Software Defined Networking: Network Intrusion Detection System

Tuan Anh Tang

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

The University of Leeds

Department of Electronic and Electrical Engineering

June 2019

Declaration

The work in this thesis is based on research carried out at the Institute of Robotics, Autonomous Systems and Sensing (IRASS), School of Electronic and Electrical Engineering, Leeds University, UK. The candidate confirms that no part of this thesis has been submitted elsewhere for any other degree or qualification and it is all his own work except where work which has formed part of jointly authored publications. The contribution of the candidate and other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others. It is to assert that the candidate has contributed solely to the technical part of the joint publication under the guidance of his academic supervisors. The detailed contributions of the authors are as the following:

Author: Tuan Anh Tang
Contributions: Proposed and implemented the SDN-based intrusion detection models. Wrote first and final drafts of the manuscripts.

Co-Author: Dr. Des McLernon
Contributions: Provided overall supervision on the whole project. Provided feedback on technical analyses and final drafts of the manuscripts.

Co-Author: Dr. Lotfi Mhamdi
Contributions: Provided feedback on technical analyses related to SDN aspects and drafts of the manuscripts.

Co-Author: Dr. Syed Ali Raza Zaidi
Contributions: Provided feedback on technical analyses and drafts of the manuscripts.

Co-Author: Professor Mounir Ghogho
Contributions: Provided feedback on technical analyses and drafts of the manuscripts.

This thesis is dedicated to my family.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Dr. Des McLernon for all of his support in my PhD life from the beginning. His guidance, feedback and encouragement have inspired and motivated me a lot in my PhD journey. But above everything, thank you very much for your caring and kindness. I wholeheartedly appreciate your kindness.

I would like to express my gratitude to Dr Lotfi Mhamdi for guiding me to the field of SDN and continuously supporting me during my research.

I would like to thank Dr. Syed Ali Raza Zaidi for all of his advice and feedback. Thank you very much for your guidance since the beginning.

My appreciation also goes to Professor Mounir Ghogho for his support and guidance. Thank you very much for the constructive talks at the early stage of my research that gave me the right research direction.

I thank my fellow lab-mates in rooms 3.62 and 2.59, School of Electronic and Electrical, for all the fruitful discussions and for all the fun we have had in the last four years. I would like to especially thank Yen, Bao, Asma, Ali, Mohanad, Edmond, Bao and Naveed for all the precious things that we have shared.

Last but not least, I would like to thank my family: my parents, and my sisters for supporting and encouraging me endlessly. Thank you very much for being with me through all the ups and downs. Without you, I would not be here now.

Abstract

Software Defined Networking (SDN) is developing as a new solution for the development and innovation of the Internet. SDN is expected to be the ideal future for the Internet since it can provide controllable, dynamic and cost-effective networking. The emergence of SDN provides a unique opportunity to achieve network security in a more efficient and flexible manner. One key advantage of SDN, as compared to traditional networks, is that by virtue of centralized control, it allows better provisioning of network security. Nevertheless, the flexibility provided by the SDN architecture manifests several new network security issues that must be addressed to strengthen SDN security. The SDN has original structural vulnerabilities, which are the centralized controller, the control-data interface and the control-application interfaces. These vulnerabilities can be exploited by intruders to conduct several types of attacks.

Network Intrusion Detection System (NIDS), which is an important part of network architecture, is used to detect network intrusions and secure the whole network. In this thesis, we propose an SDN-based NIDS (DeepIDS) using Deep Learning (DL) algorithms to detect anomalies in the SDN architecture. Firstly, we evaluate the potential of DL for flow-based anomaly detection with different flow features.

Through experiments, we confirm that the DL approach has the potential for flow-based anomaly detection in the SDN environment. Secondly, we propose a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) to improve the detection rate of the DeepIDS. Our experimental results show that the proposed GRU-RNN model improves the detection rate significantly without deteriorating network performance. The performance of our system in terms of accuracy, throughput, latency and resource utilization shows that DeepIDS does not affect the performance of the OpenFlow controller, and so is a feasible approach.

Finally, we introduce an unsupervised approach (SAE-1SVM) to solve an unlabeled and imbalanced dataset problem. This approach yields a high detection rate while maintaining a significantly low processing time. Through extensive experimental evaluations, we conclude that our proposed approach exhibits a strong potential for intrusion detection in the SDN environments.

Abbreviations

ACC	Accuracy
AE	Autoencoder
AUC	Area Under the Curve
$DDoS$	Distributed Denial of Service Attack
DoS	Denial of Service Attack
DL	Deep Learning
DNN	Deep Neural Network
$F1$	F1-measure
FP	False Positive
FPR	False Positive Rate
FN	False Negative
GRU	Gated Recurrent Unit
$GRU-RNN$	Gated Recurrent Neural Network
R	Recall
ROC	Receiver Operating Characteristic Curve
RNN	Recurrent Neural Network
$OC-SVM$	One-class Support Vector Machine
P	Precision
IDS	Intrusion Detection System
$LSTM$	Long Short-Term Memory
ML	Machine Learning
NN	Neural Network
$NIDS$	Network Intrusion Detection System
SAE	Stacked Autoencoder
SDN	Software Defined Networking
SOM	Self-Organizing Map
SVM	Support Vector Machine
TP	True Positive
TPR	True Positive Rate
TN	True Negative

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In This Chapter:

The motivation behind this work is presented. Some challenges of intrusion detection are introduced, and then the critical research questions are identified. Finally, the contributions of the work are given along with the organization of the thesis.

1.1 Motivation

Software Defined Networking (SDN) is a developing architecture that is dynamic, manageable, cost-effective, and adaptable, thus making it ideal for the high-bandwidth, dynamic nature of today's applications and networks. SDNs are currently being deployed in many network environments, from home and enterprise networks to datacenters (e.g., IBM, Cisco, Google WAN B4 [9], Huawei carrier network [10]). As can be seen in Figure 1.1, the SDN market has grown to more than a $9.5 billion market in 2019 and is predicted to continue to grow to $13.8 billion by 2021. The capabilities of SDN (e.g., logically centralized controller, and global network overview) help to solve several security issues in a traditional network and bring the ability to control network traffic at a fine-grained level. However, the SDN architecture itself also introduces some new attack threats and security vulnerabilities. Kreutz *et al.* [11] introduced seven threat vectors in SDN. Several attacks can be conducted in the SDN architecture. For instance, Distributed Denial of Service (DDoS) attacks can overwhelm an SDN controller and a communication channel with artificial service calls. A Man-in-the-Middle attack can break links between the controller and the switches and claim control of the network.

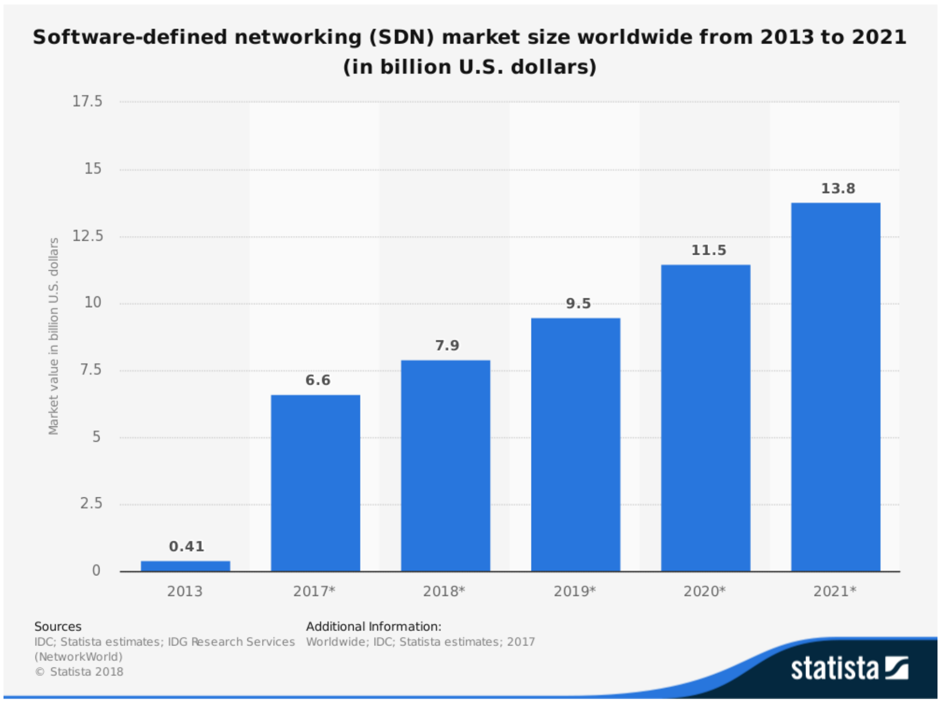


Figure 1.1: The SDN Market Size Prediction [1]

Because of the wide variety of types of SDN deployments, SDN security is a serious concern and has recently been extensively researched - see [12] and [13] for more detail. Therefore, there is a need to develop an efficient network intrusion detection system (NIDS). NIDS is one of the most crucial parts of network architecture. Several machine learning (ML) approaches have been proposed to secure the SDNs. Given a set of training data, ML algorithms learn non-linear patterns of training data and then detect attacks inside the SDNs. These algorithms can be statistical [14] [15], probabilistic [16] or Support Vector Machine [17] [18]. Despite the high accuracy and performance obtained in several fields, ML algorithms used in intrusion detection tend to have some limitations as mentioned in [19]. These include the difficulty of determining the discriminator, the availability of labelled datasets for classification and evaluation, the high cost of errors and the diversity of network traffic. In recent years, ML has been used by many researchers for intrusion detection in the SDN environments. However, these techniques usually lead to a high rate of false positives that is a significant concern for practical NIDSs.

Recently, Deep Learning (DL) has emerged and achieved significant success in the field of speech recognition [20], image recognition [21], and natural language processing [22] [23]. DL is capable of automatically finding correlations in raw data, and so it can improve the intrusion detection rate. Motivated by the development of DL, we extended this research trend to a DL-based intrusion detection approach for the SDN context. We believe DL is a promising method for the next generation of intrusion detection. DL can be used, and so we can acquire a high detection rate. The flow-based and programmable natures of SDNs also facilitate the development of NIDSs.

1.2 Challenges

In general, there are a few challenges for flow-based intrusion detection in the SDN architecture as follows:

- Network traffic is dynamic, diverse and constantly changing. In addition, network attacks keep evolving and become more intelligent and aggressive. The dynamic nature of network traffic makes intrusion detection extremely challenging.
- Traditional NIDSs use a large number of hand-crafted features to improve intrusion detection accuracy. However, an SDN provides us with a limited number of raw flow features. Therefore, it is a challenge to improve intrusion detection accuracy with these limited raw flow features.
- NIDS is supposed to provide real-time intrusion detection and mitigation. Therefore, computational complexity and network overheads also need to be seriously considered.
- For ML/DL intrusion detection approaches, the network data is significantly important. These datasets are used for training and testing systems for intrusion detection. The availability of labelled network datasets is a big problem. The SDN architecture is still a new technology, so network datasets for it are either quite rare and/or unpublished. As a result, it is difficult to train and evaluate a model in a supervised manner to detect intrusions in the SDN network.

1.3 Objective and Scope of the Thesis

In the previous sections, we have given the motivation and some of the challenges in the field. The primary objective of this thesis is to develop a NIDS under the context of SDN using DL. Figure 1.2 describes the overview of our NIDS that uses DL to detect intrusions in SDN networks. The details of the proposed NIDS will be presented in Chapter 4.

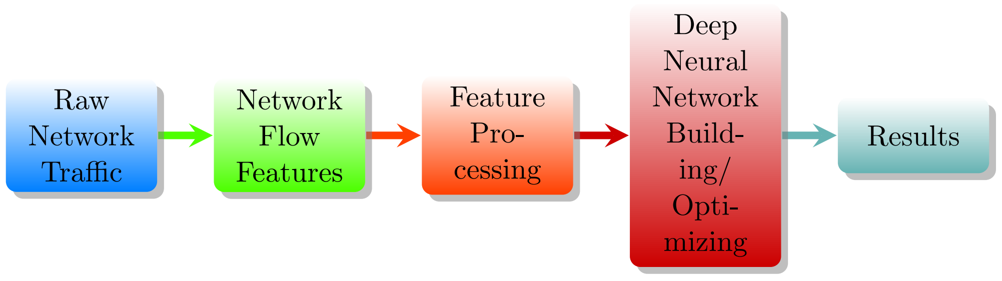


Figure 1.2: The Overview of the NIDS

We present four main research questions as follows:

- Does the SDN facilitate the development of a flow-based NIDS?
- Can we improve the intrusion detection accuracy with limited raw features in the SDN architecture using DL?
- How do we take advantage of DL to solve the dataset problem in SDNs?
- Can we develop an end-to-end system to effectively detect intrusions in the SDN paradigm?

1.4 Limitations and Constraints

In the proposed methods, we have a few limitations and assumptions.

- In this thesis, we have to adapt some conventional datasets to the SDN architecture. This adaption may not be close enough to a real SDN-based dataset but many researchers in the same field still use it. In Chapter 4 and 5, the packet-based NSL-KDD dataset is adapted for our experiments. This adaption may affect the generalization and application of our approach in the SDN context. However, we selected some most basic network features having similar characteristics in both packet-based and flow-based datasets to minimize this gap. We assume that these network features are interchangeable for both traditional and SDN-based networks.
- Because of the use of published datasets, all the attacks concerned in this thesis are just related to network and application layers (i.e., L2, L3 and L7). This thesis does not consider any kind of attacks related to L1 physical layer which is also an important part of the network. In addition, because of the nature of pulished datasets, we have not considered all types of network topologies in our experiment.

1.5 Thesis Outline and Contributions

This PhD thesis describes the research carried out in the development of a NIDS in the SDN environment using DL. Chapter 2 discusses the SDN architecture and its security issues. A literature overview about DL is presented in Chapter 3. Chapter 4 looks at the potential of applying DL for intrusion detection in the SDN and the effect of different features on the detection rate. Chapter 5 is devoted to the use of a Deep Recurrent Neural Network to improve the detection accuracy of an SDN-based NIDS. A hybrid unsupervised approach for Distributed Denial of Service (DDoS) attacks detection in SDN is proposed in Chapter 6. Chapter 7 concludes the thesis and gives further research directions.

The content of each chapter is summarized below:

Chapter 2: Software Defined Networking

In this chapter, we introduce the SDN architecture, and the OpenFlow protocol. The NIDS and its detection performance evaluation methodology are also introduced in this chapter. We also present some related work within the use of DL for intrusion detection in the SDN.

Chapter 3: Deep Learning

Chapter 3 presents a literature overview of the DL algorithm. At the end of this chapter, we introduce network datasets used in this thesis.

Chapter 4: DeepIDS: Deep Learning Approach for Intrusion Detection in Software Defined Networking

In this chapter, we propose an SDN-based NIDS (DeepIDS). We answer the following research question: *Does the SDN facilitate the development of a flow-based NIDS?* We implement a DL approach for intrusion detection in the SDN architecture. A deep neural network (DNN) is created for binary classification. Through experiments, we confirm that the DL approach has the potential for flow-based anomaly detection in the SDN environment. We also evaluate the performance of our system in terms of throughput, latency and resource utilization. Our test results show that DeepIDS does not affect the performance of the OpenFlow controller, and so is a feasible approach.

Chapter 5: Deep Recurrent Neural Networks for SDN-based Intrusion Detection Systems

In this chapter, we address the following research question: *Can we improve* *the intrusion detection accuracy with limited raw features in the SDN architecture using DL?* To improve the detection accuracy, in this chapter, we propose a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) enabled intrusion detection for SDN. The proposed approach was tested using several datasets, and we achieved a quite high detection accuracy with low dimensional feature sets that can be extracted at SDN controllers. We also evaluated network performance of our proposed approach in terms of throughput and latency. Our test results show that the proposed GRU-RNN model does not deteriorate the network performance. Through extensive experimental evaluation, we conclude that our proposed approach exhibits a strong potential for intrusion detection in the SDN environment.

Chapter 6: Deep Learning Approach Combining Stacked Autoencoder with One-class SVM for DDoS Attack Detection in SDNs

Recently, several ML/DL intrusion detection approaches have been proposed to secure SDN networks. However, these approaches cannot deal adequately with imbalanced and unlabeled datasets. So, the goal of this chapter is to detect the network attacks in an unsupervised manner by using the flow table information. In this chapter, we propose a hybrid approach using the Stack Autoencoder and One-class Support Vector Machine (SAE-1SVM) for DDoS attack detection. The experimental results show that the proposed algorithm can achieve an average accuracy of 99.35% with a small set of flow features. The SAE-1SVM shows that it can reduce the processing time significantly while maintaining a high detection rate. In summary, the SAE-1SVM can work well with imbalanced and unlabelled datasets and yields a high detection accuracy. The research question *"How do we take advantage of DL to solve the dataset problem in SDNs?"* has been addressed in this chapter.

Chapter 7: Conclusions and Future Work

In this chapter, we provide a summary of the work in the thesis. We also give final conclusions and discuss some drawbacks and limitations of this work. Then further extensions and future research directions are also presented.

1.6 Publications

The work undertaken in this thesis has resulted in the following publications.

- **Tang, T. A.**, Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016, October). Deep learning approach for network intrusion detection in software defined networking. In *Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on* (pp. 258-263). IEEE.

- **Tang, T. A.**, Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2018, June). Deep recurrent neural network for intrusion detection in SDN-based networks. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)* (pp. 202-206). IEEE.

- **Tang, T. A.**, McLernon, D., Mhamdi, L., Zaidi, S. A. R., & Ghogho, M. Intrusion Detection in SDN-based Networks: Deep Recurrent Neural Network Approach. In: Tang M ed. Deep Learning Applications for Cyber Security. (In Press)

Finally, as a result of the WINCOM paper (and its 111 citations as of 10/06/2019), Tuan Anh Tang was offered a three-month internship at the world-famous BT labs in Martlesham Heath, Ipswich to work on a project “Anomaly Detection for the Internet of Things Devices using Software Defined Networking and Deep Learning”.

Chapter 2

Software Defined Networking

In This Chapter:

Firstly, the SDN architecture and its security issues are described in detail. Secondly, an overview of the NIDS is introduced, and some related works are also discussed. Finally, we demonstrate a toy example of SDN-based NIDS in detecting DoS attacks.

2.1 Software Defined Networking

2.1.1 Definition

Modern networks were developed many decades ago and remain mostly unchanged over the past decade. It is mainly decentralized, autonomous and built from a large number of network devices such as routers, switches and numerous types of middleboxes (e.g., firewall, and load balancing) with several complex protocols implemented on them. This network equipment is traditionally developed by several manufacturers. Each manufacturer has its own designs, firmware, and software to operate their own hardware in a proprietary and closed way. Network operators have to configure policies to respond to a wide range of network events and applications. They have to manually transform these high-level policies into low-level configuration commands with very limited tools. In fact, network devices are usually vertically-integrated black boxes that make network management and performance tuning quite challenging and error-prone. Because of its vast development base and its essential role in our society's critical infrastructure, the modern network has become extremely difficult to evolve both in term of its physical infrastructure as well as its protocol and performance. In practice, it is quite difficult to deploy a new version of an existing protocol (e.g., IPv6). These issues lead to a need for a new network paradigm that makes the network more scalable, dynamic and easier to manage and configure.

The idea of "programmable networks" has been proposed as a way to facilitate the evolution of current networks. The concept of programmable networks and decoupled control logic has been around for several years. In the past, various technologies were developed to enable the programmability of communication networks. In the mid-1990s, Active Networks (AN) [24] were developed with the basic idea of injecting program into data packets. Switches extract and execute programs from data packets so that new routing mechanisms and network services can be implemented without the modification of the forwarding hardware. However, AN did not gain much attention because of its security and performance concerns. Also in mid 1990s, Devolved Control of ATM Network (DCAN) [25] was aimed at designing and developing the necessary infrastructure for scalable control and management of ATM networks. The premise of DCAN is that control/management functions of the various network devices (e.g., ATM switches) should be decoupled from the device themselves and delegated to external entities dedicated to that purpose. In the first half of the 2000s, the separation of the control and data plane had been considered as one of the central points of simplifying network design. Forwarding and Control Element Separation (ForCES) [26] is a pioneer in this area. ForCES classified the network components into two distinct types which are the forwarding element and the control element. The forwarding element only forwards and filters traffic. The control plane provides instructions for processing packets. These elements communicate with each other via a standardised open interface, which is considered to be a core feature of the ForCES protocol. Although ForCES is still under active development, it is not widely adopted by major vendors. The IETF Network Configuration Working Group proposed NETCONF [27], which is a management protocol for modifying the configurations of network devices, in 2006. Network devices can expose an API that helps to send and to retrieve extensible configuration data. However, there is no separation between control and data planes. In 2006, the SANE/ Ethane project [28] proposed a new architecture for enterprise networks. Ethane focuses on using a centralized controller to manage policy and security in a network. It can be said that Ethane is the predecessor and the foundation for what would become SDN today.

SDN is proposed as a solution for all network challenges. SDN is based on the idea of decoupling the control plane and data plane and producing less sophisticated data plane devices. In general, SDN is built under four principles:

- **Separation of control and data plane:** these planes must be logically separated and connected via an interface. The control aspect is removed from forwarding devices and delegated to an external entity.

- **Network programmability**: The provision of an open API is a core aspect of the SDN architecture. Software and scripts should be able to access, configure, and modify network elements with ease.
- **Network abstraction:** the view of the network is virtualized for any elements of a higher hierarchy. Services and applications are aware of the state of the whole network, but physical attributes and resources are irrelevant for configurations and computations.
- **Logically centralized control:** all forwarding devices of a domain are linked to a controlling entity and are subjected to its enacted policies

This way, SDN provides network-wide visibility and flexible programmability to network administrations. This also removes the differences, makes the network administration independent of data plane devices vendors and allows network administrators to design and control the network with their own applications and respond quickly to changing business needs. SDN opens up the means for new innovation and applications.

SDN is defined by the Open Networking Foundation (ONF) which was founded in 2011 by Microsoft, Google, Facebook, Yahoo, Verizon and Deutsche Telekom. As of 2015, the organization has more than 150 industry members and receives endorsement by several network equipment vendors such as Cisco, Dell, Brocade and HP. An SDN architecture decouples the network control and forwarding functions enabling the network control to become directly programmable. The separation of the control plane from the data plane lays the ground for the SDN architecture. Network switches become simple forwarding devices, and the control logic is implemented in a physical or logical centralized controller. The logically centralized controller dictates the network behaviour and offers several benefits. Firstly, it is simpler and less error-prone to modify network policies through software from a single place without reconfiguring individual devices. Secondly, a control program can automatically react to dynamic changes in the network and thus maintain the high-level policies in place. Thirdly, the centralised control logic has global knowledge of the whole network, including the network topology and the state of the network resources, thus giving flexibility and simplifying the development of more sophisticated network functions. For example, the controller can dynamically adjust flow tables to avoid congestion or apply different routing algorithms to different types of traffic. The ability to program the network to control the underlying data plane is, therefore, the crucial value proposition of SDN [11]. The advantages of SDN in various scenarios (e.g., the enterprise, and the datacenter) and across various backbone networks have already been proven (e.g., Google B4 [9], Huawei carrier network [10]).

As depicted in Figure. 2.1, SDN architecture is divided into three layers, which are infrastructure layer, control layer and application layer.

- **Infrastructure layer (Tier-1)**: This layer consists of the forwarding hardware such as switches/routers and all software interfaces and hardware components in the forwarding hardware. OpenFlow switches can be classified into two types: Software-based and Hardware-based implementation. Software switches are typically well-designed and comprise complete features. Software switches are emerging as one of the most promising solutions for data centers and virtualized network infrastructures. Examples of software-based OpenFlow switches include ofsoftswitch13 [29], Open vSwitch [30], Pantou [31], and Pica8 [32]. Hardware-based Open-

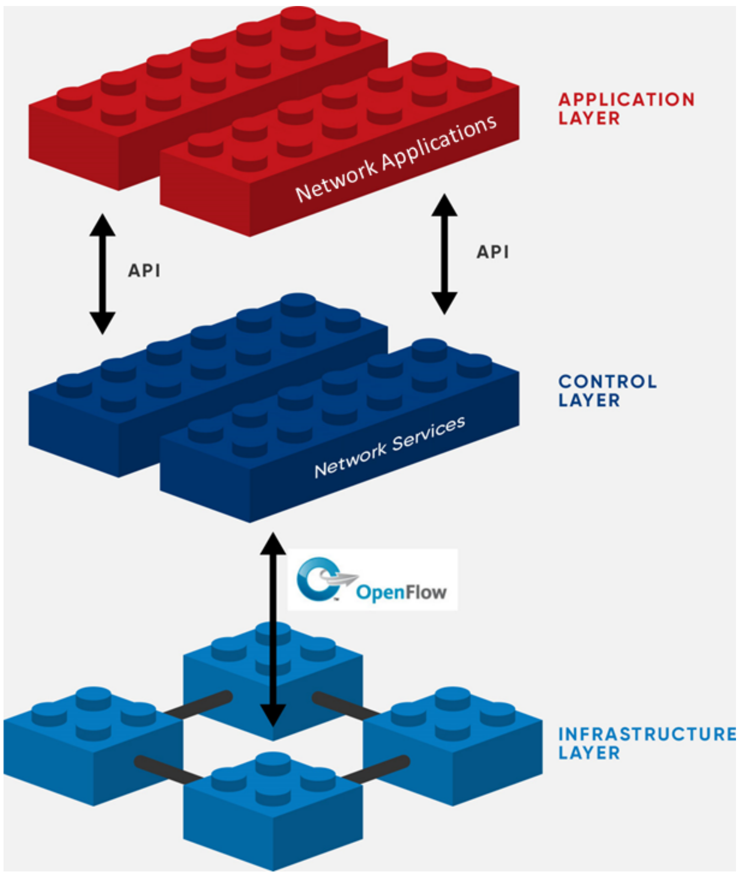


Figure 2.1: A three-layer SDN Architecture [2]

Flow switches are typically implemented as Application-Specific Integrated Circuits (ASICs). They provide line rate forwarding for large numbers of ports but lack the flexibility of software implementations. Various commercial vendors are supporting OpenFlow in their hardware switches (e.g., HP, Pronto, Cisco, Dell, Intell, NEC, and Juniper).

- **Control layer (Tier-2)**: Network intelligence is installed in a software base logically centralized SDN controller. The control layer regulates and manages forwarding hardware, i.e., Tier-1. The controller is the core of SDN networks. It lies between the network devices at the one end and the applications at the other end. An SDN controller takes the responsibility of establishing every flow in the network by installing flow entries on switch devices. One can distinguish two flow setup modes: Proactive vs Reactive. In proactive settings, flow rules are pre-installed in the flow tables. Thus, the flow setup occurs before the first packet of a flow arrives at the OpenFlow switch. The main advantages of this approach is a negligible setup delay and a reduction in the frequency of contacting the controller. However, it may overflow the flow table of the switches. With respect to a reactive approach, a flow rule is set by the controller only if no entry exists in the flow tables and this is performed as soon as the first packet of a flow reaches the OpenFlow switch. Thus, only the first packet triggers a communication between the switch and the controller. The control plane acts as an intermediary layer between the application and data plane. The control plane in SDN is called a controller; it communicates with the application plane via the northbound communication channel and with the switches via the southbound communication channel. In this thesis, we focus on an OpenFlow-related controller because OpenFlow is prominently successful, whereas other approaches are not as successful in practice. To date, various OpenFlow controllers have been publicly released, and most of the controllers currently support OpenFlow version 1.0 [33]. The controllers are summarized in Table 2.1.

- **Application layer (Tier-3)**: Application and services take advantage of control and infrastructure layers. Conceptually, the application layer is above the control layer, and this enables easy development of network

Controller	Open Source	Language	Origin
NOX [34]	Yes	C++/Python	Nicira Networks
POX [35]	Yes	Python	Nicira Networks
Maestro [36]	Yes	Java	Rice University
Beacon [37]	Yes	Java	Stanford University
SNAC [38]	No	C++/Python	Nicira Networks
RISE [39]	Yes	C and Ruby	NEC
Floodlight [40]	Yes	Java	Big Switch Networks
McNettle [41]	Yes	Nettle/Haskell	Yale University
MUL [42]	Yes	C	KulCloud
RYU [43]	Yes	Python	NTT OSRG and VA Linux
OpenDaylight [44]	Yes	Java	Multiple contributors
ONOS [45]	Yes	Java	Multiple contributors

Table 2.1: SDN Controllers

applications. These applications perform all network management tasks . Some examples of SDN application are load balancers, network monitors, and intrusion detection systems (IDS).

2.1.2 OpenFlow Protocol

The OpenFlow protocol [46] is one of the first standardized protocols for SDNs. Even though OpenFlow is not the only available protocol (e.g., Extensible Messaging and Presence Protocol XMPP [47], ForCES [26], Open vSwitch Database (OVSDB) [48], OpFlex [49]), it is considered as standard and supported by multiple companies in their SDN ready solution. OpenFlow was first proposed by McKeown et al. with an objective to enable easy network experiments in campus networks [46] and is currently used in most practical SDNs.

Different versions of the OpenFlow protocol specification are available. The most widely deployed version of OpenFlow is OpenFlow version 1.0 [33], which was released on 31st December 2009. Other versions are 1.1 [3], 1.2 [50], 1.3 [51], 1.4 [52] , and 1.5 [53]. A detailed list of changes included in every version is available in the OpenFlow 1.5.1 specification document [54].

The OpenFlow specification describes an open protocol to allow software applications to program the flow table of different switches. An OpenFlow consists of three mains components: An OpenFlow-compliant switch, a secure channel and a controller. Switches use flow tables to forward packets. A flow table is a list of flow entries. Each entry has match fields, counters and instructions.

According to SDN architecture, an OpenFlow switch is a simple forwarding device that processes incoming packets based on its flow table. As illustrated in Figure 2.2, OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. The switch communicates with the controller, and the controller manages the switch via the OpenFlow protocol.

