electronics

MDPI

Article

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

Tuan Anh Tang [1,*], Lotfi Mhamdi [2], Des McLernon [2], Syed Ali Raza Zaidi [2], Mounir Ghogho [3] and Fadi El Moussa [4]

[1] Faculty of Electronics and Telecommunication Engineering, Danang University of Science and Technology, Da Nang 550000, Vietnam
[2] School of Electronic and Electrical Engineering, the University of Leeds, Leeds LS2 9JT, UK; l.mhamdi@leeds.ac.uk (L.M.); d.c.mclernon@leeds.ac.uk (D.M.); s.a.zaidi@leeds.ac.uk (S.A.R.Z.)
[3] College of Engineering & Architecture, International University of Rabat, Rabat 11103, Morocco; m.ghogho@ieee.org
[4] BT Security Futures Practice, Adastral Park, Ipswich IP5 3RE, UK; fadiali.el-moussa@bt.com
* Correspondence: tanganhtuan@dut.udn.vn

Received: 2 August 2020; Accepted: 10 September 2020; Published: 19 September 2020

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 a controllable, dynamic, and cost-effective network. The emergence of SDN provides a unique opportunity to achieve network security in a more efficient and flexible manner. However, SDN also has original structural vulnerabilities, which are the centralized controller, the control-data interface and the control-application interface. These vulnerabilities can be exploited by intruders to conduct several types of attacks. In this paper, we propose a deep learning (DL) approach for a network intrusion detection system (DeepIDS) in the SDN architecture. Our models are trained and tested with the NSL-KDD dataset and achieved an accuracy of 80.7% and 90% for a Fully Connected Deep Neural Network (DNN) and a Gated Recurrent Neural Network (GRU-RNN), respectively. 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.

Keywords: deep learning; intrusion detection; network security; software defined networking; SDN

1. Introduction

Software Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This 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 [1]. The detail of the SDN architecture is described in Figure 1. The OpenFlow protocol [2] is one of the most popular protocols and is a foundation element in building SDN solutions.

SDNs are currently being deployed in many network environments, from home and enterprise networks to datacenters (e.g., Google WAN B4 [3], Huawei carrier network, Amazon, Cisco, Facebook). The capabilities of SDN (e.g., logical centralized controller, and global network overview) help us to solve several security issues in a traditional network and bring us the ability to control network traffic at a fine-grained level. However, the SDN architecture itself also introduces some new attack threats

and security issues. Kreutz et al. [4] introduced seven threat vectors in SDN. Three of them are specific to SDN and related to the controller: the control-data interface and the control-application interface. The controller itself is vulnerable, because it creates a single point of attack and failure. The encryption of the communication channel, using Transport Layer Security protocol (TLS), is also ineffective and optional. Several attacks can be conducted in the SDN architecture. For instance, Distributed Denial of Service (DDoS) attacks can overwhelm the controller and the communication channel with artificial service calls. A Man-in-the-Middle attack can break the link between the controller and the switches and claim control of the network.

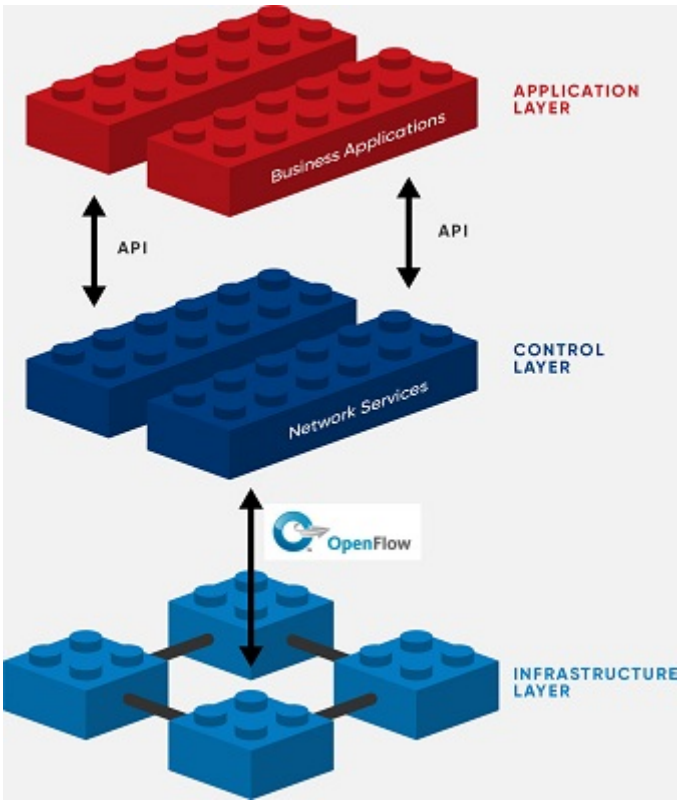


Figure 1. Software Defined Networking (SDN) Architecture [1].

An intrusion detection system (IDS) is one of the most crucial parts of network architecture. Based on the difference that data are processed, IDS can be categorised into misuse detection and anomaly detection. Misuse 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 really well this way with low false alarm rate. However, this method cannot detect zero-day attacks which are new and unknown. On the other hand, anomaly detection creates a normal behaviour baseline model and then tries to detect any behaviour that deviates from this baseline model. Thus, this method can detect zero-day attacks. Our work in this paper focuses on anomaly detection.

Because of the wide variety of types of SDN deployment, SDN security is a serious concern and has recently been extensively researched—see [5,6] for more detail. Several machine learning (ML) approaches have been proposed to secure the SDNs. Despite the high accuracy and performance obtained in several fields, ML algorithms used in intrusion detection tend to have some limitations as mentioned in [7]. 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 network intrusion detection systems (NIDS) in the SDN environment. However, these techniques usually lead to a high rate of false positives. We extended this research trend to a deep learning (DL) approach for the SDN context.

