Article

Effects of Machine Learning Approach in Flow Based Anomaly Detection on Software Defined Networking

Samrat Kumar Dey [1], and Md. Mahbubur Rahman [2]

[1] Dhaka International University (DIU); samrat.cse@diu-bd.net
[2] Military Institute of Science and Technology (MIST); mahbubucse@yahoo.com
* Correspondence: sopnil.samrat@gmail.com; Tel.: (+88-01823-26-79937)

Abstract: Recent advancements in Software Defined Networking (SDN) makes it possible to overcome the management challenges of traditional network by logically centralizing control plane and decoupling it from forwarding plane. Through centralized controllers, SDN can prevent security breach, but it also brings in new threats and vulnerabilities. Central controller can be a single point of failure. Hence, flow-based anomaly detection system in OpenFlow Controller can secure SDN to a great extent. In this paper, we investigated two different approaches of flow-based intrusion detection system in OpenFlow Controller. The first of which is based on machine-learning algorithm where NSL-KDD dataset with feature selection ensures the accuracy of 82% with Random Forest classifier using Gain Ratio feature selection evaluator. In the later phase, the second approach is combined with Gated Recurrent Unit Long Short-Term Memory based intrusion detection model based on Deep Neural Network (DNN) where we applied an appropriate ANOVA F-Test and Recursive Feature Elimination feature selection method to improve the classifier performance and achieved an accuracy of 88%. Substantial experiments with comparative analysis clearly show that, deep learning would be a better choice for intrusion detection in OpenFlow Controller.

Keywords: software defined networking; random forest; gain ratio; gru-lstm; anova f-rfe; open flow controller; machine learning

1. Introduction

Today's more advanced and more dynamic applications can no longer be handled effectively by traditional network architecture. Traditional network architecture has been in use for past decades. But it is becoming less effective gradually for modern day applications. To deal with the inadequacy of traditional network architecture, a dynamic and scalable networking architecture namely Software Defined Networking architecture (SDN) [1] has emerged. SDN shifts from the fully distributed conventional networking model to a more centralized one. Separation of the data and control planes is a defining characteristic of SDN. Forwarding components namely switches, routers, etc. are elements of the data plane and the controller is element of the control plane. Decoupling of the network control and forwarding functions, and direct programmability of the network give network managers enough control over the network. Separation of the routing and forwarding activities of networking components (e.g. switches, routers, etc.) from the data plane, makes the administration and management of the network straightforward because the control plane now only has to handle logical network topology related information, traffic routing, and so forth while the data plane only needs to manage the network traffic using the configuration provided by the control plane. SDN has been gaining attention recently for both academia and researchers. But, the concept of SDN is not something new. In fact, from the mid-1990s it started to evolve. A concrete implementation of SDN has been possible thanks to OpenFlow protocol. From the beginning, OpenFlow has been the leading standardized interface between SDN controllers and switches or other data paths [2]. OpenFlow protocol standardizes traffic management and specifies method for controller communication with network devices.

In SDN, network policies are determined by the centralized controller where all the control operations take place. Real-time network information retrieval is straightforward using OpenFlow [3] protocol due to the flow-based forwarding model of SDN. The blessings of SDN have already been demonstrated by SDN backbone like Google B4 [4] and Huawei carrier network [5]. NOX [6], Ryu [7], Beacon [8], Open Daylight [9], and Floodlight [10] are some examples of SDN controller software that open source communities have already devised. However, with these major advances, SDN also brings in various security issues related to the application interface and control data [11]. Consequently, security has become an important point of concern for SDN, hence, deserve significant consideration from industry and academia. Intrusion detection guards a network from being affected by malicious pieces of software. There are, conventionally, two kinds of intrusion detection scheme namely, signature-based detection and anomaly-based detection. In signature-based detection, new data is compared with database of known and discovered intrusions. Therefore, detecting new and unknown intrusion is not feasible in this type of intrusion detection system. In contrast, in anomaly-based detection, new data is compared against a model of normal activity and flagged as anomaly only if the data diverges considerably from the model. Consequently, new and unknown intrusion can be detected successfully in anomaly-based approach. When anomaly-based intrusion detection system is integrated with flow-based traffic monitoring, which it normally is, only needs to inspect packet headers. A direct implication is that, flow-based intrusion detection system needs to deal with moderate amount of data. In many areas of computer science such as, image processing, speech recognition and face detection, machine learning and deep learning are being used successfully. Intrusion detection approaches leveraging machine learning and deep learning are also gaining success gradually. These approaches can find interconnection in data automatically and therefore, can be used to build powerful intrusion detection system. In this research we employed both machine learning and deep learning approach to build a network intrusion detection system and evaluated their performance with the support of NSL-KDD dataset in order to measure the comparative strengths and weaknesses of both approaches.