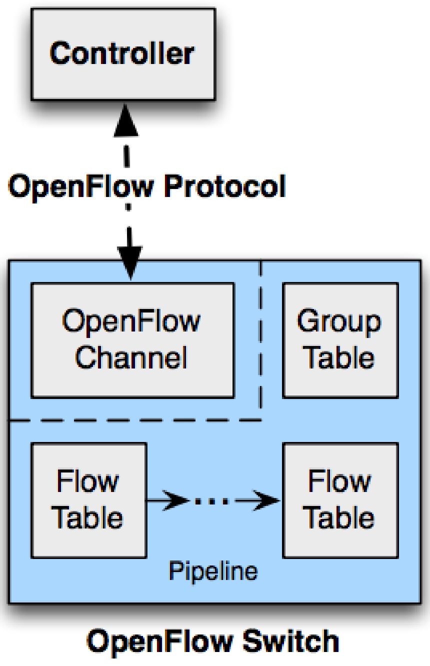


Figure 2.2: OpenFlow Switch [3]

In the OpenFlow switch, flow tables consist of flow entries, each of which determines how packets which belong to a flow, will be processed and forwarded. As illustrated in Table 2.2, flow entries typically consist of three fields:

Match fields	Counters	Instructions

Table 2.2: Main Components of a Flow Entry

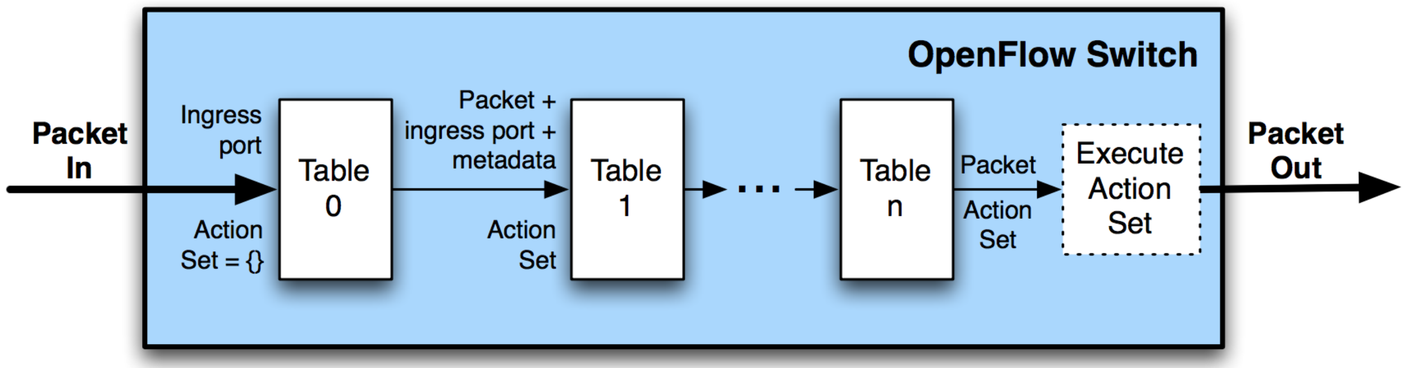
- Match fields: used to match incoming packets. Match fields may contain information found in the 15-tuple packet's header. The packet's header fields are listed in Table 2.3.
- Counters: used to collect statistics for a particular flow, such as the number of received packets, number of bytes and duration of the flow.
- Instructions: a set of instructions or actions, to be applied upon a match; they dictate how to handle matching packets. Two example instructions are forwarding or dropping the packet.

The match fields describe with which packets this entry is associated. They include the ingress port and some specific header fields of packets, such as IP address and Mac address. These fields are set by the network administrator from the controller. They can be set with a specific value or can be wild-carded to match with any flow.

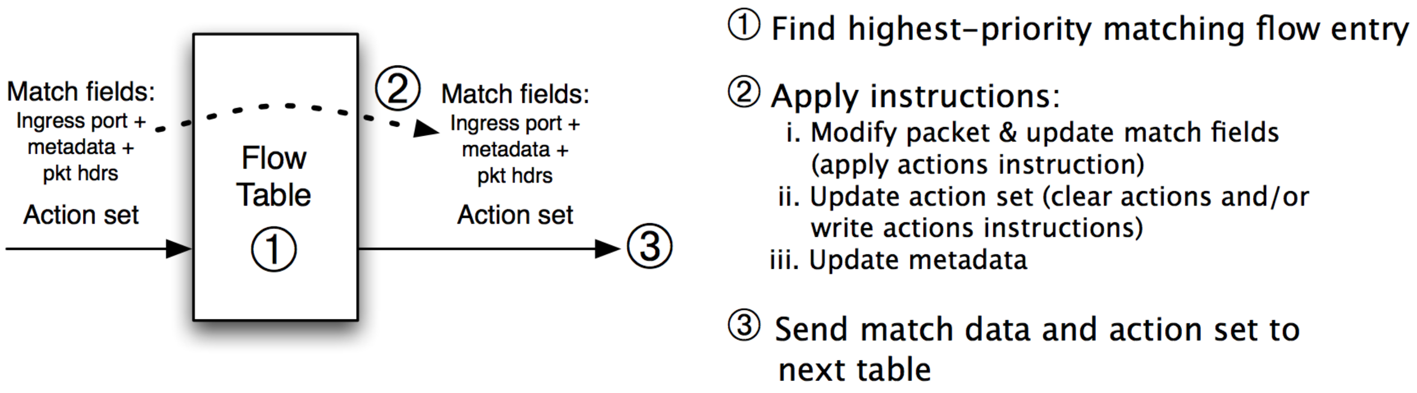
Ingress Port	Metadata	Ethernet Src.	Ethernet Dest.	Ethernet Type	VLAN ID	VLAN Priority	MPLS Label	MPLS Traffic Class	IP Src.	IP Dest.	IP Proto.	IP TOS Field	Transport Src. Port	Transport Dest. Port

Table 2.3: Match Fields in OpenFlow

Figure 2.3 illustrates the packet processing of the pipeline. Before the pipeline begins, the metadata field and the action set for the incoming packet are initialized as empty. The matching process starts from the first flow table (Table 0) to the last flow table (Table n). The packet is matched against the consecutive flow tables from each of which the highest-priority matching flow entry is selected. The pipeline ends when no matching flow table entry is found or no "Goto" instruction is set in the matching flow table entry. At each flow table, the packet will be processed in three steps. Firstly, the packet will be matched with highest-priority matching flow entry. Secondly, a set of instructions will be applied to the packet as seen in Figure 2.3. Finally, the matched packet will be sent to the next flow table for further actions.



(a) Packets are matched against multiple tables in the pipeline


Match fields:
Ingress port + metadata + pkt hdrs
Action set
Flow Table ①
② Match fields:
Ingress port + metadata + pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table

(b) Per-table packet processing

Figure 2.3: OpenFlow Packet Processing Pipeline [3]

The details of the matching process are explained in Figure 2.4. When a packet arrives at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If any matching entries are found, the switch will select the highest prioritized entry, and it applies the appropriate set of instructions, or actions, associated with the matched flow entries. If the flow table look-up procedure does not result in a match, the action taken by the switch will depend on the instructions defined by the table-miss flow entry. This particular entry specifies a set of actions that will be performed when no match is found for an incoming packet, such as dropping the packet, continue the matching process on the next flow table, or forwarding the packet to the controller over the OpenFlow channel. As for the default action, the switch will send an OFPT_PACKET_IN message to the controller to request for a rule or a specific action for the packet. After that, the controller will issue either an OFPT_FLOW_MOD or an OFPT_PACKET_OUT message containing the instructions on how the switch should process that packet.

An essential aspect of SDN architecture is the link between the control and data planes. As forwarding elements are controlled by an open interface, it is important that this link remains available and secure. The control and data planes exchange control messages with each other via the southbound interface using standardised protocols. The OpenFlow protocol, which is standardised by the ONF, is one of the most popular implementations of controller-switch interactions. An OpenFlow switch has to initiate a communication channel with the controller for exchanging control messages. This connection can use plain TCP or be encrypted with TLS. When the connection is established successfully, the switch and the controller send an OFPT_HELLO message to each other to ne-

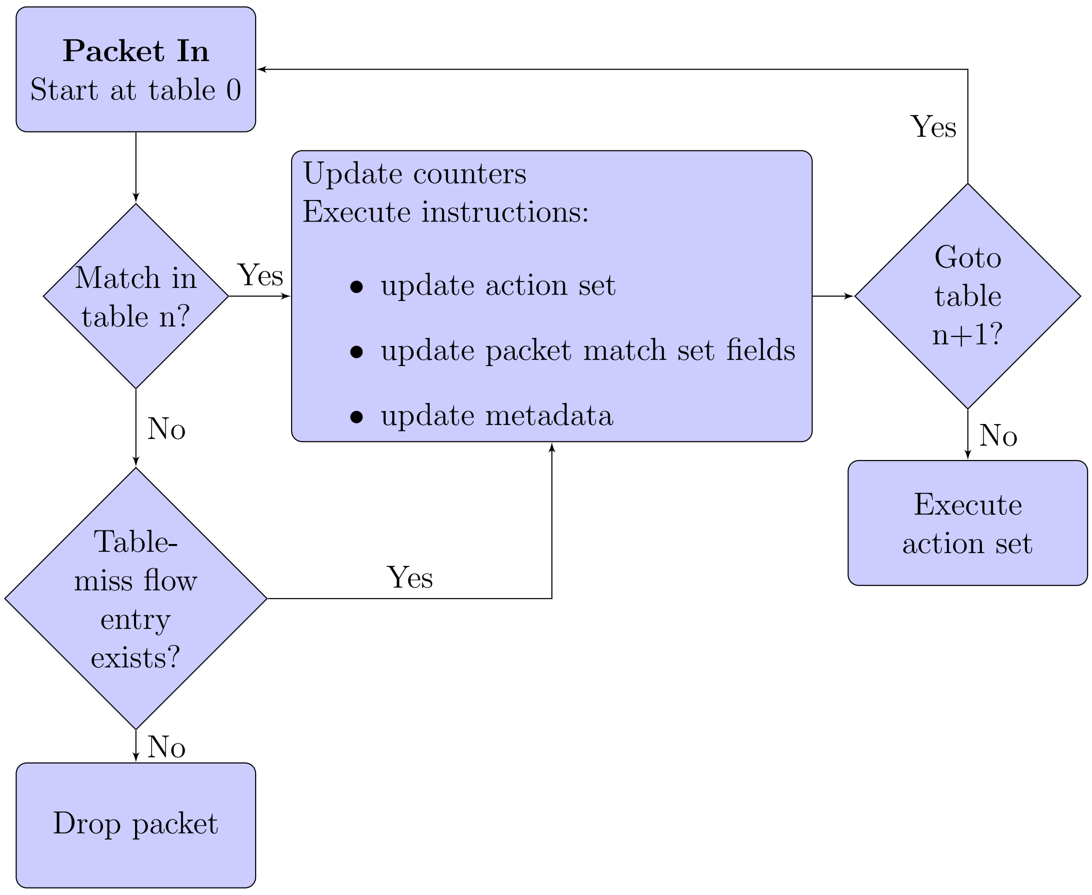


Figure 2.4: OpenFlow Packet Matching Process [3]

gotiate the protocol version for the communication channel. An OFPT_ERROR will be sent to the recipients if any failure happens. After the switch and the controller have configured the channel successfully, OpenFlow messages can be exchanged over the channel.

The OpenFlow controller is responsible for managing the whole network. It consists of one or more software controllers. It controls the forwarding table, gathers information from the data plane and provides information to the services and applications in the application tier about the network operations. Network statistics monitoring is one of the most critical factors of SDN. An OpenFlow switch calculates statistics of the network traffic and provides the information to the controller as requested. In OpenFlow version 1.0, the network statistics consist of flow statistics, table statistics, port statistics, and queue statistics. In the latest OpenFlow version 1.5.1 [54], the switch can also provide other types of statistics and meters to support network monitoring, such as group statistics, action bucket statistics, and per-flow meter. With all of these statistics, the network administrator can know the traffic status of each switch and link and can adjust the network efficiently.

2.1.3 Security in SDN

The SDN concept was initially designed with significant advantages over the traditional network. One of the crucial benefits of SDN is to make the highly vulnerable traditional network more secure and robust. By centralizing the control plane, SDN considerably simplifies the way that we integrate security mechanisms into our network. The evolution of networking brings several advantages, but it also brings the development of the network attacks. Attacks can be initiated from malicious management applications, the controller, and compromised hosts or switches. The main causes of concern lie in the SDN's main benefits: network programmability and control logic centralization. These capabilities introduce new faults and attack planes, which open the doors for new threats that did not exist before or are harder to exploit. The security issues of SDNs has been researched extensively in [11], [12], [13] and [55]. According to Kreutz et al. [12], there are seven main potential threat vectors identified in SDN and summarized in Table 2.4.

No.	Threat Vector
1	Forged or faked traffic flow
2	Attacks on vulnerabilities in switches
3	Attacks on control plane communications
4	Attacks on and vulnerabilities in controllers
5	Lack of mechanisms to ensure trust between the controller and management application
6	Attacks on and vulnerabilities in administrative stations
7	Lack of trusted resources for forensics and remediation

Table 2.4: SDN Threat Vectors

Among these seven threat vectors, number 3,4 and 5 are not present in the traditional network. They are specific to SDNs as they arise from the separation of the control and data plane and the logical centralization of the controllers. Other vectors were already presented in traditional networking. Threat vector number 5 would have the most severe impact on the SDN architecture because it could affect the entire network. Attacks in SDNs can be categorized into three categories: Control plane specific, Data plane specific and Communication channel specific. All of these attacks are summarized in Table 2.5. More details about these attacks can be found in [55].

Attack Plane	Attack Type
Control Plane	DoS Network overview manipulation Unauthorized network mangement Network service neutralization
Communication Channel	Eavesdropping Man-in-the-Middle
Data Plane	Flow table flooding Malformed control message injection Data leakage

Table 2.5: SDN Attack Summary

The controller emerges as a potential single point of attack. Attackers can attack vulnerabilities of controllers and run several dangerous scripts. Therefore, if there are no security settings to protect the controller, it would be under the control of attackers. The controller provides an interface for SDN applications to manage the network. However, this also gives chances for malicious SDN applications to take over the network because of the lack of trust between the controller and SDN applications. Malicious hosts also can cause severe damage to the SDN architecture. Malicious hosts can perform Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks to controllers and other hosts. DoS and DDoS attacks are some of the most dangerous attacks in SDN. It can be done by flooding the network with a large number of forged packets. These packets would trigger the switches to send a large number of requests to the controller for new flow rules. Therefore the control channel bandwidth and the controller CPU resources will be heavily consumed. As a result, the controller would respond slowly to legitimate requests. At the same time, the switches would also suffer from traffic congestion because the packets could quickly exhaust the memory of the flow table storage in the switches. Compromised switches not only have the same capabilities as the malicious hosts, but they are also capable of performing more dynamic and severe attacks. Firstly, they can be used for traffic eavesdropping. Both data and control flows passing through the compromised switches can be replicated and sent to the attacker for further processing. Furthermore, the attacker can interfere with the control traffic passing through the compromised switches to perform man-in-the-middle attacks. By doing so, the attacker can act as the controller to some target switches.

The research in the field of SDN and general security in SDN is still in the early phase. Currently, there are several papers analyzing the security aspect of SDN and proposing new security solutions in the SDN environment. Most of them focus on the security analysis of SDN applications or network policy verification.

2.2 Network Intrusion Detection System

Intrusion detection is the process of monitoring events occurring in a computer or network and analysing them for signs of intrusion. These signs of interventions can be referred to as anomalies. Anomalies can be put into three categories as follows:

- **Point Anomalies**: A point anomaly is an object that is considerably different from other data points.
- **Collective Anomalies**: If there are some linked objects observed as an anomaly against other objects, then they are collective anomalies. Individually observed they might appear normal.
- **Contextual Anomalies**: If an object is abnormal viewed in a defined context (e.g., temporal or spatial attribute), it is regarded as a contextual anomaly.

An Intrusion Detection System (IDS) is based on the hypothesis that an intruder's behaviour will be noticeably different from legitimate behaviour and so it is detectable. The IDS identifies illegal behaviours or attacks and report on them. The IDS has been studied for over 40 years since Anderson's report [56].

Basically, there are six types of intrusions: attempted break-ins, masquerade attacks, penetration of the security control system, leakage, denial of service and malicious user. An overview of an IDS is depicted in Figure 2.5.

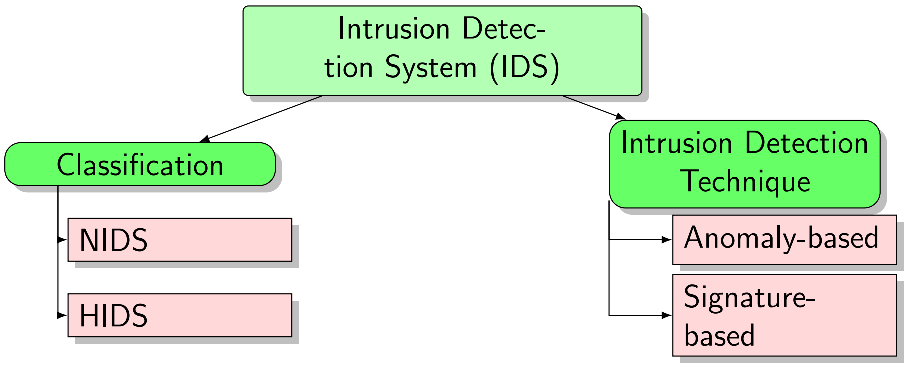

Intrusion Detection System (IDS)
Classification
NIDS
HIDS
Intrusion Detection Technique
Anomaly-based
Signature-based

Figure 2.5: Overview of Intrusion Detection System

There are two popular types of IDS: network-based IDS (NIDS) and host-based IDS (HIDS). In a host-based IDS, some applications will be installed on the individual host or the device on the network and monitor the character of the single host such as the integrity of the system, file changes, network traffics and system logs. Host-based IDS can access information in the host so that it can make more accurate decisions about attacks. However, this system can add performance overheads to the host and thus degrade the overall performance. In network-based IDS, the system will monitor all the network traffic and analyse it for signs of intrusions. It tries to detect malicious activities such as DoS attacks, DDoS attacks and other network attacks. Network-based IDS includes a number of sensors to monitor packet traffic and a number of servers for network management functions.

Based on what data is actually processed, the IDS can be categorized into either signature-based detection or anomaly-based detection. Signature-based detection generally takes intrusion behaviour as patterns and establishes a signature database based on these known patterns. This method then monitors and matches the user's behaviour with the database to detect intrusion. Known attacks can be detected well this way with a low false alarm rate. However, this method cannot detect zero-day attacks which are new and unknown. On the other hand, anomaly-based detection creates a normal behaviour baseline model and then tries to identify any behaviour that deviates from this baseline model. Thus, this method can detect zero-day attacks. The majority of current NIDSs is signature-based IDS. There are several reasons for this reluctance to switch, including the high false alarm rate, difficulty in obtaining reliable training data and network dynamics.

In this thesis, we focus on the development of an anomaly-based Network Intrusion Detection System (NIDS). One of the significant challenges of developing a NIDS is ensuring robustness and effectiveness.

2.2.1 Signature-based Detection

Signature-based detection is the process of detecting packets that have a signature (a pattern of known attacks) in the network traffic corresponding to the rules established in the IDS. Well-written signature rules can perform detection of known attacks with high probability and without mistake. Snort [57] is an open source and well-known example of a signature-based IDS. A signature-based system is at the heart of most commercial IDSs. In an IDS, there are typically at least several thousand rules that are used for detecting potentially malicious attacks and codes. These signatures also help network administrators easily recognize what kind of attack the system is under.

However, a signature-based detection system also has some drawbacks. It cannot detect unknown attacks or known attacks with small variations, so the database must be updated frequently. Many variations of one signature can also make the database grow big and slow down the whole system. It also takes time to create signatures for new exploits. A new attack must be analyzed before developing any new signature to detect it. Attackers will take advantages of this time-lapse, and this can cause serious problems. Therefore, IDS manufacturers have to rush the signature to the market and update their systems regularly. Signature-based IDS can also be bypassed when the attackers encrypt the code. For a higher detection rate, signature-based detection is commonly combined with packet-based detection. In packet-based detection, also named "Deep Packet Inspection" (DPI), both header and payload of a packet are scanned to determine whether a packet is an intrusion or not. All common kinds of known attacks and intrusions can be detected with DPI if the data source delivers an entire network packet for analysing. Header information is mainly helpful to detect attacks aiming at vulnerabilities of the network stack. Payload information is used for detecting attacks aiming at vulnerable applications. To process all payload information, packet-based IDS requires a considerable amount of resources for computation, and it is very time-consuming. These are the disadvantages of packet-based IDS. Several researchers have expended much effort to design an efficient packet-based IDS.

Signature-based intrusion detection systems are implemented using four techniques: pattern matching techniques, rule-based techniques, data mining techniques, and state-based techniques. Pattern matching techniques are commonly deployed on network intrusion detection systems. The attack patterns are usually modelled based on packet headers, packet contents or both of them and then the IDSs to match and identify them. Pattern matching is computationally expensive since there are new types and varied forms of attacks emerging every day. A rule-based technique is one of the earliest used methods for signature-based intrusion detection. The intrusion scenarios are encoded as a set of rules, which are then matched to a network or host traffic data. IDES [58] (Intrusion Detection Expert System) is a rule-based system. IDES is trained to detect known intrusions, vulnerabilities. NIDES [59] (Next-generation Intrusion Detection Expert System) is an extension of the IDES system. NIDES is a hybrid system that has both a rule-based component and a statistical model component for anomaly detection. MIDAS [60] (Multics Intrusion Detection and Alerting System) is an expert system that was developed to perform real-time intrusion detection and misuse detection for Dockmaster, the National Computer Security Center (NCSC) networked mainframe system. State-based techniques use system states and state transitions to detect intrusions. They require finite state machine construction for depicting the states and transitions. The states depict the IDS states, and transitions characterize events that cause the IDS states to change.

2.2.2 Anomaly-based Detection

Anomaly-based detection is an alternative to signature-based detection. It uses the tendency of the attack traffic to determine whether an attack is occurring or not. It will compare definitions of normal activities against observed events to find deviations. According to Maxion and Tan [61], "An anomaly is an event (or object) that differs from some standard or reference event, in excess of some threshold, in accordance with some similarity or distance metric on the event". Anomaly-based IDS was originally proposed by Denning in her report [62]. It is based on the idea that intrusion behaviors differ from legitimate behaviors and are detectable.

A variety of techniques are used for anomaly detection. However, they can be categorized into two main techniques: statistical analysis and ML. ML approaches build models of normal data and then attempt to detect deviations from the normal model in the observed data. ML approaches can be categorized into two strategies: supervised learning and unsupervised learning. Supervised anomaly detection algorithms require a set of labeled network data from which they train their model. However, we do not have either labeled or purely normal data readily available! We must deal with huge volumes of network data, and it is difficult to classify it manually. If the data contains some mislabeled intrusions buried within the training data, future instances of the attacks which will be assumed "normal data" and so may not be detected. We can obtain labeled data by simulating intrusions, but then we would be limited to the set of known attacks. Thus our detection system is restricted to identify only known attacks. Unsupervised anomaly detection takes a set of unlabelled data as input and attempts to find intrusions buried within the data. The basic idea is that since the intrusions are both different from normal and rare, they will be detected as outliers in the data. These algorithms have the major advantage of being able to process unlabelled data and identify some of the intrusions. Anomaly-based IDS can detect a variety of abnormal patterns such as attempted break-ins, DoS attacks, worm, and scanning. This method can help detect unknown attacks, but it also has some disadvantages such as complexity, low detection rate, and a high false alarm. Figure 2.6 shows some common ML techniques for classification of intrusive and non-intrusive behavior.

Nowadays, with the constant increasing of network traffic and network speed, it is challenging for traditional packet-based NIDSs to work well. In high-speed environments, flow-based approaches are introduced as a way to solve this problem. A flow is a set of IP packets that have a set of common properties. The common properties are called flow keys which are the source and destination addresses, source and destination port numbers and IP protocol. The NIDS looks at aggregated information of related packets of network traffic in the form of flow, so the amount of the data to be analyzed is reduced, and network overhead is minimized. The flows provide information about the network connection, which can be represented in a few bytes, to be analyzed rather than packet payload. Flow-based supports anomaly-based NIDS to have the ability to detect anomalies. Flow-based methods focus mostly on DoS attack, scan, worm, and viruses.

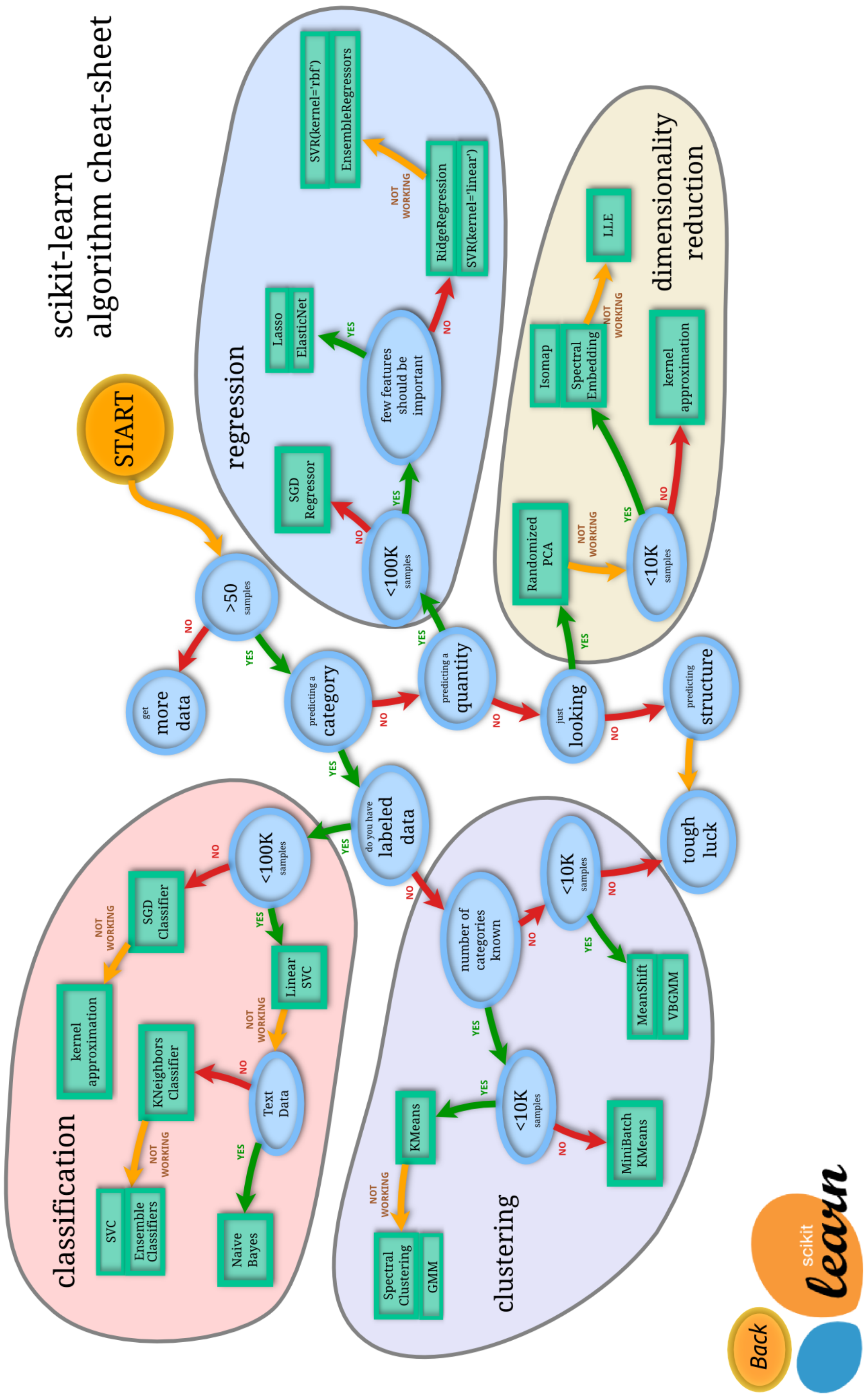


Figure 2.6: Commonly Used Machine Learning Techniques [4]

2.3 Intrusion Detection in the SDN

Researchers have employed classical ML approaches such as Support Vector Machine (SVM), K-Nearest Neighbour (KNN), artificial neural networks (ANNs), and Random Forest for intrusion detection for several years. These proposed methods have achieved various degrees of success while also exhibiting some inherent limitations. These techniques had a quite high false alarm rate and associated computational cost as mentioned in [19]. Researchers have adopted these techniques for intrusion detection under the context of SDN. For anomaly-based detection approaches, SOM and SVM are frequently used because of their high detection accuracy. Braga et al. [63] present a lightweight approach using a Self Organizing Map (SOM) to detect Distributed Denial of Service (DDoS) attacks in the SDN. This approach based on six traffic flow features (Average of Packets per flow, Average of Bytes per flow, Average of Duration per flow, Percentage of Pair-flows, Growth of Single-flows, Growth of Different Ports) gives a quite high detection accuracy. Nam *et al.* [64] propose an approach combining SOM and K-Nearest Neighbors to detect several kinds of DDoS attacks in SDN. This approach can reduce computational overheads while maintaining a suitable ACC of 98.24%. In [14], the authors use four traffic anomaly detection algorithms (threshold random walk with credit-based rate limiting, rate limiting, maximum entropy and NETAD) in the SDN environments. The experiments indicate that these algorithms perform better in the SOHO (Small Office/Home Office) network than in the ISP (Internet Service Provider) and they can work at line rates without introducing any new performance overhead for the home network. In [17] and [18], SVM is introduced to detect DDoS attacks quite efficiently. Winter *et al.* [65]

train a one-class SVM with a malicious dataset in order to reduce the false alarm rate. K-Nearest Neighbour and graph theory are combined to classify DDoS attacks from benign flows in SDNs by AlEroud *et al.* in [66]. In [16], Mukherjee *et al.* investigated the combination of Correlation-based Feature Selection, Information Gain and Gain Ratio for feature reduction. Then the reduced dataset was classified by a Naive Bayes algorithm that yields an accuracy of 97.78%. Javaid *et al.* [67] used a self-taught learning algorithm on the NSL-KDD dataset for intrusion detection. They used soft-max regression as a classifier and achieved an accuracy of 92.98%. Gabriel *et al.* [68] combined neural networks and danger theory for DDoS attack resiliency.