Recently, DL has emerged and achieved significant success in the field of speech recognition and face detection. DL is capable of automatically finding correlation in data and so it is a promising method for the next generation of intrusion detection. DL can be used to efficiently detect zero-day attacks and so we can acquire a high detection rate. In our previous work [8], we evaluated the potential of the DL approach for flow-based intrusion detection and the results were promising. We achieved an accuracy of 75.75% with just six basic features.

In this paper, which is a part of my PhD thesis [9], we present the structural design and implementation of DeepIDS (a flow-based anomaly detection system using DL) in the SDN architecture. In the previous work, we focused on the structure design of DeepIDS and features selection. The attack detection mechanism is the main research contribution of this paper. A comparison with several ML/DL algorithms proves the enhancements of our approach. Our main contributions are summarized in the following points:

- We trained and evaluated our models with different sets of features of the NSL-KDD dataset [10]. Through experiments, our models yield a detection accuracy of 80.7% and 90% using just six basic flow features among the forty one features of this dataset. The results show the ability of our models in dealing with low level network features.
- This method is scalable and the structure of the DL algorithm 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.

Accordingly, this paper is structured, as follows: in Section 2, we discuss relevant work in the field of intrusion detection in SDN environments, while Section 3 presents the implementation details of the proposed DeepIDS, with respect to data gathering, anomaly detection, and mitigation. Section 4 presents detection performance of our DL approach. Section 5 describes our performance evaluation experiments and results. Finally, Section 6 concludes the paper and discusses future work.

2. Related Work

ML has been used by several researchers for intrusion detection in the traditional network. Ibrahim et al. [11] used a Self Organizing Map (SOM) in their intrusion detection research and achieved an accuracy of 75.49%. In [12], Mukherjee et al. investigated the combination of Correlation-based Feature Selection, Information Gain, and Gain Ratio for feature reduction. Subsequently, the reduced dataset was classified by a Naive Bayes algorithm that yields an accuracy of 97.78%.

DL has also been applied for intrusion detection, but only in traditional networks. We will briefly summarize these results to distinguish from our work with SDNs. Javaid et al. [13] used a self-taught deep 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%. Yin et al. [14] proposed a deep learning approach while using recurrent neural networks for intrusion detection. They got an accuracy of 83.28% with their experiment on the NSL-KDD dataset. Shone et al. proposed a nonsymmetric deep autoencoder for unsupervised feature learning in [15]. They achieved quite promising results on both the KDD Cup 99 and NSL-KDD datasets. However, these approaches cannot be applied directly in the context of SDN. The SDN architecture is not considered in these approaches and some features used in these methods are not flow-based features.

Several intrusion detection algorithms and approaches have been proposed to secure OpenFlow-based SDN networks. For anomaly-based detection approaches, SOM and Support Vector Machine (SVM) are frequently used because of their high detection accuracy. A lightweight method for DDoS attack detection based on traffic flow features is presented in [16] with an extraction of

six-tuple features: 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. SOM—a neural network-based technique—is used as the classification method. SVM was used in [17,18] to detect DDoS attacks in SDNs quite efficiently. One-class SVM was trained with a malicious dataset for a low false alarm rate in [19]. AlEroud et al. [20] combined k-Nearest Neighbor and graph theory to classify DDoS attacks from benign flow in SDNs. Gabriel et al. [21] combined neural networks and danger theory for DDoS attack resiliency. S. M. Mousavi proposed in [22] 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 amount of hosts. The entropy will drop dramatically when any attack happens. Trung et al. [23] 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 server) under normal and attack states.

In order to improve the scalability of the native OpenFlow protocol, a combination of OpenFlow and sFlow was proposed in [24] for an effective and scalable anomaly detection and mitigation mechanism in an SDN environment. By using the programmability of the SDN architecture, the authors of [25] show that a programmable home network router can provide the ideal platform and location in the network for detecting security problems in a SOHO (Small Office/Home Office) network. The authors implemented four prominent traffic anomaly detection algorithms (threshold random walk with credit based rate limiting, rate limiting, maximum entropy detector, and NETAD) in an SDN context. The experiments indicate that these algorithms are significantly more accurate in identifying malicious activities in the SOHO network than the ISP (Internet Service Provider) and also the anomaly detector can work at line rates without introducing any new performance overhead for the home network traffic. SPHINX was introduced in [26] 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. SPHINX dynamically learns new network behavior and raises alerts when it detects suspicious changes to existing network control plane behavior. The authors evaluate the accuracy, latency, and throughput of SPHINX to show that SPHINX is capable of detecting attacks in real time with a low performance overhead. A summarize of some popular related work is presented in Table 1. Most of the works use a full dataset of several features or just focus on one type of attacks, like DDoS attacks.

Table 1. The related work summarization and comparison.

Article	Algorithm	Types of Attacks	Accuracy (%)
Ibrahim et al. [11]	SOM	All	75.49
Mukherjee et al. [12]	Feature Selection & Naive Bayes	All	97.78
Javaid et al. [13]	Self-taught Deep Learning	All	92.98
Yin et al. [14]	Recurrent Neural Network	All	83.28
Braga et al. [16]	SOM	DDoS	98
Kokila et al. [18]	SVM	DDoS	95
Our Work	Deep Learning	All	90

Nowadays, current research trends focus on detecting DDoS attacks that are some of the most dangerous attacks in SDNs. In our research, we focus not only on detecting DDoS attacks, but also on detecting all types of network attacks and discovering zero-day attacks in SDNs. Our DL approach only uses a small set of basic flow features as compared to others for SDN architecture.

