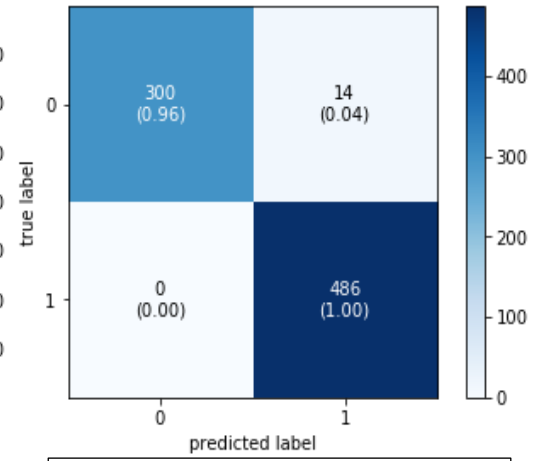


Figure 6.16. Restricted Boltzmann machine and mean shift (400 samples)

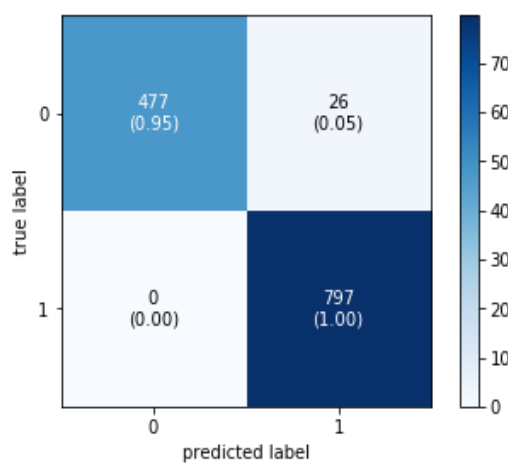


Figure 6.14. Restricted Boltzmann machine and k-means (800 samples)

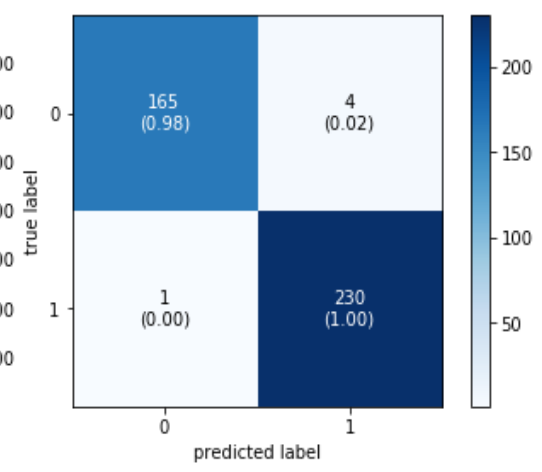


Figure 6.17. Restricted Boltzmann machine and mean shift (800 samples)

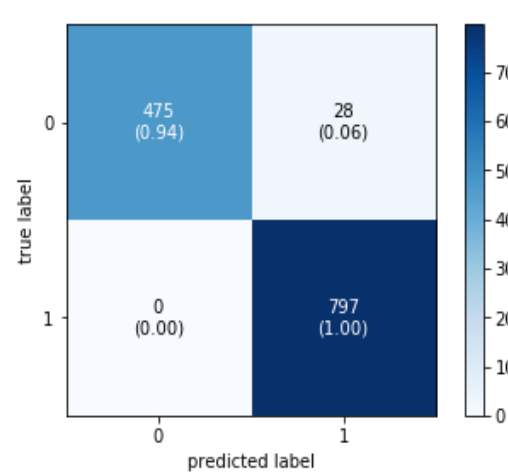


Figure 6.15. Restricted Boltzmann machine and k-means (1300 samples)

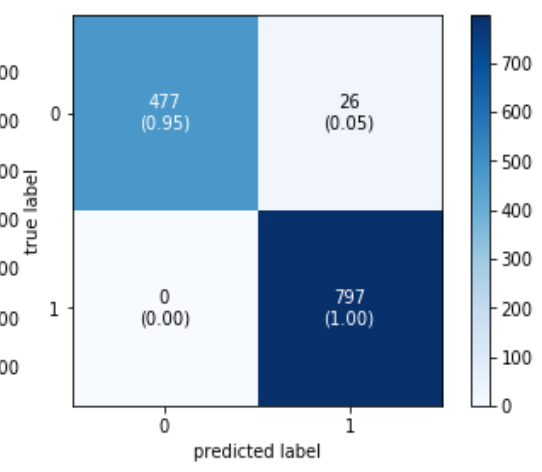
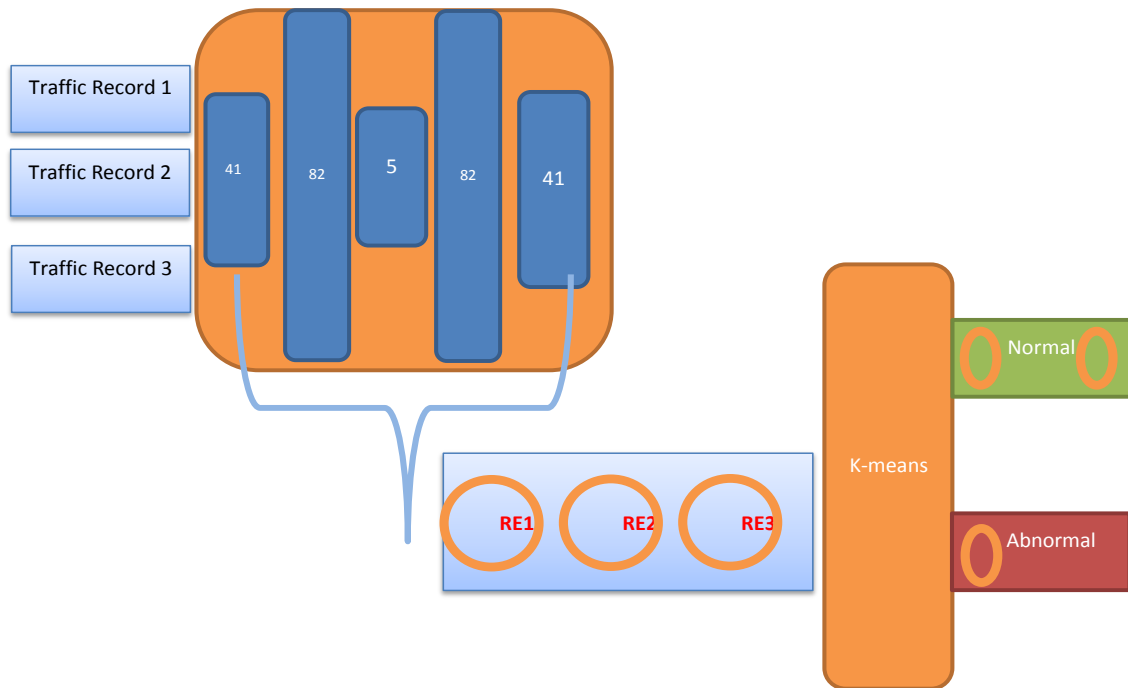


Figure 6.18. Restricted Boltzmann machine and mean shift (1300 samples)



## 6.4 Design Principles in Action

In section 4, six principles were proposed for the framework design. In Chapters 5 and 6, principles were translated to implementation and results, respectively. This section provides a discussion of the principles according to the output results analysis.