Entropy is used to detect DDoS attacks quite effectively in [15], [69] and [70]. S. M. Mousavi proposed in [15] an early detection method for the DDoS attacks against the SDN controller. This method is based on the entropy variation of the data flows' destination IP addresses. It assumes that the destination IP addresses are evenly distributed in the normal flows, while the malicious flows are destined for a small number of hosts. The entropy will drop dramatically when an attack happens. In [69], a distributed algorithm for entropy-based DDoS attack detection is introduced to reduce the communication overhead. Trung *et al.* [71] combined hard thresholds of detection and a fuzzy inference system to detect the risk of DDoS attacks based on the real traffic characteristics (Distribution of Inter-arrival Time, Distribution of Packet Quantity per Flow and Flow Quantity to a Sever) under normal and attack states. In order to improve the scalability of the native OpenFlow protocol, a combination of OpenFlow and sFlow had been proposed in [72] for effective and scalable anomaly detection and mitigation mechanism in an SDN environment. SPHINX was introduced in [73] to detect both known and potentially unknown attacks within an SDN by leveraging the novel abstraction of flow graphs, which closely approximate the actual network operations.

Recently, DL has developed as an important research trend in the field of intrusion detection. Yin *et al.* [74] propose a DL approach using recurrent neural networks for detection intrusion. They got an accuracy of 83.28% with their experiments on the NSL-KDD dataset. Fu *et al.* [75] propose an IDS using Long short term memory RNN (LSTM-RNN). They achieved an accuracy of 97.52% with the NSL-KDD dataset. An autoencoder (AE) - a form of Artificial Neural Network - is extensively exploited by many researchers for anomaly detection in SDNs. Zhang *et al.* [76] propose a method combining Sparse Autoencoder and Xgboost algorithms to deal with a high-dimensional and unlabelled dataset. They achieve an F1-measure of 91.97%, but their precision is still quite low compared to other state-of-the-art approaches. In [77], the authors propose a DL based approach using a stacked autoencoder (SAE) for detecting DDoS attacks in the SDN. A Non-symmetric deep AE and Random Forest algorithm are combined to detect DDoS attacks in [78]. The authors claim that they can obtain a good classification result whilst significantly reducing the training time.

Nowadays, current research trends focus on detecting DDoS attacks which are some of the most dangerous attacks in SDNs. The controller is a single point of failure in the SDN architecture. Therefore, if intruders trigger the DDoS attacks on the controller and take control of it, they can also control the whole network. In this thesis, our approach aims to detect all types of attacks and discover zero-day attacks in SDNs.

2.4 Intrusion Detection Performance Evaluation

The performance and effectiveness of the NIDS are evaluated by several metrics. A confusion matrix is a table that is often used to describe the performance of the NIDS. A binary confusion matrix is described in Table 2.6.

		Predicted Classes	
		Anomaly	Legitimate
Actual Classes	Anomaly	True Positive (TP)	False Negative (FN)
	Legitimate	False Positive (FP)	True Negative (TN)

Table 2.6: A Binary Confusion Matrix

- TP: the number of anomaly records correctly classified.
- TN: the number of normal records correctly classified.
- FP: the number of normal records incorrectly classified.
- FN: the number of anomaly record incorrectly classified.

For the evaluation purpose, Accuracy (ACC), Precision (P), Recall (R) and F1-measure (F1) metrics are applied. These metrics are calculated as follows:

- Accuracy (ACC): shows the percentage of true detection over total traffic trace,

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%. \tag{2.1}$$

- Precision (P): shows how many intrusions predicted by a NIDS are actual intrusions. The higher P then the lower false alarm is:

$$P = \frac{TP}{TP + FP} \times 100\%. \tag{2.2}$$

- Recall (R): shows the percentage of predicted intrusions versus all intrusions presented. We want a high R value,

$$R = \frac{TP}{TP + FN} \times 100\%. \tag{2.3}$$

- F1-measure (F1): gives a better measure of the accuracy of a NIDS by considering both the precision (P) and the recall (R). We also aim for a high F value,

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} \times 100\%. \tag{2.4}$$

A Receiver Operating Characteristic (ROC) curve is also used to evaluate the detection performance. The ROC curve is created by plotting False Positive Rate (FPR) against True Positive Rate (TPR). The FPR and TPR are calculated as follows:

$$FPR = \frac{FP}{FP + TN} \times 100\%. \tag{2.5}$$

$$TPR = \frac{TP}{FP + TN} \times 100\%. \tag{2.6}$$

The area under the ROC curve (AUC) is a standard measure for classifier comparison. The higher the AUC, then the better is the performance of the method. Figure 2.7 shows an example of the ROC curve in which the AUC is the area below the yellow curve. In practice, we expect to get high AUC and TPR values.

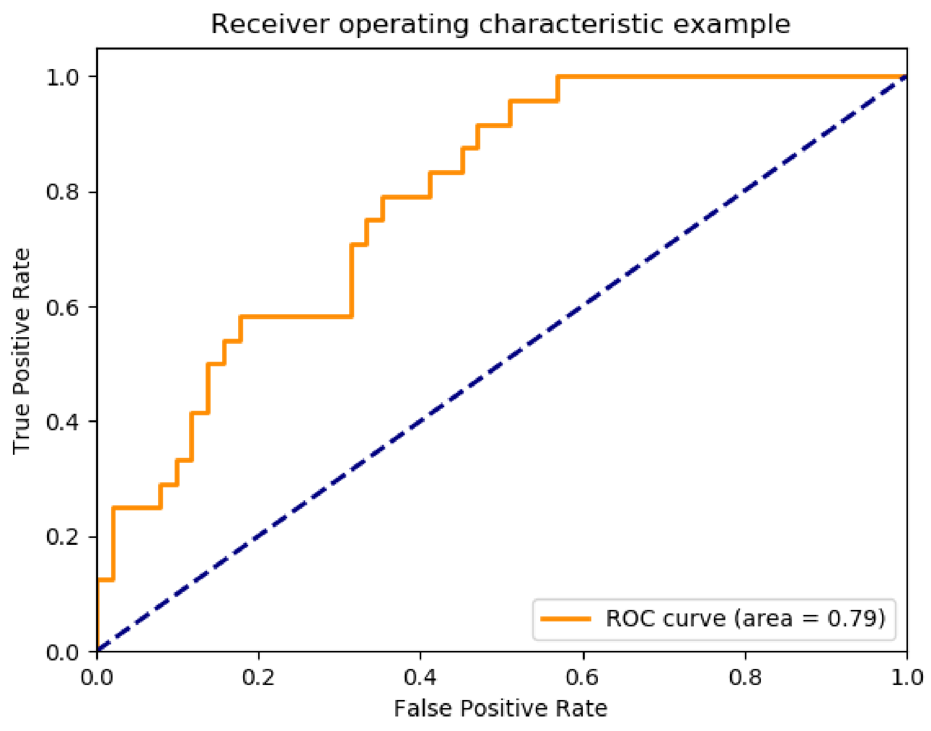


Figure 2.7: The ROC Curve Example. The closer the ROC curve to the top left corner, the better the result is

2.5 SDN-based NIDS: An Example

In this section, we present a simple example of SDN-based NIDS to show the flexibility of SDN in network monitoring and detecting intrusions. As mentioned in the previous sections, the DoS attack is one of the most severe attacks in the SDN. During the DoS attack, many spoofed packets with fake source IP addresses are sent to one host or one specific destination IP address. Therefore, in this experiment, we monitor one type of network information that is the IP destination addresses. As described in the previous section, any unmatched packet will be sent to the SDN controller for routing advice. The IP destination address will be extracted from PACKET_IN messages sent to the SDN controller.

Entropy is commonly used to detect randomness. The less the randomness is, the less the entropy is and vice versa. Therefore, entropy-based approaches have significant advantages in DoS attack detection ([15], [69], [70]). When legitimate traffic is sent in a network, the entropy values are relatively close and smooth. In the case of DoS attacks, the entropy values will drop significantly when a large number of packets are attacking one host or a subnet of hosts.

Let I be a set of destination IP addresses and let W be the window containing a packets' destination IP address x and the occurrence number y. Then, the probability of x_i happening in W is p_i and the entropy is denoted as H.

$$I = \{x_1, x_2, x_3, ..., x_n\} \tag{2.7}$$

$$W = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n)\} \tag{2.8}$$

$$p_i = \frac{y_i}{n} \tag{2.9}$$

$$H = -\sum_{i=1}^{n} p_i \log p_i \tag{2.10}$$

The entropy will be at its maximum if all elements have equal probabilities. If an element appears more than others, the entropy will be lower. As mentioned above, a non-match packet will be sent to the controller for further processing. In DoS attacks, hackers usually use spoofed IP packets so that these packets will be sent to the SDN controller for forwarding advice. The controller can calculate the entropy with information extracted from these packets and detect DoS attacks.

2.5.1 Experimental Setup

In this experiment, Mininet [79] is used to virtualize a complete SDN network. Mininet is an open-source network emulator which runs a collection of end-hosts, switches, routers and links on a single Linux kernel. A large network with different topologies can be emulated and tested with Mininet. Mininet network behaves just like a real network, so the code developed in Mininet can be applied to any real network. Mininet also supports OpenFlow version 1.0, so it is quite easy for us to do SDN experiments. In this experiment, a small scale network with 1 controller, 9 switches and 64 hosts is emulated by Mininet (see Figure 2.8). A POX controller [35], which is written in Python, is used in this experiment to control the emulated SDN network and collect network information. Scapy [80] is used here for packets and spoofing IP address generation.

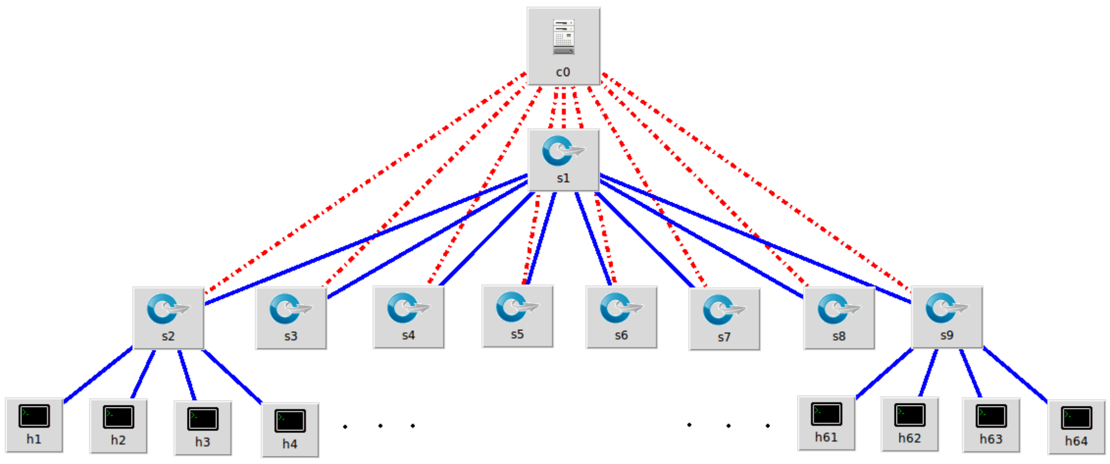


Figure 2.8: The Emulated Network Topology

As seen in Figure 2.8, all the OpenFlow switches are controlled by the POX controller (c0). The POX controller controls traffic flows between hosts in the emulated network.

In all test cases, Scapy sends randomly 4000 UDP packets to 64 hosts in the network to create legitimate traffic. DoS attacks are generated by Scapy by sending 500 UDP packets to a single host (i.e., 10.0.0.1) in the network. Every 50 PACKET_IN messages will be parsed for their destination IP address, and the entropy of the list will be computed. Normal traffic is run on all switches with randomly generated packets going to all hosts. Attack traffic is run manually from one host. Each test case was run ten times to get mean values. Two types of attack rate are tested in this experiment.

- In the first test case, an attack with 25% rate attack is performed. This results in having about 12 attack packets in a window of 50 packets.
- In the second test case, an attack with 75% rate attack is performed. This results in having about 39 attack packets in a window of 50 packets.

2.5.2 Simulation Results

In this section, we examine the result of the attack. Figure 2.9 is a result of 10 runs with 4000 packets per test. Each point on the horizontal axis shows a window of 50 packets and the vertical axis indicates the entropy for that window. The graph's data are the mean values over ten runs.

In Figure 2.9, the differences among legitimate, 25% rate attack and 75% rate attack traffic are shown clearly. When the DoS attack begins, the entropy starts dropping and remains quite low during the attack. The entropy drops significantly so that we can identify the time and duration of the attack easily.

Comparing the 25% rate attack and 75% rate attack entropies, it can be seen that the entropy drops deeper. The window of the attack is also narrower compared to 25% rate DoS attack. Because of the increase of the attack rate, the randomness of the attacked IP address decreases and the entropy drops dramatically. In this graph, we can see that legitimate and attack traffic are significantly different from each other. A threshold which is 1.3 can be set for attack detection. With this threshold, all attack packets can be detected with 100% detection rate.

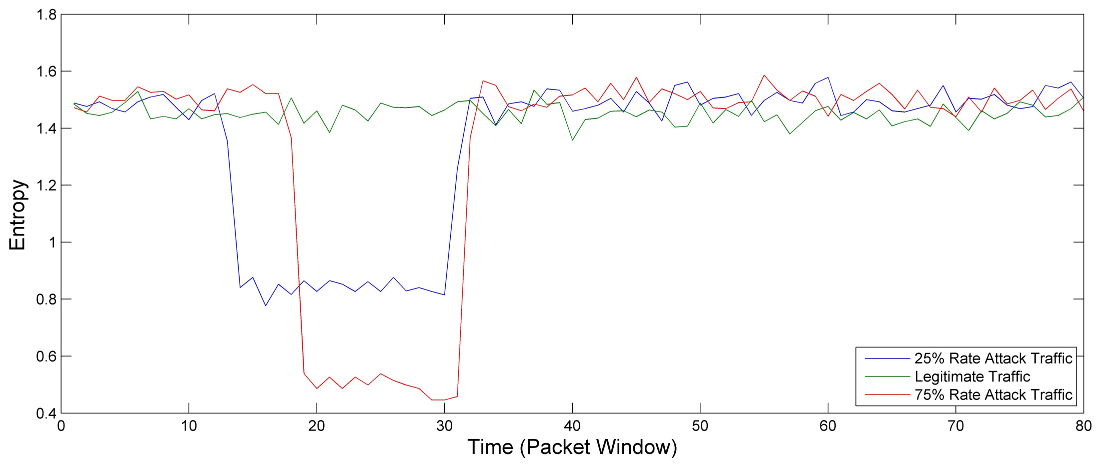


Figure 2.9: Entropy Comparision for Legitimate, 25% Rate DDoS Attack and 75% Rate Attack Traffic

The entropy method is quite lightweight and flexible for detecting some simple networks attacks but it cannot detect more complex attacks. Although the entropy algorithm can identify the attack patterns, there are some benign network anomalies, which affect the network metrics in the same way of malicious anomalies. For example, a Flash Crowd anomaly would cause a massive drop in the destination IP and destination port metric, like a DDoS. These benign network anomalies can increase the false positive rate of the detection system. In another case, Slow DDoS attacks can be performed by attackers. These attacks use slow traffic that mimics legitimate traffic, so they are tough to detect and mitigate. Besides, choosing a proper hard threshold is also a big problem for this method. It requires significant pre-knowledge about the network to decide the alarm threshold. Network traffic is dynamic and constantly changing so the hard threshold cannot detect attacks effectively.

2.6 Conclusion

In this chapter, we discussed in detail the SDN architecture and the OpenFlow protocol. Some major security concerns in the SDN architecture are also presented in this chapter. This chapter also provided an overview of the NIDS and some related work in SDNs. The flow-based NIDS will be focused as the primary research of the thesis. We describe and explain the use of some metrics in evaluating the performance of the DL algorithm as well as the NIDS. At the end of this chapter, we introduced an example of SDN-based NIDS to show the flexibility of the SDN paradigm in monitoring and detecting network attacks.

Chapter 3

Deep Learning

In This Chapter:

The DL overview is introduced for easy understanding of this thesis. In addition, we also introduce the evaluation methodology and the network datasets used in this thesis.

3.1 Notations

Throughout this thesis, we use the following mathematical notations. In general, the following rules are used for number and arrays:

- Non-bold letters (e.g., x, y, I) are for scalars.
- Bold small letters (e.g., $\mathbf{w}$) denote column vectors. A row vector is denoted by its transpose (e.g., $\mathbf{w}^T$).
- Bold capital letters (e.g., $\mathbf{W}$) denote matrices.

3.2 Introduction

In recent years, DL has emerged as one of the hottest technologies and somewhat of a buzzword. So what is DL? And what is the difference between Artificial Intelligent (AI), ML and DL? Figure 3.1 shows the relationship of these terms. As we can see, DL is a subset of ML, and then ML is a subset of AI.

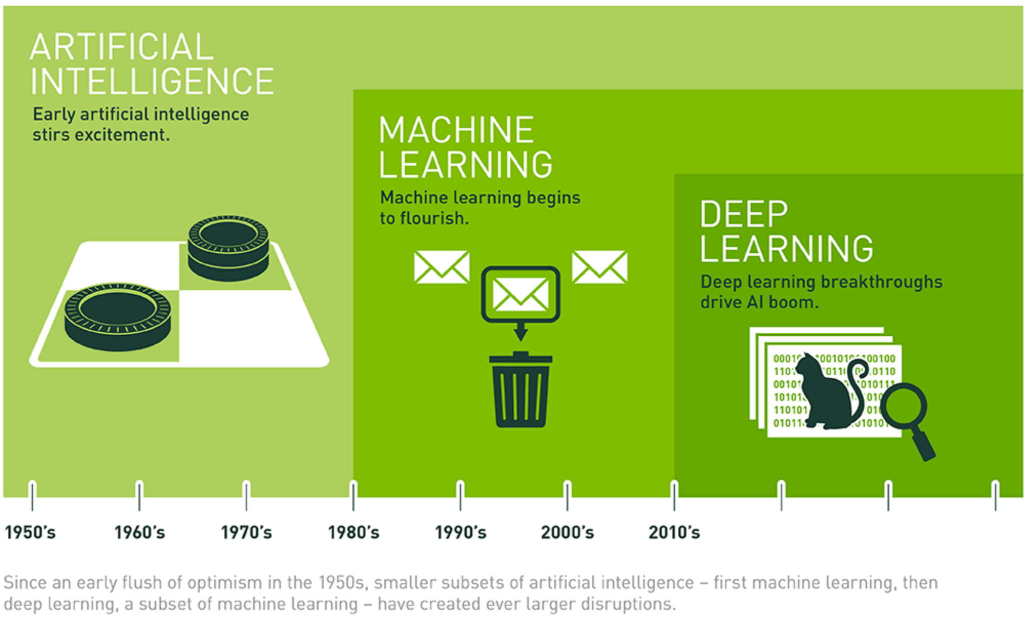

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Figure 3.1: The AI, ML and DL Relationships [5]

AI is a broad concept that means incorporating human intelligence into machines. The intention of ML is to give machines the ability to learn and make decisions by themselves using the provided data. ML was inspired by the structure and function of the human brain which is a highly interconnected system of neurons. A neuron is a basic computational unit of the human brain. Figure 3.2 shows a drawing of a biological neuron and how it works in practice. The neuron receives input signals from its dendrites and then produces output signals through its axon. The axon connects to dendrites of other neurons, so the output signals of this neuron will become the input signals of the others.

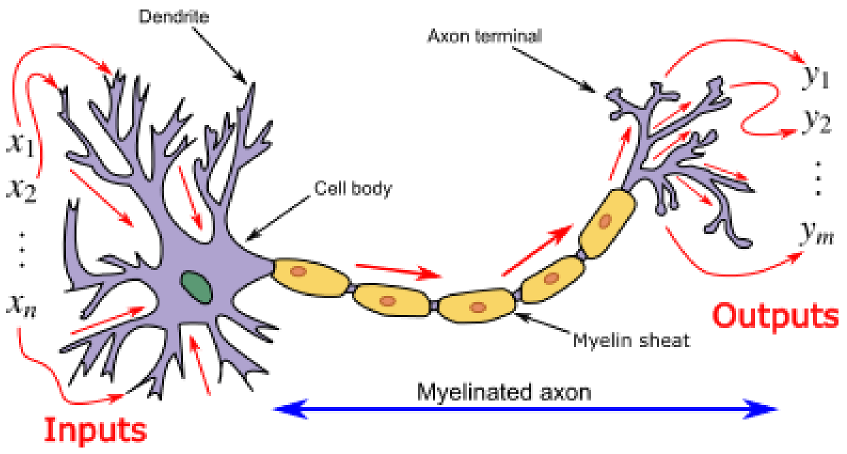


Figure 3.2: A Drawing of a Biological Neuron [6]

A common model of an artificial neuron is described in Figure 3.3. The artificial neuron also receives inputs and then computes an output as the human neuron.

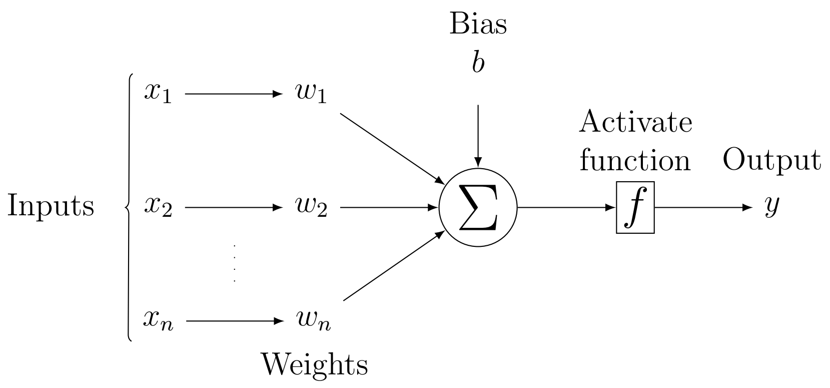


Figure 3.3: The Single Artificial Neuron Structure

The working of a neuron can be presented mathematically as in Equation 3.1.

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b), \tag{3.1}$$

where $f(\cdot)$ is a non-linearity activation function, $\mathbf{w}$ is a weight vector, $\mathbf{x}$ is an input vector and b is a bias. Table 3.1 shows some of the most common activation functions.

Function f(x)	**Definition**	**Range**
Sigmoid	$\frac{1}{1+e^{-x}}$	(0,1)
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1,1)
Rectified Linear Units (ReLU)	$max(0,x)$	(0,∞)

Table 3.1: Definitions of Activation Function

The history of ML development can be dated back to 1957 with an introduction of the perceptron learning algorithm by Frank Rosenblatt. However, this perceptron algorithm could not solve some simple non-linear problems at that time. This lead to a period of time called “The first AI Winter”. In 1986, the Multi-layer Perceptron (MLP), which is a neural network with many hidden layers, was introduced in [81] to solve non-linear problems. The MLP attracted many researchers at that time, but several problems like lack of processing power, lack of data, overfitting and vanishing gradient prevented the development of the MLP. Besides, traditional MLP algorithms are limited in their ability to process raw data. Considerable domain expertise is required to design features as inputs for the MLP system. Despite some achievements in this period, “The second AI Winter” happened, and many researchers changed their focus to Kernel Machine methods like Support Vector Machine. In 2012, Convolutional Neural Networks

(CNN) were introduced in [82] and have had achieved many achievements since then [21] [83] [84].

In general, DL is an ML approach. DL algorithms use a neural network (NN) to learn associations between inputs and outputs. DL refers to creating a deep NN (DNN) that has a huge amount of hidden layers. The NN consists of several artificial neurons arranged in a series of layers. The most common type of the NN is a fully-connected NN in which neurons between two adjacent layer are fully connected. A basic structure of the fully-connected NN can be seen in Figure 3.4.

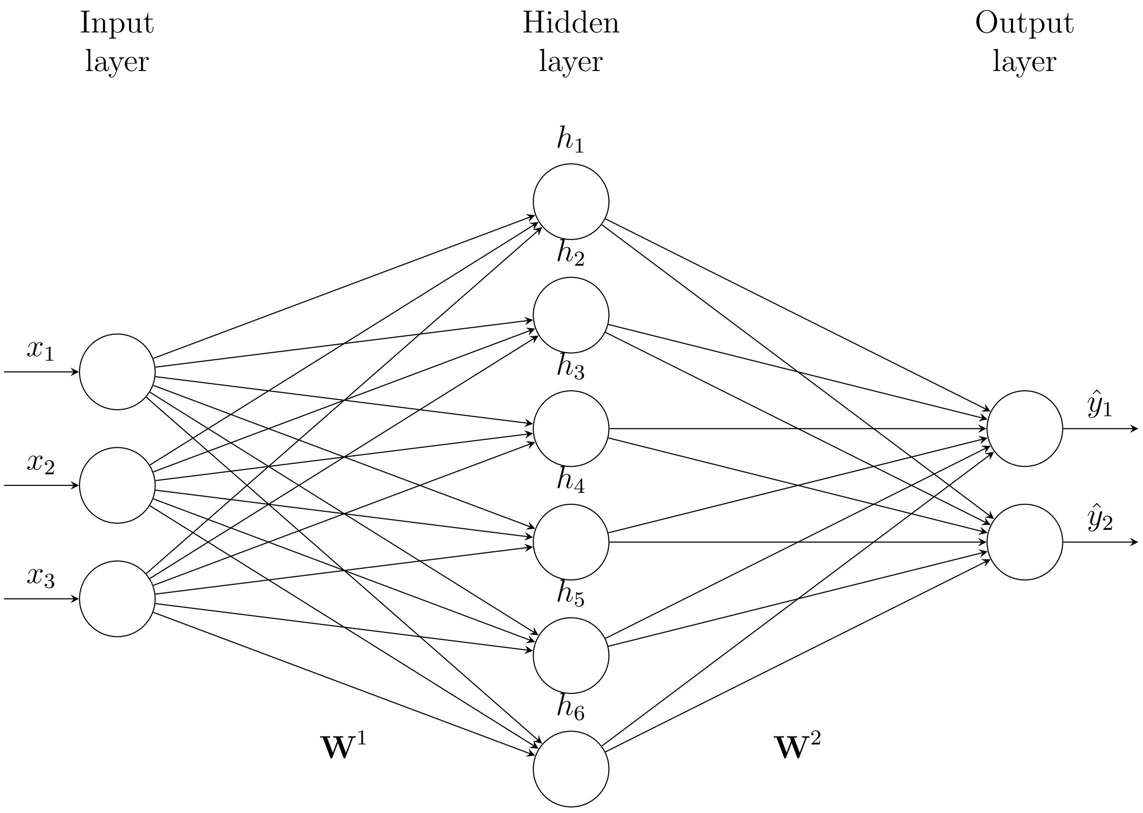


Figure 3.4: The NN Architecture with an Input Layer, a Hidden Layer and an Output Layer

The first layer, known as the input layer, receives information (x_1, x_2, x_3) from the outside environment for the learning process. On the opposite side, the last layer of the NN, known as the output layer, gives us responses (y_1, y_2) to the learned information. In between the input and output layers are hidden layers. The connection from one neuron to another is represented by a number called a weight.

The DL algorithm can be categorized into different types based on its architectural designs. Figure 3.5 presents an overview of DL algorithms that are categorized into two categories: Generative and Discriminate Architectures.

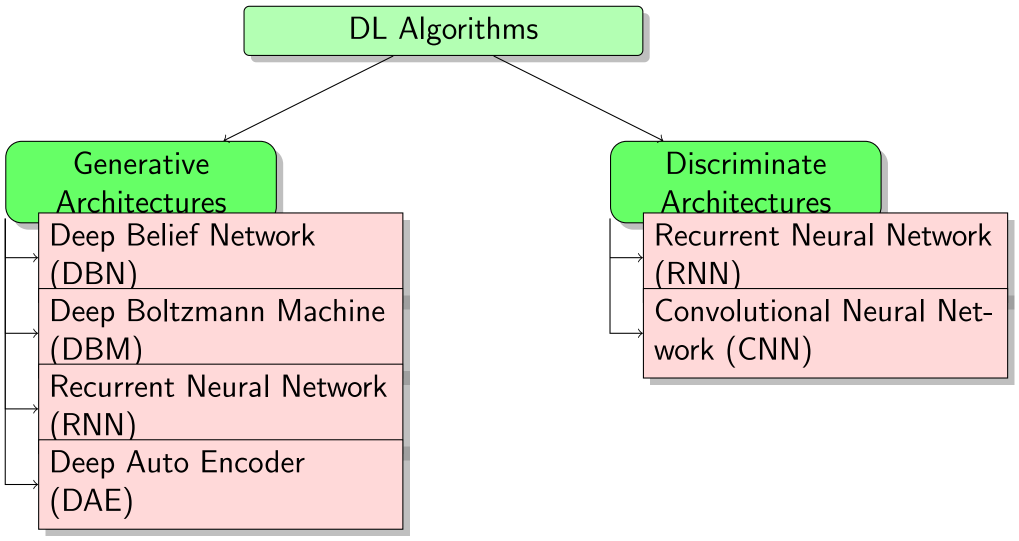

DL Algorithms
Generative Architectures
Deep Belief Network (DBN)
Deep Boltzmann Machine (DBM)
Recurrent Neural Network (RNN)
Deep Auto Encoder (DAE)
Discriminate Architectures
Recurrent Neural Network (RNN)
Convolutional Neural Network (CNN)

Figure 3.5: Overview of The DL Algorithm

The DL yields state-of-the-art results for a wide range of applications such as speech recognition [20], natural language processing [22], and image classification [82]. The success of DL has been facilitated by several factors: better hardware including GPUs, bigger datasets, better regularization and optimization methods. The DL algorithm is expected to improve the field of intrusion detection.

3.3 Advantages and Disadvantages of DL

DL has the ability to learn high-level representations automatically from raw data, while ML requires handcrafted features as input. Each layer in a DNN is a non-linear module that transforms the representation at one layer to a more complex/abstract level. By that way, very complex functions can be learned. With the development of the DL, we now no longer spend time for feature engineering and focus on optimizing the DNN architecture. This is one of the main benefits of DL over traditional ML algorithms. The difference between the ML and the DL is depicted in Figure 3.6.