3. DeepIDS System Architecture

We propose an SDN-based NIDS architecture (DeepIDS), as depicted in Figure 2. Our DeepIDS consists of three main modules: the Collector, the Anomaly Detector and the Counter Measure Deployment. Our 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.

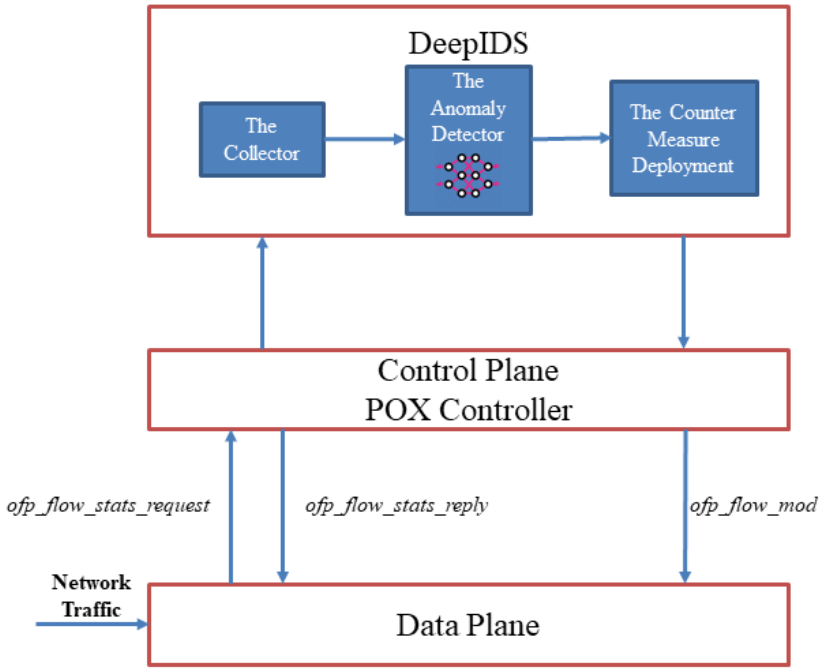


Figure 2. The DeepIDS Architecture.

3.1. 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 that are 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 optimal selection of 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 extremely large 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 overhead for the controller in terms of resource efficiency. While several researchers [27,28] focus on traffic monitoring and sampling, this is not the main focus of our work in this paper.

Currently, most of the 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 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 will be prepared as an input for the DL model in the Anomaly Detector module.

3.2. The Anomaly Detector

Data produced by the Collector are subsequently fed to the Anomaly Detector. In this module, we choose the DL algorithm as a core of the module. In general, the DNN is constructed with an input layer, several hidden layers and an output layer, as described in Figure 3. The number of hidden layers and the dimension of each hidden layer can be varied to find the best structure for attack detection. We construct the DNN model to minimize the overhead to the controller.

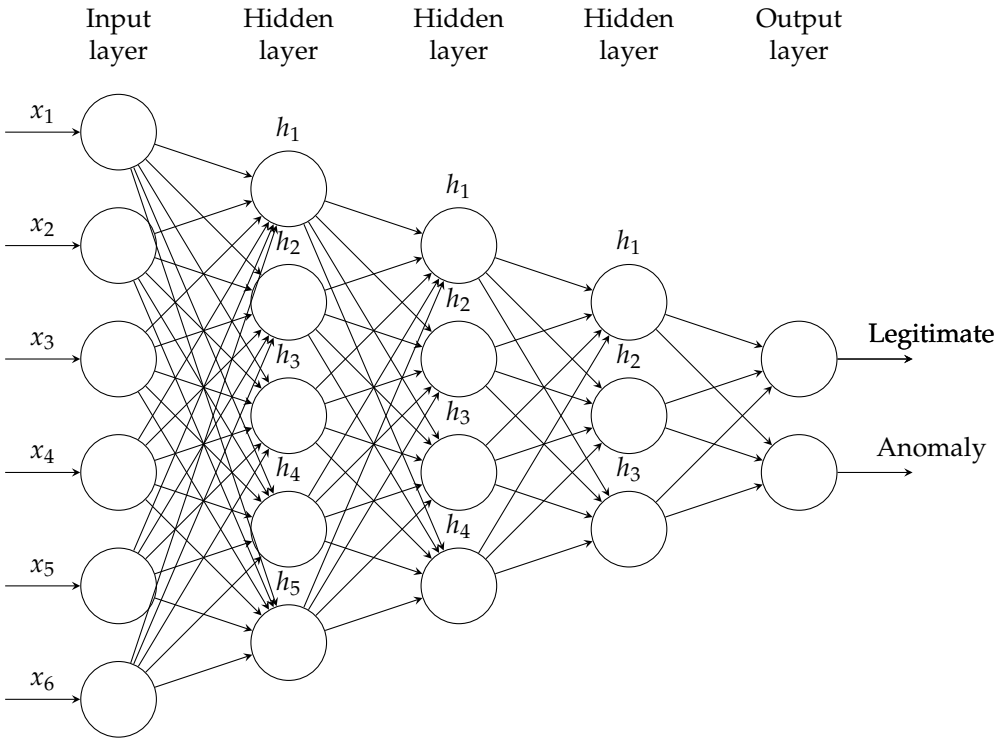


Figure 3. The DNN Model.