*Figure 6.19 Reconstruction errors used for forming clusters.*

Figure 6.19 depicts the framework components and the use of reconstruction errors in the clustering at the second phase. One of the main principles of the framework is dimensionality reduction, which was used to find patterns in the data. In the past, datasets have been subject to redundancy issues attributable to repetition of records, which is a component of DoS attacks. Additionally, the number of the features was not sufficiently complex for DL algorithms—this problem was not only limited to the selected dataset, but also to the limited number of the fields in the IP/TCP packet (or Overflow header). For example, in image recognition applications, the number of features is represented by the number of pixels in the image—for a simple image with 600\*800, the number of input features will be extremely large compared with 41 features in an IP/TCP packet. Additionally, the dataset or network traffic records generally only vary in a small number of fields; for example, in the records shown



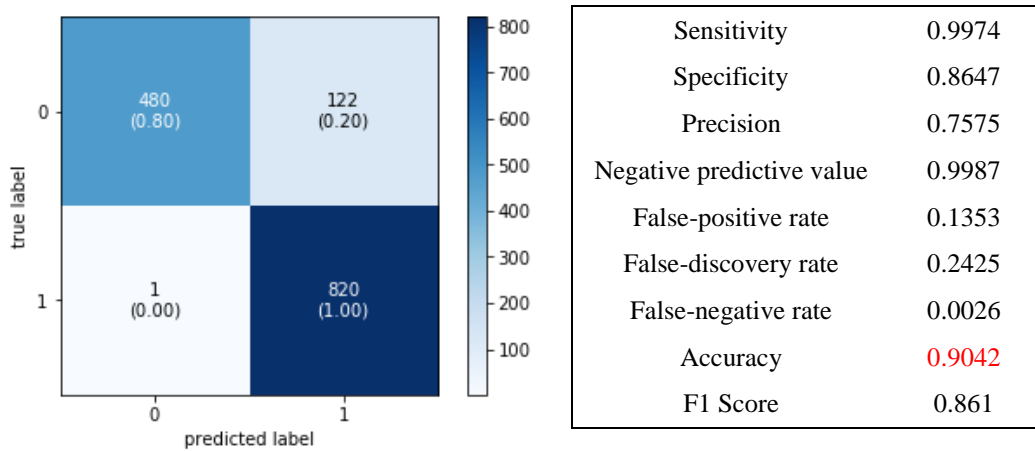


Figure 6.20. System accuracy without increasing neurons in the first hidden layer.

Figure 6.21 shows the correlation propagated to the cluster distribution in which the clusters are tightly distributed over the reconstruction errors, with spikes for some clusters (which are slightly different from the other clusters). These strong relationships complicate the separation on two levels: samples and clusters.

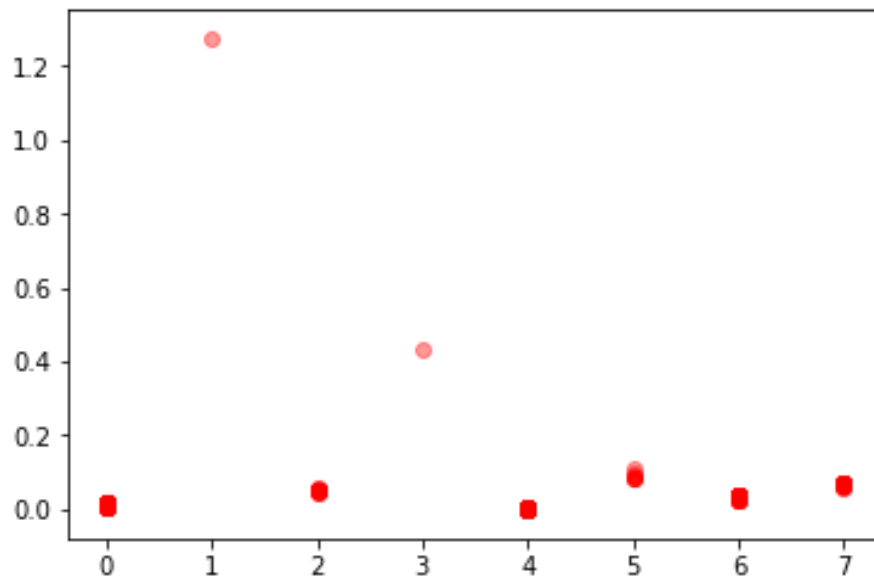


Figure 6.21. Reconstruction error distribution without increasing dimensionality in the first layer.

## 6.5 Evaluations

To highlight the contributions of this research, a comparative study with similar and related work was done. For the selected work, which was closely related to this research, an in-depth study of these approaches, experiment and results were conducted. These factors were discussed in the previous section in which a comprehensive comparative study was presented to evaluate and emphasise the intersection and distinction of this work.

*Table 6.8. Related Work Performance*

	<b>Algorithms</b>	<b>Dataset</b>	<b>Performance Statistics</b>
[139]	Discriminative RBM	KDD99	Accuracy $\approx$ 84%
[149]	Autoencoder + classifier	KDD99	Accuracy = 97.85% Precision = 99.99% Recall = 97.85% F-score = 98.15% False alarm = 2.15%
[140]	Sparse autoencoder	NSL-KDD	Accuracy $\approx$ 98% F-score $\approx$ 98.84%
[150]	SVM and $k$ - means	KDD99	Accuracy up to 90%
[142]	Autoencoder	Different dataset	Accuracy 70–93%
[151]	$k$ -means	KDD99	Accuracy 85–95% for different samples

Unlike the generative RBMs used in this research, the authors in [139] used a discriminative RBM as the unsupervised pre-step. In their work, RBMs were deployed as discriminative classifiers or as a standalone supervised classifier, which adds classes to the input records at the training phase.

In RBM, the  $p(v, h)$  formula is:

$$p(v, h) = \frac{e^{-E(v, h)}}{\sum_{u, g} e^{-E(u, g)}} \quad (31)$$

where  $v$  and  $h$  are visible and hidden units, respectively.

Discriminative RBMs consider the output at the input for the probability distribution:

$$p(v, h, y) \propto \exp(-E(v, h, y)) \quad (32)$$

where  $y$  is the output and  $E$  is the energy function.

The goal of the classifier is to optimise  $p(y/v)$  instead of  $p(y, v)$  in RBM.

During the implementation, the authors adopted a semi-supervised approach in which the discriminative RBM was trained on normal records only. The KDD99 dataset was used for training and only around 97 k instances were used. Additionally, from 41 features, 28 were selected. In their results for accuracy, the algorithm showed 84% on KDD testing data. As one of the proposed scenarios in this thesis, RBM was deployed as a pre-training phase for  $k$ -means and mean shift. The RBM worked as pre-training feature extractor rather than a single classifier. Similar to discriminative RBM, KDD99 was used. However, all 41 features were selected, and the system was trained using 800 k instances of the training data. RBM with  $k$ -means had an accuracy of over 98%, which represents a significant improvement over the proposed approach in.

In this paper, the authors proposed an anomaly detection framework based on a variation of the autoencoder known as a non-symmetric deep autoencoder (NDAE) [149]. Figure 6.22 depicts the framework architecture. The framework was composed of two phases: an NDAE was used for the first phase and a random forest classifier was used for the second phase.

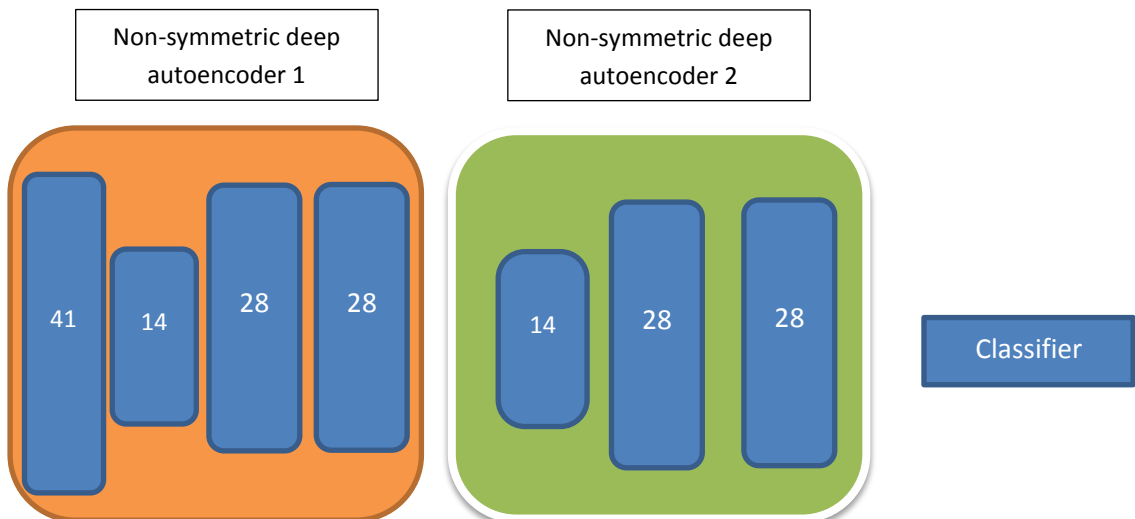


Figure 6.22. Using the encoder from deep autoencoders with a classifier.

Figure 6.23 shows the difference between a typical autoencoder and an NDAE, which, unlike the autoencoder, does not include a decoder. At the first layer, a stacked NDAE was used to encode the input vector, which was then forwarded to the classifier. A stack of NDAEs was used to increase the depth (i.e., to discover more features) and reduce the computation complexity of increasing hidden layers.