2. Background and Related Work

In recent years, Flow-based anomaly detection systems have been extensively researched. In [12] a Flow-based anomaly detection system was proposed which is based on Multi-Layer Perceptron (MLP) neural network with one hidden layer and Gravitational Search Algorithm (GSA). This model can differentiate between normal and malicious flows with reasonable accuracy. In [13], the authors introduce an inductive NIDS using One-Class Support Vector Machines for analysis. Unlike other systems, their model is trained with malicious network data, thus, achieved a low false alarm rate. Authors of [14] implemented four leading traffic anomaly detection algorithms (threshold random walk with credit-based rate limiting, rate limiting, maximum entropy detector and NETAD) in an SDN environment using NOX controller and OpenFlow compliant switches. But, according to the results of their experiments, it is evident that, these algorithms are more successful in identifying malicious activities in the small office or home office (SOHO) networks but not in the Internet Service Provider (ISP). A traffic flow features based, and lightweight DDoS attack detection system is demonstrated in [15]. Leveraging programmatic interface of NOX platform traffic flow information is obtained with a very low overhead. Using Self Organizing Maps this method detects DDoS attacks at a high rate. Kokila RT et al. [16] also devised method for detecting DDoS attack analyzing the SVM classifier. According to their experiments, this method produces less false positive rate in comparison with other techniques. An optimized protection mechanism (OpenFlowSIA) using SVM classifier is proposed by Trung et al. [17]. Along with SVM classifier, this mechanism uses Idle-timeout Adjustment (IA) algorithm devised the same authors and can detect DDoS attact in SDN. The entropy variation of the destination IP address is used in a lightweight solution [18] which can detect DDoS attacks within first 250 malicious packets of the traffic. Stacked autoencoder (SAE) based DL model designed by Niyaz et al. [19] can detect multi-vector DDoS attacks in SDN. Tang et al. [20] present a

Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) enabled intrusion detection systems and achieve a high degree of accuracy. However, in very recent times two different approaches of feature selection-based intrusion detection model has been proposed by Dey S.K in [21] and [22] were they employed both deep learning and machine learning approaches for finding the higher accuracy in terms of intrusion detection. Apart from that, Dey S.K [23] also showed the performance analysis of SDN-based intrusion detection model for various machine leaning based classifier with different feature selection approaches.

3. Materials and Methods

In this section of article, we will briefly discuss about our research methodology. We will start from machine learning based classification model approach with different feature selection methods and evaluation procedure. Deep learning-based model development will also be discussed in this section in order to capture a translucent comparative idea regarding these two approaches.

3.1. Proposed Machine Learning Based Model of Classification

Here, we present our proposed ML based model that establishes an effective intrusion detection scheme for classifying intrusion data. This intrusion detection scheme will provide high detection rate and low false alarm rate. A hybrid classification model based on two layers is shown in Figure. 1. The first layer which is based on some common feature selection methods eliminates unrelated and inconsequential features and provides the selected features to the second layer. The second layer, then, classifies the abridged data set using some useful machine learning algorithm. The model is trained and tested further using 10-fold cross-validation technique. We evaluated the model using different accurate measures.

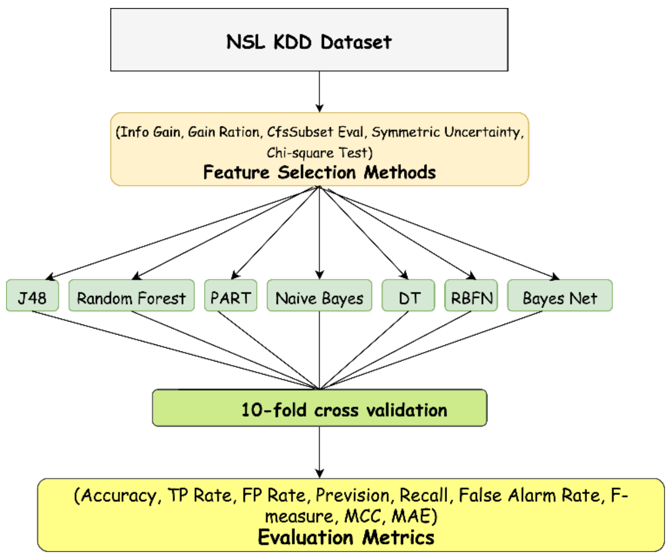
NSL KDD Dataset
(Info Gain, Gain Ration, CfsSubset Eval, Symmetric Uncertainty, Chi-square Test)
Feature Selection Methods
J48
Random Forest
PART
Naive Bayes
DT
RBFN
Bayes Net
10-fold cross validation
(Accuracy, TP Rate, FP Rate, Prevision, Recall, False Alarm Rate, F-measure, MCC, MAE)
Evaluation Metrics

Figure 1. Machine learning based multi-layer classification model [22]

3.2. Machine Learning Approach

According to [24] the general idea behind most machine learning system is that the system learns to perform a task by being trained using an example set of training data. This training enables the system of distributed computers and controllers to accomplish similar tasks where the system is confronted with completely new sets of data it has not encountered before. Therefore, machine learning can be used for flow-based anomaly detection system to automatically build a predictive model based on the training dataset. To solve numerous classification and prediction problems machine learning algorithms have been used [25]. A complete flow chart of anomaly detection mechanism in OpenFlow controller is shown in Figure 2.