In this paper, we implemented a Fully Connected Deep Neural Network (DNN) and a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN). The details of a Gate Recurrent Unit can be found in [29]. In our experiments, we use Keras [30] to implement our DL models. The details of our models can be seen in Table 2. The DNN and GRU-RNN are implemented with a same structure.

This model can be trained offline and be deployed for online detection. In addition, we can update our model anytime without interfering with the 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 of the network metrics of the identified attack for further forensics and send all related information to the Counter Measure Deployment module.

Table 2. Deep Neural Network (DNN) Model Structure.

Variable	Parameters
Input Layer	6
Hidden Layer	5, 4, 3
Output Layer	2
Activation Function	Tanh
Loss Function	Mean Squared Error
Learning Rate	0.001
Batch Size	10
Epoch	100

3.3. The Counter Measure Deployment

The Counter Measure Deployment module aims to neutralize identified attacks. All of the information about detected attacks (e.g., source IP address, destination IP address, source port, and destination port) is collected from the Anomaly Detector module. Subsequently, 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 in order to drop all of 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 attack 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. Algorithm 1 summarizes the complete process for counter measure deployment.

Algorithm 1 The Counter Measure Deployment

```
for all Anomaly Packets do

    msg = of.ofp_packet_out()                          ▷ Create packet_out message

    msg.buffer_id = event.ofp.buffer_id

    msg.in_port = packet_in.in_port

    msg.match = of.ofp_match.from_packet(packet)

    action = of.ofp_action_output(port = of.OFPP_DROP)   ▷ Add an action to drop the packet

    msg.actions.append(action)

    self.connection.send(msg)                          ▷ Send message to switches

end for
```

4. Detection Evaluation

4.1. Dataset Description

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 (i.e., the KDD Cup 99 dataset [31], the NSL-KDD dataset, DAPRA [32], and the ISCX 2012 Intrusion Detection [33]). Among these datasets, the KDD Cup 99 dataset and NSL-KDD dataset have been commonly used in the literature to assess the performance of NIDSes. The KDD Cup 99 dataset is one of the most popular datasets and it 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. [10] to resolve this redundancy issue. SDNs are a new environment, and so the dataset for it is still very rare and unpublished. The NSL-KDD dataset is still considered as a state-of-the-art dataset by many researchers to evaluate their approaches, and so we also choose it for our experiment. Each traffic sample in this dataset has 41 features that are categorized into three types of features: basic features,

content-based features, and traffic-based features. Attacks in the dataset are divided into four categories according to their characteristics. The details of each category are described in Table 3. 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. This dataset contains 125,973 records for training and 22,544 records for testing.

Table 3. Attacks in The NSL-KDD Dataset.

Category	Training Set	Testing Set
DoS	back, land, neptune, pod, smurf, teardrop	back, land, neptune, pod, smurf, teardrop, **mailbomb, 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, snmpgetattack, httptunnel, sendmail, named**
U2R	buffer-overflow, loadmodule, perl, rootkit	buffer-overflow, loadmodule, perl, rootkit, **sqlattack, xterm, ps**
Probe	ipsweep, nmap, portsweep, satan	ipsweep, nmap, portsweep, satan, **mscan, saint**

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 accuracy. 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. Table 4 outlines details of each feature set.

Table 4. Feature Set Description.

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

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 a numerical value. The NSL-KDD dataset contains both the 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:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \quad (1)$$

where x' is the normalized value and x is the original value.

4.2. Evaluation Metrics

The performance and effectiveness of the NIDS are evaluated by several experiments. For the evaluation purpose, Accuracy (ACC), Precision (P), Recall (R), and F1-measure (F1) metrics are applied. These metrics are calculated by using four different measures—True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN), defined, as follows:

- 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.

These metrics are evaluated as formulas (2)–(5) below:

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

$$AC = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%. \quad (2)$$

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

$$P = \frac{TP}{TP + FP} \times 100\%. \quad (3)$$

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

$$R = \frac{TP}{TP + FN} \times 100\%. \quad (4)$$

- 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,

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} \times 100\%. \quad (5)$$

4.3. Experimental Results

Our DNN model is implemented for binary classification (normal and anomaly class). The performance of the model for each feature set is shown in Table 5. 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 when compared to others. The Basic Feature Set's ACC is just slightly lower than that of the Mixed Feature Set.

Table 5. Accuracy Evaluation.

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

Table 6 gives you a more detailed view about the performance of DNN on each feature set. As seen in Table 6, the R and P values of the Mixed Feature Set are higher than the other sets.

Table 6. Accuracy Metric Evaluation of the Three Feature Sets.

Feature Set	Precision (%)	Recall (%)	F1-Measure (%)
Basic Feature Set	84	80	80
Traffic Feature Set	77	73	72
Mixed Feature Set	85	81	81

In the following, Receiver Operating Characteristic (ROC) curves are presented in Figure 4, and False Positive Rate (FPR) is plotted against True Positive Rate (TPR) for each feature set. The area under the ROC curve (AUC) is a standard measure for classifier comparison. The higher the AUC, then the better is the system. Figure 4 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 to reduce the FPR, which is an important factor of NIDS.