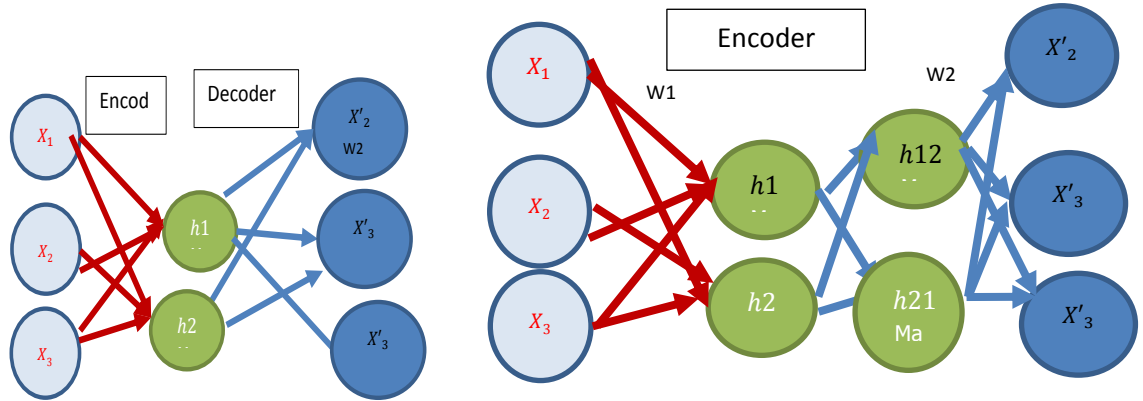


Figure 6.23. Autoencoders vs. non-symmetric deep autoencoders.

This thesis proposed a similar approach with respect to using two phases, an autoencoder and a simple algorithm— $k$ -means or mean shift. Additionally, this approach adopted dimensionality reduction to simplify classification during the second phase. However, the design considered in this thesis utilised a single autoencoder that included an encoder and the decoder for the output layers. In contrast to the abovementioned work, the proposed framework reduced dimensions to a single value for reconstruction errors.

The authors used the KDD99 dataset for their simulation, using 125 k sample records in the training phase. Results were analysed using a confusion matrix and contrasted with another DL algorithm, a deep belief network.

Table 6.9 compares results from NDAE, deep belief network and autoencoder using  $k$ -means (1.3 k samples). The analysis shows a significant improvement of  $\approx 15\%$  in accuracy and recall and a similar decrease in the false alarm rate of  $\approx 15\%$ . NDAE resulted in slightly better precision. F-score, which represents recall and precision, was superior in the proposed framework. The algorithm used in the second phase involved a random forest classifier, which is in supervised mode; however, the system lacks the essential feature of zero-day attack classification. The work proposed in this thesis used

two unsupervised algorithms, which, as discussed in Chapter 4, increases the probability of detecting new attacks.

Table 6.9. Performance of NDAE vs. DBN and proposed system in this research.

	Accuracy	Precision	Recall	F-Score	False alarm rate
NDAE	85.42	<b>100.00</b>	85.42	87.37	14.58
DBN	80.58	88.10	80.58	84.08	19.42
Autoencoder + <i>k</i> -means	<b>0.9931</b>	0.9933	<b>0.9917</b>	<b>0.9925</b>	<b>0.0067</b>

Note: NDAE: non-symmetric autoencoder; DBN: deep belief network

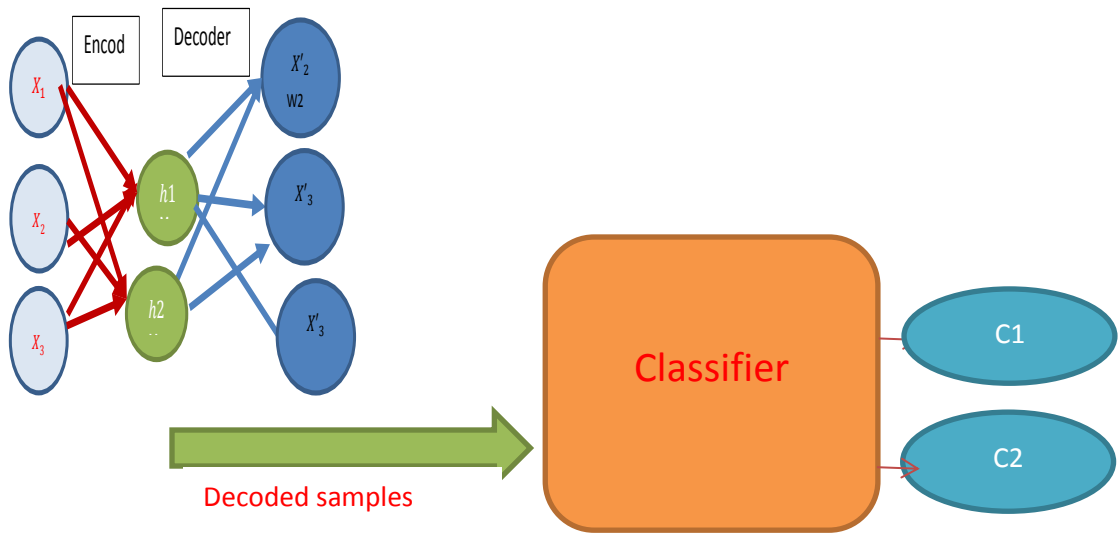


Figure 6.24. Using the encoder output for classification.

In [140] sparse autoencoders were used in the first layer as a feature extractor; subsequently, the learned features from the encoder were forwarded to a classifier. Sparsity improves the generalisation of the algorithm in which a constraint is imposed on the activation function of each neuron (*j*) to be close enough to sparsity ( $\rho$ ) as in the formula below:

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m [a_j^{(\text{Bn})}(x^{(i)})] \quad (33)$$

where  $a_j^{(\text{Bn})}$  is activation of the *j*th neuron of the autoencoder, and  $a_j^{(\text{Bn})}(x)$  is neuron activation linked to the input.

The learned features of the sparse autoencoder were classified using a softmax regression layer, as shown in figure 6.25 [140], which is an extension of classical

logistic regression for multiclass classification. The authors benchmarked a revised version of KDD99 dataset—NSL-KDD—by removing redundant records.

A confusion matrix was used for the analysis of the study, showing an accuracy of  $\approx 98\%$ . Similar to other approaches discussed, this framework used the entire extracted features from the first phase as input to the second phase.

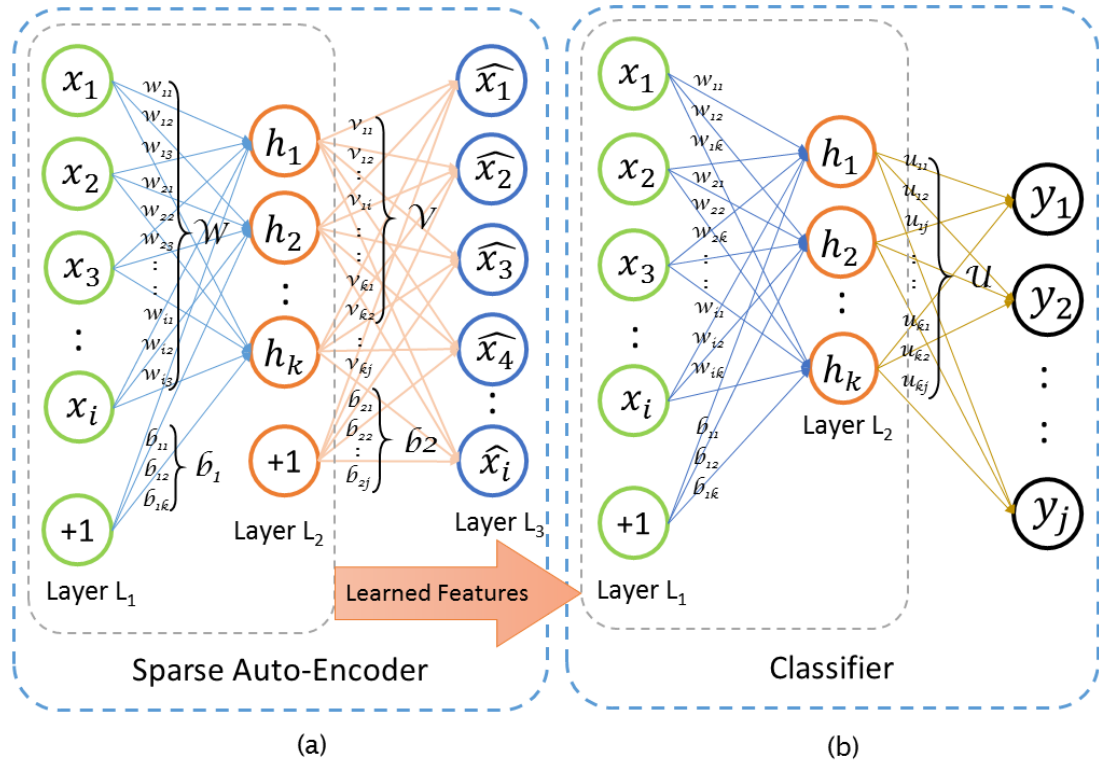


Figure 6.25. The proposed classification system based on sparse autoencoder [143].