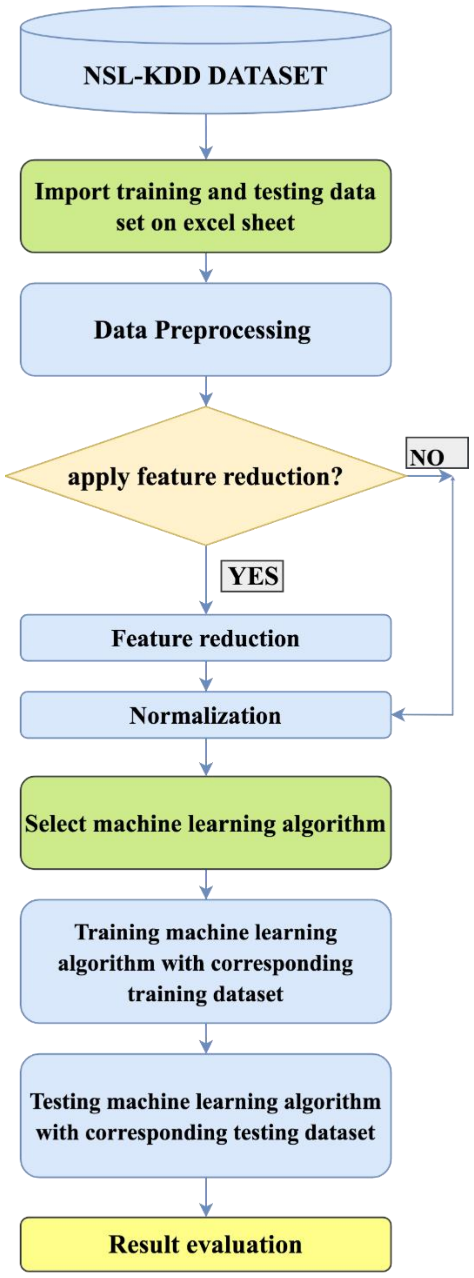
NSL-KDD DATASET
Import training and testing data set on excel sheet
Data Preprocessing
apply feature reduction?
NO
YES
Feature reduction
Normalization
Select machine learning algorithm
Training machine learning algorithm with corresponding training dataset
Testing machine learning algorithm with corresponding testing dataset
Result evaluation

Figure 2. Flow diagram of anomaly detection using machine learning approach [22]

3.3. Overview of NSL-KDD Dataset

Choosing an accurate data set for establishing and evaluating a model is by no means straightforward. Our intent is to select an extensive, free of noise, consistent and redundancy-free data set. For different anomaly detection systems, various types of dataset have been used; some are self-made while others are publicly accessible. KDD-99 is a publicly accessible dataset among others which is most extensively used and accepted dataset. According to authors of [26] KDD-99 is practically free of downside, but the whole dataset is so big that it dramatically increases the cost of computation of the Intrusion Detection. Further, the authors of [27] illustrates that only 10% of the dataset is usually used. In training dataset there are much extraneous data against which there are identical records in testing dataset. Consequently, the process of learning of135the system is sometimes distracted. NSL-KDD is a refined version of KDD-99 which minimizes the redundancies between training dataset and testing dataset. Tavallace et al. [26] recommended NSL-KDD dataset which consists of intrusion data. Many researchers are using the dataset for their experiment. There are41 features in NSL-KDD that contains both normal and attack patterns. There are 5 normal classes and 4 attack classes. The features of NSL-KDD Dataset are summarized below in Table 1.

Table 1. Features and attribute position of NSL-KDD dataset [22].

Type	Features	Attributes position
Nominal	protocol type, service and flag	2, 3, 4
Numeric	duration, src bytes, st bytes, wrong fragment, urgent, hot, num _failed logins, num _ compromised, num root, num file creations, num shells, num access _ files, num _outbound cmds, count srv count, serror rate, srv serror rate, rerror rate, srv _rerror rate, same srv rate, diff srv rate, srv diff host rate, dst host count, dst host srv count, dst _host same srv rate, dst host diff srv rate, dst host same src port rate, dst host srv diff host rate, dst host serror rate, dst host srv serror rate, dst host rerror rate and dst _host srv rerror rate	1, 5, 6, 8, 9, 10, 11, 13, 16, 17, 18, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41
Binary	land, logged in, root shell, su attempted, is host login and is guest login	7, 12, 14, 15, 21, 22

3.4. Selection of Features for Machine Learning Approach

Accuracy of anomaly detection decreases in the presence of redundant attributes in the intrusion dataset. Therefore, developing methods for suitable feature selection that can sort out unrelated attributes can be a research challenge. Feature selection is the technique of choosing relevant features and removing irrelevant features from the dataset to accomplish a certain task [28]. Decreasing the number of redundant, irrelevant and noisy features can lead to a better model by speeding up a data-mining algorithm with an improvement in learning accuracy [29]. We refined dataset features using Info Gain, Gain Ratio, CFS Subset Evaluation, Symmetric Uncertainty and Chi-Square Test. Attribute selection procedures with different evaluator and their search strategies are shown in Table 2.

Table 2. Feature selection with different evaluator and search method [23]

Evaluator	Search	Selected Attributes
Info Gain	Ranker	5,6,3,4,33,35,34,40,41,23,30,29,12,27,28:15
Gain Ratio	Ranker	28,12,41,27,4,6,5,30,29,40,3,25,26,39,34:15
CFS Subset Evaluator	Best First	5,6,12,25,28,30,31,37,41:9
Symmetric Uncertainty	Ranker	6,5,4,41,28,12,27,30,3,40,29,34,35,33,37:15

Chi-Square Test	Ranker	5,6,3,33,35,34,4,40,23,12,41,30,29,27,37:15

3.5. Evaluation Metrics

Performance of intrusion detection rate is measured in terms of accuracy (AC), precision (P), recall(R), F-measure (F), False Alarm Rate (FAR) and Mathews correlation coefficient (MCC). To calculate the value of these techniques, some performance metrics derived from confusion matrix are taken into account. Confusion matrix illustrates the performance of the algorithm according to Table 3. True positive is the total number of samples predicted as normal while they were actually normal and false negative is the total number of samples predicted as normal while they were actually attack. On the other hand, false positive is opposite of true positive and it is the total number of samples predicted as attack while they were actually normal. True negative is the total number of samples predicted as attack while they were actually attack.