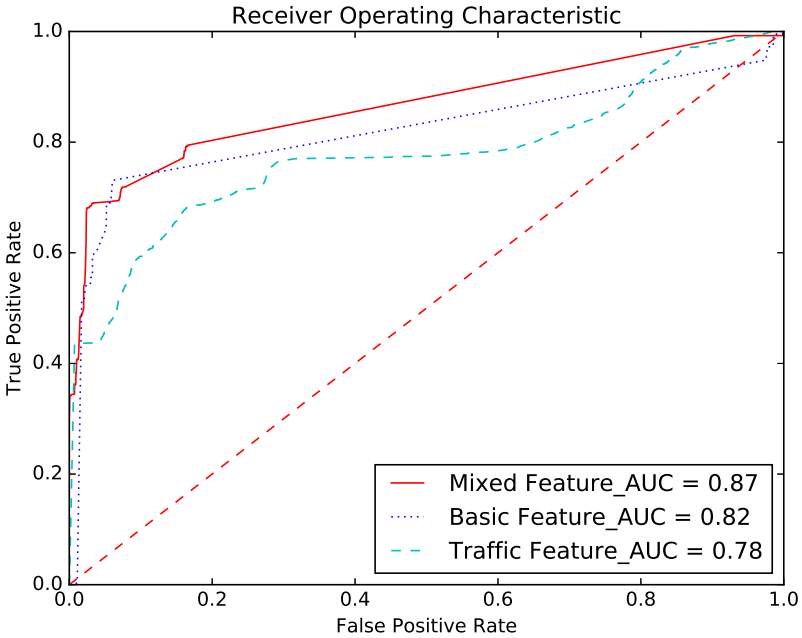
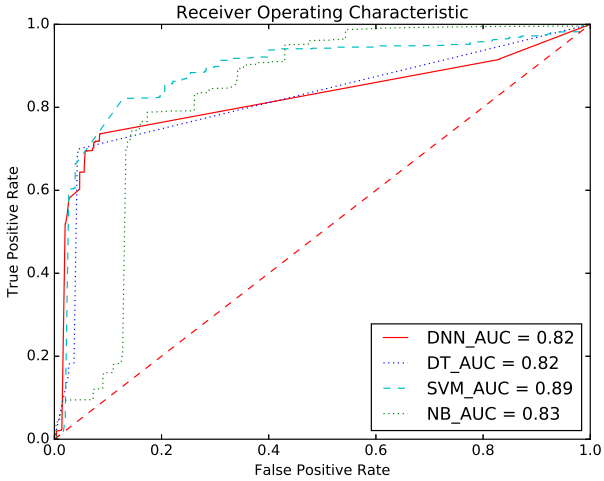
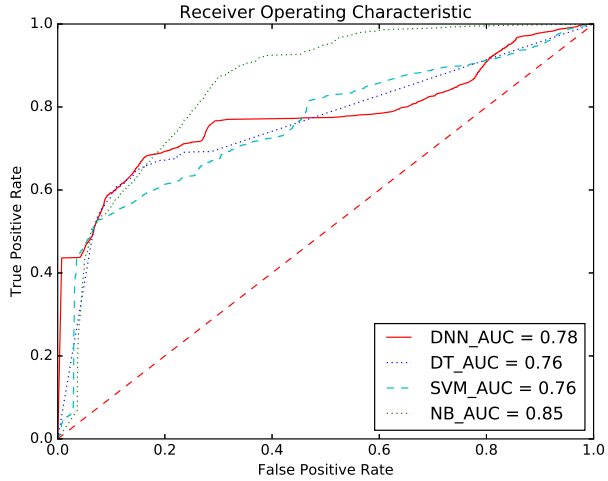


Figure 4. Receiver Operating Characteristic (ROC) Curve Comparison for Different Feature Sets.

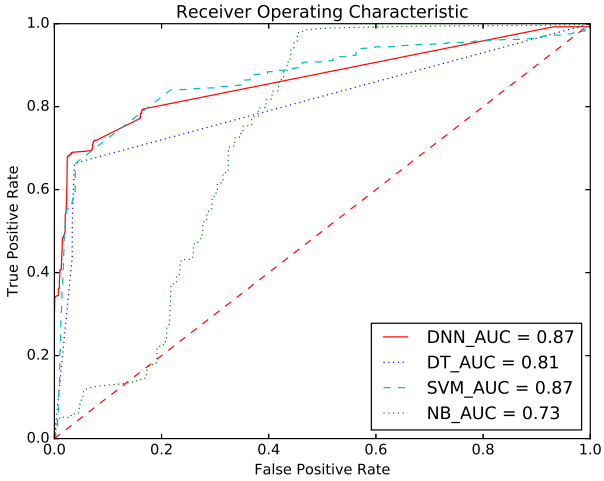
In the next experiment, we evaluate the sensitivity of each feature set with some existing algorithms: Naive Bayes (NB), Decision Tree (DT), and SVM. The NB and DT are quite simple and cost-effective. The SVM was proposed by Phan et al. [17] for DDoS detection. They used their own dataset, and so we have to regenerate the results on the considered NSL-KDD dataset. The ROCs of each test case are shown in Figure 5. Each feature set has a different effect on the performance of each algorithm. Our DNN works quite well with the Basic and Mixed Feature Set, with quite low FPR and high AUC. SVM also achieves quite high AUC, but it also has some drawbacks like high FPR and low computational efficiency. We will discuss the efficiency of these algorithms in due course. NB performs very poorly with the Basic and Mixed Feature Set, but it achieves 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 the NIDS. The Mixed Feature Set gives us the best combination of AUC and the FPR.



(**a**) Basic Feature Set



(**b**) Traffic Feature Set



(**c**) Mixed Feature Set

Figure 5. ROC Curve Comparison for Different Algorithms.

From the above evaluation, we can see the potential of DNN for intrusion detection. The Mixed Feature Set will be chosen for further evaluation. In the next experiment, we evaluate performance of the DNN and GRU-RNN algorithms. We compare the P, R, and F1 (see (3)–(5)) of these algorithms for more understanding. As we can see in Table 7, the DNN model can detect almost all the attacks mentioned above with a high R value, and our results are promising, with a high P of 85%. Table 7 also shows that the more complex GRU-RNN can improve the P, R, and F1 significantly. These results show that our proposed approach is a strong candidate for anomaly detection.