Autoencoders have been used for anomaly detection based on dimensionality reduction. However, accuracy has been relatively weak compared with traditional PCA and its variation achieved a significant improvement in some of the datasets [142]. However, as proposed in this thesis, adding an extra simple algorithm to cluster the pre-processed data significantly improved accuracy.

Another approach used  $k$ -means clustering in the pre-processing step, with results then being fed to an SVM classifier. However, the performance was poor, with 90% accuracy [150]. Another work focused on improving the  $k$ -means for intrusion detection in KDD99, showing unstable results ranging between 85% and 95% [151], compared with the proposed framework, in which accuracy is more stable and noticeably improved.



Compared with all previously mentioned related works, the proposed framework (autoencoder +  $k$ -means) in this thesis outperformed with respect to accuracy. The frameworks in [149] and [140] used a similar approach; however, technically, the proposed framework in this research improves dimensionality reduction, while others have forwarded the learned features from the DL approach. Additionally, the comparative study shows the superiority of the adopted approach. Dimensionality reduction in this work did not involve a complicated second phase, negating the need for an expensive computation algorithm and improving the performance of available resources.

## 6.6 Summary

This chapter presented the execution results for several samples of different sizes. An output was depicted for every single step execution for the different scenarios discussed in the simulation chapter. The outputs showed the convergence of the framework (for the autoencoder and RBM) in which cost optimisation was downgraded during epoch sweeps. The results, which included samples of the identified reconstruction errors and their distributions graphs, supported verification of the model in the testing phase. Further, the clustering results for different scenarios were presented. Each cluster was shown in two components (normal and abnormal samples), and results verified the labels associated with the original dataset after clustering.

This chapter presented an analysis of results that evaluated the various combinations using DL algorithms and clustering algorithms. Given that related works have used the same approach, a confusion matrix was selected for the analysis process, facilitating comparison and evaluation. Accuracy and other related measures showed that autoencoder and  $k$ -means outperformed other scenarios.

Additionally, this chapter offered a solution to the problem of highly correlated data and the limited number of features in network traffic packets. The solution was adopted from the kernel trick in PCA algorithms in which the original data were projected into a new dimension where they became separable, followed by an application for dimensionality reduction in subsequent layers. In a comparison of results for direct dimensionality reduction and the application of projecting the data in new dimensions,

the analysis showed that the first approach achieved 90% accuracy, where the second approach scored more than 99%.

To highlight the thesis contributions, a comparative analysis was conducted against related works. At the implementation level, similar research has adopted DL as a pre-training step before forwarding outputs (decoded hidden layers) to a classifier in the second phase. Technically, the adopted approach in this thesis differed in that the DL phase was used as a dimensionality reduction approach, which reduced the output to a single value for each record. From the results, the framework outperformed all similar frameworks with regard to accuracy and precision.

## Chapter 7: Conclusions

SDN architecture provides flexibility and programmability by separating the controller from the data planes. While this offers many advantages, the risk of attacks on the flexible control plane makes SDN highly vulnerable to serious security breaches. The impacts of such attacks are intensified as a direct consequence of the increased agility and flexibility of SDN that arises from consolidating the control planes of multiple networking devices into a single central controller.

SDN has introduced a novel networking paradigm. SDN architecture separates the data planes from the control plane, which generates the flow rules required for data plane devices to forward packets. Logic is detached from devices to form a new plane known as the controller. The SDN model simplifies the traditional network and leverages the management of flexibility and scalability.

SDN had been used in various applications, including in network traffic engineering, network monitoring and virtualisation. The centralisation and providing a global view of the entire network provide better network statistics, which support decisions in network traffic balance and network monitoring. The concept of device abstraction (separation of device logic into the controller) intersects with virtualisation in which devices are abstracted and shared. Additionally, SDN has been a driver of several networking environments, including IoT, cloud computing and wireless networks. The complexity of these environments increases the complexity of network management in which SDN boosts networking resilience and abstraction. For example, in infrastructure as a service cloud computing, tenants share physical computing and networking devices, while resource sharing is executed through the device virtualisation and abstraction adopted by SDN.

SDN is also utilised to improve security. The new model provides several applications for security purposes, including security policy enforcement and verification and threat detection systems. Programmability and a global network view enable the development of improved capabilities and abstraction improves the efficiency of hardware.

Nevertheless, security is a significant challenge in SDN networks. Given that a single entity governs the entire network, the controller is a crucial element in the SDN model.

A centralised configuration is highly vulnerable because the controller is an attractive target for intruders. The severity of the traditional attacks is higher in SDN networks. Additionally, SDN has an extended attack vector because of the introduction of the controller.

This thesis investigated current security solutions and their limitations. The study provided an empirical analysis of SDN controller security to identify, formalise and quantify security concerns related to the new model. This study explored the threats related to SDN architecture, specifically those originating from the existence of the control plane. Controller security was analysed in three stages. The first stage defined potential threats based on a review of the literature. In the second stage, threats were demonstrated and modelled using a STRIDE analysis and an in-depth attack-oriented analysis was conducted using several attack trees. The third stage introduced an experiment to simulate threats and identify consequences.

The study provides a comprehensive understanding of the problem domain by identifying the security flaws of SDN. Prior analysis has shown that the controller is the major weak point in the architecture and is vulnerable to traditional network attacks such as man-in-the-middle attacks, spoofing and DoS, which primarily arise from two factors. First, the controller is an entity with more advanced capabilities than those of classic NOS such as Cisco IOS. These capabilities arise from the more complex software, which is prone to more significant vulnerabilities. Second, as a centralised entity with governance over the entire network and the facilitator of communications between the applications and forwarding planes, the controller may be a single cause of failure.

A typical networking security measure is the use of IDS, which primarily aim to identify network threats. Several approaches have been adopted, including signature-based and anomaly-based detection methods. Signature-based detection approaches utilise databases of attack signatures, matching traffic against predefined signatures—if a match occurs, the system raises the alarm of a possible attack. This approach has limitations in detecting zero-day attacks, for which there are no signatures in the database. The anomaly-detection approach utilises various methods, including statistical and machine learning algorithms, to detect threats. Given that this approach does not require pre-knowledge of threats, it can identify zero-day attacks; however, it

commonly produces a high rate of false-positive and false-negative alarms, limiting its industrial applicability. In this thesis, we proposed a framework for network anomaly detection based on recent advances in machine learning, specifically DL.

For the solution domain, this thesis proposed a novel threat detection framework based on unsupervised DL algorithms to classify network threats as anomalies. The framework consisted of two phases: a DL algorithm and clustering algorithms (either  $k$ -means or mean shift). The DL algorithm represented a pre-training phase, which simplified the input to the clustering algorithm. The framework was focused on dimensionality reduction by compressing the dimensions of the input data to a single value, simplifying and improving the performance of the clustering algorithm during the second phase.

Dimensionality reduction involved reducing the entire input record to a single value, the reconstruction error. In other applications of USDL, dimensionality reduction involves using the encoded layer as a reduced representation of the data and reducing the dimensionality of data to improve the separation of samples into clusters. However, the use of autoencoders for dimensionality reduction is limited because they are prone to exaggerated reductions, negatively affecting model predictions and reducing generalisation. Notably, the approach presented in this thesis reduced the dimensions of the data to a single value, which could be clustered using a fast algorithm such as  $k$ -means rather than using expensive computational algorithms, leading to improved performance.

The use of the DL algorithm in the pre-training phase contributed to solving the curse of dimensionality related to  $k$ -means, which is based on calculating distances between samples—as the number of samples increases, the distance between them reduces. The framework solved this problem by reducing the number of inputs based on a key procedure using autoencoders and RBMs, resulting in more straightforward inputs being forwarded to the  $k$ -means. The framework is based on two unsupervised algorithms, meaning it can find patterns in data with no previous labelling. Hence, this approach may be used to detect zero-day attacks.