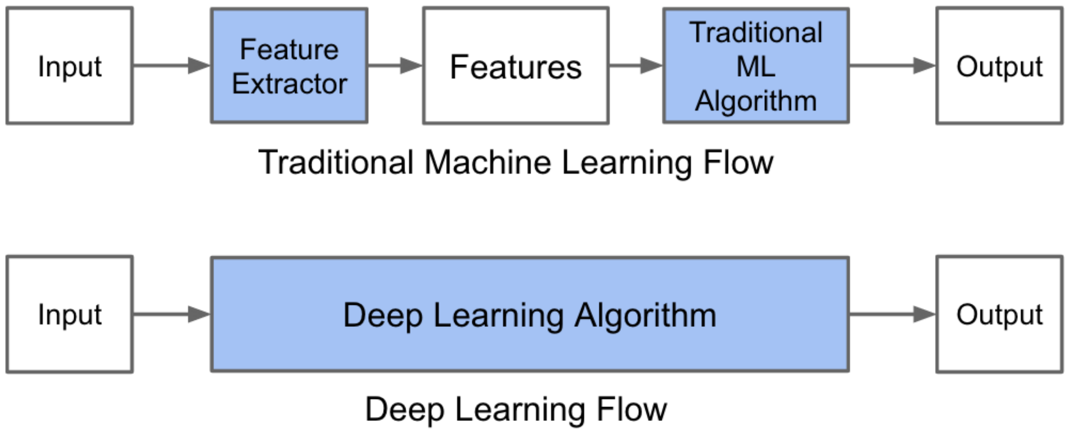


Figure 3.6: The Difference Between Traditional ML and DL Algorithms

However, DL also has several disadvantages as follows:

- DL algorithms require much more data than traditional ML algorithms.
- DL algorithms are computationally expensive. DL needs high-performance hardware and much more time to train compared to traditional ML algorithms.

- DL is a “Black Box” in nature. It means that we cannot explain what happens inside the DNN and how it comes up with its outputs.

3.4 Styles of Learning

ML/DL algorithms can be categorized into two groups by their learning style as follows:

- **Supervised Learning:** Input data or training data has a known label associated with each sample. Given a set of data points $\{x_1, x_2, ..., x_n\}$ associated to a set of outcomes $\{y_1, y_2, ..., y_n\}$, we want to build a classifier that learn how to predict y from x. During the training process, an ML/DL model will be optimized to minimize the distance (or error) between the known label and the predicted label.
- **Unsupervised Learning:** Input data or training data does not have any known label. Given a set of unlabeled data points $\{x_1, x_2, ..., x_n\}$, we want to find hidden patterns of the data. During the training process, an ML/DL model will try to learn structures, patterns or general rules of the input data.

3.5 Training and Testing Neural Networks

One of the most common forms of ML/DL is supervised learning. The NN training process of supervised learning can be separated into six steps as follows:

- **Collecting and Processing Training Data:** Training data is a crucial part of preparing for the training process. All the collected data must be cleaned, normalized and labeled.

- **Defining the NN Architecture:** The number of neurons at each layer must be determined at this step. For the input layer, the number of neurons is picked based on the size of input data (features). The number of neurons at the output layer represents the class label of the training data. The number of hidden layers and the number of neurons in each hidden layer can be varied to find the best architecture for each task. There is no architecture that fits for all different tasks.

- **Feed forward Process:** Given a training set $\{(\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_p}, \mathbf{y_p})\}$ consisting of p ordered pairs of $n-$ and $m-$dimensional vectors, which are called the inputs and outputs. The input $\mathbf{x_i}$ is fed into the NN and propagated through the network to compute the output $\mathbf{\hat{y}_i}$. Referring to Figure 3.4, the output $\mathbf{\hat{y}_i}$ can be computed by

$$\mathbf{h} = f(\mathbf{W}^1\mathbf{x_i}), \tag{3.2}$$

$$\mathbf{\hat{y}_i} = f(\mathbf{W}^2\mathbf{h}), \tag{3.3}$$

where $f(\cdot)$ is a nonlinearity function, and $\mathbf{W}^1$ and $\mathbf{W}^2$ are weight matrices. During the training process of supervised learning, a loss function is introduced to compute the distance (or error) between the predicted output and the desired output. The most common loss function is the Mean Square Error (MSE) which is computed in Equation 3.4.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3.4}$$

where N is the number of observations, y_i and $\hat{y}_i$ are the true and predicted values respectively.

- **Backpropagation Process:** The goal of the training process is to minimize the loss function by modifying the NN's internal adjustable parameters (or weights). Let $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ is a loss function in which $\mathbf{W}$ is a weight matrix, $\mathbf{b}$ is a bias vector, and $\mathbf{X}, \mathbf{Y}$ are matrices of input and output samples respectively. The gradient is computed by the backpropagation procedure [81] which uses the chain rule to compute the derivative. In practice, a stochastic gradient descent (SGD) is a commonly used optimization algorithm to compute errors, gradients and then adjust the weights. The weight will be adjusted in the opposite direction to the gradient vector as in Equation 3.5. For a large dataset, computing the loss and gradient over the whole dataset is infeasible. Therefore, in practice, data is normally divided into small batches for training.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha \frac{\partial J}{\partial \mathbf{W}_t}, \tag{3.5}$$

  where α is a scalar variable called learning rate.

  The weight is computed and updated backwards from the output layer to the input layer. That is the reason why it is called backpropagation.

- **Validation Process:** After training, another set of samples (testing set) is used to evaluate the trained model performance. This testing set is not used for training. Thus, the model performance on the testing set shows the generalization of then trained model. It is expected to achieve high accuracy on both training and testing sets.

- **Hyper-parameter Tuning:** All parameters like learning rate, batch size, number of neurons, and number of layers are termed as hyper-parameters. After the validation process, hyper-parameters can be tuned to achieve better performance for the NN. Tuning network hyper-parameters is a quite challenging task and is one of the main research of the DL. Choosing the learning rate is an important step in the training process. It takes a lot of time and many experiments to decide the best learning rate for each problem.

During the training process, we usually have to encounter an overfitting problem. Overfitting happens when the model tries to fit all the training data including noise (see Figure 3.7). As a result, the trained model performs well on the training data but very poor with new data. In order to solve this problem, several techniques have been introduced such as regularizer, dropout, and early stopping. Dropout technique is one of the most common methods among them. A predefined percentage of NN will be terminated randomly at each epoch in Dropout technique.

Figure 3.8 describes the complete training and testing processes of ML/DL algorithms.

3.6 Network Datasets

Anomaly detection techniques require large numbers of existing benign and suspicious activities to build detection models. Currently, there are only a few public datasets available for intrusion detection evaluation (e.g., the KDD Cup 99 dataset [85], the NSL-KDD dataset [86], DAPRA [87], the ISCX 2012 Intrusion

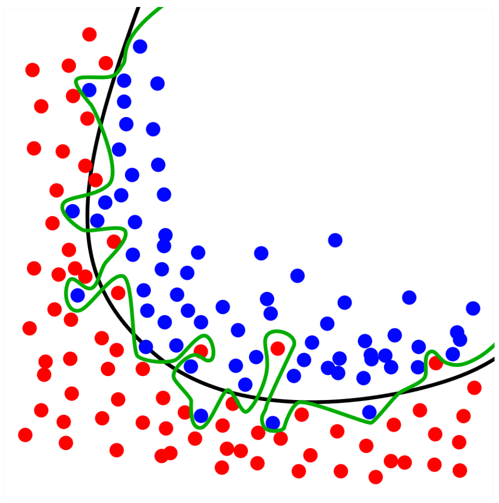


Figure 3.7: The green line represents an overfitted model, and the black line represents a regularized model. While the green line best follows the training data, it is too dependent on that data, and it is likely to have a higher error rate on new unseen data, compared to the black line [7]

Detection [88] and the CICIDS2017 dataset [89]). In this thesis, the NSL-KDD and CICIDS2017 datasets are chosen for evaluation purposes.

3.6.1 NSL-KDD Dataset

Among the above datasets, the KDD Cup 99 dataset and the NSL-KDD dataset have been commonly used in the literature to assess the performance of NIDSs. The KDD Cup 99 dataset is one of the most popular datasets and is widely applied to evaluate the performance of intrusion detection systems. However, this dataset suffers from the redundancy of records that makes the classifier fail to deliver better accuracy. The NSL-KDD dataset is introduced by Tavallaee *et al.* to resolve this redundancy issue.

SDNs are a new environment, and so the dataset for them are still very rare and unpublished. The NSL-KDD dataset is still considered as a state-of-the-art

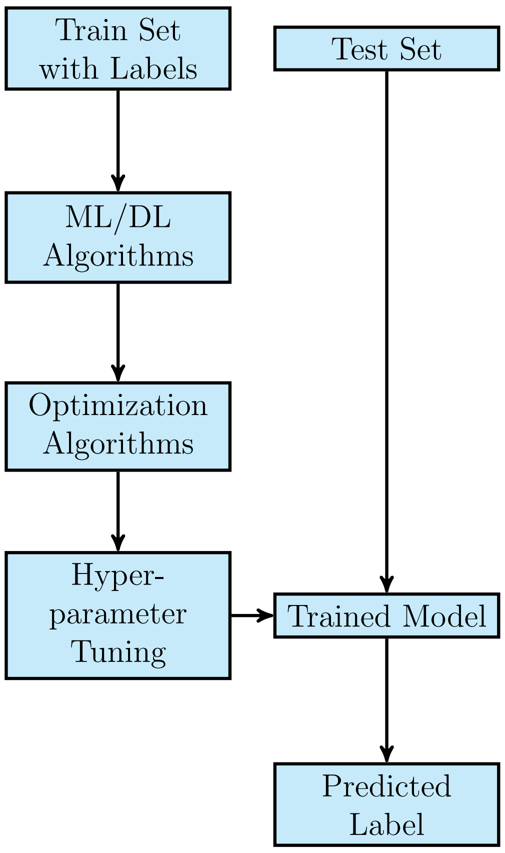


Figure 3.8: ML/DL Training and Testing Phase

dataset by many researchers to evaluate their approaches, and so we also choose it for our experiment. This dataset contains 125,973 records for training and 22,544 records for testing. Details about the distribution of this dataset are presented in Table 3.2. The number of legitimate and anomaly traffic is balanced to make sure that the ML/DL algorithms can be trained properly.

Dataset	Legitimate Traffic	Anomaly Traffic
KDDTrain+	67,343	58,630
KDDTest+	9,711	12,833

Table 3.2: The NSL-KDD Dataset Distribution

Each traffic sample in this dataset has 41 network features (see Table 3.3) that are categorized into three types of features: basic features (1 - 9), content-based features (10 - 22) and traffic-based features (23 - 41). The details of these features can be found in [86].

No.	Feature Name	No.	Feature Name
1	Duration	**22**	Is_guest_login
2	Protocol_type	**23**	Count
3	Service	**24**	Serror_rate
4	Flag	**25**	Rerror_rate
5	Src_bytes	**26**	Same_srv_rate
6	Dst_bytes	**27**	Diff_srv_rate
7	Land	**28**	Srv_count
8	Wrong_fragment	**29**	Srv_serror_rate
9	Urgent	**30**	Srv_rerror_rate
10	Hot	**31**	Srv_diff_host_rate
11	Num_failed_logins	**32**	Dst_host_count
12	Logged_in	**33**	Dst_host_srv_count
13	Num_compromised	**34**	Dst_host_same_srv_rate
14	Root_shell	**35**	Dst_host_diff_srv_rate
15	Su_attempted	**36**	Dst_host_same_src_port_rate
16	Num_root	**37**	Dst_host_srv_diff_host_rate
17	Num_file_creations	**38**	Dst_host_serror_rate
18	Num_shells	**39**	Dst_host_srv_error_rate
19	Num_access_files	**40**	Dst_host_error_rate
20	Num_outbound_cmds	**41**	Dst_host_srv_rerror_rate
21	Is_host_login		

Table 3.3: The NSL-KDD Dataset Features

Attacks in the dataset are divided into four categories according to their characteristics. The details of each category are described in Table 3.4. Some specific attack types (written in bold) in the testing set do not appear in the training set, and that makes the detection task more realistic.

The DNN requires each record in the input data to be represented as a vector of real numbers. Thus, every symbolic feature in a dataset is first converted into

Category	**Training Set**	**Testing Set**
DoS	back, land, neptune, pod, smurf, teardrop	back, land, neptune, pod, smurf, teardrop, **mail-bomb, processtable, udpstorm, apache2, worm**
R2L	fpt-write, guess-passwd, imap, multihop, phf, spy, warezclient, warezmaster	fpt-write, guess-passwd, imap, multihop, phf, spy, warezmaster, **xlock, xsnoop, snmpguess, sn-mpgetattack, httptun-nel, sendmail, named**
U2R	buffer-overflow, loadmod-ule, perl, rootkit	buffer-overflow, load-module, perl, rootkit, **sqlattack, xterm, ps**
Probe	ipsweep, nmap, portsweep, satan	ipsweep, nmap, portsweep, satan, **mscan, saint**

Table 3.4: Attacks in The NSL-KDD Dataset

a numerical value. The NSL-KDD dataset contains both numerical and symbolic features. These symbolic features include the type of protocol (TCP, UDP, and ICMP), the service type and the TCP status flag. After converting all symbolic attributes into numerical values, every feature within each record is normalized by the respective maximum value and therefore falls into the same range of [0-1] by Min-Max scaling. Its mathematical equation is given as:

$$x^{'} = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{3.6}$$

where $x^{'}$ is the normalized value, and x is the original value.

In Table 3.5, we give some examples of the NSL-KDD dataset for more understanding.

Duration	**Protocol Type**	**Service**	**Flag**	**Src_Bytes**	**Dts Bytes**	**Land**	**...**	**Dst_host_srv_ rerror_rate**	**Label**
0	TCP	ftp_data	SF	491	0	0	...	0	normal
0	UDP	other	SF	146	0	0	...	0	normal
0	TCP	private	S0	0	0	0	...	0	neptune
0	TCP	http	SF	232	8153	0	...	0.01	normal
0	ICMP	ecr_i	SF	1032	0	0	...	0	smurf

Table 3.5: The NSL-KDD Dataset Examples

3.6.2 CICIDS2017 Dataset

As mentioned in [89], most of the current network dataset are out-of-date and not reliable enough, so the CICIDS2017 dataset was proposed as a new benchmark dataset. The CICIDS2017 dataset is claimed to be most updated with all common attacks and real-world traffic. This dataset covers seven types of common attack families (i.e., Brute Force Attack, Heatbleed Attack, Botnet, DoS Attack, DDoS Attack, Web Attack, and Infiltration Attack). This dataset contains seven small datasets with different attack scenarios. All of these datasets are labeled and saved in CSV format. Each flow sample in the CICIDS2017 dataset contains 80 flow features which are defined and explained in detail in [90].

In this thesis, we choose Wednesday and Friday datasets focusing on DoS, Heartblead, Slowloris, Slowhttptest, Hulk, GoldenEye, and DDoS attacks. These types of attacks are on the rise and are major threats to the SDN architecture. The Wednesday dataset will be used for training and testing purposes. The Friday dataset will be used for testing purposes only. The details of each dataset are described in Table. 3.6.

Dataset	Legitimate Traffic	Anomaly Traffic
Wednesday Dataset	439,683	251,723
Friday Dataset	183,877	41,834

Table 3.6: The CICIDS2017 Dataset Description

This dataset is also normalized into the range of [0-1] by the Min-Max scaling as Equation 3.6.

In Table 3.7, we give some examples of the CICIDS2017 dataset for more understanding.

Flow ID	**Source IP**	**Source Port**	**Destination IP**	**Destination Port**	**Protocol**	**Flow Duration**	**...**	**Flow Bytes/s**	**Label**
192.168.10.14-209.48.71.168-49459-80-6	192.168.10.14	49459	209.48.71.168	80	6	38308	...	313.250496	BEGNIGN
192.168.10.3-192.168.10.17-389-49453-6	192.168.10.3	389	192.168.10.17	49453	6	479	...	1039665.971	BEGNIGN
172.16.0.1-192.168.10.50-55832-80-6	172.16.0.1	55832	192.168.10.50	80	6	105686432	...	24.09959303	DoS slowloris
172.16.0.1-192.168.10.50-55830-80-6	172.16.0.1	55830	192.168.10.50	80	6	105686759	...	24.09951846	DoS slowloris
192.168.10.14-199.96.57.6-51016-80-6	192.168.10.14	51016	199.96.57.6	80	6	13687752	...	1039.469447	BEGNIGN

Table 3.7: The CICIDS2017 Dataset Examples

3.7 Conclusion

In this chapter, we have presented some background knowledge about the DL. The process of training and testing the DNN has been demonstrated to make the work in this thesis clearer and easier to understand. In the last section of this chapter, network datasets that are used to train the DL algorithm are also introduced in this chapter.

Chapter 4

DeepIDS: Deep Learning Approach for Intrusion Detection in Software Defined Networking

In This Chapter:

An SDN-based NIDS is introduced with three core modules: The Collector, The Anomaly Detector and The Counter Measure Deployment. In this chapter, we focus on developing the Anomaly Detector module in which DL algorithms are implemented. We evaluate the potential of DL in detecting network anomaly with different learning rates and feature sets. Experimental results show that a simple DNN can detect network anomalies effectively with just some basic network features. Network overheads are an important factor when designing the NIDS, so they are also evaluated in detail in this chapter.

4.1 Introduction

4.1.1 Motivation

The SDN architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The SDN controller is centralized and separated from its underlying switches, and so it simplifies network management and facilitates network evolution. Instead of deploying a hardware-based NIDS as in the traditional network, we can implement a software-based NIDS on the SDN controller and then take the advantages of SDNs. We can now collect network information, detect any sign of intrusions and then deploy a flow rule to mitigate attacks in a real-time manner. Considering the success of DL in many fields [20] [22] [82], a combination of SDN and DL can improve the NIDS performance and then secure the network better.

In this chapter, we present the structural design and implementation of DeepIDS (a flow-based anomaly detection system using DL) in the SDN architecture. The attack detection mechanism is the main research contribution of this chapter. A comparison with several machine learning algorithms proves the enhancements offered by our approach.

4.1.2 Contribution

Our main contributions are summarized in the following points:

- We propose and implement an SDN-based NIDS with three core modules: The Collector, The Anomaly Detector and The Counter Measure Deployment.

- We have implemented a Deep Neural Network (DNN) model in DeepIDS. We trained and evaluated the model with different sets of features of the NSL-KDD dataset [86]. Through experiments, our model yields a detection ACC of 80.7% using just six basic network features among the 41 features of this dataset.
- This method is scalable, and the structure of the DNN can be changed according to the characteristic of the data features, which makes our method applicable to detect other kinds of attack.
- We have also evaluated the network performance of DeepIDS in the SDN environment. We implemented our DeepIDS in a POX controller, and stress tested DeepIDS through extensive simulation. The test results demonstrate that our approach does not degrade the POX controller's performance.

4.1.3 Chapter Organization

This chapter is structured as follows: In Section 4.2, we present the implementation details of the proposed DeepIDS with respect to data gathering, anomaly detection and mitigation. The experimental methodology is introduced in Section 4.3. Section 4.4 presents the detection performance of our DL approach. Section 4.5 describes our network performance evaluation experiments and results. Finally, Section 4.6 concludes the work.

4.2 DeepIDS System Architecture

In this section, we propose an SDN-based NIDS architecture (DeepIDS) as depicted in Figure 4.1. Our DeepIDS consists of three main modules: the Collector, the Anomaly Detector and the Counter Measure Deployment. The DeepIDS is written in Python programming language and deployed as an application in the SDN controller.

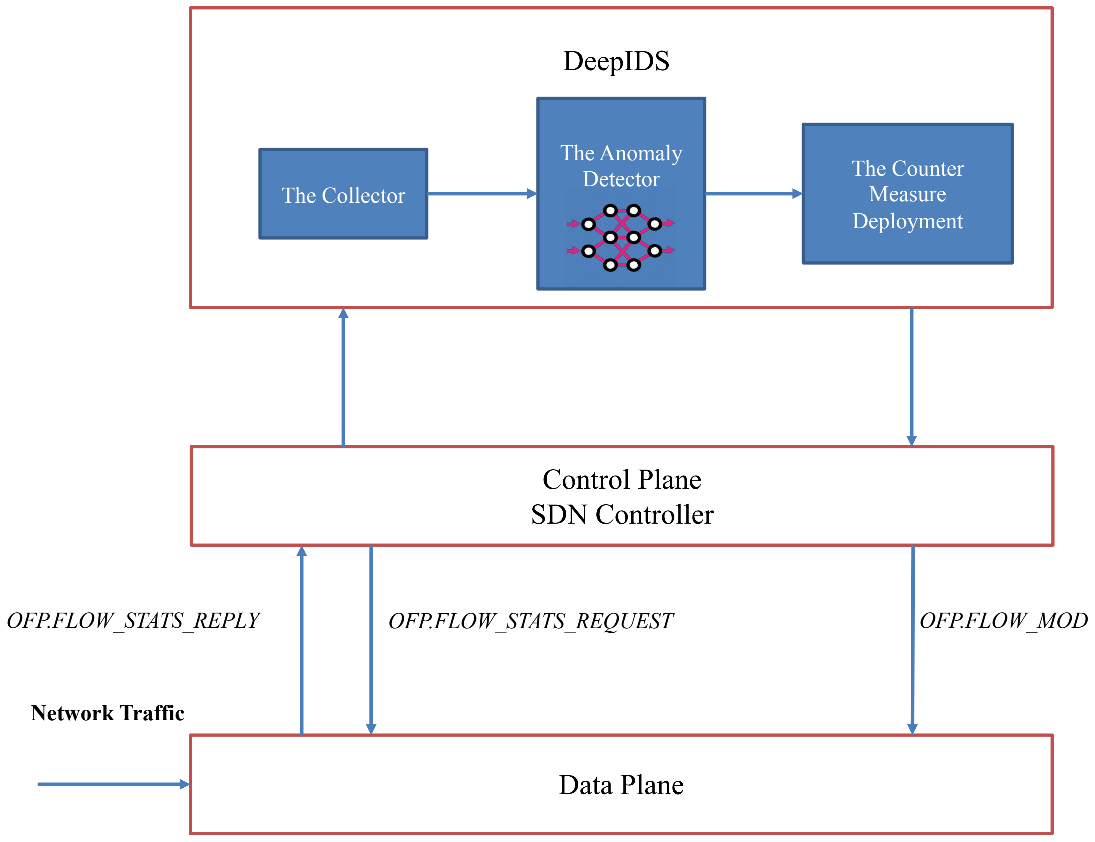


Figure 4.1: The DeepIDS Architecture

The DeepIDS is designed to fulfil the following properties:

- Flexibility: The DeepIDS is developed as an application for ease of deployment, configuration and interaction. It can be implemented on top of any SDN controller. Our DL models can be modified and optimized based on network requirements. New threat models can also be easily added/updated.

- Scalability: The DeepIDS is designed with a goal to facilitate not only small scale networks but also large scale networks. The overhead of our approach does not degrade the performance of the whole network. The overhead on the SDN controller is evaluated with different network sizes.

For ease of understanding, details of the network intrusion detection framework are presented in Figure 4.2. This framework describes a complete cycle of data flow in the DeepIDS.

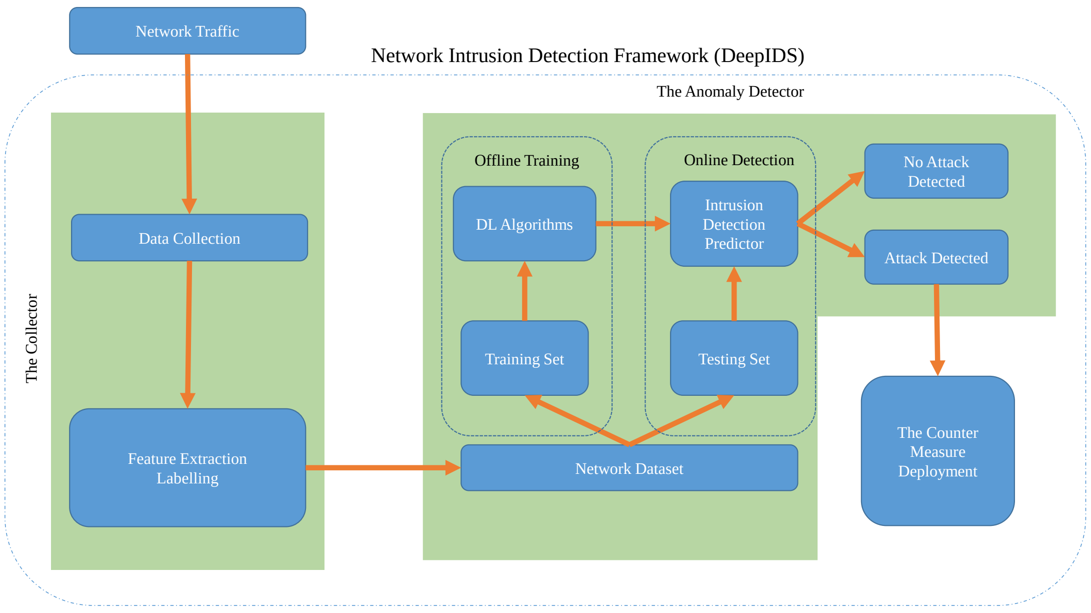


Figure 4.2: Network Intrusion Detection Framework

At the first stage of the intrusion detection process, The Collector collects network information and then extracts flow features from them. These flow features are inputs of the DL algorithm. Some examples of flow features are shown in Table 4.1. For the DL algorithm training purpose, these flow features will be cleaned, labeled and saved as a network dataset.

No.	Flow Feature
1	Source IP Address
2	Destination IP Address
3	Source Port
4	Destination Port
5	Number of Flow per Second
6	Number of Packet per Second
7	Packet Duration
8	Bytes per Packet

Table 4.1: Flow Feature Examples

After preparing the network dataset, we will move to the second stage which is the most critical part of intrusion detection and also is the primary research of this thesis. At this stage, we will train the DL algorithm to detect all types of network attacks. As mentioned in Chapter 2, the network dataset will be separated into two subsets: Training Set and Testing Set. The training set will be used for training, and the testing set will be used for evaluating trained models. If there is any attack detected, the Counter Measure Deployment module will be activated to mitigate the attack.

4.2.1 The Collector

We consider network traffic logs as a time series. For each time interval in a series, we extract various per-flow information. Based on this information, flows transferred during each time interval are classified. For this reason, the analysed time period is divided into equal overlapping time bins of length T. The length of each time bin should be selected in such a manner that it contains enough information to detect anomalies. Notice also that the optimal selection of the detection loop period is a complex problem. If the selected period is too large, then the response time will be long, which makes the controller and the switches handle an enormous amount of attack packets and even destroys the controller and the switches. If the period is too small, the attack detection will occur more frequently, which incurs significant overheads for the controller in terms of resource efficiency. While several researchers [91] - [92] focus on traffic monitoring and sampling, this is not the main focus of our work in this thesis. In this thesis, we are focusing on improving the anomaly detection rate. In addition, because of a lack of network facilities, most of the experiment at the early stage of this work are done in an offline manner. All the ML/DL models are trained and tested with pulished datasets. Therefore, we are not focusing on this process now.

Currently, most approaches use the periodic trigger to start the detection of the attack based on inspection of flow entries, whereby, the collection of flow entries is performed at predetermined time intervals by the controller. It is hard to choose the time interval, but the system also gives the network manager the right to change that time interval to suit the network. In our experiment, the time interval is set at $T = 1$ second for stress testing the controller.

The Collector module is responsible for collecting flow information and periodically exporting it to the Anomaly Detector module. The OpenFlow protocol provides us with a proactive way to collect network information. An OFP_FLOW_STATS_REQUEST message will be sent to all switches by the controller after a fixed time-window to request the network statistics. The OpenFlow switches will reply with an OFP_FLOW_STATS_REPLY message. All the statistics in the OFP_FLOW_STATS_REPLY message will be extracted and recorded by the Collector module. The 6-tuple basic feature willbe prepared as an input for the DL model in the Anomaly Detector module.

4.2.2 The Anomaly Detector

Data produced by the Collector is subsequently fed to the Anomaly Detector. In this module, we choose the DL algorithm as a core of the module. The DL model can be trained offline and can be deployed for online detection. In addition, we can update our model anytime without interfering with real-time detection. For every time-window, the Anomaly Detector module inspects all the flow entries to identify any potential attack traffic. As soon as an anomaly is detected in the network, our algorithm will record all the network metrics of the identified attack for further forensics and send all related information to the Counter Measure Deployment module.

4.2.3 The Counter Measure Deployment

The Counter Measure Deployment module aims to neutralize identified attacks. All information about detected attacks (e.g., source IP address, destination IP address, source port and destination port) is collected from the Anomaly Detector module. Then, this information will become an input for the Counter Measure Deployment module. This module will insert new flow-entries into the flow table or modify current flow-entries of the OpenFlow switch to drop all the malicious traffic from attacking source IP addresses. A new flow rule can be sent from the controller to the switches with an attack IP address in a matching field and an "of.OFPP_DROP" action in an action field to drop all attack packets. The attacking packet can be redirected to a honey pot with a specific destination IP address in the same way. A warning message will also be sent to the network administrators for further actions. The complete process for countermeasure deployment is summarized in Algorithm. 1.

Algorithm 1 The Counter Measure Deployment

```
1: for all Anomaly Packets do
2:     msg = of.ofp_packet_out()                          ▷ Create packet_out message
3:     msg.buffer_id = event.ofp.buffer_id
4:     msg.in_port = packet_in.in_port
5:     msg.match = of.ofp_match.from_packet(packet)
6:     action = of.ofp_action_output(port = of.OFPP_DROP)  ▷ Add an action
   to drop the packet
7:     msg.actions.append(action)
8:     self.connection.send(msg)                          ▷ Send message to switches
9: end for
```

4.3 Experimental Methodology

4.3.1 DL Experimental Setup

In this chapter, we implement a fully-connected DNN model which is presented in Chapter 3 and is the simplest form of DL algorithms. The DNN will be trained with flow features, and it then classifies network traffic as legitimate traffic or anomaly traffic. The anomaly traffic includes all types of network attacks appearing in the training set. The number of hidden layers and the dimension of each hidden layer can be varied to find the best structure for intrusion detection.

In our experiment, six input features will be used, so we define an input layer with six neurons. As introduced in Chapter 3, the KDDTrain++ and KDDTest++ of the NSL-KDD dataset are used for training and testing the DNN respectively. We use Keras [93] to implement the DNN model. Details of all initiated parameters of the DNN model can be seen in Table 4.2.

Variable	Parameters
Input Layer	6
Hidden Layer	5,4,3
Output Layer	2
Activation Function	Tanh
Loss Function	Mean Squared Error
Batch Size	10
Epoch	1000

Table 4.2: The DNN Model Structure

In general, the DNN is constructed with an input layer, three hidden layers and an output layer as described in Figure 4.3.