Table 7. Accuracy Metric Evaluation.

Algorithm	Precision (%)	Recall (%)	F1-measure (%)
DNN	85	81	81
GRU-RNN	89	89	89

Table 8 gives us an overview of the ACC and our proposed DNN and GRU-RNN give the best results among all the algorithms. The GRU-RNN yields the highest ACC of 89%.

Table 8. Accuracy Comparison for Different Algorithms.

Algorithm	Accuracy (%)
NB	45
DT	74
SVM	70.9
DNN	80.7
GRU-RNN	89

From all of the above, we have demonstrated that our proposed DL 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 in the SDN environment.

4.4. 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 9 provides relevant parameters for these algorithms. The NB and DT algorithms have really low training and testing times in ms as compared to the SVM and our DNN. However, they also yield low detection accuracy. 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 architecture. The total processing time takes about 2.5 h. Our DNN works much faster than the SVM and it also gives a better detection accuracy. The GRU-RNN has higher training and testing times when compared to the DNN. However, the GRU-RNN still works much faster than the other ML algorithms and also outperforms them in terms of ACC.

Table 9. Running Time Evaluation.

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
GRU-RNN	2500 s	4 s

As mentioned in [16], the SOM algorithm was used for flow-based intrusion detection in the SDN context. The system was trained and tested by a six-tuple feature dataset that consists of 8608 samples and 62,888 samples in the training set and testing set respectively. The system takes about 7.16 h 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 [16].

5. Network Performance Evaluation

In this section, we provide detailed network performance analysis of our DeepIDS in a POX [34] controller. We also compare the network performance of our DeepIDS with the SVM algorithm. First, we describe the evaluation testbed and then present the throughput and latency results. We also evaluate the resource utilization and briefly discuss our observations in this section.

5.1. Experimental Setup

The DeepIDS and SVM are 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 POX, we used a Virtual Machine having Intel Core i5-4460 3.2 GHz processor with three cores available and 8 GB of RAM memory.

We ran Cbench [35] which is the standard tool used for evaluating SDN OpenFlow controller. We measured the performance of the controller in terms of throughput and latency. Cbench tests can run in either throughput or in 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 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 before sending another packet. The latency results represent the average number of milliseconds that a flow consumes to be installed in each switch.

We tested the controller's throughput and latency with a different number of virtual OpenFlow switches emulated by Cbench. We vary the size of network from 8, 16, 32, 64, 128, and 256 switches, with each switch having 1000 unique source MACs. Each Cbench run consists of 10 test loops with a duration of 10 s. We ran the controller with a typical layer 2 learning switch application. The obtained results are the average values from the 10 tests. We evaluate the performance of the controller in the following scenarios.

- The POX controller runs without DeepIDS. This scenario serves as the baseline.
- The POX controller runs with DeepIDS or the SVM being enabled.

5.2. Analysis of Results

5.2.1. Throughput Evaluation

For throughput mode tests, we evaluate how many packets a controller can process per second. This metric indicates the performance of the controller under heavy traffic conditions. Figure 6 depicts the average response rate of the network in three testing scenarios. 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 when compared to other SDN controllers. However, the running stand-alone POX controller still performs best and it is taken as a baseline for performance evaluation in our experiment. For overall evaluation, the bigger the size of the network the higher the performance overhead. The throughput slightly decreases with the increasing network size according to the results in Figure 6. The network performance drops when the number of forwarding devices increases to 64, showing that the controller throughput degrades by roughly 3% from 32 to 64 switches. The base line results for the smallest network scale and the largest network scale are 4852 responses/s

and 4607 responses/s, respectively. The DNN, GRU-RNN, and SVM overheads are about 3%, 3.4%, and 4%, respectively, in the case of eight switches. We can see that in the case of high *"packet-in"* message rate with 256 switches, overall throughputs decreases significantly. The reason for this behavior is that the NIDS sends "*ofp_flow_stats_request*" messages and processes "*ofp_flow_stats_reply*" messages during processing "*packet-in*" messages. The NIDS has to process a large amount of "*ofp_flow_stats_reply*" messages when the network size increases, resulting in the overhead. Figure 6 shows that the SVM algorithm has affected the controller and resulted in low network throughput in all test cases. The SVM algorithm's throughput decreases by about 6.4%. The GRU-RNN has lower throughput than the DNN, but the difference is not significant. As we can see, DeepIDS performs better than the SVM algorithm in terms of throughput. The DeepIDS's throughput degradation (which is around 4% on average) is quite low and it has almost no effect on the POX controller's performance.