Further, the study provided a comparative evaluation between a generative energy-based model (RBM) and a non-probabilistic algorithm (autoencoder). Implementation of the proposed framework design was done using TensorFlow.

DL has achieved unparalleled results in image, speech, signal, text and natural language processing applications. Network anomaly detection is an area in which DL can improve detection precision. However, research in this area is limited. In this thesis, we proposed a semi-supervised DL-based detection framework for discovering network abnormalities.

The framework employed USDL for the first phase and a simpler algorithm, *k*-means or mean shift, for the second phase. Additionally, we experimentally demonstrated the prediction accuracy of the main USDL algorithms (i.e. autoencoders and RBMs).

Simulation of several scenarios was conducted using the KDD99 network dataset. During many executions over several testing cycles, data were collected and statistically analysed. We used TensorFlow as the DL development library. As the name indicates, TensorFlow expresses matrix flows in a graph model. A TensorFlow graph includes nodes and edges, with nodes representing mathematical operations and edges representing multidimensional data arrays (or tensors).

The first stage of the experiment involved building the autoencoder network. The autoencoder consisted of two passes—the encoder and the decoder—both of which comprise multiple layers. The dataset (41 training samples) was loaded into TensorFlow's tensor dimension. Weight and bias tensors were created for the encoder and decoder. The dimensions of weights and biases depended on the number of neurons (or units in the hidden layer). For example, if the input was decoded into five units, this meant that there would be (41, 5) tensors, with 41 representing some input units (features of one network traffic record) plus 41 biases. The same dimensions were used for the decoder. The second step was to train the network. In the forward pass, logits were used as the activation function, which reconstructed records from the decoded units, weights and biases for the output. The third step was to compare the original data against the reconstructed output. A cost function, such as the squared error function was used to compute data loss. The fourth step was to minimise the cost (in this case, data loss). Several optimisation algorithms, including Adam optimisers, were used to

minimise loss or reconstruction rate. Once the network had settled after various sweeps of the data chunks (batches), the second phase of testing was conducted. During the testing, we fed the network with the testing samples to attempt to reconstruct the data.

A systematic analysis using confusion matrices was conducted to evaluate results and compare them with those of other related works. The simulation showed a significant accuracy of  $\approx 99\%$  for the integration of the autoencoder and  $k$ -means clustering algorithm.

Given that the use of DL in IDS reduces data dimensionality and the number of features in network traffic data, we recommend that further investigations be conducted into the application of DL in IDS.

## References

- [1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, ‘Software-defined networking: A comprehensive survey’, in *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.
- [2] A. Pras *et al.*, ‘Key research challenges in network management’, *IEEE Commun. Mag.*, vol. 45, no. 10, pp. 104–110, Oct. 2007.
- [3] T. Benson, A. Akella and D. Maltz, ‘Unraveling the complexity of network management’, in *Proc. 6th USENIX Symp. Netw. Syst. Design Implement.*, Boston, MA, USA, 2009, pp. 335–348.
- [4] *Open Networking Foundation (ONF)*. [Online]. Available: <https://www.opennetworking.org/>
- [5] B. Raghavan *et al.*, ‘Software-defined internet architecture: Decoupling architecture from infrastructure’, in *Proc. 11th ACM Workshop Hot Topics Netw.*, Redmond, WA, USA, 2012, pp. 43–48.
- [6] H. Kim and N. Feamster, ‘Improving network management with software defined networking’, *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.
- [7] S. Sezer *et al.*, ‘Are we ready for SDN? Implementation challenges for software-defined networks’, *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.
- [8] D. Kreutz, F. M. Ramos and P. Verissimo, ‘Towards secure and dependable software-defined networks’, in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Hong Kong, China, 2013, pp. 55–60.
- [9] M. H. Bhuyan, D. K. Bhattacharyya and J. K. Kalita, ‘Network anomaly detection: Methods, systems and tools’, *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 303–336, First Quarter 2014, doi: 10.1109/SURV.2013.052213.00046.
- [10] I. Mukhopadhyay, M. Chakraborty and S. Chakrabarti, ‘A comparative study of related technologies of intrusion detection and prevention systems’, *J. Inf. Secur.*, vol. 2, no. 1, pp. 28–38, Jan. 2011.
- [11] A. A. Ghorbani, W. Lu and M. Tavallaee, *Network Intrusion Detection and Prevention Concepts and Techniques*. Boston, MA, USA: Springer, 2010.
- [12] A. Patcha and J.-M. Park, ‘An overview of anomaly detection techniques: Existing solutions and latest technological trends’, *Comp. Netw.*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007, doi: 10.1016/j.comnet.2007.02.001.



- [13] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis and R. C. Atkinson, ‘Shallow and deep networks intrusion detection system: A taxonomy and survey’, *CoRR*, vol. abs/1701.02145, pp. 1–43, Jan. 2017. [Online]. Available: <http://arxiv.org/abs/1701.02145>
- [14] G. E. Hinton, S. Osindero and Y. W. Teh, ‘A fast learning algorithm for deep belief nets’, *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006, doi: 10.1162/neco.2006.18.7.1527.
- [15] A. Krizhevsky, I. Sutskever and G. E. Hinton, ‘ImageNet classification with deep convolutional neural networks’, in *Proc. 25th Int. Conf. Neural Proc. Syst.*, Lake Tahoe, NV, USA, 2012, pp. 1097–1105.
- [16] V. Chandola, A. Banerjee and V. Kumar, ‘Anomaly detection: A survey’, *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009, Art. no. 15, doi: 10.1145/1541880.1541882.
- [17] I. Ahmad, S. Namal, M. Ylianttila and A. Gurtov, "Security in Software Defined Networks: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317-2346, Fourth quarter 2015. doi: 10.1109/COMST.2015.2474118
- [18] L. Schehlmann, S. Abt and H. Baier, ‘Blessing or curse? Revisiting security aspects of software-defined networking’, in *10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, 2014, Rio de Janeiro, Brazil, 2014, pp. 382–387.
- [19] OpenDaylight: A Linux Foundation Collaborative Project. [Online]. Available: <http://www.opendaylight.org>
- [20] ‘Defense4All: User Guide’, in *OpenDaylight*. [Online]. Available: [https://wiki.opendaylight.org/view/Defense4All:User\\_Guide](https://wiki.opendaylight.org/view/Defense4All:User_Guide)
- [21] H. Shawn, L. Scott, O. Tomasz and S. Adam, ‘Uncover security design flaws using the STRIDE approach’, *MSDN Mag.*, Mar. 2015. [Online]. Available: <http://msdn.microsoft.com/en-gb/magazine/cc163519.aspx>
- [22] V. Saini, Q. Duan and V. Paruchuri, ‘Threat modeling using attack trees’, *J. Comput. Sci. Coll.*, vol. 23, no. 4, pp. 124–131, Apr. 2008.
- [23] M. Abadi *et al.*, ‘TensorFlow: Large-scale machine learning on heterogeneous systems’, Preliminary White Paper, Nov. 2015. [Online]. Available: <https://www.tensorflow.org/about/bib>
- [24] ‘KDD Cup 1999 Data’, in *University of California, Irvine*. [Online]. Available: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- [25] B. Raghavan *et al.*, ‘Software-defined internet architecture: Decoupling architecture from infrastructure’, in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, Redmond, WA, USA, pp. 43–48.

- [26] A. Ghodsi *et al.*, ‘Intelligent design enables architectural evolution’, in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, Cambridge, MA, USA, pp. 3:1–3:6.
- [27] N. McKeown *et al.*, ‘OpenFlow: Enabling innovation in campus networks’, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
- [28] S. Schenker, *The Future of Networking, and the Past of Protocols*. (26 Oct. 2011). Accessed: [Online Video] Available: <http://www.youtube.com/watch?v=YHeyuD89n1Y>
- [29] R. Klöti, V. Kotronis and P. Smith, ‘OpenFlow: A security analysis’, in *2013 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Goettingen, Germany, 2013, pp. 1–6.
- [30] S. Kaur, J. Singh and N. S. Ghumman, ‘Network programmability using POX controller’, in *Proc. Int. Conf. Commun. Comput. Syst. (ICCCS)*, 2014, pp. 134–138.
- [31] ‘What is software-defined networking? Definition’, in *SDX Central*. [Online]. Available: <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/> (accessed 1 Jun. 2019).
- [32] D. Levin, A. Wundsam, B. Heller and N. Handigol, ‘Logically centralized? State distribution trade-offs in software defined networks’, in *Proc. 1st Workshop on Hot Topics in Software-Defined Networks (HotSDN 12)*, 2012, Helsinki, Finland, pp. 1–6.
- [33] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman and R. Kompella, ‘Towards an elastic distributed SDN controller’, in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, New York, NY, USA, 2016, pp. 7–12, doi: <https://doi.org/10.1145/2491185.2491193>.
- [34] Y. Takamiya and N. Karanatsios, ‘TremaOpenFlow controller framework’, in *Trema*. [Online]. Available: <https://github.com/trema/trema>
- [35] ‘RYU network operating system’, in *Nippon Telegraph and Telephone Corporation*. [Online]. Available: <http://osrg.github.com/ryu/>
- [36] ‘Floodlight’ in *Project Floodlight*. [Online]. Available: <http://www.projectfloodlight.org/floodlight/>
- [37] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey and G. Wang, ‘Meridian: An SDN platform for cloud network services’, *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 120–127, Feb. 2013.