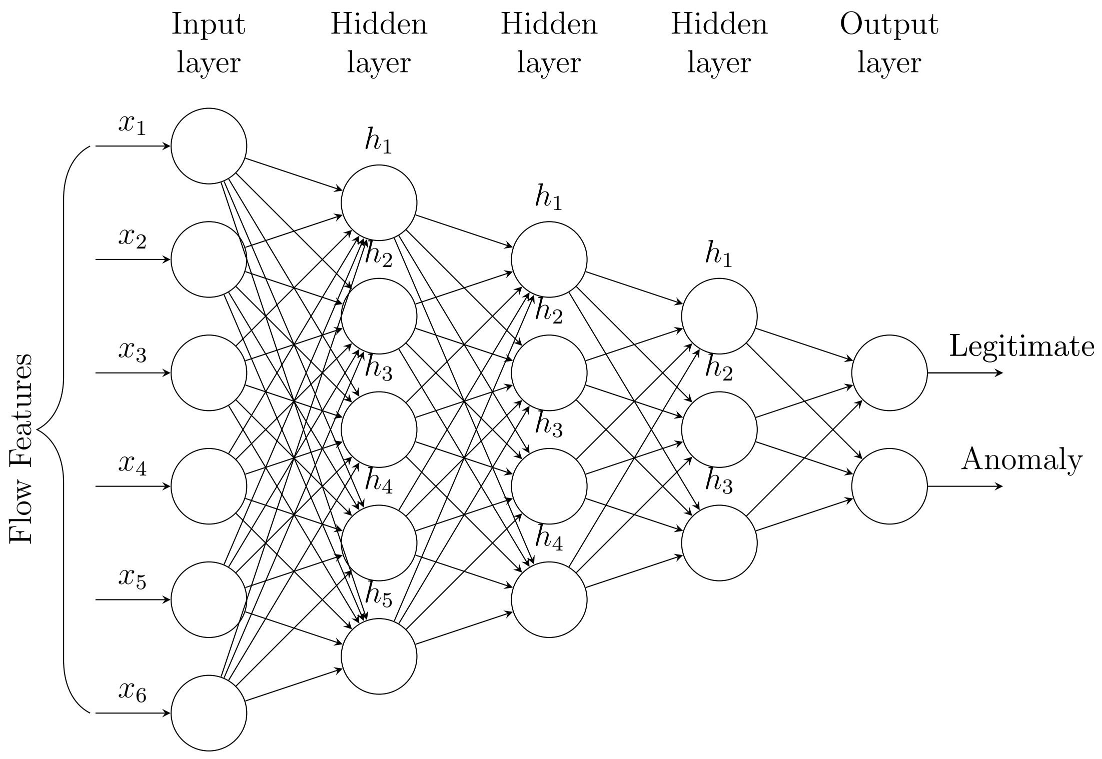


Figure 4.3: The DNN Structure

Under the SDN context, we just focus on the basic features and traffic-based features. For traditional networks, an anomaly-based NIDS is trained with a huge amount of features including packet content-based features. This consumes a lot of computer resources and time. Many researchers have developed feature selection algorithms to reduce the feature dimension and gain higher detection ACC. However, the packet content is not directly accessible in the current OpenFlow protocol, and so we do not employ any content-based features of this dataset. In our experiment, we created three distinct sub-datasets that contain traffic samples with six features extracted from the NSL-KDD dataset:

- Basic Feature Set: contains basic features of individual TCP connections
- Traffic Feature Set: contains features of network traffic
- Mixed Feature Set: contains both basic features and traffic-based features

The main purpose of these subsets is to evaluate the role of each type of features to the detection performance. Details of each feature set can be found in Table 4.3.

Feature Set	**Description**
Basic Feature Set	duration, protocol_type, src_bytes, dst_bytes, land, wrong_fragment
Traffic Feature Set	count, srv_count, same_srv_rate, dst_host_count, dst_host_same_srv_rate, dst_host_same_src_port_rate
Mixed Feature Set	duration, protocol_type, src_bytes, dst_bytes, srv_count, dst_host_same_src_port_rate

Table 4.3: Feature Set Description

4.3.2 Network Performance Evaluation Setup

Cbench [94] is a standard tool used for evaluating SDN OpenFlow controllers. In this chapter, Cbench is used to evaluate the overheads of the DeepIDS on the SDN controller. We measured the performance of the controller in terms of throughput and latency. Cbench tests can run in either throughput or latency mode.

- In throughput mode, Cbench sends a stream of PACKET_IN messages to the controller for a specified period of time and then records the number of responses (PACKET_OUT messages) for the request that it has sent to the controller. Throughput data reflects the average number of flows the controller could treat per second in each switch.
- In latency mode, Cbench sends a PACKET_IN message to the controller and waits for the response (PACKET_OUT message) before sending another packet. The latency results represent the average number of milliseconds that a flow consumes to be installed in each switch.

Figure 4.4 shows the flow diagram of network performance evaluation process. At the beginning of the evaluation process, Cbench will be connected to the SDN controller. Cbench emulates a network topology by itself, then sends PACKET_IN messages to the SDN controller and waits for PACKET_OUT messages from the SDN controller.

We tested the controller's throughput and latency with a different number of virtual OpenFlow switches emulated by Cbench. We evaluate the network performance in six scenarios. Details about network parameters of each scenario

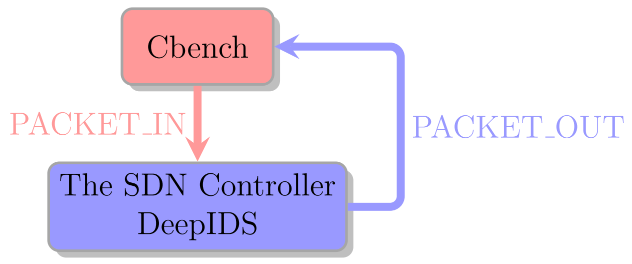


Figure 4.4: The Network Evaluation Process

are described in Table 4.4. In all the testing scenarios, each switch is connected with a thousand host in a star topology. Then all the switches are connected to the SDN controller.

Number of Switches	Number of Hosts in Each Switch
8	1000
16	1000
32	1000
64	1000
128	1000
256	1000

Table 4.4: Network Parameters

Each Cbench run consists of 10 test loops with a duration of 10 seconds. We ran the controller with a typical layer 2 learning switch application. The obtained results are the average values of the 10 tests. We evaluate the performance of the controller in the following scenarios.

- The SDN controller runs stand-alone. This scenario serves as a baseline for our evaluation.
- The SDN controller runs with DeepIDS or other ML algorithms being enabled.

4.4 Detection Performance Evaluation

4.4.1 Effect of The Learning Rate

The DNN architecture is a very flexible architecture that consists of several layers with a different number of neurons. Different network architectures and hyper-parameter setting will yield different results. Tuning the DNN architecture and hyper-parameters is a challenging task. The learning rate is one of the most important hyper-parameters that we have to tune in each training process.

In this section, we firstly analyze the effect of different learning rates on detection performance. Thus, we tried to optimize the model by varying the value of the learning rate in a range of {0.1, 0.01, 0.001, 0.0001}. When training the DNN model, we tried to minimize the loss and maximize the ACC (see Equation 2.1).

Learning Rate	**Train Set**		**Test Set**	
	Loss	**ACC (%)**	**Loss**	**ACC (%)**
0.1	11.49	88.04	31.26	72.05
0.01	8.41	90.9	20.15	73.03
0.001	8.26	91.62	15.51	80.7
0.0001	7.45	91.7	20.3	74.67

Table 4.5: Loss and Accuracy Evaluation for Different Learning Rates

By comparing the loss and ACC of the training phase (see Table 4.5), we can see that along with the decrease of the learning rate, the loss will decrease, and the ACC will increase. However, in the testing phase, when we lower the learning rate to 0.0001, the results are not as good as the learning rate of 0.001. This is because if the learning rate is too small, the NIDS model will be trained too accurately and the overfitting problem starts happening. That is the reason for the best results in the training phase of the learning rate of 0.0001. Because of this too accurate training phase, the model cannot generalize the characteristic of the training samples well. So while the model can easily catch the intrusion instances in the training set, it cannot capture the new intrusion instances in the testing set. As a result, DNN performance decreases in the testing phase. In practice, the ACC of the testing phase is one of the most important evaluation criteria. The higher testing ACC is, the better the DNN model is. This is to make sure that the trained model can work well with new data and unseen attacks.

Secondly, we evaluate the P, R, and F1 (see Equations 2.2, 2.3, and 2.4) of the model for more details. The performance of the DNN algorithm was evaluated using the test data provided. The performance of the model with each learning rate is shown in Table 4.6. As we can see, the learning rate of 0.001 gives us the best results among the four learning rates in all evaluation metrics. All evaluation metrics are in a growing trend when we decrease the learning rate from 0.1 to 0.001, but the metrics suddenly decrease at a learning rate of 0.0001. From evaluating the evaluation metrics, the loss, and the ACC, we can see that the performance of the DNN model decreases when the learning rate is decreased to 0.0001.

Learning Rate	**P (%)**	**R (%)**	**F1 (%)**
0.1	79	72	72
0.01	82	73	72
0.001	85	81	81
0.0001	83	75	74

Table 4.6: Accuracy Metrics for Different Learning Rates

From the above evaluations, we can see that the learning rate has a substantial impact on DNN performance. It must be analyzed carefully at the beginning of the training process. In this chapter, the learning rate of 0.001 will be taken as the best parameter for other evaluations.

4.4.2 Effect of The Feature Set

We first analyze the impact of different feature sets on the ACC metric. The performance of the model for each feature set is shown in Table 4.7. As we can see, the Mixed Feature Set gives us the highest ACC with 80.7%. The ACC of the Traffic Feature Set is quite low compared to others. The Basic Feature Set's ACC is just slightly smaller than that of the Mixed Feature Set.

Feature Set	ACC (%)
Basic Feature Set	80
Traffic Feature Set	71
Mixed Feature Set	80.7

Table 4.7: Accuracy Evaluation for Different Feature Set

Table 4.8 gives you a more detailed view about the performance of DNN on each feature set. As seen in Table 4.8, the R and P values of the Mixed Feature Set are higher than the other sets. The evaluation result shows that with just a small set of basic network features, we still can achieve high detection ACC. The DNN shows its power in learning network characteristics from raw features. The combination of basic and traffic features helps improve the DNN performance. However, the traffic feature above does not give much information about the network and attack characteristic, so the detection ACC is low.

In the following, ROC curves and their AUC are presented in Figure 4.5. Figure 4.5 shows that the Mixed Feature Set gives the best result with the highest AUC. The Basic and Traffic Feature Sets have quite high FPRs. As we can see, the combination of basic features and traffic features helps reduce the FPR that is an

Feature Set	P (%)	R (%)	F1 (%)
Basic Feature Set	84	80	80
Traffic Feature Set	77	73	72
Mixed Feature Set	85	81	81

Table 4.8: Performance Metric Evaluation of Three Feature Sets

essential factor of NIDS. The high FPR means that there will be many legitimate traffic classified as attacks. Therefore, our network will be more vulnerable. For the Traffic Feature set, we can conclude that the difference between legitimate and anomaly traffic is not significant, so the DNN cannot learn their characteristics well. This leads to a very high FPR.

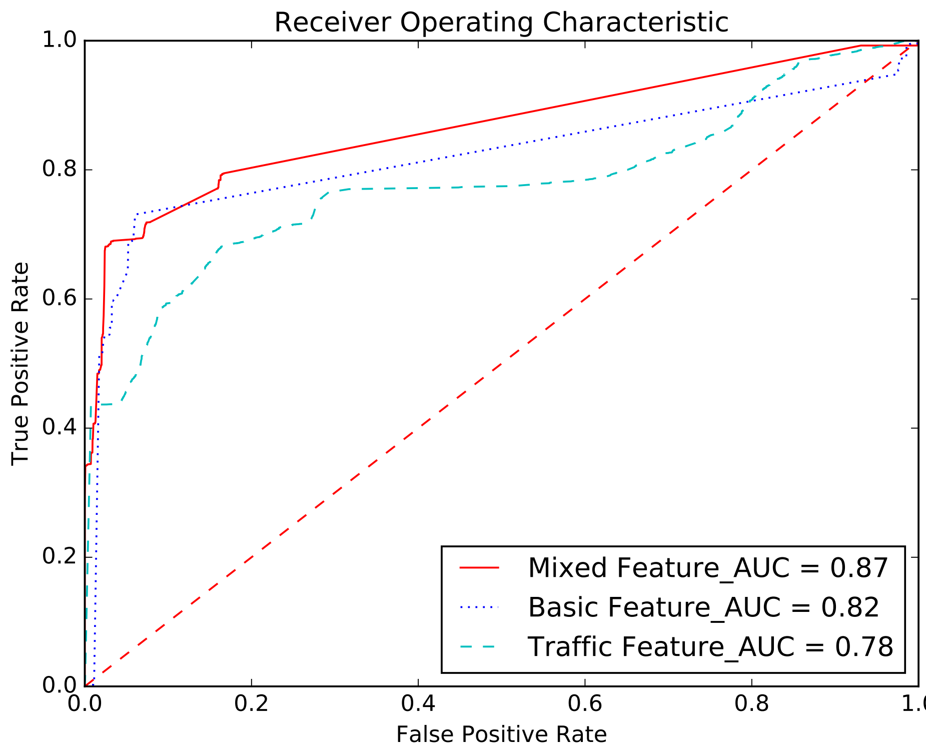


Figure 4.5: ROC curve comparison for different feature sets. The Mixed Feature Set achieves the best result with highest AUC and lowest FPR

In the next experiment, we evaluate the sensitivity of each feature set with some existing algorithms: Naive Bayes (NB), Decision Tree (DT) and SVM. The NB and DT are quite simple and cost-effective. The SVM was proposed by Phan *et al.* [18] for DDoS detection in SDNs. They used their own dataset, and so we have to regenerate the results on the considered NSL-KDD dataset. The ROCs of each test case are shown in Figure 4.6, 4.7 and 4.8. Each feature set has a different effect on the performance of each algorithm.

For the Basic Feature set, all the algorithms perform quite well and are comparable to each other. The SVM yields a slightly better result than the others. As mentioned in Chapter 2, The SVM is a popular intrusion detection approach and has shown good results in detecting intrusion. Thus, the results in Figure 4.6 are expected. However, the SVM also has some drawbacks that will be discussed in the next sections.

For the Traffic Feature set, following the results of previous evaluations, the AUC results are quite low for all the algorithms except the NB. This phenomenon is quite common in the field of ML/DL. One algorithm can work well with one dataset but it can be bad for another dataset. Therefore, we had to experiment with different algorithms to find the most suitable one for our problem.

In the case of the Mixed Feature set, the DNN and SVM have the same AUC result and are the best amongst all the algorithms. In this case, we can see that the overall FPR is quite low compared to those of other feature sets. This confirms our conclusion about the contribution of basic and traffic features in improving detection ACC.

All in all, NB achieves a quite high AUC in the case of the Traffic Features Set. Despite the high AUC of the NB, its FPR is quite high and not feasible for

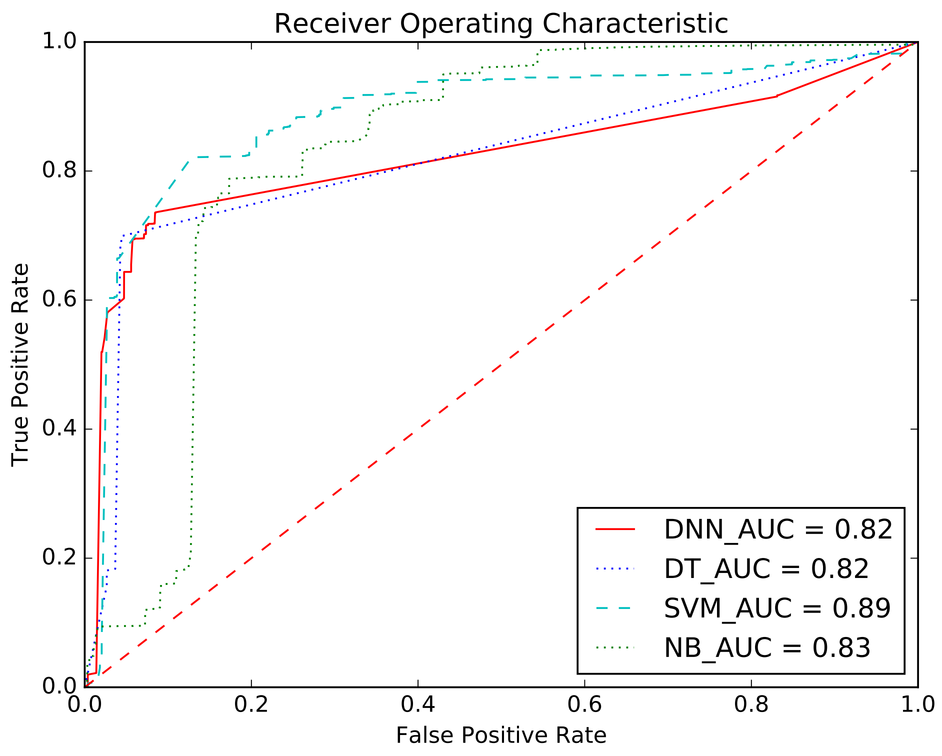


Figure 4.6: ROC Curve Comparison of Different Algorithms for Basic Feature Set

the NIDS. In general, the DNN works quite well with all of the feature set, with quite low FPR and high AUC. SVM also achieves a quite high AUC, but it also has some drawbacks like high FPR and low computational efficiency. We will discuss the effectiveness of these algorithms in due course. The Mixed Feature Set gives us the best combination of AUC and the FPR.

From the above evaluation, we can see the potential of DNN for intrusion detection. The Mixed Feature Set will be chosen for further evaluation. We also evaluated the P, R, and F1 of the model for more understanding. The performance of the model is described in Table 4.9. As we can see, our model performs well in the training phase with high ACC. However, the ACC decreases in the testing

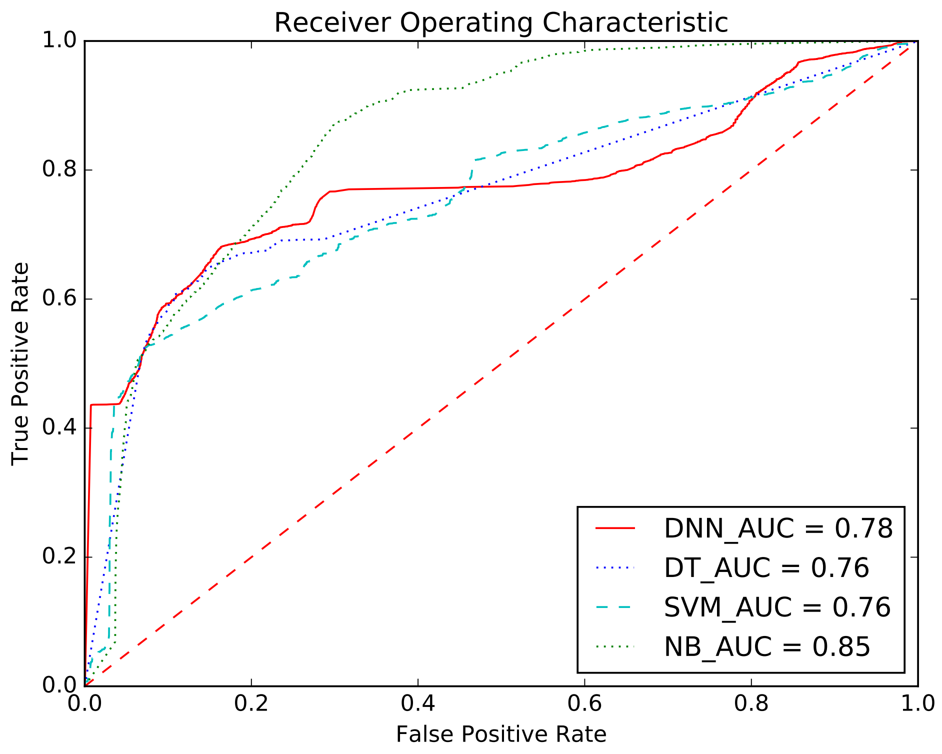


Figure 4.7: ROC Curve Comparison of Different Algorithms for Traffic Feature Set

phase. This is because several attacks in the testing set are new and complex. These attacks cannot be fully generalized by our existing DNN model using just six basic network features. The R result of the testing phase shows that our DNN model can detect almost all the attack mentioned in the KDDTest++ set, and our results are significantly promising with a high P of 85%.

Dataset	P (%)	R (%)	F1 (%)
Training Set	96	96	96
Testing Set	85	81	81

Table 4.9: Performance Metric Evaluation of the Mixed Feature Set

Table 4.10 gives us an overview of the ACC comparison amongst popular in-

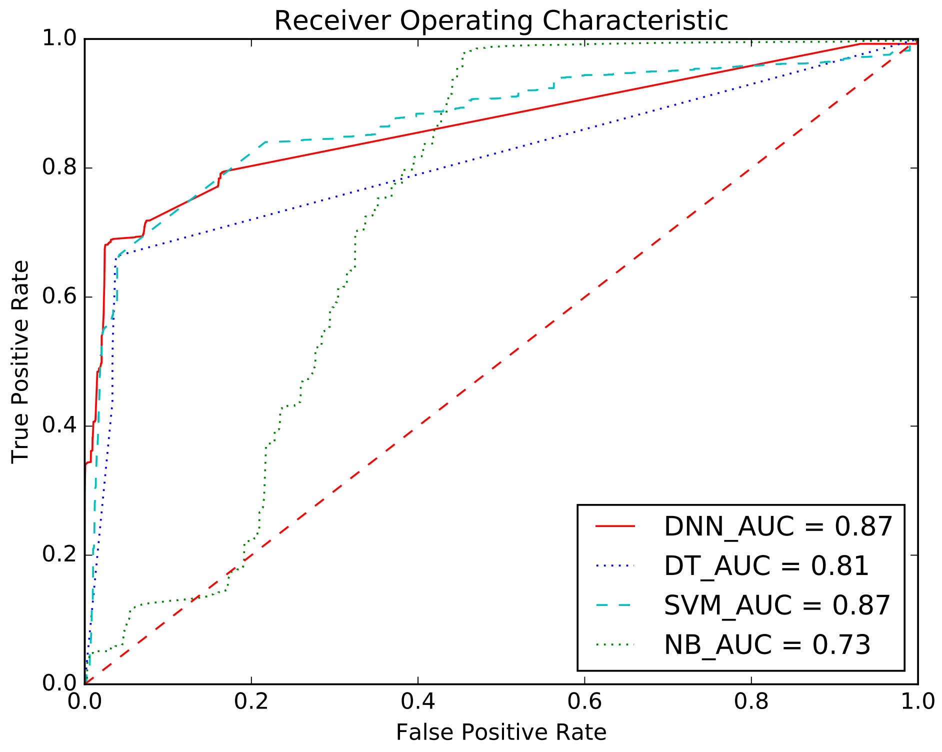


Figure 4.8: ROC Curve Comparison of Different Algorithms for Mixed Feature Set

trusion detection algorithms. The proposed DNN gives the best results amongst all the algorithm. This is because of the limitation of traditional ML methods. Without a large set of carefully hand-crafted input features, traditional ML methods perform very poorly compared to the DL method.

In addition, we compared our results with the result in [86] from different machine learning algorithms. In [86], the authors train and test different algorithms with a full training and testing set of forty-one features. From these experiments, the authors can evaluate the performance of these algorithms in their dataset. From Table 4.11, the DNN approach gives quite good ACC compared with other algorithms. The most accurate machine learning algorithm is the NB tree with

Algorithm	**Accuracy (%)**
NB	45
DT	74
SVM	70.9
DNN	**80.7**

Table 4.10: Accuracy Comparison for Different Algorithms Using the Mixed Feature Set

82.02%. This is the ACC obtained from the full feature training set, and so the NB tree algorithm can generalize the characteristics of the normal and anomaly traffic very well.

Algorithm	**Accuracy (%)**
J48	81.05
Naive Bayes (NB)	76.56
NB Tree	82.02
Random Forest	80.67
Random Tree	81.59
Multi-layer Perceptron	77.41
Support Vector Machine (SVM)	69.52
DNN	**80.7**

Table 4.11: Accuracy comparison of different algorithms. The DNN uses the Mixed Feature Set with 6 features. The others use 41 features

In our experiments, it must be noted that only the six basic features in the Mixed Feature Set are used for training and testing. This sub-feature set is quite small compared to the full feature set, so it cannot provide enough information for our DNN algorithm to generalize the characteristics of some sophisticated or new attacks. However, it can be seen that the DNN performs reasonably compared with other algorithms.

In summary, we have demonstrated that our proposed DNN approach can generalize and abstract the characteristics of the normal and anomaly traffic with a small number of features and gives us a promising ACC. The result of DL algorithm facilitates the development of flow-based NIDS under the SDN context. All the basic network feature, that can be collected easily in the SDN architecture, can be used directly by the DL algorithm without any feature engineering process.

4.4.3 Efficiency Evaluation

We also evaluate the efficiency of our model compared to other algorithms by comparing the training time and the testing time of these algorithms. Table 4.12 provides relevant parameters for these algorithms. The NB and DT algorithms have really low training and testing times in ms compared to the SVM and our DNN. However, they also yield low detection ACC. The SVM is one of the most popular algorithms in the field of intrusion detection, but it is not fast enough for real-time detection, especially under SDN architectures. The total processing time takes around 2.5 hours. Our DNN works much faster than the SVM and also gives a better detection ACC, and so it is feasible for real-time detection in SDNs.

Algorithm	**Training Time**	**Testing Time**
NB	19.8 ms	4 ms
SVM	2 h	30 min
DT	256.9 ms	2.74 ms
DNN	**500 s**	**811.6 ms**

Table 4.12: Running Time Evaluation

As mentioned in [63], the SOM algorithm was used for flow-based intrusion detection in the SDN context. The SOM algorithm was trained and tested by a 6-tuple feature (Average of Packets per flow, Average of Bytes per flow, Average of

Duration per flow, Percentage of Pair-flows, Growth of Single-flows, and Growth of Different Ports) dataset that consists of 8608 samples and 62888 samples in the training set and testing set respectively. The system takes about 7.16 hours for training and 352 ms for testing. Despite the difference of the experimented systems, our DeepIDS computation cost is really low (with the same number of features and a larger number of samples) when compared to [63].

4.5 Network Performance Evaluation

In this section, we provide detailed network performance analysis of our DeepIDS in a POX [35] controller. We also compare the network performance of the DeepIDS with the SVM algorithm. We also evaluate resource utilization and briefly discuss our observations in this section.

The DeepIDS is implemented in the POX controller as an application written in Python language. Currently, POX supports OpenFlow 1.0 protocol and includes special support for the Open vSwitch/Nicira extensions. In order to test the performance of DeepIDS in the POX controller, we used a Virtual Machine having Intel Core i5-4460 3.2 GHz processor with three cores available and 8 GB of RAM memory. The evaluation setup for this experiment has been described in Section 4.3.2.

4.5.1 Throughput Evaluation

For throughput mode tests, we evaluate how many packets a controller can process in a second. This metric indicates the performance of the controller under heavy traffic conditions. Figure 4.9 depicts the average response rate of the network in three testing scenarios.

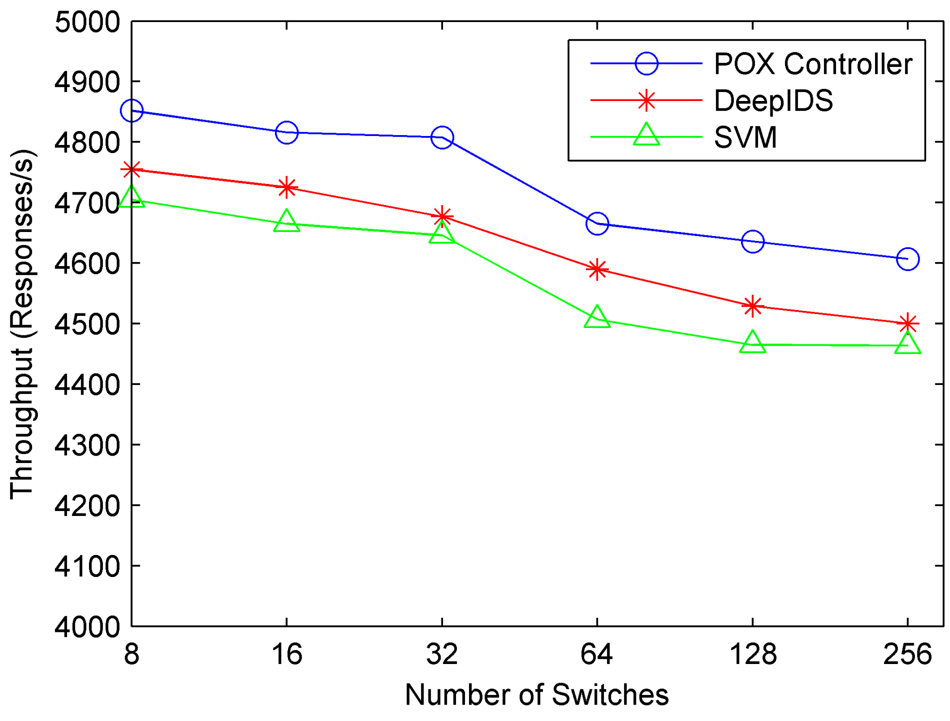


Figure 4.9: Throughput Evaluation (log scale on x-axis)

As we can see, the performance of the POX controller itself is quite poor with a maximum throughput around 4900 responses/s. This is one of the limitations of the POX controller compared to other SDN controllers. However, the running stand-alone POX controller still performs best and is taken as a baseline for performance evaluation in our experiment. Of overall assessment, the bigger the size of the network then the higher the performance overhead. The throughput slightly decreases with the increasing network size according to the results in Figure 4.9. The network performance drops when the number of forwarding devices increases up to 64, showing that the controller throughput degrades by roughly 3% from 32 to 64 switches. The baseline results for the smallest network scale and the largest network scale are 4852 responses/s and 4607 responses/s respectively.

The DeepIDS and SVM overheads are about 2% and 3.12% respectively in the case of 8 switches.

We can see that in the case of high PACKET_IN message rate with 256 switches, DeepIDS's throughput decreases by 2.7% with 4465 responses/s. The SVM algorithm's throughput decreases by about 3.2% in the same situation. The reason for this behaviour is that the NIDS sends OFP_FLOW_STATS_REQUEST messages and processes OFP_FLOW_STATS_REPLY messages during processing PACKET_IN messages. When the network size increases, the NIDS has to process a large amount of OFP_FLOW_STATS_REPLY messages resulting in the overhead. Figure 4.9 shows that the SVM algorithm has affected the controller and resulted in the low network throughput in all test cases. As we can see, DeepIDS performs better than the SVM algorithm in terms of throughput. DeepIDS's throughput degradation (which is around 4% on average) is quite low and has almost no effect on the POX controller's performance.