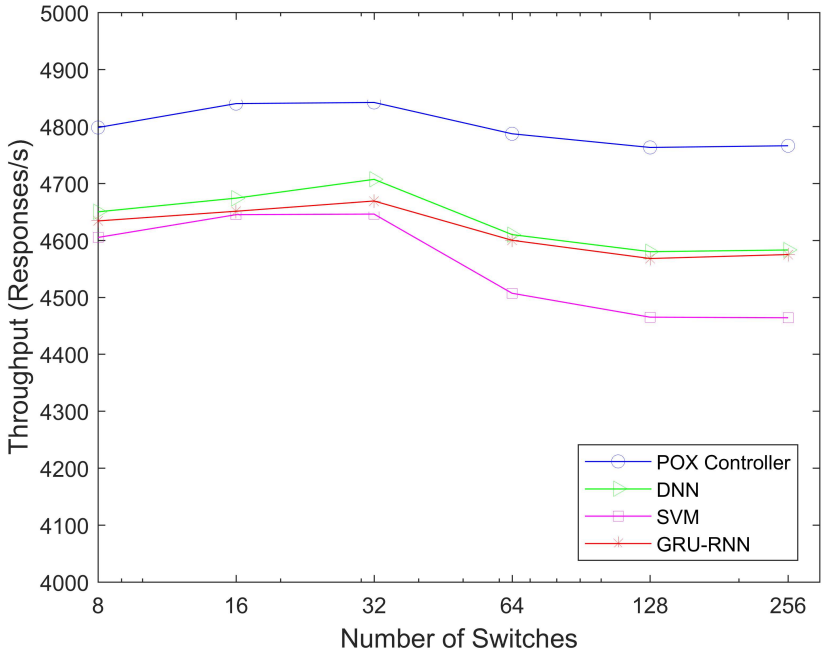


Figure 6. Throughput Evaluation (log scale on x-axis).

5.2.2. Latency Evaluation

For latency mode tests, the controller is evaluated for the length of time that 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 7, the latency increases with the increasing network size. This is an expected behavior: increasing the number of devices will increase the load on the controller, causing a latency increase. In general, we can see in Figure 7 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 so it results in a higher latency under high traffic rates. The DNN, GRU-DNN, and SVM overheads are about 2.6%, 4%, and 5.8%, respectively, in the case of 256 switches. The DeepIDS's latency degradation is quite low and it can be improved in the future.

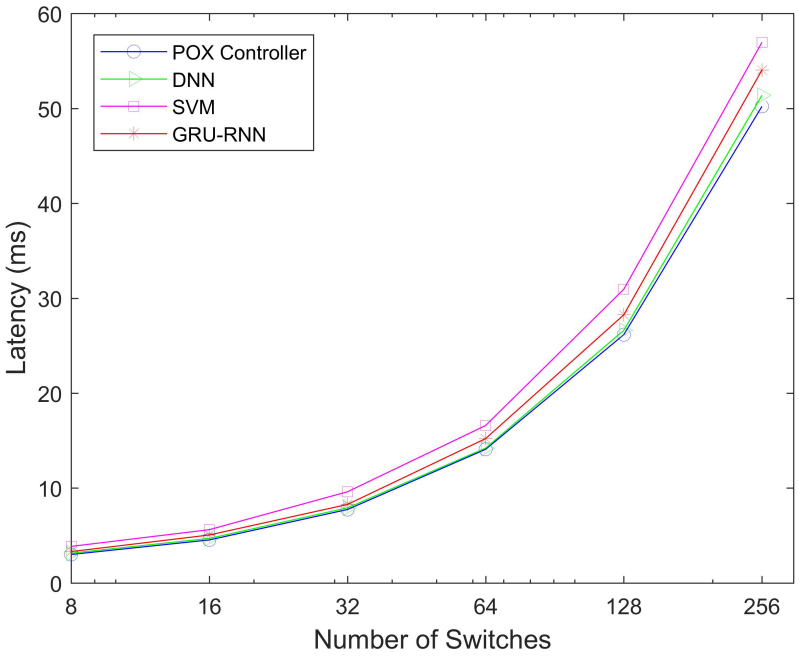


Figure 7. Latency Evaluation (log scale on x-axis).

5.2.3. Resource Utilization

We also evaluate the resource utilization of the DeepIDS in terms of CPU and memory usage. These information is monitored by System Monitor application of Linux. The POX controller itself resource utilization is a benchmark for our comparison. DeepIDS does not use a lot of computer resources, as we can see in Table 10. The GRU-RNN uses more computer resources than the DNN as expected. DL normally requires a significant computational cost. However, our model is quite simple and the input is small, and so it uses computational resources optimally.

Table 10. Resource Utilization Evaluation.

Algorithm	CPU Usage (%)	Memory Usage (%)
POX	15	7.5
DNN	17	8.2
GRU-RNN	20	10.5

In summary, we can see that the DeepIDS has the good network performance and low overhead on the system. In this scenario, we just considered a network with one SDN controller and thousands of connected devices that is suitable for SO/HO networks. For a larger scale network, a single SDN controller can be a bottleneck with a huge number of OpenFlow packet message. Therefore, a scenario with multiple SDN controllers updated consistently should also considered in the future work.

6. Conclusions and Future Work

In this paper, 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 shown 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 approach does not significantly affect the network performance. Therefore, our approach is quite promising and it 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 accuracy of the

NIDS. As shown above, our DeepIDS gives quite low ACC when compared to 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. Our DeepIDS also needs to be implemented in other OpenFlow controllers and a real testbed for further evaluation. Semi-supervised and unsupervised learning algorithms are also promising directions for our future research.