- [38] D. Erickson, ‘The Beacon OpenFlow controller’ in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Hong Kong, China, 2013, pp. 13–18.
- [39] T. Koponen *et al.*, ‘Onix: A distributed control platform for large-scale production networks’, in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement.*, Vancouver, Canada, 2010, pp. 1–6.
- [40] U. Krishnaswamy *et al.*, ‘ONOS: An open source distributed SDN OS’. [Online]. Available: <http://www.slideshare.net/umeshkrishnaswamy/open-networkoperating-system>
- [41] B. Pfaff and B. Davie, ‘The Open vSwitch database management protocol’, Internet Engineering Task Force, Fremont, CA, USA, RFC 7047, Dec. 2013. [Online]. Available: <http://www.ietf.org/rfc/rfc7047.txt>
- [42] *Open vSwitch*. [Online]. Available: <http://vswitch.org/>
- [43] R. Enns, ‘NETCONF configuration protocol’, Internet Engineering Task Force, Fremont, CA, USA, Dec. 2004. [Online]. Available: <http://tools.ietf.org/html/rfc4741>
- [44] Y. Rekhter, T. Li and S. Hares, ‘A border gateway protocol 4 (BGP-4)’, Internet Engineering Task Force, Fremont, CA, USA, RFC 4271, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4271.txt>
- [45] D. Harrington, R. Presuhn and B. Wijnen, ‘An architecture for describing simple network management protocol (SNMP) management frameworks’, Internet Engineering Task Force, Fremont, CA, USA, Dec. 2002. [Online]. Available: <http://www.ietf.org/rfc/rfc3411.txt>
- [46] J. Vasseur and J. L. Roux, ‘Path computation element (PCE) communication protocol (PCEP)’, Internet Engineering Task Force, Fremont, CA, USA, RFC 5440, Mar. 2009. [Online]. Available: <http://www.ietf.org/rfc/rfc5440.txt>
- [47] F. Maino, V. Ermagan, Y. Hertoghs, D. Farinacci and M. Smith, ‘LISP control plane for network virtualization overlays’, Internet Engineering Task Force, Fremont, CA, USA, Oct. 2013. [Online]. Available: <http://tools.ietf.org/html/draft-maino-nvo3-lisp-cp-03>
- [48] M. Casado *et al.*, ‘SANE: A protection architecture for enterprise networks’, in *Proc. 15th Conf. USENIX Security Symp.*, 2006, vol. 15, Article 10.
- [49] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja and X. Xiao, ‘Overview and principles of Internet traffic engineering’, RFC, vol. 3272, pp. 1–71, May 2002.

- [50] E. Rosen, A. Viswanathan and R. Callon, ‘Multiprotocol label switching architecture’, *RFC*, vol. 3031, Jan. 2001. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc3031.txt>
- [51] D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell and J. McManus, ‘Requirements for traffic engineering over MPLS’, *RFC*, vol. 2702, pp. 1–29, Sep. 1999.
- [52] M. V. Neves, C. A. F. De Rose, K. Katrinis and H. Franke, ‘Pythia: Faster big data in motion through predictive software-defined network optimization at runtime’, in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, Phoenix, AZ, USA, May 2014, pp. 82–90.
- [53] K. Jeong, J. Kim and Y.-T. Kim, ‘QoS-aware network operating system for software defined networking with generalized OpenFlows’, in *Proc. IEEE Netw. Oper. Manage. Symp.*, Maui, HI, USA, Apr. 2012, pp. 1167–1174.
- [54] N. Handigol *et al.*, ‘Aster\*x: Load-balancing web traffic over wide-area networks’, in *GENI Eng. Conf. 9*, Washington, DC, USA, 2009.
- [55] B. Heller *et al.*, ‘ElasticTree: Saving energy in data center networks’, in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, San Jose, CA, USA, 2010, pp. 17–17.
- [56] A. I. Coates, A. O. Hero, III, R. Nowak and B. Yu, ‘Internet tomography’, *IEEE Signal Process. Mag.*, vol. 19, no. 3, pp. 47–65, May 2002. (Monitoring)
- [57] M. Cheikhrouhou and J. Labetoulle, ‘Efficient instrumentation of management information models with SNMP’, in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, Honolulu, HI, USA, 2000, pp. 477–490. (Monitor Process)
- [58] A. Voellmy, H. Kim and N. Feamster, ‘Procera: A language for high-level reactive network control’, in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Helsinki, Finland, 2012, pp. 43–48.
- [59] J. Suh, T. Kwon, C. Dixon, W. Felter and J. Carter, ‘OpenSample: A low-latency, sampling-based measurement platform for commodity SDN’, in 2014 IEEE 34th Int. Conf. Distrib. Comput. Syst., Madrid, Spain, 2014, pp. 228–237. (OpenSample)
- [60] N. L. M. van Adrichem, C. Doerr and F. A. Kuipers, ‘OpenNetMon: Network monitoring in OpenFlow software-defined networks’, in 2014 IEEE Netw. Oper. Manage. Symp., Krakow, Poland, 2014, pp. 1–8. doi: 10.1109/NOMS.2014.6838228. (OpenNetMon)
- [61] A. Leon-Garcia and L. G. Mason, ‘Virtual network resource management for next-generation networks’, *IEEE Commun. Mag.*, vol. 41, no. 7, pp. 102–109, Jul. 2003. (Virtualisation improve performance)

- [62] N. M. M. K. Chowdhury and R. Boutaba, 'A survey of network virtualization', *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2008. (Virtualisation improve performance)
- [63] R. Jain and S. Paul, 'Network virtualization and software defined networking for cloud computing: A survey', *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, Nov. 2013. (SDN for Virtualisation why)
- [64] M. V. Malik and C. Barde, 'Survey on architecture of leading hypervisors and their live migration techniques', *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 11, pp. 65–72, Nov. 2014. (Hypervisors)
- [65] T. Koponen et al., 'Network virtualization', U.S. Patent 8,959,215 B2, 17 Feb. 2015. (Hypervisors)
- [66] A. Blenk, A. Basta, M. Reisslein and W. Kellerer, 'Survey on network virtualization hypervisors for software defined networking', *IEEE Commun. Surv. Tut.*, vol. 18, no. 1, pp. 655–685, First Quarter 2016, doi: 10.1109/COMST.2015.2489183. (Virtualisation survey)
- [67] D. Drutskoy, E. Keller and J. Rexford, 'Scalable network virtualization in software-defined networks', *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar./Apr. 2013. (FlowN)
- [68] Z. Bozakov and P. Papadimitriou, 'AutoSlice: Automated and scalable slicing for software-defined networks', in *Proc. 2012 ACM Conf. CoNEXT Student Workshop*, Nice, France, 2012, pp. 3–4. (AutoSlice)
- [69] M. El-Azzab, I. L. Bedhiaf, Y. Lemieux and O. Cherkaoui, 'Slices isolator for a virtualized OpenFlow node', in *2011 First Int. Symp. Netw. Cloud Comput. Appl.*, Toulouse, France, 2011, pp. 121–126. (Slice Isolator)
- [70] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu and D. Qiu, 'Security of the Internet of Things: Perspectives and challenges', *Wireless Netw.*, vol. 20, no. 8, pp. 2481–2501, Nov. 2014.
- [71] F. A. Alaba, M. Othman, I. A. T. Hashem and F. Alotaibi, 'Internet of Things security: A survey', *J. Netw. Comput. Appl.*, vol. 88, pp. 10–28, Jun. 2017.
- [72] D. E. Kouicem, A. Bouabdallah and H. Lakhlef, 'Internet of things security: A top-down survey', *Comput. Netw.*, vol. 141, pp. 199–221, Aug. 2018.
- [73] O. Flauzac, C. González, A. Hachani and F. Nolot, 'SDN based architecture for IoT and improvement of the security', in *2015 IEEE 29th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Gwangju, South Korea, 2015, pp. 688–693.