Table 3. Tabular form of a confusion matrix [22]

	Predicted as Normal	**Predicted as Attack**
Normal Class (Actually)	True Positive (TP)	False Positive (FP)
Attack Class (Actually)	False Negative (FN)	True Negative (TN)

A good intrusion detection scheme involves high rate of accuracy and high detection rate with a very low false alarm rate. False alarm rate is directly proportional to miss classification rate. A brief discussion and calculating formula 170 for the metrics that are used to evaluate the model are given below:

Accuracy (AC): Shows the proportion of the classification over all N Examples that were correct.

$$AC = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision (P): Shows the proportion of network intrusion detection system detected intrusion that are real intrusion. The higher the value of *P*, the lower the false alarm rate.

$$P = \frac{TP}{TP + FP}$$

Recall (R): Shows the proportion of correctly classified positive examples. We are in search of a high value of *R*.

$$R = \frac{TP}{TP + FN}$$

F-measure (F): By conveying balance between accuracy and recall, it gives a better measure of accuracy. We are in quest of a high F-measure value.

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

Mathews correlation coefficient (MCC): It returns a binary value between −1 and 1 by showing the value correlation coefficient between the predicted and observed binary classifications.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

3.6. Deep Learning Approach

3.6.1. Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is a type of artificial neural network which has the capability of learning from previous time-steps. RNNs are extended form of typical Feed-forward neural network (FNN). But, in contrast with FNN, RNNs use their internal state while processing sequential data. Using internal state, here, refers to the fact that, RNN takes advantage of previous computations for output. As they carry out same task for each element in the sequence, they are called recurrent. The structure of a simple of RNN is depicted in Figure 3.

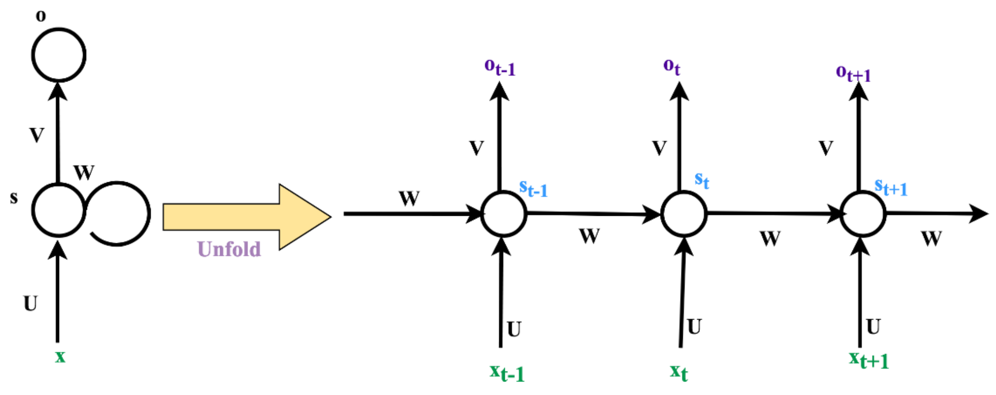

Figure 2. Structure of a plain Recurrent Neural Network (RNN) [21]

In the above figure, x_t is input and o_t is output. s_t is considered as hidden state. f indicates nonlinear function, such as *tanh* or *ReLU*. s_t is calculated using previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. To calculate the initial hidden state, s_{-1}, is required which is initialized 185 to zeroes by default. Hidden sates of RNN are computed as:

$$s_t = f(Ux_t + Ws_{t-1}), \text{for } t = T, \ldots, 1 \quad (1)$$

A gradient-based algorithm namely Backpropagation through time (BPTT) is generally applied for training the RNN. RNN training is considerably faster using BPTT algorithm than other existing optimization techniques. However, RNN Model with backpropagation has a significant drawback, called vanishing gradient problem. It happens when the gradient is so small that it seems vanished. Consequently, it prevents the value of weight from changing and in some cases, stops further training. According to [30], vanishing gradient problem prevents RNN from being accurate. To solve these problems, more powerful combined models like Long short-term memory (LSTM) [31] and Gated Recurrent Units (GRUs) [32] were suggested.

3.6.2. Long short-term Memory RNN (LSTM)

A deep neural network is unfolded in time and for every time-step an FNN is constructed. Then, weights and biases for each hidden layer are updated by the gradient rule. These updates minimize the loss between the expected and actual outputs. But, when the time-steps are more than 5-10, standard RNNs don't perform better. Weights fluctuate due to the prolonged back-propagation vanishing or blowing up error signals, making the network performance poor. Accordingly, to deal with this vanishing gradient problem, researchers suggested the Long-Short-Term-Memory (LSTM) network. LSTM bridges the minimal time gaps and makes use of a gating mechanism to handle long-term dependencies. The LSTM structure can be seen in Figure 4.