4.5.2 Latency Evaluation

For latency mode tests, the controller is evaluated for the length of time it takes to process a single packet. This metric reflects the performance of the controller in light traffic conditions. The testing scenarios are the same as the throughput test. As depicted in Figure 4.10, the latency increases with the increasing network size. This is expected behavior: increasing the number of devices will increase the load at the controllers, causing a latency increase. In general, we can see in Figure 4.10 that the SVM always has higher latency than the DeepIDS. The NIDS takes time to process the PACKET_IN message, the OFP_FLOW_STATS_REPLY message and detect anomaly flows, and so the performance overhead is unavoidable. The SVM algorithm takes a longer time to process the information, and thus it results in a higher latency under high traffic rates. The DeepIDS and SVM overhead are about 2.6% and 5.8% respectively in the case of 256 switches. The DeepIDS's latency degradation is quite low and can be improved in the future.

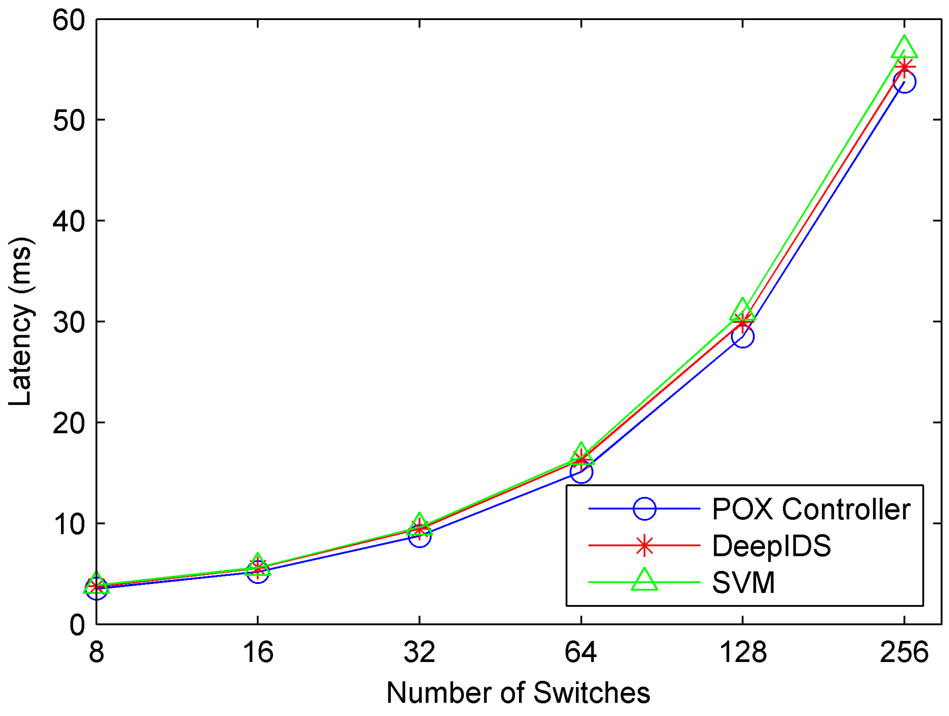


Figure 4.10: Latency Evaluation (log scale on x-axis)

4.5.3 Resource Utilization

We also evaluate the resource utilization of DeepIDS. The POX controller itself resource utilization is a benchmark for our comparison. As we can see in Table 4.13, DeepIDS does not use a lot of computer resources. In general, DL requires a significant computational cost. DL models usually have high-dimensional inputs and a large number of hidden layers. However, in our experiment, we minimize the input with just six basic network features and use three hidden layers. As a result, our DNN model is quite simple, and so it uses computational resources optimally.

Algorithm	CPU Usage (%)	Memory Usage (%)
POX	15	7.5
DNN	17	8.2

Table 4.13: Resource Utilization Evaluation

4.6 Conclusion

In this chapter, we proposed the DeepIDS as an SDN-based NIDS architecture. We have implemented a DL algorithm for detecting network intrusion and evaluated our DeepIDS. Our results show that our approach has significant potential and advantages for further development. By comparing the results with those of other classifiers, we have demonstrated the potential of using DL for the flow-based anomaly detection system. In the context of the SDN environment, the DL-based IDS approach is promising. Regarding the above-mentioned evaluations, we can see that our method does not affect network performance significantly. Therefore, our approach is quite promising and can be improved in many ways. With the flexibility of the SDN structure, we can extract features focused on one specific type of attack, like DDoS, to increase the ACC of the NIDS. As shown above, our DeepIDS gives the slightly lower ACC than other approaches that use a full 41 feature NSL-KDD dataset. In the future, we will optimize our model to improve the detection rate and decrease the false alarm rate.

Chapter 5

Deep Recurrent Neural Networks for SDN-based Intrusion Detection Systems

In This Chapter:

We introduce a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) for time series network traffic classification. As Internet traffic traces are a kind of time series data, we believe that GRU-RNNs are suitable for this task. The aim of this chapter is to improve the intrusion detection ACC on the Internet traffic. We demonstrate through our experiment and analysis that the GRU-RNN can improve the ACC significantly without degrading the network performance.

5.1 Introduction

5.1.1 Motivation

The DNN in the previous chapter shows the potential of DL under the context of SDN. This approach is relatively lightweight, but the detection ACC is quite low compared to other state-of-the-art results and not applicable for real NIDS. Therefore, there is a need to improve the detection rate. Recently, Recurrent Neural Networks (RNNs) have shown great success in language modelling, text generating and speech recognition [20] [22]. Following the trajectory of current research, we believe that deep RNNs can potentially offer better solutions for implementation of IDS under the context of SDN.

Network traffic often shows temporal correlations. As a result, these sequential traffic traces lead to the generation of time series data. For the DNN, this temporal information can be lost during the training process. The RNN addresses this issue. It is a powerful technique that can represent the relationship between a current event and previous events. Some network anomalies are collective anomalies, so they are difficult to detect by the DNN. We believe that the RNN enhances the anomaly detection rate. In this chapter, a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) is proposed to take advantage of network time series data to detect anomaly traffic.

5.1.2 Contribution

In summary, the major contributions of this chapter are the following:

- We introduce a NIDS in the SDN environment using GRU-RNN.

- Our GRU-RNN approach yields a detection rate of 89% in the NSL-KDD dataset using a minimum number of features compared to other state-of-the-art approaches. This gives our approach significant potential for real-time detection. The GRU-DNN achieves an impressive detection rate of 99% dealing with DDoS attacks in the CICIDS2017 dataset.
- We also evaluate the network performance of our proposed approach in the SDN environment. The results show that our approach does not significantly degrade the SDN controller's performance.

5.1.3 Chapter Organization

The remainder of this chapter is organized as follows. Section 5.2 presents an introduction about RNNs and Gated Recurrent Units (GRU). In Section 5.3, we give a system description and an analysis of detection performance. Section 5.4 presents the network performance analysis. Finally, Section 5.5 concludes the chapter.

5.2 Recurrent Neural Networks

An RNN, which is an extension of a conventional NN mentioned in Chapter 3, makes use of the sequential information. The idea is that the RNN can use information from previous stages to understand information at the current stage. This is also the way that the human brain works. The RNN is called a "recurrent" because it performs the same task for every element of a sequence, with the output being dependent on the previous computations. Figure 5.1 shows the

RNN unfolded in time.

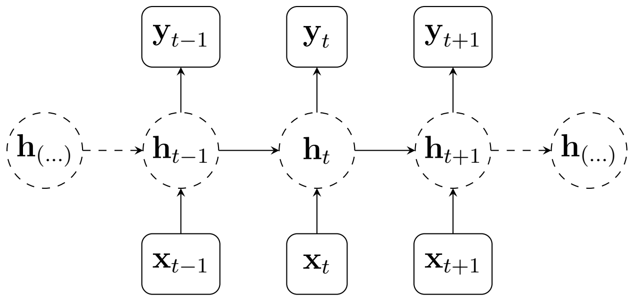


Figure 5.1: The RNN Unfolded in Time

Mathematically, the hidden states and output of the RNN are computed as:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_h),\ t = 1, 2,\ \ldots,\ T, \tag{5.1}$$

$$\mathbf{y}_t = \sigma(\mathbf{V}\mathbf{h}_t), \tag{5.2}$$

where $\sigma(\cdot)$ is a non-linearity function, $\mathbf{x}_t$ is an input vector at time t, $\mathbf{h}_t$ is a hidden state vector at time t, $\mathbf{W}$ is an input to hidden weight matrix, $\mathbf{U}$ is a hidden to hidden weight matrix, $\mathbf{V}$ is a hidden to label weight matrix, and $\mathbf{b}_h$ is a bias vector.

The Backpropagation Through Time (BPTT) [95] algorithm is used for training the RNN. However, BPTT for the traditional RNN is usually difficult due to a problem known as vanishing/exploding gradient [96]. Long Short Term Memory (LSTM) [97] networks and GRUs [8] were proposed to solve this problem and are among the most widely used models in DL algorithms.

GRUs are selected in our research because of their simplicity and faster training phase compared to LSTMs [98]. Figure 5.2 shows architecture detail of a single GRU cell.

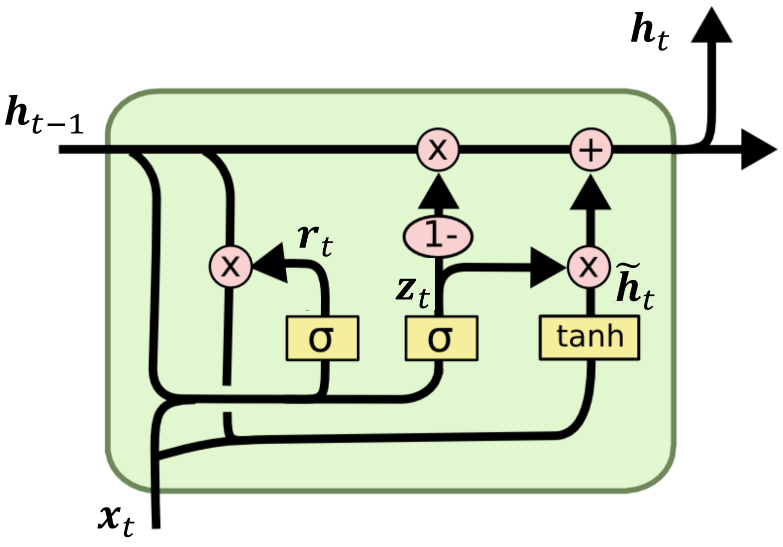


Figure 5.2: Gated Recurrent Unit Structure [8]

Reffering Figure 5.2, the activation h_t^j of each $j-$th GRU unit at time t is computed differently from Equation 5.1 as follows:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j,\ t = 1, 2,\ \ldots,\ T, \tag{5.3}$$

where an update gate z_t^j defines how much of the previous memory to keep, $\tilde{h}_{t-1}^j$ is the previous hidden state and $\tilde{h}_t^j$ is the candidate activation. The update gate is computed by

$$z_t^j \ = \sigma(\mathbf{W}_z \mathbf{x}_t \ + \mathbf{U}_z \mathbf{h}_{t-1})^j, \tag{5.4}$$

where $\sigma(\cdot)$ is a non-linearity function, $\mathbf{W}_z$ and $\mathbf{U}_z$ are learned weight matrices.

The candidate activation $\tilde{h}_t^j$ is computed by

$$\tilde{h}_t^j \ = \tanh(\mathbf{W}_h \mathbf{x}_t \ + \mathbf{U}_h(\mathbf{h}_{t-1} \ \odot \ \mathbf{r}_t))^j, \tag{5.5}$$

where a set of reset gates $\mathbf{r}_t$ determines how to combine the new input with the previous activation vector, $\odot$ is an element-wise vector multiplication, and $\mathbf{W}_h$ and $\mathbf{U}_h$ are learned weight matrices . The reset gate is computed by

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}), \tag{5.6}$$

where $\sigma(\cdot)$ is again a non-linearity function, and $\mathbf{W}_r$ and $\mathbf{U}_r$ are learned weight matrices.

5.3 Detection Performance Evaluation

5.3.1 Experimental Methodology

In this chapter, the experimental methodology is the same as Chapter 4. The Mixed Feature Set, that gave us the best ACC in Chapter 4, is used for training and testing in this chapter. In addition, the CICIDS2017 dataset is also used in this chapter for further evaluation. We extract a subset of six features out of 80 features of the CICIDS2017 dataset for our research. These features have an SDN-related nature and can be extracted easily by SDN controllers. Details of these features can be seen in Table 5.1.

According to our experiments, A GRU-RNN with three hidden layers gives the best results in all experimental cases. Therefore, we implemented a GRU-RNN with three hidden layers in this chapter. A DNN is also implemented with the same structure as the proposed GRU-RNN for evaluation purposes. We used Keras [93] to implement our GRU-RNN, DNN, and VanilaRNN models. The Scikit-learn library [99] is used to implement the SVM algorithm and measure all

Feature Name	Description
Source Port	Source port of the flow
Destination Port	Destination port of the flow
Protocol	Protocol type of the flow
Flow Duration	Duration of the flow in microseconds
Flow Bytes/s	Number of flow bytes per second
Flow Packet/s	Number of flow packets per second

Table 5.1: The CICIDS2017's Feature Description

the evaluation metrics. The details of all models can be seen in Table 5.2.

Algorithm	Input Layer	Hidden Layer	Output Layer	Activation Function	Learning Rate
GRU-RNN	6	6,4,3	2	tanh, tanh, tanh, sigmoid	0.001
DNN	6	6,4,3	2	tanh, tanh, tanh, sigmoid	0.001
VanilaRNN	6	4	1	tanh, sigmoid	0.001

Table 5.2: Neural Network Model Structures

For the training phase, the batch size and epoch number are 100 and 10000 respectively. We use a Nadam optimizer [100] and Mean Squared Error (MSE) function for the GRU-RNN model. In addition, we added L1-regularization to our model to prevent overfitting during the training phase.

In general, the RNN architecture is the same as the DNN architecture (as in Figure 5.3). The main difference is that each neuron is now a GRU cell.

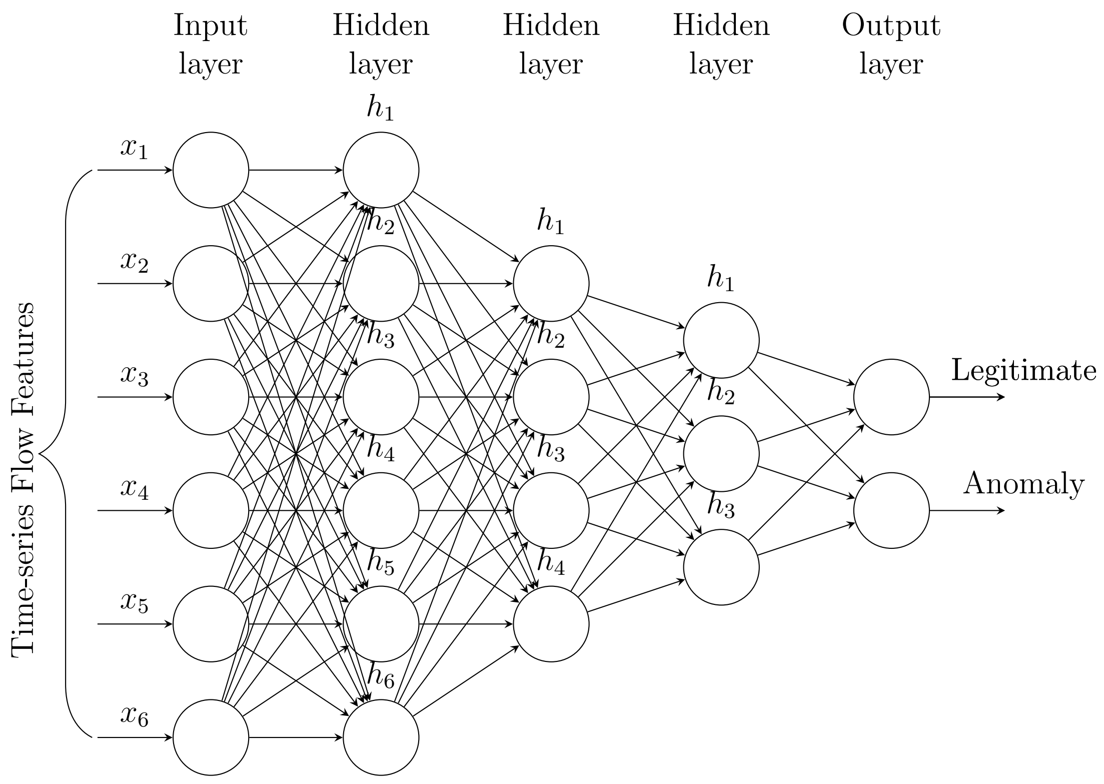


Figure 5.3: The RNN Structure

5.3.2 Experimental Result Analysis

To start with, we analyze training and testing processes of the GRU-RNN. As shown in Figure 5.4, it takes us a considerable amount of time to train the GRU-RNN. At the beginning, we can see that the GRU-RNN achieves a very high ACC of 95% in training, but the model performance is very poor with testing data. However, after around 6000 epochs, the GRU-RNN performance starts improving significantly. We can see that the ACC in training starts dropping gradually, but the ACC in testing starts increasing significantly. This is what we expect when training a DNN in general. The GRU-RNN is a quite complex algorithm, so it takes quite a long time to find its optimal point that gives the highest ACC. In the previous chapter, the DNN is quite lightweight and simple, so it just takes around 1000 epochs to achieve its best result. It is also interesting to see that if we keep training the GRU-RNN for a longer time, the model performance drops again. This is the time when overfitting happens.

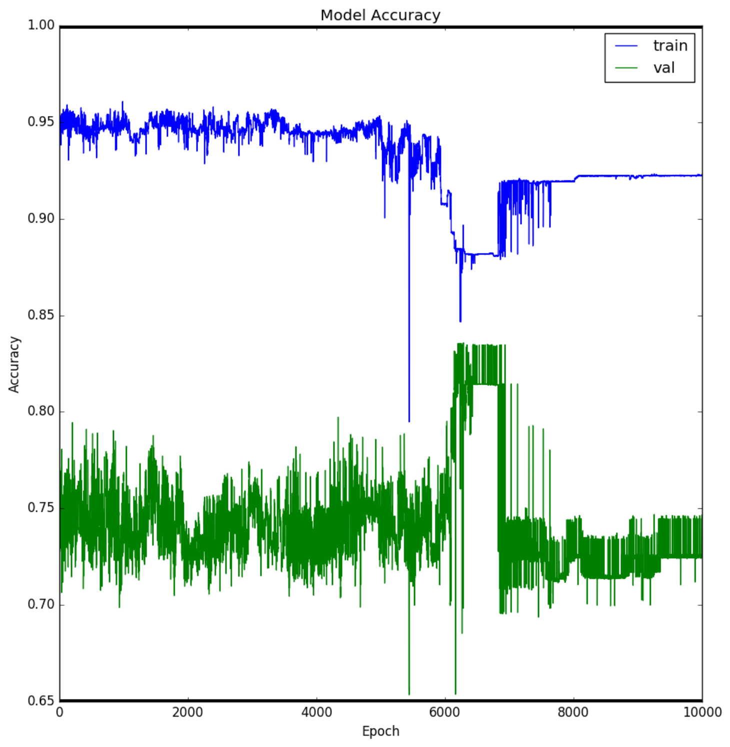


Figure 5.4: Traing and Testing Phase Evaluation

The best model from our training process will be taken for further evaluation. The results in Table 5.3 show that our approach outperforms other methods in terms of detection ACC. The DNN, coming in second place, shows the potential of the DL approach in anomaly detection. The VanilaRNN gives the worst result compared with its counterpart GRU-RNN. This is the expected result. As mentioned in the previous section, the VanilaRNN suffers from the vanishing gradient problem that makes the detection ACC becomes worst after a long training time.

The result of the GRU-RNN shows that it solves the vanishing gradient problem very well and then improves the ACC significantly.

Algorithm	ACC (%)
VanilaRNN	44.39
SVM	65.67
DNN	80.7
GRU-RNN (Proposed Model)	**89**

Table 5.3: Accuracy Comparison with Other Algorithms

For further evaluation, we present the anomaly detection performance of the GRU-RNN model in terms of ACC, P, R, and F1 (see Equations 2.1, 2.2, 2.3, and 2.4) on the NSL-KDD dataset. The performance of each class is calculated based on the TP and TN of each class. Details of the results given in Table 5.4 show that our GRU-RNN performs well for all the evaluation metrics. Both the legitimate and anomaly traffic traces are detected really well by the GRU-RNN. The detection rates of the legitimate and anomaly traffic traces are 89% and 90% respectively. The anomaly detection ACC of 90% shows that the GRU-RNN is good at detecting zero-day attacks.

Class Name	P (%)	R (%)	F1 (%)
Legitimate	87	89	88
Anomaly	91	90	90

Table 5.4: Performance Metric Evaluation for the NSL-KDD Dataset

We also compare the performance of our proposed model with other popular algorithms like VanilaRNN, Support Vector Machine (SVM) and DNN using the same subset of six features. As we can see in Figure 5.5, the GRU-RNN outperforms other algorithms in all the evaluation metrics. The GRU-RNN yields good results for both legitimate and anomaly traffic traces, while other algorithms just work well in only one class. The GRU-RNN outperforms other algorithms in detecting anomaly traffic as we expected. Flooding DDoS attacks in the NSL-KDD dataset are collective anomalies, so they can be characterized and detected by the GRU-RNN. These attacks can be missed by the DNN and other algorithms.

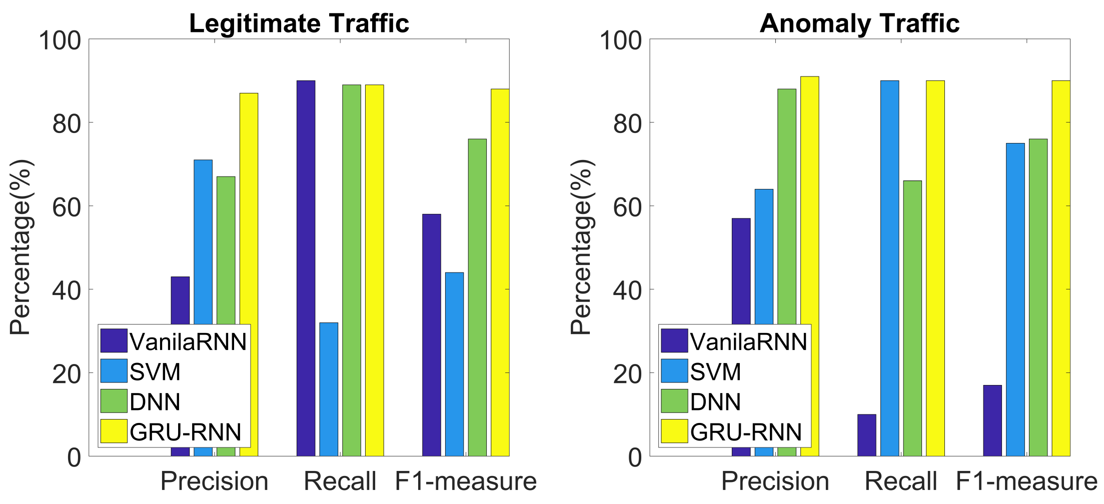


Figure 5.5: Performance Metric Comparison

The ROC curve is introduced to evaluate our proposed approach. Figure. 5.6 shows that the proposed GRU-RNN achieves the highest AUC amongst all the tested algorithms with a TPR of 90% and an FPR of 10%. As we can see, the GRU-RNN has the lowest FPR which is an essential factor of the IDS. The VanilaRNN gives the worst result as expected. The VanilaRNN suffers from the gradient vanishing problem, so it cannot give us a promising result after a long training time.

The P vs R curve shows the trade-off between P and R for different thresholds. A high area under the curve represents both high R and high P. An ideal system with a high area under the curve will return many results, with all results labelled

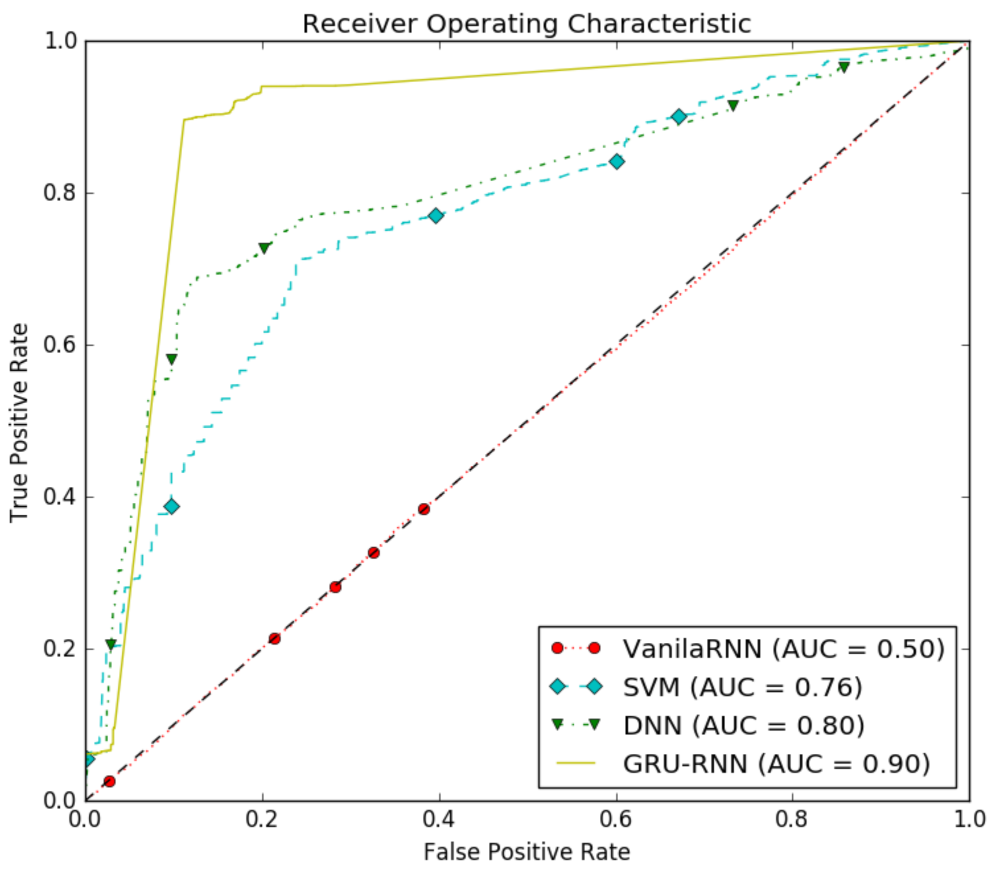


Figure 5.6: ROC Curve Comparison for Different Algorithms

correctly. As seen in Figure. 5.7, the GRU-RNN gives us the best results amongst all the algorithms. As the R threshold increases, the P decreases significantly for all the algorithms, except for GRU-RNN where increases P increases to 89%. This means the GRU-RNN can classify the network traffic with a high ACC.

Furthermore, we also compared the performance of our proposed model with others in the literature. Our GRU-RNN is compared with other state-of-the-art algorithms like SVM, DNN, and NB Tree algorithms. The NB Tree gave the best result in [86]. The results in Table 5.5 show that our proposed model outperforms all the previous methods. Our GRU-RNN performs better than the SVM and

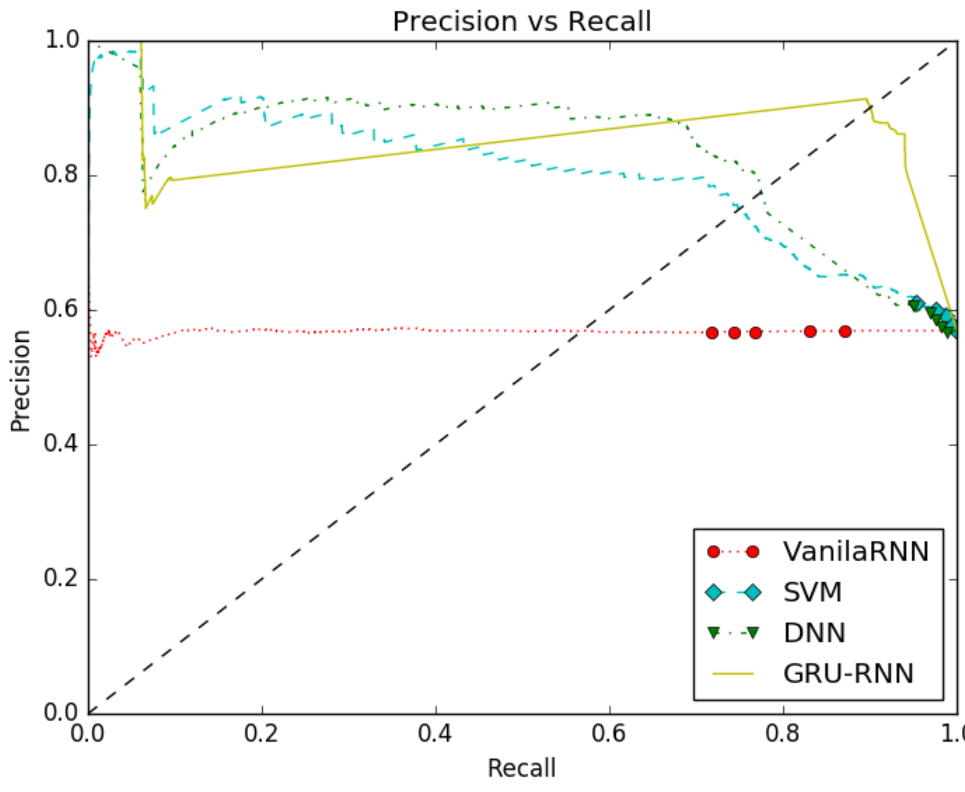


Figure 5.7: P vs R Curves

NB Tree algorithms that use the whole set of 41 features for training and testing. Even when we provide more network features, the other algorithms cannot detect the collective anomalies as good as the GRU-RNN. The GRU-RNN results also indicate a significant improvement in the ACC compared to the basic DNN in our previous chapter.