- [74] C. González, O. Flauzac, F. Nolot and A. Jara, ‘A novel distributed SDN-secured architecture for the IoT’, in *2016 Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Washington, DC, USA, 2016, pp. 244–49.
- [75] C. González, S. M. Charfadine, O. Flauzac and F. Nolot, ‘SDN-based security framework for the IoT in distributed grid’, in *2016 Int. Multidisciplinary Conf. Comput. Energy Sci. (SpliTech)*, Split, Croatia, 2016, pp. 1–5.
- [76] S. S. Bhunia and M. Gurusamy, ‘Dynamic attack detection and mitigation in IoT using SDN’, in *2017 27th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Melbourne, Australia, 2017, pp. 1–6.
- [77] S. Chakrabarty, D. W. Engels and S. Member, ‘A secure IoT architecture for smart cities’, in *2016 13th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, 2016, pp. 812–813.
- [78] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk and A. Rindos, ‘SDIoT: A software defined based internet of things framework’, *J. Ambient Intell. Humanized Comput.*, vol. 6, no. 4, pp. 453–461, Aug. 2015.
- [79] P. Bull, R. Austin, E. Popov, M. Sharma and R. Watson, ‘Flow based security for IoT devices using an SDN gateway’, in *2016 IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, Vienna, Austria, 2016, pp. 157–63.
- [80] K. Kalkan and S. Zeadally, ‘Securing internet of things with software defined networking’, *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 186–192, Sept. 2018
- [81] A. A. Diro and N. Chilamkurti, ‘Distributed attack detection scheme using deep learning approach for Internet of Things’, *Future Gener. Comput. Syst.*, vol. 82, pp. 761–768, May 2018.
- [82] E. Hodo *et al.*, ‘Threat analysis of IoT networks using artificial neural network intrusion detection system’, in *2016 Int. Symp. Netw. Comput. Commun. (ISNCC)*, Yasmine Hammamet, Tunisia, 2016, pp. 1–6.
- [83] A. Vahdat, D. Clark and J. Rexford, ‘A purpose-built global network: Google’s move to SDN’, *ACM Queue*, vol. 13, no. 8, Dec. 2015, Art. no. 100, doi: 10.1145/2838344.2856460. (GOOGLE SDN)
- [84] T. Benson, A. Akella, A. Shaikh and S. Sahu, ‘CloudNaaS: A cloud networking platform for enterprise applications’, in *Proc. 2nd ACM Symp. Cloud Comput.*
- [85] S. Racherla *et al.*, *Implementing IBM Software Defined Network for Virtual Environments*. Durham, NC, USA: IBM RedBooks, May 2014.
- [86] C. Beard and W. Stallings, *Wireless Communication Networks and Systems*, 1st ed. Hoboken, NJ, USA: Pearson, 2016.

- [87] D. P. Agrawal and Q.-A. Zeng, *Introduction to Wireless and Mobile Systems*, 4th ed. Boston, MA, USA: Cengage Learning, 2016.
- [88] U. Doestch et al., ‘Final report on architecture’, Mobile and Wireless Communications Enablers for the Twenty-twenty Information Society (METIS), Stockholm, Sweden, Proj. no. ICT-317669, Feb. 2015. [Online]. Available: [https://www.metis2020.com/wpcontent/uploads/deliverables/METIS\\_D6.4\\_v2.pdf](https://www.metis2020.com/wpcontent/uploads/deliverables/METIS_D6.4_v2.pdf) (SDN Vi and 5h)
- [89] L. Li, Z. Mao and J. Rexford, ‘Toward software-defined cellular networks’, in Proc. 2012 Eur. Workshop SDN, Washington, DC, USA, 2012, pp. 7–12.
- [90] R. El Hadachi and J. Erfanian, ‘NGMN 5G white paper’, NGMN Alliance, Frankfurt, Germany, White Paper, Feb. 2015. [Online]. Available: [https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN\\_5G\\_White\\_Paper\\_V1\\_0.pdf](https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf)
- [91] A. Gudipati, D. Perry, L. Li and S. Katti, ‘SoftRAN: Software defined radio access network’, in Proc. 2nd ACM SIGCOMM Workshop Hot Topics SDN, Hong Kong, China, 2013, pp. 25–30.
- [92] X. Jin, L. Erran Li, L. Vanbever and J. Rexford, ‘SoftCell: Scalable and flexible cellular core network architecture’, in *Proc. 9th Int. Conf. Emerging Netw. Exp. Technol.*, Santa Barbara, CA, USA, 2013, pp. 163–174.
- [93] S. Schmid and J. Suomela, ‘Exploiting locality in distributed SDN control’, in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Hong Kong, China, 2013, pp. 121–126.
- [94] C. Monsanto, J. Reich, N. Foster, J. Rexford and D. Walker, ‘Composing software-defined networks’, in Proc. 10th USENIX Conf. Netw. Syst. Design Implement., Berkeley, CA, USA, 2013, pp. 1–14.
- [95] C. J. Casey, A. Sutton and A. Sprintson, ‘tinyNBI: Distilling an API from essential OpenFlow abstractions’, in Proc. 3rd Workshop Hot Topics SDN, New York, NY, USA, 2014, pp. 37–42. [Online] Available: <http://arxiv.org/abs/1403.6644>
- [96] Open Networking Foundation (ONF), ‘Charter: Forwarding abstractions working group’, Apr. 2014. [Online]. Available: <https://www.opennetworking.org/images/stories/downloads/working-groups/charter-forwarding-abstractions.pdf>
- [97] *Mininet*. [Online]. Available: <http://mininet.org/>
- [98] Open Networking Foundation, ‘Software-defined networking: The new norm for networks’ ONF White Paper, Palo Alto, CA, USA, Apr. 2012.

- [99] A. Pras *et al.*, ‘Key research challenges in network management’, *IEEE Commun. Mag.*, vol. 45, no. 10, pp. 104–110, Oct. 2007.
- [100] M. Walfish *et al.*, ‘Middleboxes no longer considered harmful’, in *Proc. Fifth USENIX 467 Conf. Operating Syst. Des. Implement.*, San Francisco, CA, USA, 2004, p. 15–15.
- [101] K. Wang, Y. Qi, B. Yang, Y. Xue and J. Li, ‘LiveSec: Towards effective security management in largescale production networks’, in *2012 IEEE Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Macau, China, 2012, pp. 451–460.
- [102] M. Casado *et al.*, ‘Ethane: Taking control of the enterprise’, in *Proc. 2007 Conf. Appl. Technol. Architectures Protocols Comp. Commun.*, Kyoto, Japan, 2007, pp. 1–12.
- [103] R. Hand, M. Ton and E. Keller, ‘Active security’, in *Proc. 12th ACM Workshop Hot Topics Netw.*, College Park, MD, USA, Nov. 2013, p. 17.
- [104] H. Hu, W. Han, G.-J. Ahn and Z. Zhao, ‘FlowGuard: Building robust firewalls for software-defined networks’, in *Proc. 3rd Workshop Hot Topics Software-Defined Netw.*, Chicago, IL, USA, 2014, pp. 97–102.
- [105] S. Son, S. Shin, V. Yegneswaran, P. Porras and G. Gu, ‘Model checking invariant security properties in OpenFlow’, in *Proc. IEEE Int. Conf. Commun.*, Budapest, Hungary, 2013, pp. 1974–1979.
- [106] R. Braga, E. Mota and A. Passito, ‘Lightweight DDoS flooding attack detection using NOX/OpenFlow’, in *IEEE 35th Conf. Local Comp. Netw. (LCN)*, Denver, CO, USA, 2010, pp. 408–415.
- [107] M. Yu, L. Jose and V. Miao, ‘Software defined traffic measurement with OpenSketch’, in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement.*, Lombard, IL, USA, 2013, pp. 29–42.
- [108] S. Shin *et al.*, ‘FRESCO: Modular composable security services for software-defined networks’, in *NDSS Symp.*, San Diego, CA, USA, 2013.
- [109] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu and G. Jiang, ‘NetFuse: Short-circuiting traffic surges in the cloud’, in *Proc. IEEE Int. Conf. Commun.*, Budapest, Hungary, 2013.
- [110] A. Sapiro *et al.*, ‘MAPPER: A mobile application personal policy enforcement router for enterprise networks’, in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, London, United Kingdom, 2014.
- [111] S. Fayazbakhsh, V. Sekar, M. Yu and J. Mogul, ‘FlowTags: Enforcing network-wide policies in the presence of dynamic middlebox actions’, in *Proc. 2nd*