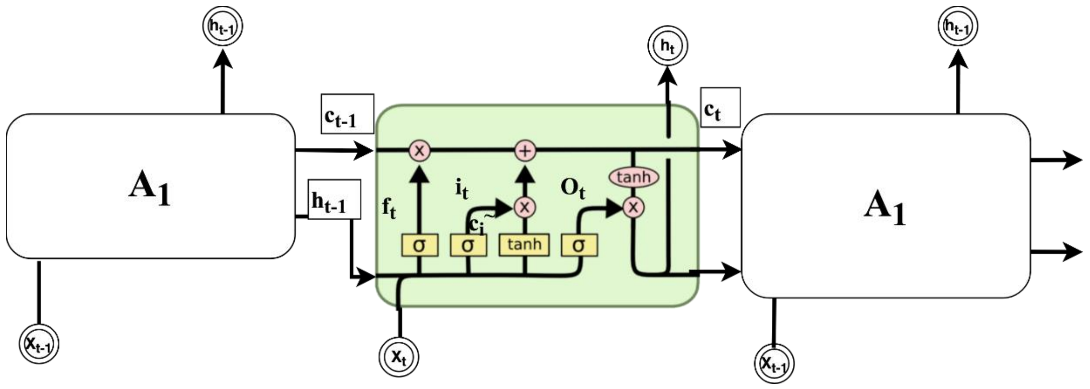

Figure 4. Simple structure of a Long Short-Term Memory Unit [21]

3.6.3. Gated Recurrent Unit (GRU)

A Gated Recurrent Unit (GRU) is a lighter version of an LSTM. The reduced complexity in the structure of GRU is obtained by decreasing the gates in the architecture. To solve the vanishing gradient problem of a standard RNN, both update gate and reset gate are used by GRU. These, essentially, are two vectors which decides which information should be passed to the output. Because the training phase of GRU is smoother and faster than that of LSTM, we have selected GRU for our model development [33]. Both the "forget gate" and "input gate" in an LSTM are merged to an "update gate" in GRU and the hidden state and cell state are combined, resulting in a simpler architecture as shown in Figure 5.

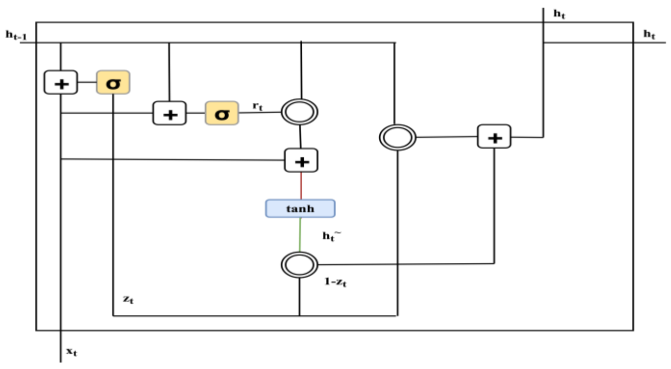

Figure 5: Single layer Gated Recurrent Unit [21]

The following relationship can be obtained from Figure 5.

$$\text{Update gate } z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right) \tag{2}$$

$$\text{Candidate activation } \widetilde{h_t} = \tanh(Wx_t + r_t \odot Uh_{t-1}) \tag{3}$$

$$\text{Reset gate } r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} \tag{4}$$

$$\text{Activation function } h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{5}$$

3.6.4. Multi-Layer GRU RNN

The performance of algorithm greatly depends on the numerous architectures of deep neural network. A deep structure called Multi-layer RNN is constructed stacking different layers of RNN (plain RNN, LSTM, GRU). An RNN using GRU cells in each hidden layer is called GRU RNN. Apart from Back-propagation through time, network input is passed through multiple GRU layers in the multilayer structure. In [34], it has been proven that multilayered RNNs learn from the different time lengths of input sequences. To achieve optimized performance, multi-layered RNNs share the hyper parameters, weights, and biases across the layers.

3.6.5. Overview of Scikit-Learn

During experiment, we used to scikit-learn which is a python-based machine learning library for data mining and data analysis [35]. Data of most machine learning algorithm needs to be stored in 2D array or in a matrix shape. In scikit-learn, these 2D shaped data can be stored effectively. Following Figure 6 shows the scikit-learn data representation, where there are N samples and D features.

$$feature\ matrix: x = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$label\ vector: y = [y_1, y_2, y_3, \dots, y_N]$$

Figure 6: Data representation in *scikit-learn*

3.6.6. Appropriate Feature Selection for Deep Learning Approach

Feature selection mechanism is a required process to get rid of the irrelevant and extraneous data form the dataset. According to [35] feature selection is a process of deriving a subset of relevant features from the complete feature set without decaying presentation. Intrusion dataset containing superfluous attributes often prevents detection from being accurate. Numerous reasons were analyzed to show why restricting the features is obligatory. Irrelevant features increase computation time without contributing to classifier improvement and sometimes incorrectly indicate correlation between feature and desired class. In our experiment, we have used a univariate feature selection with Analysis of Variance (ANOVA) F-test. ANOVA is used to determine whether the means of some groups are different using F-test which statistically checks the equality of means. Each feature is individually analyzed which calculates the strength of feature-labels relationship. Percentile of the highest scores based feature selection is performed by SelectPercentile method (*sklearn.feature selection).* Upon finding a subset Recursive Feature Elimination (RFE) is applied. REF frequently builds a model where features are kept aside and reiterates the process until all features in dataset are removed. Feature ranking is developed by using the weight of a classifier. Following table represents the selected features after applying both ANOVA F-test and RFE.