For further investigation, we evaluate the GRU-DNN performance as regards detecting DDoS attacks in the CICIDS2017 dataset. We compare the proposed GRU-DNN with DNN and ID3 algorithms. Results of ID3 algorithm are the best achieved from the CICIDS2017 dataset in [89]. Table 5.6 gives details of our evaluation. As can be seen, the proposed GRU-DNN has better results in all the evaluation metrics compared with the best result from [89]. The DNN yields

Method	**ACC (%)**
SVM [86]	69.52
DNN [101]	75.75
NB Tree [86]	82.02
GRU-RNN (Proposed Model)	**89**

Table 5.5: Accuracy comparison with previous studies. The GRU-RNN and DNN use the Mixed Feature Set with 6 features. The SVM and NB Tree use the NSL-KDD dataset with 41 features

slightly lower results than that of the ID3. The proposed GRU-DNN can work well with diverse and complex traffic traces and detect almost all types of DDoS attacks in the CICIDS2017 dataset.

Method	**P (%)**	**R (%)**	**F1 (%)**
DNN	97	97	97
ID3 [89]	98	98	98
GRU-RNN (Proposed Model)	**99**	**99**	**99**

Table 5.6: Performance Metric Evaluation for the CICIDS2017 dataset

From the above results, the GRU-DNN shows its strong potential in dealing with low-dimensional time series traffic. Therefore, it is a potential solution for intrusion detection in the SDN paradigm.

5.4 Network Performance Evaluation

In this section, we evaluate the effect of our proposed GRU-RNN on the performance of the POX controller in the SDN environment. The network evaluation setup is also as in Chapter 4 (Section 4.3.2).

5.4.1 Throughput Evaluation

Throughput evaluation indicates the performance of the controller under heavy traffic conditions. Figure. 5.8 depicts the average response rate of the controller under three testing scenarios.

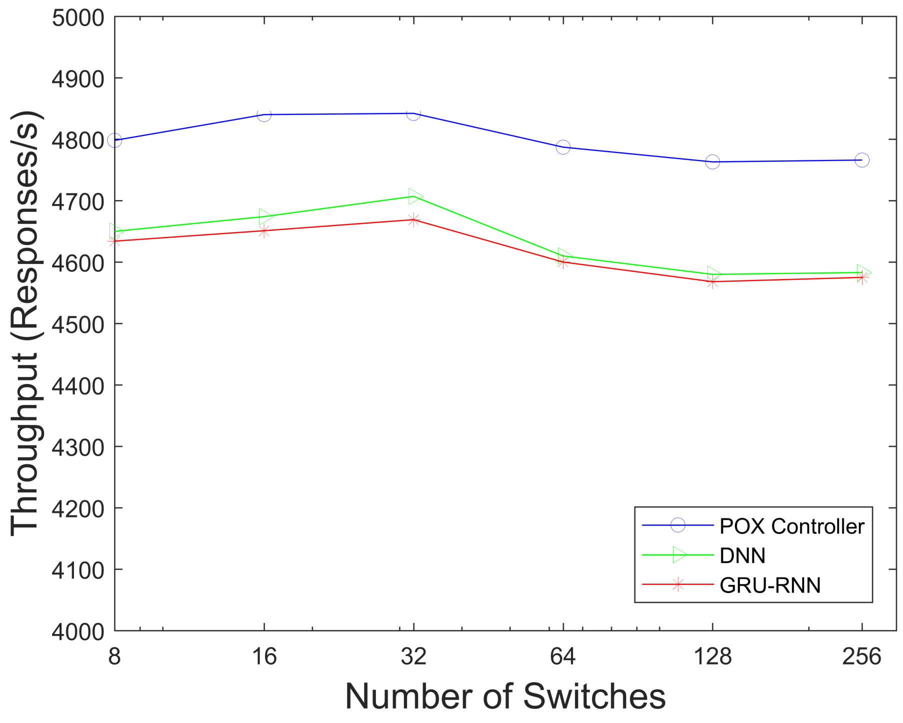


Figure 5.8: Throughput Evaluation

As we can see, both the DNN and GRU-RNN cause overhead on the controller. The DNN algorithm is simpler than the GRU-RNN, and so it gives a slightly better network performance than that of the GRU-RNN. However, the GRU-RNN outperforms the DNN in terms of the detection ACC. The effect of the GRU-RNN on the controller performance is predictable. The network throughput decreases slightly when the network size increases from 32 switches to 64 switches. The network performance degrades by about 3.5%

when the network size is under 32 switches. When we increase the size to over 64 switches, the throughput drops by about 4%. The GRU-RNN module has to send several OFP_FLOW_STATS_REQUEST messages and process OFP_FLOW_STATS_REPLY messages while processing PACKET_IN messages. Thus, the overhead on the controller of the GRU-RNN module is unavoidable. However, throughput degradation is quite low and can be improved in the future.

5.4.2 Latency Evaluation

Latency evaluation indicates the length of time that the controller takes to process one single packet. Figure. 5.9 shows the average response time of the controller under three testing scenarios. As we can see, the network latency increases along with increasing the network size. When we increase the network size, the load on the controller is increased as well and this causes the overhead. The GRU-RNN still has the highest overhead amongst all. It takes time for the GRU-RNN to process the OFP_FLOW_STATS_REPLY messages, PACKET_IN messages and detect anomaly flows, so the overhead is unavoidable. The overall degradation is about 7% in all cases. This overhead is not significant and can be improved in the future.

All in all, the overhead caused by the GRU-RNN on the SDN controller is quite low, and so our proposed approach has significant potential for real-time intrusion detection in the SDN paradigm.

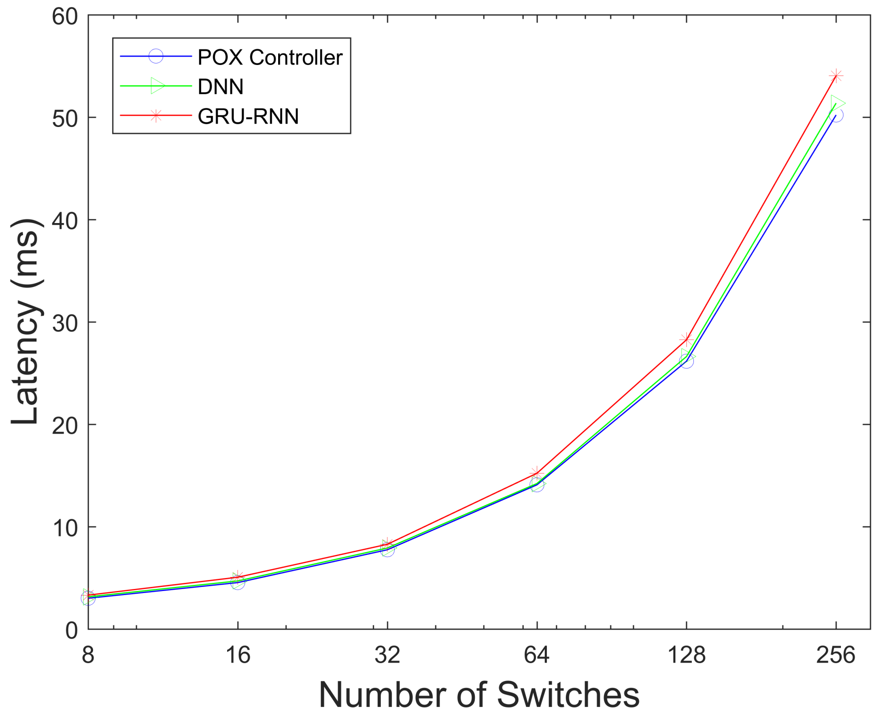


Figure 5.9: Latency Evaluation

5.5 Conclusion

In this chapter, we presented an Anomaly-based NIDS in the SDN environments using the GRU-RNN algorithm. We showed that our proposed approach outperforms other state-of-the-art algorithms with an ACC of 89% and 99% for the NSL-KDD and CICIDS2017 datasets. Although the GRU-RNN is more complex and takes a longer time than the DNN in Chapter 3 for training and testing. The detection ACC has been improved significantly compared to the work in Chapter 3. There is a trade-off between performance and security. However, the performance still can be improved in the future. Our scheme uses a minimum number of features compared to other state-of-the-art approaches so computational costs can be reduced significantly. In addition, the network performance evaluation showed that our proposed approach does not significantly affect the controller performance. This makes our model a strong candidate for real-time detection. In the future, we will optimize our model and use other features to increase the ACC and reduce the overhead on the controller. We will also try to extend our research to unsupervised intrusion detection approaches.

Chapter 6

Deep Learning Approach Combining Stacked Autoencoder with One-class SVM for DDoS Attack Detection in SDNs

In This Chapter:

As mentioned in Chapter 1, the lack of network datasets is one of the main problems for anomaly detection in SDNs. Currently, most of the proposed ML/DL intrusion detection approaches are in a supervised manner that requires labelled and well-balanced datasets for training. However, this needs a lot of time and human expertise to prepare these datasets. Besides, network attacks keep evolving, especially DDoS attacks, so it is challenging to get good labeled datasets. The goal of this chapter is to detect the DDoS attacks in an unsupervised manner by using the flow table information. In this chapter, we propose a hybrid approach using the Stack Autoencoder and One-class Support Vector Machine (SAE-1SVM) for DDoS attack detection. The SAE-1SVM shows that it can reduce the processing time significantly while maintaining the high DDoS attack detection rate.

6.1 Introduction

6.1.1 Motivation

Because of the SDN architecture, the SDN controller is a single point of failure and a target for attackers. The SDN controller is vulnerable to DoS and DDoS attacks. These attacks can take down the SDN controller and then the whole network. Therefore, DDoS attacks directly threaten the network and service availability. Detecting and mitigating DDoS attacks effectively become an important part of the NIDS. Several works [66] [15] [77] [78] have been done to tackle this problem in SDNs. Most of these works are for the supervised approach. This approach requires balanced and labelled datasets for training. However, these datasets are not always available for researchers, and they are especially rare under the context of SDN.

Unlike supervised learning approaches, the unsupervised learning approach does not need label information for the data and can address the imbalanced classification problems. One-class Support Vector Machine (OC-SVM) for a long time has been one of the most effective anomaly detection methods and is widely adopted in both research and industrial applications. However, the biggest issues for OC-SVM is the capability to operate with large and high-dimensional datasets due to inefficient features and optimization complexity. Their training speed is also heavily affected by the size of the dataset. As a result, conventional OC-SVM may not be desirable in big data and high-dimensional anomaly detection applications. Besides, Autoencoder recently emerges as an effective intrusion detection approach in different fields [102] [103] [104]. In these researches, a reconstruction error is used to detect anomalies. In this chapter, we propose an unsupervised hybrid approach combining Stack Autoencoder with OC-SVM (SAE-1SVM) for DDoS attack detection in the SDN.

6.1.2 Contribution

Our main contributions are as follows:

- We introduce an unsupervised DDoS attack detection approach in the SDN paradigm using the SAE-1SVM. To the best of our knowledge, this is the first attempt to use the SAE-1SVM for DDoS attack detection in the SDN environment. In our work, the Stack Autoencoder learns the patterns of legitimate traffic and also compresses input data into a lower dimension. These lower-dimensional and higher-level data is now more suitable for the OC-SVM to process.
- Our SAE-1SVM approach yields a detection rate of 99.35% using a minimum number of features compared to other state-of-the-art approaches.
- We also evaluate the computational overhead of the proposed approach. The results show that our approach has significant potential for real-time intrusion detection.

6.1.3 Chapter Organization

This chapter is organized as follow. In section 6.2, we introduce DDoS attacks and its taxonomy. Section 6.3 describes our proposed hybrid approach for DDoS attack detection. Section 6.4 shows the performance evaluation of our approach. Finally, conclusions and future work are discussed in section 6.5.

6.2 Denial-of-Service Attacks

DoS and DDoS attacks are serious threats in current networks. Even though DDoS attacks have been recognized in networks, since the origins of the Internet the number of DDoS attacks in today's networks still increase. The victim of DDoS attacks can be any business or critical infrastructure. For example, on 23rd October 2015, TalkTalk in England had been hit by a DDoS attack that allowed the attacker access to banking accounts and personal information of over four million UK customers. A DDoS attack is an attempt to exhaust a victim's bandwidth or disrupt legitimate users' access to services. This attack is relatively easy to perform, hard to defend against, and the attacker is rarely traced back because of the distributed nature of DDoS attacks. The attacker launches a DDoS attack using a botnet-group of zombies-to generate a vast amount of traffic against a victim's web server. Zombies or computers that are part of a botnet are usually recruited through the use of worms, Trojan horses or back doors. Figure 6.1 describes a topology of DDoS attacks.

The complexity of the attack increases due to the zombies modifying the packets, commonly spoofing the source. As a consequence, it becomes even more difficult to trace the origin of the attack. Some well-known bandwidth-consuming DDoS attacks are UDP flood, TCP flood and SYN flood attack. These high rate DDoS attacks significantly deviate from the legitimate traffic so we can detect them comparatively easier. However, unlike bandwidth-consuming DDoS attacks, the Slowloris attack uses a low amount of bandwidth and mimics the legitimate traffic, so it is more difficult to detect. DDoS flooding attacks can be classified into two categories based on the protocol level that is targeted:

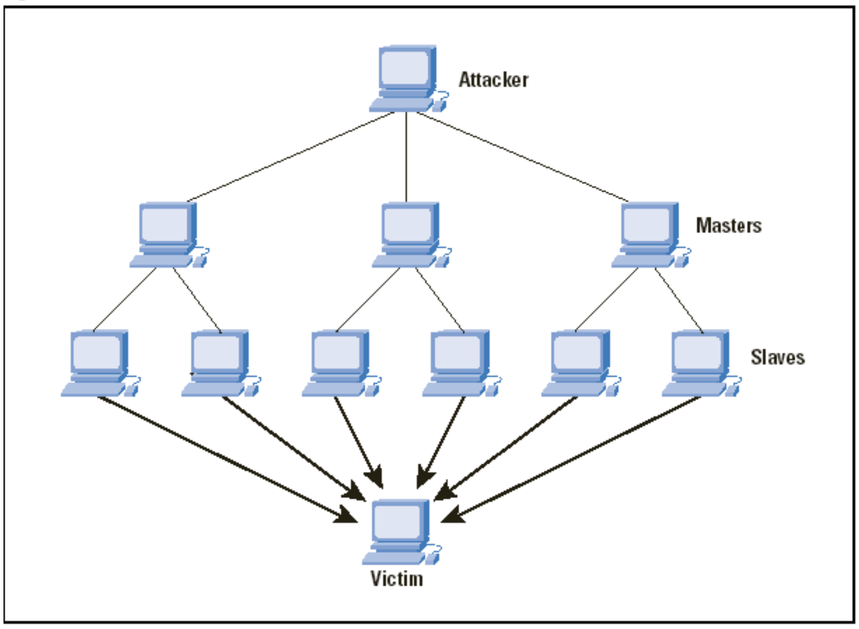


Figure 6.1: DDoS Attack Topology

- Network/transport-level DDoS flooding attacks: these attacks have been mostly launched using TCP, UDP, ICMP and DNS protocol packets. There are four types of attacks in this category: flooding attacks, protocol exploitation flooding attacks, reflection-based flooding attacks, amplification-based flooding attacks

- Application-level DDoS flooding attacks: these attacks focus on disrupting legitimate user's services by exhausting the server resources (e.g., Sockets, CPU, memory, disk/database bandwidth, and I/O bandwidth). Application-level DDoS attacks generally consume less bandwidth and are stealthier in nature compared to benign traffic. However, application-level DDoS flooding attacks usually have the same impact on the services since they target specific characteristics of applications such as HTTP, DNS, or Session Initiation Protocol (SIP). There are two types of attacks in this category:

reflection/amplification based flooding attacks and HTTP flooding attacks.

Within the operation of SDN, there are two options for the handling of new flow when no flow match exists in the flow table. Either the complete packet or a portion of the packet header is transmitted to the controller to resolve the query. In this case, it would be easy for attackers to execute a DoS attack on the node by setting up many new and unknown flows. As the memory element of the node can be a bottleneck due to the high cost, an attacker could potentially overload the switch memory. Therefore attackers might simply flood the flow tables of the switch using conventional DDoS methods. The controller and data plane communication channel are also vulnerable to various DDoS attacks. Zombie hosts generate unknown packets with bogus header fields which switches then forward to the controller for advice. Because of large amounts of traffic, legitimate packets might be dropped by the controller, and so there is a possibility to slow the processing power significantly enough to cause an overload in control operations.

Defending against DDoS attacks is a challenging issue, and in order to do so, we have to first detect them. There are several methods for detecting DDoS attacks like statistics-based method [105], and clustering method [106]. DDoS attacks can be mitigated by some defence mechanisms like firewall, load balancing. However, these defence mechanisms have their own limitations. Despite all the effort to tackle these attacks, DDoS attack strategies are evolving, so it is tough to detect and mitigate all of these attacks. There are several companies providing commercial DDoS protection services (e.g., Radware, CloudFlare).

6.3 SAE-1SVM for DDoS Attack Detection

An autoencoder (AE) consists of one input layer, one or more hidden layers and one output layer. The input and output layers always have the same sizes. A general structure of an AE is shown in Figure 6.2.

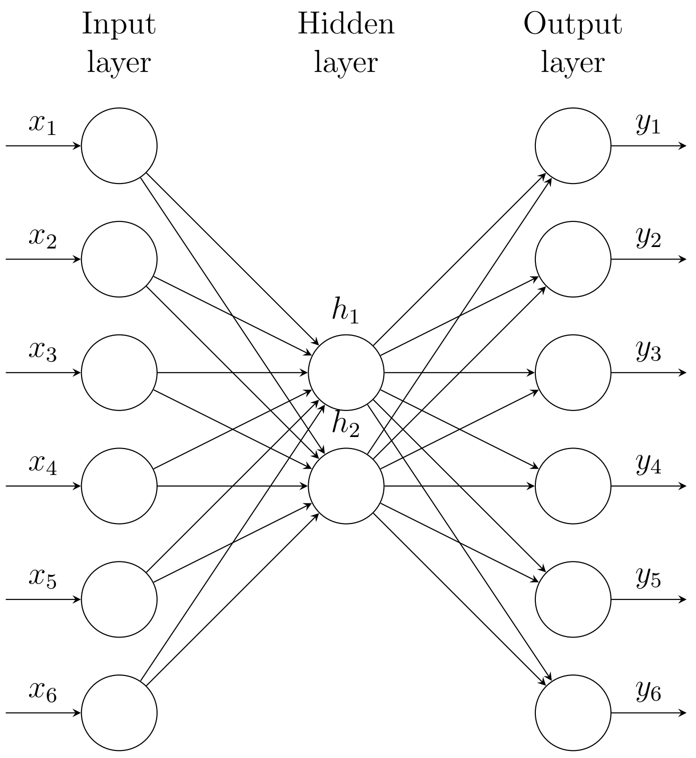


Figure 6.2: A General Structure of an AE

The AE has two phases which are encoding and decoding. For encoding process, input data $\mathbf{x}$ is compressed into a low-dimensional representation $\mathbf{h}$ and then the decoder reconstructs the input based on the low-dimensional representation.

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{6.1}$$

$$\mathbf{y} = f(\mathbf{W}'\mathbf{h} + \mathbf{b}'), \tag{6.2}$$

where $f(\cdot)$ is a non-linearity activation function, $\mathbf{W}$ and $\mathbf{W}'$ are hidden weight matrices, $\mathbf{b}$ and $\mathbf{b}'$ are biases and $\mathbf{y}$ is output vector.

The main goal of training the AE is to minimize the difference between the input $\mathbf{x}$ and output $\mathbf{y}$. Therefore, a MSE loss function is used as follows:

$$L(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2 . \tag{6.3}$$

In order to learn feature representations of input data, AEs are stacked successively to form a deep AE (SAE). The learned feature representations will be used as inputs for other classifiers. In this work, the OC-SVM is used as the classifier for anomaly detection.

The OC-SVM [107] is an unsupervised approach for classification. The OC-SVM tries to learn a hyperplane that best separates all the data points from the origin:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) - \rho, \tag{6.4}$$

where $\phi(\cdot)$ is a feature projection function that maps an input vector $\mathbf{x}$ into a higher dimensional feature space, $\mathbf{w}$ is a decision hyperplane normal vector which is perpendicular to the hyperplane, and ρ is an intercept term. We can obtain $\mathbf{w}$ and ρ by solving an objective function:

$$\min_{\omega,\xi,\rho} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho, \tag{6.5}$$
$$\text{subject to: } \mathbf{w}^T \phi(\mathbf{x}_i) \geq \rho - \xi_i, \xi_i > 0,$$

where the meta-parameter $\nu \in (0, 1]$ determines the upper bound on the fraction of outliers and the lower bound on the number of training samples used as support vectors, and ξ_i are non-zero slack variables for penalizing the outliers.

By using Lagrangian techniques and a kernel function for the dot-product calculations, the decision function becomes:

$$f(\mathbf{x}) = \sum_{i}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho, \tag{6.6}$$

where α_i is a Lagrange multiplier, and $k(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$ is a kernel function. A Radial Basic Function (RBF) kernel is employed in our experiment:

$$k(\mathbf{x}_i, \mathbf{x}) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}\|^2}, \gamma > 0. \tag{6.7}$$

In this chapter, we propose a hybrid approach combining SAE with OC-SVM for DDoS attack detection. Figure 6.3 gives a general structure of the proposed SAE-1SVM. The SAE-1SVM is trained with legitimate traffic traces. At first, the legitimate traffic traces are trained with the SAE to extract the low-dimensional representation, and then the low-dimensional representation is trained with OC-SVM for DDoS attack classification. Because the SAE-1SVM is trained with the legitimate traffic only, the anomaly traffic will be considered as outliers and then detected.

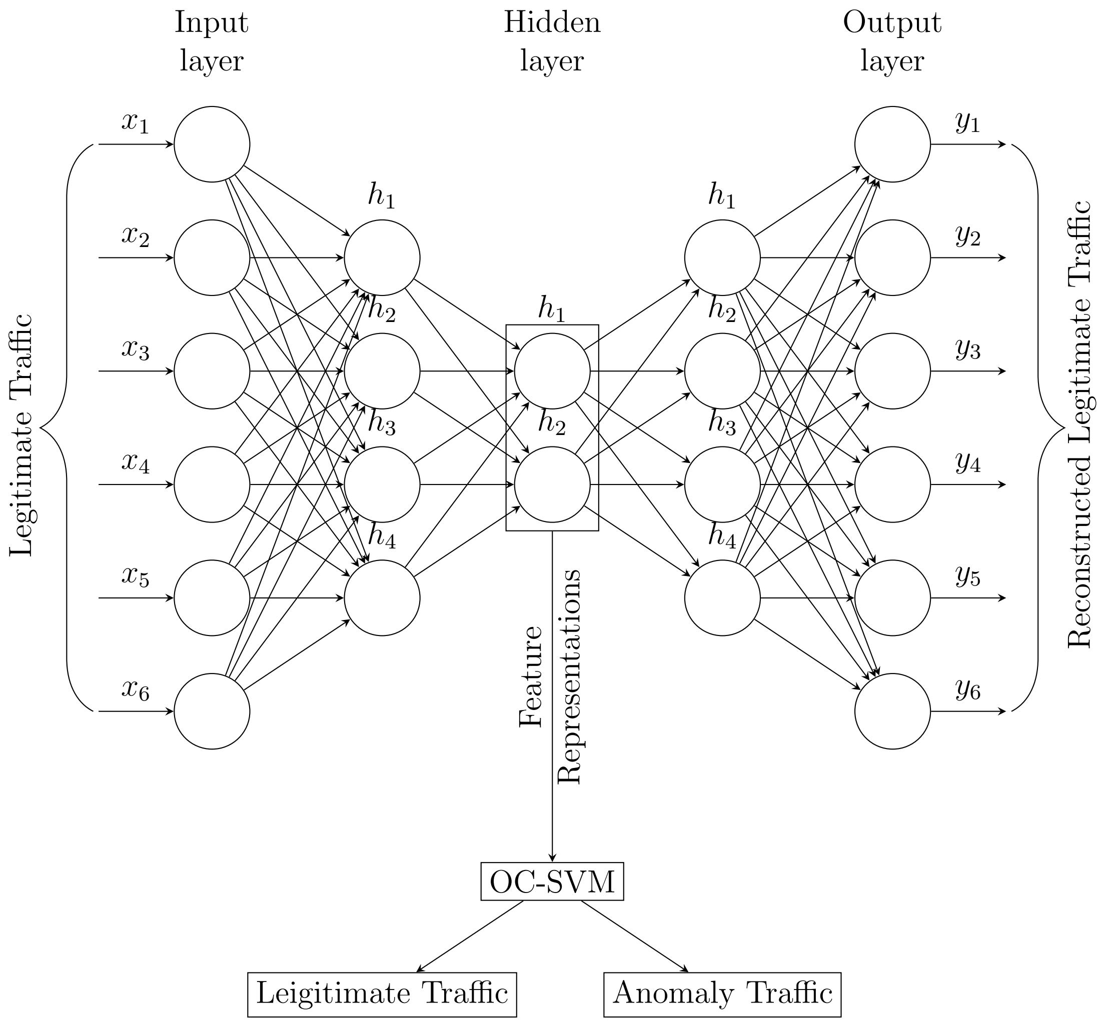


Figure 6.3: SAE-1SVM System Detail

6.4 Detection Performance Evaluation

6.4.1 Experimental Setup

In our experiment, the SAE architecture is implemented with all hyper-parameters given in Table 6.1. For the OC-SVM model, the parameters ν and γ are chosen from the range $\{10^{-10}, 10^{-9}, \ldots, 10^{0}\}$. After the optimizing process, the parameters $\nu = 10^{-2}$ and $\gamma = 10^{-2}$ are choosen for the experiment.

Variable	**Parameters**
Activation Function	Tanh
Loss Function	Mean Squared Error
Learning Rate	0.001
Batch Size	10
Epoch	1000

Table 6.1: The SAE Architecture

In this chapter, we just focus on DDoS attack detection, so we only use the CICIDS2017 dataset for training and testing. This dataset contains the most recent types of DDoS attacks. In Table 6.2, we describe thirteen features used in this experiment.

Feature Name	**Description**
Source Port	Source port of the flow
Destination Port	Destination port of the flow
Protocol	Protocol type of the flow
Flow Duration	Duration of the flow in microseconds
Total Fwd Packets	Total packets in the forward direction
Total Length of Fwd Packets	Total size of packet in forward direction
Fwd Packet Length Mean	Standard deviation size of packet in forward direction
Flow Bytes/s	Number of flow bytes per second
Flow Packet/s	Number of flow packets per second
Flow IAT Mean	Mean time between two packets sent in the forward direction
Flow IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd Packets/s	Number of forward packets per second
Subflow Fwd Bytes	The average number of packets in a sub flow in the forward direction

Table 6.2: The CICIDS2017's Feature Description

Again, these features are selected under the context of SDN architectures.

As a result, the input and output layers of our SAE contain thirteen neurons respectively. Details about the number of neurons of each network architecture used in this experiment are shown in Table 6.3.

Architecture	Input Layer	Hidden Layer	Output Layer
AE	13	2	13
SAE_1	13	6,2,6	13
SAE_2	13	10,8,6,4,2,4,6,8,10	13

Table 6.3: Network Architecture Details

6.4.2 DDoS Attack Detection with a Hard Threshold

The AE is commonly used to detect anomaly with the idea that behaviors of attacks are different from those of legitimate traffic. Therefore, the AE will be trained only with the legitimate traffic and then tries to reconstruct them with the highest ACC. The anomaly traffic is not used for training, so the AE cannot reconstruct them correctly. We can detect anomaly traffic based on this difference. In this section, we analyze the effect of network architecture on the reconstruction performance.

We compare the reconstruction ACC of the AE, SAE_1, and SAE_2 in Table 6.4. As we can see, the SAE_2 yields the best reconstruction ACC at 98.6%. The AE gives a quite low reconstruction ACC at 85%. With just one hidden layer, we cannot learn good feature representations, so the reconstructed input is just a lossy version of the original inputs. It shows that a deeper SAE can learn feature representations better and then reconstructs the inputs with a higher ACC. Therefore, the SAE_2 will be chosen for further experiment.

As in [102], [103], and [104], we also employ the reconstruction error to detect

Architecture	Reconstruction ACC (%)
AE	85
SAE_1	96
SAE_2	98.6

Table 6.4: Reconstruction ACC Comparison

anomalies. The SAE_2 is trained to minimize the reconstruction error, so the error rate should be quite small with the legitimate input traffic. If any anomaly traffic is fed into the SAE_2, the SAE_2 could not recognize it and reconstruct it correctly. In this case, the reconstruction error is higher than normal, and so we can detect the network attack.

Figure 6.4 shows the difference in the reconstruction error rate between the legitimate and anomaly traffic. As seen in Figure 6.4, some anomaly error rates dramatically deviate from the standard legitimate error rate. It is quite easy for us to detect these kinds of attacks with a hard threshold. However, we also can see that some anomaly traffic has quite the same reconstruction error as the legitimate ones. These attacks cannot be detected with a hard threshold. The legitimate and anomaly reconstruction error rates are not linearly separated, so we cannot define any good hard threshold to detect anomaly traffic. If we set a high hard threshold, the FPR will increase significantly, and our network will become vulnerable to attacks. If we set a low hard threshold, the FAR will increase and so the NIDS can block the legitimate traffic. In Table 6.5, we compare the performance of different threshold values in terms of ACC, P, R, and F1 (see Equations 2.1, 2.2, 2.3, and 2.4). As we can see, with a higher threshold, we get a higher detection ACC. However, the other evaluation metrics drop dramatically with high threshold values. The reason for this trend is that more legitimate traffic is classified correctly with a higher threshold, but we also misclassify anomaly traffic. Even with a small threshold of 0.01, the detection P is still worst.

Threshold	**ACC (%)**	**P (%)**	**R (%)**	**F1 (%)**
0.01	54.9	21	85	33.67
0.03	55.3	13.9	4.3	6.5
0.05	58.2	1.7	0.26	0.45

Table 6.5: Accuracy Metrics for Different Thresholds

The AE approach with a hard threshold for anomaly detection works quite well in [102], [103], and [104] but it does not perform well in our experiments. This phenomenon can be because of the complexity of DDoS attacks in the CICIDS2017 dataset. Some DDoS attacks in this dataset try to mimic behaviours of legitimate traffic, so they are hard to detect. The construction error rate in Figure 6.4 can explain our theory quite well with the reconstruction error rate of both legitimate and anomaly traffic are quite close to each other.