- Workshop Hot Topics Software Defined Netw., Seattle, WA, USA, 2013, pp. 533–546.
- [112] Z. A. Qazi *et al.*, ‘SIMPLE-fying middlebox policy enforcement using SDN’, in *Proc. ACM SIGCOMM 2013 Conf.*, Hong Kong, China, 2013, pp. 27–38.
- [113] K. Giotis, G. Androulidakis and V. Maglaris, ‘Leveraging SDN for efficient anomaly detection and mitigation on legacy networks’, in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014.
- [114] S. Shin and G. Gu, ‘CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)’ in *20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Austin, TX, USA, 2012, pp. 1–6.
- [115] A. Khurshid, W. Zhou, M. Caesar and P. Godfrey, ‘VeriFlow: Verifying network-wide invariants in real time’, *ACM SIGCOMM Comp. Commun. Rev.*, vol. 42, no. 4, pp. 467–472, 2012.
- [116] K. Wang, Y. Qi, B. Yang, Y. Xue and J. Li, ‘LiveSec: Towards effective security management in large-scale production networks’, in *2012 IEEE Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, 2012, pp. 451–460
- [117] J. R. Ballard, I. Rae and A. Akella, ‘Extensible and scalable network monitoring using OpenSAFE’, in *Proc. 2010 Internet Netw. Manage. Conf. Res. Enterprise Comput.*, San Jose, CA, USA, 2010, p. 8.
- [118] P. Porras *et al.*, ‘A security enforcement kernel for OpenFlow networks’, in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 121–126
- [119] J. H. Jafarian, E. Al-Shaer and Q. Duan, ‘OpenFlow random host mutation: Transparent moving target defense using software defined networking’, in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 127–132.
- [120] A. Zaalouk, R. Khondoker, R. Marx and K. Bayarou, ‘OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions’, in *Proc. IEEE Netw. Oper. Manage. Symp.*, 2014.
- [121] J. Matias, J. Garay, A. Mendiola, N. Toledo and E. Jacob, ‘FlowNAC: Flow-based network access control’, in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, 2014.
- [122] X. Wen, Y. Chen, C. Hu, C. Shi and Y. Wang, ‘Towards a secure controller platform for OpenFlow applications’, in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics in Software Defined Netw.*, 2013, pp. 171–172.

- [123] A. Tootoonchian and Y. Ganjali, ‘HyperFlow: A distributed control plane for OpenFlow’, in *Proc. 2010 Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.
- [124] S. Scott-Hayward, G. O’Callaghan and S. Sezer, ‘SDN security: A survey’, in *2013 IEEE SDN Future Netw. Services*, Trento, Italy, 2013, pp. 1–7.
- [125] F. Ruffy, W. Hommel and F von Eye, ‘A STRIDE-based security architecture for software-defined networking’, in *ICN 2016: Fifteenth Int. Conf. Netw.*, Lisbon, Portugal, 2016, pp. 95–101.
- [126] L. Yao, P. Dong, T. Zheng, H. Zhang, X. Du and M. Guizani, ‘Network security analyzing and modeling based on Petri net and attack tree for SDN’, in *2016 Int. Conf. Comput. Netw. Commun. (ICNC)*, Kauai, HI, USA, 2016, pp. 1–5.
- [127] ‘SDL threat modeling tool’, in *Microsoft*. [Online]. Available: <https://www.microsoft.com/en-us/sdl/adopt/+threatmodeling.aspx> (accessed 13 Mar. 2017).
- [128] V. Saini, Q. Duan and V. Paruchuri, ‘Threat modeling using attack trees’, *J. Comput. Sci. Coll.*, vol. 23, no. 4, pp. 124–131, Apr. 2008.
- [129] ‘Our most advanced penetration testing distribution, ever’, in *Kali Linux*. [Online]. Available: <https://www.kali.org/> (accessed 18 Jun. 2017).
- [130] ‘Nmap’, in *Nmap*. [Online]. Available: <https://nmap.org/> (accessed 18 Jun. 2019).
- [131] ‘SDN-Toolkit’, in *Hellfire Security*. [Online]. Available: <http://www.hellfiresecurity.com/tools.html>
- [132] D. Mudzingwa and R. Agrawal, ‘A study of methodologies used in intrusion detection and prevention systems (IDPS)’, in *2012 Proc. IEEE Southeastcon*, Orlando, FL, USA, 2012, pp. 1–6.
- [133] A. Mohamed, G. Dahl and G. E. Hinton, ‘Acoustic modelling using deep belief networks’, *IEEE Trans. Audio Speech Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2011.
- [134] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2017.
- [135] J. Schmidhuber, ‘Deep learning in neural networks: An overview’, *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
- [136] Y. Bengio, ‘Learning deep architectures for AI’, *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009.

- [137] L. Zamparo and Z. Zhang, ‘Deep autoencoders for dimensionality reduction of high-content screening data’, *CoRR*, vol. abs/1501.01348, 2015.
- [138] S. Zhai, Y. Cheng, W. Lu and Z. Zhang. ‘Deep structured energy based models for anomaly detection’, in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1100–1109.
- [139] U. Fiore, F. Palmieri, A. Castiglione and A. De Santis, ‘Network anomaly detection with the restricted Boltzmann machine’, *Neurocomputing*, vol. 122, pp. 13–23, Dec. 2013.
- [140] A. Javaid, Q. Niyaz, W. Sun and M. Alam, ‘A deep learning approach for network intrusion detection system’, in *Proc. 9th EAI Int. Conf. Bio-inspired Inf. Commun. Tech.*, New York, NY, USA, 2015, pp. 21–26.
- [141] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish and A. E. Hassanien, ‘Hybrid intelligent intrusion detection scheme’, in *Soft Computing in Industrial Applications (Advances in Intelligent and Soft Computing, vol. 96)*, A. Gaspar-Cunha, R. Takahashi, G. Schaefer and L. Costa, Eds., Berlin, Germany: Springer, 2001, pp. 293–303.
- [142] M. Sakurada and T. Yairi. ‘Anomaly detection using autoencoders with nonlinear dimensionality reduction’, in *Proc. MLSDA 2014 2nd Workshop Mach. Learn. Sensory Data Anal.*, Gold Coast, Australia, 2014, p. 4, doi: 10.1145/2689746.2689747.
- [143] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," in *IEEE Access*, vol. 7, pp. 41525-41550, 2019. doi: 10.1109/ACCESS.2019.2895334
- [144] B. Dong and X. Wang, ‘Comparison deep learning method to traditional methods using for network intrusion detection’, in *2016 8th IEEE Int. Conf. Commun. Softw. Netw. (ICCSN)*, Beijing, China, 2016, pp. 581–585.
- [145] H. Hoffmann, ‘Kernel PCA for novelty detection’, *Pattern Recognit.*, vol. 40, no. 3, pp. 863–874, Mar. 2007.
- [146] J. Zacharias, M. Barz, and D. Sonntag, ‘A Survey on Deep Learning Toolkits and Libraries for Intelligent User Interfaces’. arXiv preprint arXiv:1803.04818, 2018.
- [147] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay E. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825-2830, 2011.

- [148] M. Tavallae, E. Bagheri, W. Lu and A. A. Ghorbani, 'A detailed analysis of the KDD CUP 99 data set', in *Proc. 2nd IEEE Symp. Comput. Intell. Secur. Defence Appl.*, Ottawa, Canada, 2009, pp. 53–58.
- [149] N. Shone, T. N. Ngoc, V. D. Phai and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, Feb. 2018.
- [150] A. Shrivastava and R. R. Ahirwal, "A SVM and K-means Clustering based Fast and Efficient Intrusion Detection System," *International Journal of Computer Applications*, vol. 72, no. 6, pp. 25–29, 2013.
- [151] L. Tian and W. Jianwen, "Research on Network Intrusion Detection System Based on Improved K-means Clustering Algorithm," *2009 International Forum on Computer Science-Technology and Applications*, Chongqing, 2009, pp. 76-79.