Table 4: Selected Features after Applying ANOVA F-Test and RFE [21]

Attack Category	Selected Features
DoS	(1, 'flag SF'), (2, 'dst host serror rate'), (3, 'same srv rate'), (4, 'count'), (5, 'dst host srv count'), (6,'dst host same srv rate'), (7, 'logged in'), (8, 'dst host count'), (9, 'serror rate'), (10, 'dst host srv serror rate'), (11,'srv serror rate'), (12, 'service http'), (13, 'flag S0')
Probe	(1, 'service private'), (2, 'service eco i'), (3,dst host srv count'), (4, 'dst host same src port rate'), (5, 'dst host srv rerror rate'), (6, 'dst host diff srv rate'), (7, 'dst host srv diff host rate'), (8, 'dst host rerror rate'), (9, 'logged in'), (10, 'srv rerror rate'), (11,'Protocol type icmp'), (12, 'rerror rate'), (13, 'flag SF')
R2L	(1, 'src bytes'), (2, 'hot'), (3, 'dst host same src port rate'), (4,'dst host srv count'), (5, 'dst host srv diff host rate'), (6, 'dst bytes'), (7, 'service ftp data'), (8, 'num failed logins'), (9, 'is guest login'), (10, 'service imap4'), (11, 'service ftp'), (12,'flag RSTO'), (13, 'service http')
U2R	(1, 'hot'), (2, 'dst host srv count'), (3, 'dst host count'), (4,'num file creations'), (5,'root shell'),(6,'dst host same src port rate'),(7,'dst host srv diff host rate'), (8, 'service ftp data'), (9,'service telnet'), (10, 'num shells'), (11, 'urgent'), (12,'service http'), (13, 'srv diff host rate')

3.7. Designed Algorithm and Proposed SDN-based Anomaly Detection Architecture

Generally, OpenFlow switches are monitored by SDN controller. SDN controller can request for all network data whenever necessary. Therefore, we implemented our proposed intrusion detection segment in SDN controller for both machine learning and deep learning approaches, which is illustrated in Figure 7. Our suggested approach for ML based classification model is summarized in the following Algorithm 1.

```
1  Procedure FLOW BASED ANOMALY DETECTOR BASED ON ML MODEL
2      Selection of appropriate machine learning algorithm;
3      Train the ML-based model using benchmark dataset NSL-KDD;
4      Nomination of appropriate feature after using different feature selection methods;
5      if The trained model predicts an anomaly class on an OpenFlow Controller by the ML based Intrusion Detection Model then
6          Update the SDN OpenFlow controller rules to block that class attack type;
7      else
8          Allow the normal class to pass through SDN controller and access the available resources;
9      end
10 end
```

Algorithm 1: Machine Learning based anomaly class detector for SDN attacks

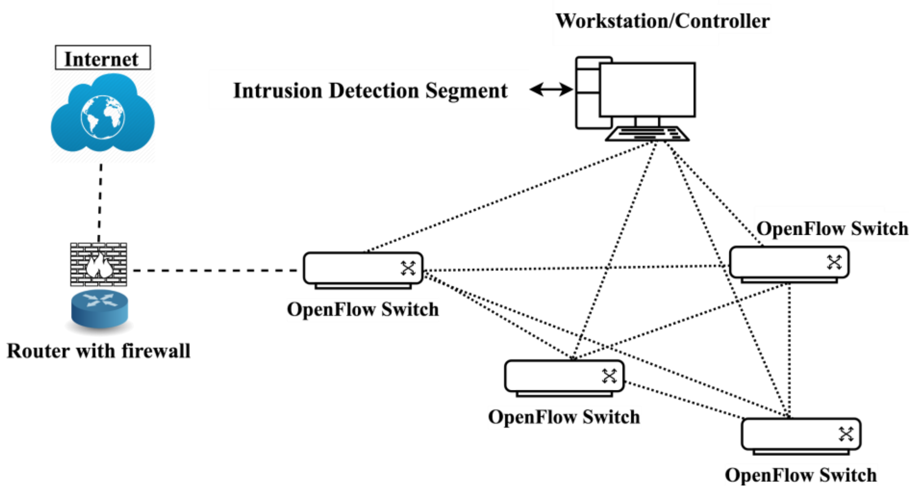

Figure 7: Proposed flow-based anomaly detection architecture in SDN [23]

For requesting network data, an OpenFlow stats request message will be sent from the controller to all OpenFlow switches. As controller request for all the available statistics, an OpenFlow stats reply message with all available data send back to the controller by OpenFlow switch. Figure 8 clearly describes the architecture of how OpenFlow switch handles the incoming packet and responds according to the availability of data in Flow table by using Open Flow protocol. One of the noticeable behaviors of SDN that its centralized controller can take Opportunities of the complete network to evaluate and associate feedback from the network. Therefore, Open Flow protocol can effectually alleviate an intrusion via flow table adjustment if once a network anomaly is discovered and recognized. Our suggested approach for Deep learning based classification model is summarized in the following Algorithm 2.