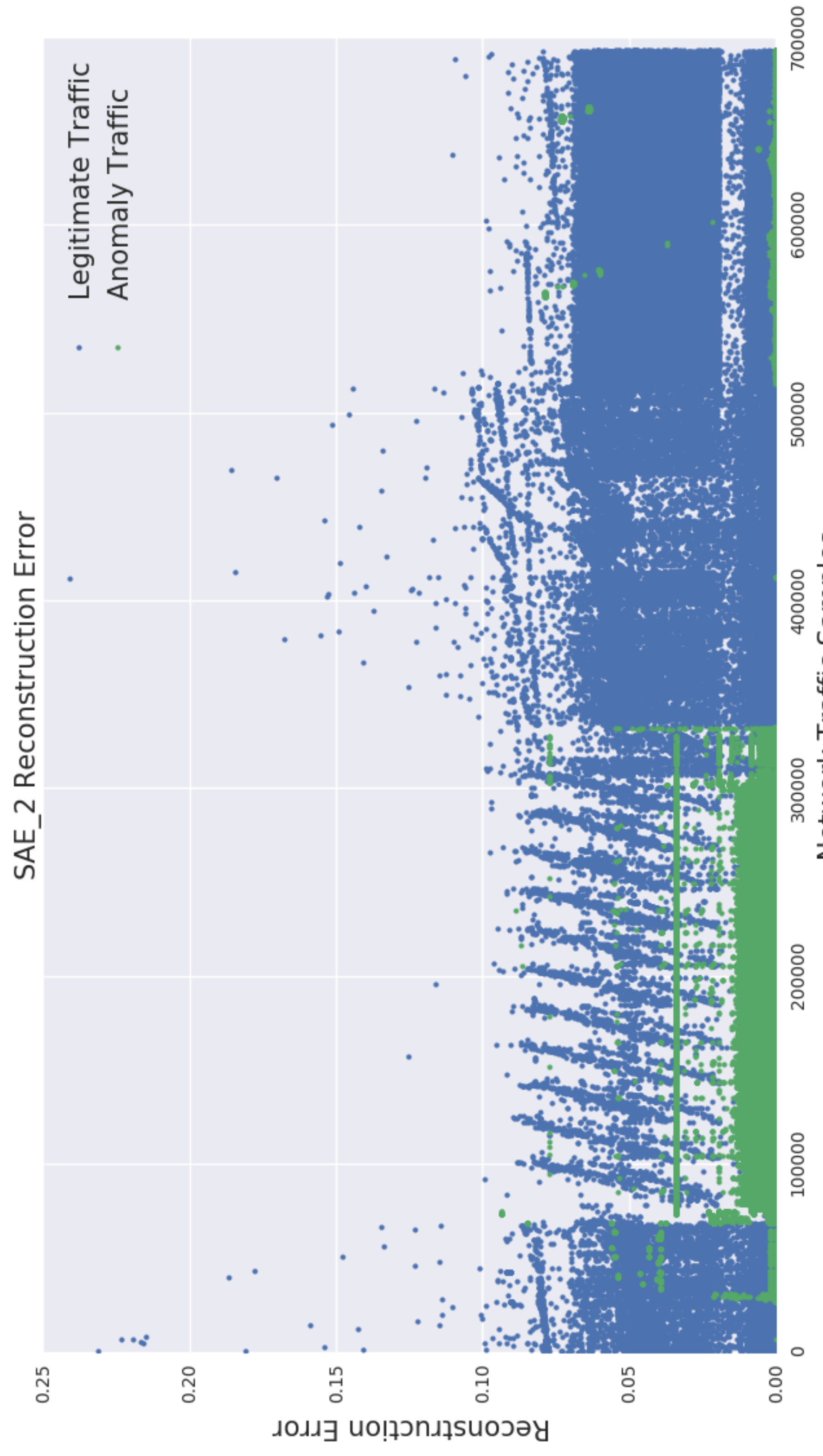


Figure 6.4: Reconstruction Error Rate Comparision

6.4.3 DDoS Attack Detection with the SAE-1SVM

In this section, we analyze the detection performance of the SAE-1SVM. Instead of using the hard threshold as the previous section, we employ a completely unsupervised detection approach. The general architecture of the SAE-1SVM has been described in Figure 6.3. In this experiment, we employ the SAE_2 architecture from the previous experiment for feature representation learning. To begin with, we present the detection performance of the SAE-1SVM in term of ACC, P, R and F1 with the Wednesday dataset. We evaluate the proposed model for binary-classification. We compare the performance of SAE-1SVM with classical OC-SVM. We also compare the SAE-1SVM with a DL algorithm combined Convolution Neural Network (CNN) and Long-Short Term Memory (LSTM) proposed by Abdurraman Pektas and Tankut Acarman [108]. They also use the CICIDS2017 dataset for performance evaluation.

The overall detection performance comparison is depicted in Table 6.6. According to the experimental results shown in Table 6.6, we can see that the SAE-1SVM outperforms the OC-SVM in all of the evaluation metrics. Specifically, the SAE-1SVM achieves a much higher P than the OC-SVM. The SAE-1SVM also achieves better results than the CNN+LSTM algorithm. The main reason for this high performance is that the OC-SVM in the SAE-1SVM is trained with the low dimensional representation. The low dimensional representation helps the OC-SVM characterize the network traffic better, so the detection ACC can be improved significantly.

For further evaluation, we examine the SAE-1SVM with the Friday dataset. The details of our evaluation are shown in Table 6.7. The detection ACC for the

Algorithm	ACC (%)	P (%)	R (%)	F1 (%)
OC-SVM	98	96.26	98.21	97.22
CNN+LSTM [108]	98.87	98.89	98.83	98.86
SAE-1SVM	**99.35**	**99.97**	**98.28**	**99.11**

Table 6.6: The Evaluation Metric Comparison

Friday dataset is 91.43%. This result indicates that the SAE-1SVM can classify unseen traffic traces as well. The P and R results for this dataset are lower than those of the Wednesday dataset. Some legitimate samples have been classified as anomaly samples. This is because the SAE cannot generalize legitimate traffic traces well enough. The SAE performance can be improved in the future with further tuning and optimization. However, the overall results still show that the SAE-1SVM can detect both DoS and DDoS attacks with a high ACC.

Evaluation Metric	Result (%)
ACC	91.43
P	88.6
R	71.79
F1	79.31

Table 6.7: The Detection Performane Results with the Friday Dataset

The computational time is an important factor in evaluating the performance of a classifier. In the era of big data, the classifier has to process a large amount of data for training and testing. Reducing the computational time is also very important. The training and testing times of each algorithm are presented in Table 6.8. As we can see, the SAE-1SVM consumes significantly less time than OC-SVM in both training and testing processes. The SAE-1SVM is 27 and 6 times faster than the OC-SVM in training and testing respectively. The OC-SVM module in the SAE-1SVM now only processes 2-dimensional inputs compared to

13-dimensional inputs in the original OC-SVM, so the processing time has been reduced significantly. In the SAE-1SVM, the OC-SVM processes more representative but lower-dimensional inputs. Therefore, the SAE-1SVM has excellent potential for real-time NIDS.

Algorithm	Traing Time (s)	Testing Time (s)
OC-SVM	5110	141
SAE-1SVM	189	26

Table 6.8: The Training and Testing Time Comparison

6.5 Conclusion

In this chapter, we presented a hybrid unsupervised DL approach for DDoS attack detection. The above results show that our proposed approach has strong potential in detecting DDoS attacks using limited flow features. The experimental results also show that our SAE-1SVM can deal really well with imbalanced and unlabeled datasets. Although the final results have a quite high FPR rate, the SAE-1SVM can be improved in several ways. Several DL approaches can be applied to the SAE to improve generalizing capability. We can also optimize the OC-SVM by a grid search algorithm. In future research, we will deploy our proposed approach in a real SDN testbed for more detail analysis. Detecting other kinds of network attacks will be considered in future research.

Chapter 7

Conclusions and Future Work

In This Chapter:

The concluding remarks following from the aforementioned work are now presented. Some limitations of this thesis are also presented. Additionally, future directions as a result of this work are identified.

7.1 Conclusions

As mentioned in the previous chapters, SDN brings us a critical dilemma: an important potential evolution of networking architectures, along with a very dangerous increase in security problems. SDN introduces new faults, and attack planes that did not exist before or were harder to exploit. These potential security issues are because of network programmability and control logic centralization in an SDN. However, an SDN can also be utilized to strengthen network security. In this thesis, we have implemented an end-to-end NIDS - DeepIDS - for the SDN architecture. The DeepIDS can be deployed in any SDN and which then takes advantages of global network overview for intrusion detection. As mentioned in

Chapter 1, several challenges must be solved when developing a NIDS. Therefore, in this thesis, we researched approaches to address these problems. In chapters 4 and 5, we studied the potential of supervised DL algorithms for intrusion detection under the SDN context. In Chapter 4, we focused on examining the effect of different learning rates and features sets on the detection ACC of the DNN model. We showed that basic flow-based features obtained from SDN controllers can be used to detect network attacks effectively by the DNN. The DNN also does not degrade the overall network performance which is a critical evaluation factor.

Chapter 5 focused on improving the detection ACC of the DeepIDS. We proposed the GRU-RNN that takes advantage of the time-series nature of network traffic to enhance the detection ACC. We demonstrated that the GRU-RNN does improve the detection ACC significantly compared to the DNN. Although the experimental results showed that the GRU-RNN is more complex and demands more computational resources than the DNN, the trade-off between the detection ACC and the network performance is still acceptable.

In chapter 6, we explored the field of unsupervised learning. The SAE-1SVM was introduced to deal with the unlabeled and imbalanced dataset in Chapter 1. The SAE-1SVM is designed to learn the characteristic of legitimate network traffic and then detect anomalies that have a different nature. Another advantage of the SAE-1SVM is that it overcomes the high complexity problem of the OC-SVM.

The following is a brief summary of the key contributions of this thesis. These also answer the research questions raised in Chapter 1.

- An end-to-end NIDS has been proposed for the SDN paradigm. This system can monitor the whole network and collect all the necessary network information. This information can be then used to detect any abnormal activity and then mitigate it as soon as possible.

- Several DL algorithms from simple to complex have been developed for a flow-based anomaly detection approach. We have demonstrated that DL algorithms can achieve high detection ACC without degrading the performance of the SDN controller.

- The unlabeled and imbalanced dataset problem has been addressed and partly solved in this thesis.

7.2 Limitations

There are a few limitations associated with our work in this thesis.

- As mentioned in Chapter 1, the lack of an SDN-based dataset is a huge problem. Therefore, in this thesis, we have to adapt some conventional datasets to the SDN architecture. This adaption may not be close enough to a real SDN-based dataset but many researchers in the same field still use it. In Chapter 4 and 5, the packet-based NSL-KDD dataset is adapted for our experiments. This adaption may affect the generalization and application of our approach in the SDN context. However, we selected some most basic network features having similar characteristics in both packet-based and flow-based datasets to minimize this gap. We also try to use a flow-based dataset (The CICIDS2017 dataset) to fill the gap in our research.

- Because of the use of published datasets, all the attacks concerned in this thesis are just related to network and application layers (i.e., L2, L3 and

L7). This thesis does not consider any kind of attacks related to L1 physical layer which is also an important part of the network.

- Although our proposed approaches have been evaluated in term of throughput and latency, they have not been deployed in any real SDN testbed yet. Because of limited facilities and resources, a more detail network performance evaluation has not been done yet.

- Hyper-parameters of our DL models such as learning rate, batch size, number of neurons and number of layers was chosen as some state-of-the-art literature. We trained the DL models with different combinations of hyper-parameters to find the best results. However, the training process still can be further optimized to achieve a better result.

7.3 Future Work

In order to solve the above limitations, several improvements can be implemented as a part of our future work.

Analysis with Real SDN Testbed and Network Traffic

Currently, all the work has been done in an offline manner. All the DL algorithms are trained with several datasets to detect intrusion, but some of these datasets are outdated. In addition, some legitimate and anomaly traffic in these datasets are synthetic, so they cannot reflect real network scenarios. It would be better to implement our approach in an SDN testbed with real legitimate and anomaly traffic for further evaluation. It would be interesting to see how our method works with real networks and how quickly it can respond to the network attacks. Furthermore, taking into account the nature of streaming data, online ML/DL algorithms should be considered in future work. In addition, our current work is focused on developing the Anomaly Detection module of the DeepIDS. The Collector and Counter Measure module development should be done in future research.

Analysis of Hyper-parameters and Network Features

As can be seen in Chapters 3 and 4, we achieved the results for a potential NIDS. However, it would be beneficial for us to do an exhaustive grid search to optimize our DL models with different hyper-parameters. For DL algorithms, every problem and every dataset require different hyper-parameters, and so the analysis of hyper-parameters is necessary. But when we deploy the NIDS in a real network, several network features can be collected for intrusion detection. Chapter 4 shows that different feature sets have different effects on the detection ACC, and so we also need to do more research on network feature selection.

Analysis of DL Algorithms

In this thesis, we have implemented several types of DL algorithms which are both supervised and unsupervised approaches. However, these DL algorithms (i.e., DNN, RNN, SAE) are just a small part of DL techniques. Various combinations of DL algorithms can be implemented for the anomaly detection problem. Recently, CNN has yielded significant achievements in the field of image processing because of its ability to learn spatial correlations. CNN now has been gradually adapted to detect network anomalies. For example, RNN and CNN are combined in [108] to learn both temporal and spatial relations of network data that can improve the intrusion detection ACC. CNN can be combined with other DL algorithms for further research. However, network data is not an image, so we have to convert network data to an image's structure. This is not a clear process, so it would be interesting to see the application of CNN for intrusion detection in the future.

Exploiting Applications of SDN and DL in Internet of Things Framework

Finally, the Internet of Things (IoT) is an emerging technology and expected to have around 50 billion devices connected to the Internet by 2020. As the number of IoT devices is increasing dramatically, they are becoming the primary targets for hackers. Now IoT devices can be used to attack other networks on a large scale and then damage them severely. IoT devices generate a massive amount of network traffic with many types of data that could threaten the security of the whole network. Last year, Mirai's IoT DDoS attacks had taken down several vital services all over the world. These types of attacks are becoming more and more popular and are very difficult to detect and mitigate in a real-time manner. A combination of SDN and an IoT framework as in [109], [110] and [111] can strengthen the network security, so it is also a potential future research trend. SDN architectures can be used to monitor and collect network information from IoT networks. DL has the ability to learn valuable information from complex data, so it has also potential for intrusion detection in IoT networks.

7.4 Final Remarks

In summary, this thesis has introduced the application of DL in detecting intrusions under the context of an SDN architecture. According to the experimental results, we show the strong potential of DL. DL can learn the network's patterns with some limited raw netwok input features. With the use of NSL-KDD and CI-CIDS2017 datasets, we demonstrate the potential of DL in detecting anomalies in both small scale networks (i.e., Small Home/Office or Enterprise) and large networks. Although the achieved results are still far from real-life applications, this is just some first steps in solving a big problem, and several improvements can be made in the future.

Appendix **A**

Appendix

A.1 Preparing Data

- **Data Preprocessing:** formatting and cleaning data, removing all missing values, encoding all categorical values to numbers or one-hot vector.
- **Data Transformation:** transforming all preprocessed data for ML/DL by scaling, aggregating.

A.2 Performance Evaluation Processes

After preparing data and training an ML/DL, we need to evaluate the performance of the ML/DL model. The evaluation process can be done as the following:

- **Training and Testing Datasets:** during the training processing, the ML/DL will not be exposed to the testing dataset. Any prediction result in the testing dataset indicates the model performance in general. We are expecting a model with high ACC on the testing set.

- **Performance Measure:** the model performance then is further assessed by evaluation metrics (Section 2.4). These metrics are standard for classification problems, so they can be used for anomaly classification.

- **Literature Comparision:** the model performance then will be compared with some state-of-the-art literature for a final decision. In general, these literature use the same dataset as us for training and testing ML/DL models. All the classification problem also uses the same evaluation metrics, so it is really convenient for comparison.

A.3 Experiment Testbed Setup

The experiment testbed for SDN networks is implemented in an Intel PC using Mininet. Because of the lack of resources and the ease of experiment, a network with several switches and hosts in a star topology is emulated for simulation purpose. The testbed has three main components as seen in Figure A.1.

- **SDN Controller:** the POX [35] controller is used for SDN networks controlling purpose.

- **NIDS Module:** the DeepIDS module is written in Python and runs on top of the POX controller. This module is in charge of monitoring network traffic, detecting and mitigating anomalies.

- **Switches:** OpenFlow switches are emulated by Mininet and connected to the POX controller via OpenFlow protocol.

- **Hosts:** Several hosts are emulated by Mininet and connected to the switch in a star topology. These hosts act as attackers and victims in the simulation.

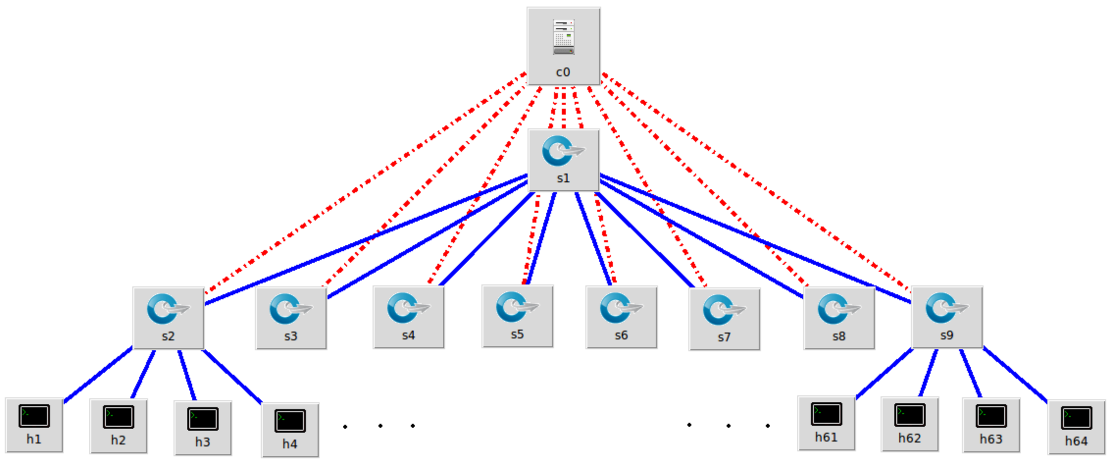


Figure A.1: The Emulated Network Topology

References

[1] "The sdn market size prediction," https://www.statista.com/statistics/468636/global-sdn-market-size/. Accessed 19 Feb 2019. vi, 2

[2] ONF, "Software-defined networking (sdn) definition," https://www.opennetworking.org/sdn-definition/. Accessed 12 Feb 2018, n.d. vi, 16

[3] OpenFlow_Switch_Specification_1.1.0, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf. vi, 18, 19, 22, 24

[4] Scikit-Learn, http://scikit-learn.org/stable/tutorial/machine_learning_map. vi, 35

[5] M. Copeland, https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. Accessed 10 Feb 2019, n.d. vi, 48

[6] Wikipedia, "Biological neuron model," https://en.wikipedia.org/wiki/Artificial_neural_network. Accessed 10 Feb 2019, n.d. vi, 49

[7] ——, “Overfitting,” https://en.wikipedia.org/wiki/Overfitting. Accessed 10 Feb 2019, n.d. vii, 58

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014. viii, 97, 98

[9] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience with a globally-deployed software defined wan,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013. 1, 15

[10] H. P. Centre, “China telecom and huawei unveil world’s first commercial deployment of sdn in carrier networks,” http://pr.huawei.com/en/news/hw-332209-sdn.htm. 1, 15

[11] D. Kreutz, F. Ramos, and P. Verissimo, “Towards secure and dependable software-defined networks,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 55–60. 2, 15, 25

[12] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015. 2, 25

[13] S. Scott-Hayward, S. Natarajan, and S. Sezer, “A survey of security in software defined networks,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016. 2, 25

[14] S. A. Mehdi, J. Khalid, and S. A. Khayam, “Revisiting traffic anomaly detection using software defined networking,” in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2011, pp. 161–180. 3, 36

[15] S. M. Mousavi and M. St-Hilaire, “Early detection of ddos attacks against sdn controllers,” in *Computing, Networking and Communications (ICNC), 2015 International Conference on.* IEEE, 2015, pp. 77–81. 3, 37, 42, 113

[16] S. Mukherjee and N. Sharma, “Intrusion detection using naive bayes classifier with feature reduction,” *Procedia Technology*, vol. 4, pp. 119–128, 2012. 3, 37

[17] R. Kokila, S. T. Selvi, and K. Govindarajan, “Ddos detection and analysis in sdn-based environment using support vector machine classifier,” in *Advanced Computing (ICoAC), 2014 Sixth International Conference on.* IEEE, 2014, pp. 205–210. 3, 36

[18] T. V. Phan, T. Van Toan, D. Van Tuyen, T. T. Huong, and N. H. Thanh, “Openflowsia: An optimized protection scheme for software-defined networks from flooding attacks,” in *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on.* IEEE, 2016, pp. 13–18. 3, 36, 83

[19] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *2010 IEEE symposium on security and privacy.* IEEE, 2010, pp. 305–316. 3, 36

[20] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012. 3, 52, 67, 95

[21] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, “Joint training of a convolutional network and a graphical model for human pose estimation,” in *Advances in neural information processing systems*, 2014, pp. 1799–1807. 3, 51

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011. 3, 52, 67, 95

[23] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, “On using very large target vocabulary for neural machine translation,” *arXiv preprint arXiv:1412.2007*, 2014. 3

[24] D. L. Tennenhouse and D. J. Wetherall, “Towards an active network architecture,” *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 5–17, 1996. 12

[25] “Devolved control of atm network.” http://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/. 12

[26] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, "Forwarding and control element separation (forces) protocol specification," Tech. Rep., 2010. 13, 18

[27] R. Enns, M. Bjorklund, and J. Schoenwaelder, "Netconf configuration protocol," *Network*, 2011. 13

[28] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12. 13

[29] E. L. Fernandes and C. E. Rothenberg, "Openflow 1.3 software switch," *https://github.com/CPqD/ofsoftswitch13.*, 2014. 15

[30] O. vSwitch, http://vswitch.org/. Accessed 10 Feb 2019, 2013. 15

[31] Y. Yiakoumis, J. Schulz-Zander, and J. Zhu, "Pantou: Openflow 1.0 for openwrt@ online," 2012. 15

[32] Pica8, http://www.pica8.org. Accessed 05 Jun 2019. 15

[33] OpenFlow_Switch_Specification_1.0.0, https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. 17, 18

[34] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008. 18

[35] POX, https://github.com/noxrepo/pox. Accessed 04 Jul 2018, 2009. 18, 43, 89, 138

[36] Maestro, http://zhengcai.github.io/maestro-platform/. Accessed 05 Jun 2019. 18

[37] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 13–18. 18

[38] SNAC, http://www.openflow.org/wp/snac. 18

[39] S. Ishii, E. Kawai, T. Takata, Y. Kanaumi, S. Saito, K. Kobayashi, and S. Shimojo, "Extending the rise controller for the interconnection of rise and os3e/nddi," in *Networks (ICON), 2012 18th IEEE International Conference on.* IEEE, 2012, pp. 243–248. 18

[40] "Floodlight," http://www.projectfloodlight.org/. Accessed 04 Jul 2018. 18

[41] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication.* ACM, 2012, pp. 289–290. 18

[42] KulCloud, "Mul openflow controller," http://sourceforge.net/p/mul/wiki/Home/. 18

[43] Ryu, http://osrg.github.io/ryu/. 18

[44] OpenDaylight, https://www.opendaylight.org/lithium. 18

[45] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open,

distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking.* ACM, 2014, pp. 1–6. 18

[46] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. 18

[47] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core," 2011. 18

[48] B. Pfaff and B. Davie, "The open vswitch database management protocol," 2013. 18

[49] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "Opflex control protocol," *IETF, http://datatracker. ietf. org/doc/draft-smith-opflex*, 2014. 18

[50] OpenFlow_Switch_Specification_1.2.0, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf. 18

[51] OpenFlow_Switch_Specification_1.3.0, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf. 18

[52] OpenFlow_Switch_Specification_1.4.0, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf. 19

[53] OpenFlow_Switch_Specification_1.5.0, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf. 19

[54] OpenFlow_Switch_Specification_1.5.1, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. 19, 25

[55] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, no. 6, pp. 3514–3530, 2017. 25, 26

[56] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980. 28

[57] Snort, https://snort.org/. 30

[58] D. Denning and P. G. Neumann, *Requirements and model for IDES-a real-time intrusion-detection expert system.* SRI International, 1985. 32

[59] R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru, and A. Valdes, "System design document: Next-generation intrusion detection expert system (nides)," Technical Report, Tech. Rep., 1993. 32

[60] M. M. Sebring, "Expert systems in intrusion detection: A case study," in *Proc. 11th National Computer Security Conference, Baltimore, Maryland, Oct. 1988*, 1988, pp. 74–81. 32

[61] R. Maxion, K. Tan *et al.*, "Anomaly detection in embedded systems," *Computers, IEEE Transactions on*, vol. 51, no. 2, pp. 108–120, 2002. 32

[62] D. E. Denning, "An intrusion-detection model," *Software Engineering, IEEE Transactions on*, no. 2, pp. 222–232, 1987. 33

[63] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 408–415. 36, 88, 89

[64] T. M. Nam, P. H. Phong, T. D. Khoa, T. T. Huong, P. N. Nam, N. H. Thanh, L. X. Thang, P. A. Tuan, V. D. Loi *et al.*, "Self-organizing map-based approaches in ddos flooding detection using sdn," in *2018 International Conference on Information Networking (ICOIN)*. IEEE, 2018, pp. 249–254. 36

[65] P. Winter, E. Hermann, and M. Zeilinger, "Inductive intrusion detection in flow-based network data using one-class support vector machines," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5. 36

[66] A. AlEroud and I. Alsmadi, "Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach," *Journal of Network and Computer Applications*, vol. 80, pp. 152–164, 2017. 37, 113

[67] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26. 37

[68] I. Mihai-Gabriel and P. Victor-Valeriu, "Achieving ddos resiliency in a software defined network by intelligent risk assessment based on neural networks and danger theory," in *Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 319–324. 37

[69] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed ddos detection mechanism in software-defined networking," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 310–317. 37, 42

[70] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "Jess: Joint entropy-based ddos defense scheme in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2358–2372, 2018. 37, 42

[71] P. Van Trung, T. T. Huong, D. Van Tuyen, D. M. Duc, N. H. Thanh, and A. Marshall, "A multi-criteria-based ddos-attack prevention solution using software defined networking," in *Advanced Technologies for Communications (ATC), 2015 International Conference on*. IEEE, 2015, pp. 308–313. 37

[72] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014. 37

[73] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *Network and Distributed System Security (NDSS)*, 2015. 37

[74] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017. 38

[75] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang, "An intelligent network attack detection method based on rnn," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 483–489. 38

[76] B. Zhang, Y. Yu, and J. Li, "Network intrusion detection based on stacked sparse autoencoder and binary tree ensemble method," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6. 38

[77] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based ddos detection system in software-defined networking (sdn)," *arXiv preprint arXiv:1611.07400*, 2016. 38, 113

[78] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018. 38, 113

[79] Mininet, http://www.mininet.org. 43

[80] Scapy, http://scapy.net/. Accessed 12 Feb 2018. 43

[81] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, p. 533, 1986. 50, 56

[82] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 51, 52, 67

[83] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in neural information processing systems*, 2014, pp. 568–576. 51

[84] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention.* Springer, 2015, pp. 234–241. 51

[85] KDDCup99, http://kdd.ics.uci.edu/databases/kddcup99/. Accessed 04 Jul 2018, 1999. 57

[86] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009. 57, 60, 68, 86, 105, 107

[87] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, “Evaluating intrusion detection systems: The 1998 darpa off-line intrusion

detection evaluation," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2. IEEE, 2000, pp. 12–26. 57

[88] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012. 58

[89] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of fourth international conference on information systems security and privacy, ICISSP*, 2018. 58, 63, 106, 107

[90] "Netflowmeter," http://netflowmeter.ca/netflowmeter.html. Accessed 19 Feb 2019. 63

[91] D. Raumer, L. Schwaighofer, and G. Carle, "Monsamp: A distributed sdn application for qos monitoring," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 961–968. 72

[92] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Computer Networks*, vol. 109, pp. 172–182, 2016. 72

[93] F. Chollet *et al.*, "Keras," https://keras.io, 2015. 75, 99

[94] Cbench, https://github.com/mininet/oflops/tree/master/cbench. Accessed 04 Jul 2018, n.d. 77

[95] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990. 97

[96] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001. 97

[97] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 97

[98] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014. 97

[99] 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 E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 99

[100] T. Dozat, "Incorporating nesterov momentum into adam.(2016)," *Dostupné z: http://cs229. stanford. edu/proj2015/054_report. pdf*, 2016. 100

[101] T. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM) (WINCOM'16)*, Fez, Morocco, Oct. 2016. 107

[102] R. C. Aygun and A. G. Yavuz, “Network anomaly detection with stochastically improved autoencoder based models,” in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud).* IEEE, 2017, pp. 193–198. 113, 123, 125

[103] Y. Kawachi, Y. Koizumi, and N. Harada, “Complementary set variational autoencoder for supervised anomaly detection,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2018, pp. 2366–2370. 113, 123, 125

[104] T. Luo and S. G. Nagarajany, “Distributed anomaly detection using autoencoder neural networks in wsn for iot,” in *2018 IEEE International Conference on Communications (ICC).* IEEE, 2018, pp. 1–6. 113, 123, 125

[105] A.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong, “A flow-based method for abnormal network traffic detection,” in *Network operations and management symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1. IEEE, 2004, pp. 599–612. 117

[106] P. Casas, J. Mazel, and P. Owezarski, “Unsupervised network intrusion detection systems: Detecting the unknown without knowledge,” *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012. 117

[107] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, “Support vector method for novelty detection,” in *Advances in neural information processing systems*, 2000, pp. 582–588. 119

[108] A. Pektaş and T. Acarman, "A deep learning method to detect network intrusion through flow-based features," *International Journal of Network Management*, p. e2050. 127, 128, 134

[109] C. Gonzalez, S. M. Charfadine, O. Flauzac, and F. Nolot, "Sdn-based security framework for the iot in distributed grid," in *Computer and Energy Science (SpliTech), International Multidisciplinary Conference on.* IEEE, 2016, pp. 1–5. 135

[110] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure iot architecture," *Internet of Things*, vol. 3, pp. 82–89, 2018. 135

[111] J. Li, Z. Zhao, R. Li, H. Zhang, and T. Zhang, "Ai-based two-stage intrusion detection for software defined iot networks," *IEEE Internet of Things Journal*, 2018. 135