Author Contributions: Investigation, T.A.T.; Supervision, L.M., D.M., S.A.R.Z., M.G. and F.E.M.; Writing—original draft, T.A.T.; Writing—review & editing, L.M., D.M. and S.A.R.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. ONF. Software-Defined Networking (SDN) Definition. Available online: https://www.opennetworking.org/sdn-definition/ (accessed on 18 July 2020).
2. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
3. Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 3–14. [CrossRef]
4. Kreutz, D.; Ramos, F.; Verissimo, P. Towards secure and dependable software-defined networks. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, 16 August 2013; ACM: New York, NY, USA, 2013; pp. 55–60.
5. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
6. Scott-Hayward, S.; Natarajan, S.; Sezer, S. A survey of security in software defined networks. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 623–654. [CrossRef]
7. Sommer, R.; Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010; pp. 305–316.
8. Tang, T.; Mhamdi, L.; McLernon, D.; Zaidi, S.A.R.; Ghogho, M. Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM) (WINCOM'16), Fez, Morocco, 26–29 October 2016.
9. Tang, T. Software Defined Networking: Network Intrusion Detection System. Ph.D. Thesis, The University of Leeds, Leeds, UK, 2019.
10. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. 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, Ottawa, ON, Canada, 8–10 July 2009.
11. Ibrahim, L.M.; Basheer, D.T.; Mahmod, M.S. A comparison study for intrusion database (Kdd99, Nsl-Kdd) based on self organization map (SOM) artificial neural network. *J. Eng. Sci. Technol.* **2013**, *8*, 107–119.
12. Mukherjee, S.; Sharma, N. Intrusion detection using naive Bayes classifier with feature reduction. *Procedia Technol.* **2012**, *4*, 119–128. [CrossRef]
13. Javaid, A.; Niyaz, Q.; Sun, W.; Alam, M. 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), Pittsburgh, PA, USA, 13–14 March 2016; pp. 21–26.
14. Yin, C.; Zhu, Y.; Fei, J.; He, X. A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access* **2017**, *5*, 21954–21961. [CrossRef]
15. Shone, N.; Ngoc, T.N.; Phai, V.D.; Shi, Q. A deep learning approach to network intrusion detection. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 41–50. [CrossRef]

16. Braga, R.; Mota, E.; Passito, A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In Proceedings of the 2010 IEEE 35th Local Computer Networks (LCN) Conference, Denver, CO, USA, 10–14 October 2010; pp. 408–415.
17. Phan, T.V.; Van Toan, T.; Van Tuyen, D.; Huong, T.T.; Thanh, N.H. OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks. In Proceedings of the 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), Ha Long, Vietnam, 27–29 July 2016; pp. 13–18.
18. Kokila, R.; Selvi, S.T.; Govindarajan, K. DDoS detection and analysis in SDN-based environment using support vector machine classifier. In Proceedings of the 2014 IEEE Sixth International Conference on Advanced Computing (ICoAC), Chennai, India, 17–19 December 2014; pp. 205–210.
19. Winter, P.; Hermann, E.; Zeilinger, M. Inductive intrusion detection in flow-based network data using one-class support vector machines. In Proceedings of the 2011 IEEE 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 7–10 February 2011; pp. 1–5.
20. AlEroud, A.; Alsmadi, I. Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach. *J. Netw. Comput. Appl.* **2017**, *80*, 152–164. [CrossRef]
21. Mihai-Gabriel, I.; Victor-Valeriu, P. Achieving DDoS resiliency in a software defined network by intelligent risk assessment based on neural networks and danger theory. In Proceedings of the 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 19–21 November 2014; pp. 319–324.
22. Mousavi, S.M.; St-Hilaire, M. Early detection of DDoS attacks against SDN controllers. In Proceedings of the 2015 IEEE International Conference on Computing, Networking and Communications (ICNC), Garden Grove, CA, USA, 16–19 February 2015; pp. 77–81.
23. Van Trung, P.; Huong, T.T.; Van Tuyen, D.; Duc, D.M.; Thanh, N.H.; Marshall, A. A multi-criteria-based DDoS-attack prevention solution using software defined networking. In Proceedings of the 2015 IEEE International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 14–16 October 2015; pp. 308–313.
24. Giotis, K.; Argyropoulos, C.; Androulidakis, G.; Kalogeras, D.; Maglaris, V. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Comput. Netw.* **2014**, *62*, 122–136. [CrossRef]
25. Mehdi, S.A.; Khalid, J.; Khayam, S.A. Revisiting traffic anomaly detection using software defined networking. In *International Workshop on Recent Advances in Intrusion Detection*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 161–180.
26. Dhawan, M.; Poddar, R.; Mahajan, K.; Mann, V. SPHINX: Detecting Security Attacks in Software-Defined Networks. In Proceedings of the Network and Distributed System Security (NDSS)'15, San Diego, CA, USA, 8–11 February 2015; Internet Society: Reston, VA, USA, 2015.
27. Raumer, D.; Schwaighofer, L.; Carle, G. Monsamp: A distributed sdn application for qos monitoring. In Proceedings of the 2014 IEEE Federated Conference on Computer Science and Information Systems (FedCSIS), Warsaw, Poland, 7–10 September 2014; pp. 961–968.
28. Ha, T.; Kim, S.; An, N.; Narantuya, J.; Jeong, C.; Kim, J.; Lim, H. Suspicious traffic sampling for intrusion detection in software-defined networks. *Comput. Netw.* **2016**, *109*, 172–182. [CrossRef]
29. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
30. Keras 2015. Available online: https://keras.io (accessed on 18 July 2020).
31. KDDCup99. 1999. Available online: http://kdd.ics.uci.edu/databases/kddcup99/ (accessed on 18 July 2020).
32. Lippmann, R.P.; Fried, D.J.; Graf, I.; Haines, J.W.; Kendall, K.R.; McClung, D.; Weber, D.; Webster, S.E.; Wyschogrod, D.; Cunningham, R.K.; et al. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In Proceedings of the IEEE 2000 DISCEX'00 DARPA Information Survivability Conference and Exposition, Hilton Head, SC, USA, 25–27 January 2000; Volume 2, pp. 12–26.
33. Shiravi, A.; Shiravi, H.; Tavallaee, M.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.* **2012**, *31*, 357–374. [CrossRef]

34. POX. 2009. Available online: https://github.com/noxrepo/pox (accessed on 18 July 2020).
35. Cbench. Available online: https://github.com/mininet/oflops/tree/master/cbench (accessed on 18 July 2020).