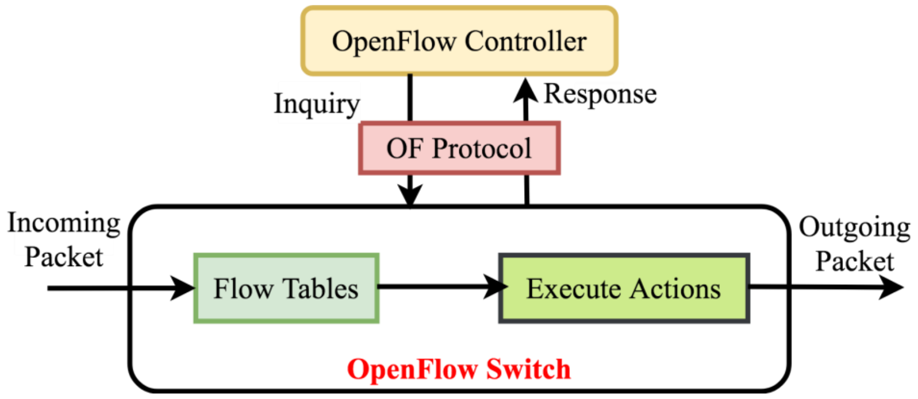

Figure 8: Diagram of Handling Incoming Packets in OpenFlow Switch [23]

```
1  Procedure FLOW BASED ANOMALY DETECTOR BASED ON
     DEEP MODEL
2      Selection of appropriate Deep learning algorithm;
3      Train the Deep Neural Network (DNN)-based model using benchmark
         dataset NSL-KDD;
4      Nomination of appropriate feature after using different feature
         selection methods;
5      if The trained model predicts an anomaly class on an OpenFlow
         Controller by the DNN based Intrusion Detection Model then
6          Update the SDN OpenFlow controller rules to block that class
             attack type;
7      else
8          Allow the normal class to pass through SDN controller and access
             the available resources;
9      end
10 end
```

Algorithm 2: Deep Learning based anomaly class detector for SDN attacks

4. Experimental Results and Findings

4.1. Experimental results of machine learning approach

We used WEKA [36] environment and NSL-KDD Dataset for carrying out the experiments. The environment consists of a machine with 6 GB of memory and a Processor Intel(R) Core(TM) i5-2410M CPU @ 2.30 GHz, 2301 MHz, Dual Core(s), and 4 Logical Processor(s). To make loading and analyzing the dataset trouble-free heap size of WEKA was increased. To train and test each selected 285 feature, NSL-KDD dataset was used. The whole dataset is publicly accessible. 10-fold cross validation technique was used to perform the experiment. Dividing the training set into 10 subsets, we tested each subset when the model is trained on the other 9 subsets. Each subset is processed only once as test data; therefore, the process repeats 10 times. For simplicity, in Table 5, we only mentioned results of the higher accuracy obtained classifier with various feature selection method.

Table 5: Random forest classifier showing the highest accuracy with gain ratio feature selection

Feature Selection Method	Classifier Techniques	Accuracy	TP Rate	FP Rate	Evaluation Criteria			F-Measure	MCC	MAE
					Precision	Recall	FAR			
Info Gain	Random Forest	79.360	0.794	0.163	0.846	0.794	0.341	0.792	0.641	0.229
CFS Subset	PART	79.249	0.792	0.167	0.839	0.792	0.333	0.791	0.633	0.264
Gain Ratio	Random Forest	81.946	0.819	0.143	0.860	0.819	0.297	0.819	0.681	0.232
Symmetric Uncertainty	Random Forest	80.708	0.807	0.153	0.853	0.807	0.317	0.806	0.661	0.221
Chi-square	Random Forest	80.132	0.801	0.157	0.850	0.801	0.328	0.800	0.653	0.222

Table 6 represents the results of entire classifier with different feature Selection method. Experimental results of accuracy, True positive rate, False positive rate, Precision, Recall, False alarm rate, F-measure, Mathews correlation coefficient and Means absolute error of different classifier are shown. Furthermore, only the results of highest accuracy of different classifier in terms of feature selection are displayed. Our aim was to achieve high Accuracy, Recall and MCC and low False alarm rate. In our experiment, we successfully attained that. From the experimental data, Random Forest with Gain Ratio feature selection method exhibits the highest accuracy of 81.946% which is illustrated from our experimental results.

Table 6: Results of all classifier with different feature selection method

Feature Selection Method	Classifier Techniques	Evaluation Criteria								
		Accuracy	TP Rate	FP Rate	Precision	Recall	FAR	F-Measure	MCC	MAE
Info Gain	J48	78.006	0.781	0.172	0.84	0.781	0.364	0.779	0.623	0.229
	Random Forest	**79.36**	**0.794**	**0.163**	**0.846**	**0.794**	**0.341**	**0.792**	**0.641**	**0.229**

	PART	77.102	0.771	0.18	0.835	0.771	0.382	0.768	0.609	0.231
	Nave Bayes	72.068	0.721	0.227	0.789	0.721	0.442	0.715	0.514	0.279
	DT	72.595	0.726	0.214	0.814	0.726	0.461	0.718	0.545	0.197
	RBFN	71.965	0.72	0.228	0.787	0.72	0.441	0.714	0.511	0.299
	Bayes Net	73.203	0.732	0.209	0.816	0.732	0.45	0.725	0.553	0.268
CFS Subset Evaluator	J48	73.984	0.74	0. 203	0.82	0.74	0.436	0.734	0.564	0.267
	Random Forest	74.84	0.784	0.197	0.823	0.748	0.42	0.743	0.575	0.345
	PART	**79.249**	**0.792**	**0.167**	**0.839**	**0.792**	**0.333**	**0.791**	**0.633**	**0.264**
	Nave Bayes	74.702	0.747	0.829	0.829	0.747	0.43	0.741	0.58	0.253
	DT	43.075	0.431	0.431	0.186	0.431	1	0.259	0	0.504
	RBFN	71.127	0.711	0.222	0.817	0.711	0.505	0.7	0.533	0.323
	Bayes Net	60.632	0.606	0.298	0.794	0.606	0.691	0.564	0.401	0.447
Gain Ratio	J48	81.871	0.819	0.145	0.858	0.819	0.293	0.818	0.677	0.193
	Random Forest	81.946	0.819	0.143	0.86	0.819	0.297	0.819	0.681	0.232
	PART	**77.905**	**0.779**	**0.179**	**0.835**	**0.779**	**0.362**	**0.777**	**0.616**	**0.231**
	Nave Bayes	76.242	0.762	0.186	0.832	0.762	0.398	0.758	0.597	0.237
	DT	72.595	0.726	0.214	0.814	0.726	0.461	0.718	0.545	0.197
	RBFN	75.177	0.752	0.193	0.828	0.752	0.419	0.747	0.584	0.272
	Bayes Net	71.517	0.715	0.221	0.812	0.715	0.483	0.705	0.532	—
Symmetric Uncertainty	J48	78.927	0.789	0.167	0.842	0.789	0.346	0.787	0.633	0.218
	Random Forest	**80.708**	**0.807**	**0.153**	**0.853**	**0.807**	**0.317**	**0.806**	**0.661**	**0.221**
	PART	80.371	0.804	0.157	0.848	0.804	0.318	0.803	0.653	0.221
	Nave Bayes	73.292	0.733	0.21	0.813	0.733	0.444	0.726	0.551	0.266
	DT	72.595	0.726	0.214	0.814	0.726	0.461	0.718	0.545	0.197
	RBFN	73.522	0.735	0.209	0.812	0.735	0.438	0.729	0.552	0.288
	Bayes Net	71.562	0.716	0.222	0.808	0.716	0.478	0.706	0.529	0.282
Chi-square Test	J48	78.051	0.781	0.173	0.838	0.781	0.363	0.778	0.621	0.229
	Random Forest	80.132	0.801	0.157	0.85	0.801	0.328	0.8	0.653	0.222
	PART	**77.989**	**0.78**	**0.173**	**0.84**	**0.78**	**0.367**	**0.777**	**0.622**	**0.218**
	Nave Bayes	72.618	0.726	24 0.224	0.79	0.726	0.43	0.722	0.521	0.273
	DT	72.595	0.726	0.214	0.814	0.726	0.461	0.718	0.545	0.197
	RBFN	70.723	0.707	0.234	0.789	0.707	0.475	0.699	0.502	0.31
	Bayes Net	72.409	0.724	0.215	0.812	0.724	0.463	0.716	0.541	0.275

4.2. Experimental results of Deep learning approach

We used Googles TensorFlow [37] to carry out the experiments. TensorFlow provides an option to visualize the network design. The experiments are per- formed in an environment of Intel i5 3.2 GHz, 16 GB RAM, and NVIDIA GTX 1070 with Linux based Ubuntu 16.10 Distribution Operating System. We have used the TensorFlow tf.train.AdamOptimizer. The following Table 7, shows the initialization of hyper parameter set. Learning rate is controlled by Kingma and Ba's Adam algorithm in *tf.train.AdamOptimizer*.

Table 7: Set of different Hyper parameter

#Hyper Parameters
learning rate = 0.001
training epochs =10
display step=1 num
layers=1

#Definition of Hyper parameters for the model
learning rate = 0.001
number of classes=2
display step=100
input features=train X.shape[1] #*No of Selected features*
training cycles=1000 #*No of time-steps to back propagate*
time steps=5
hidden units=50 #*No of LSTM units in a LSTM Hidden Layer*

The model is developed using Python programming language along with several libraries like python based numpy, machine learning based scikit-learn, pandas for data visualization, and TensorFlow for model development. We began our experiments with a light-weight GRU with one hidden layer and one hidden unit. For each hyper parameter set (learning rate, time-steps, hidden layers) 10 sets of experiments were carried out and we tuned them to obtain the optimized results. Following Table 8 represents the results of various evaluation metrics like accuracy, precision, recall, False alarm rate and F-1 score for each time steps.

Table 8: Evaluation metrics for all layer ids classifier

Time-steps	Learning rate	Train Accuracy	Precision	Recall	FAR
10	86.632	0.9994	0.99	0.9977	0.0022
20	85.534	0.9943	0.3296	0.9922	0.0077
30	84.510	0.986	0.9952	0.9418	0.05812
40	86.613	0.9996	0.9902	0.9983	0.0016
50	85.434	0.9967	0.9919	0.9865	0.0134
60	72.89	0.8914	0.9935	0.5011	0.4988
70	**87.911**	**0.9981**	**0.9939**	**0.9923**	**0.0076**

80	83.243	0.9999	0.9842	0.9997	0.0002
90	83.323	0.9995	0.9859	0.9981	0.0018
100	82.167	0.9937	0.9925	0.974	0.0257

4.3. Comparative analysis of two approaches

In this section, we will briefly discuss about experimental procedures and analyze our results for further use in the sector of intrusion detection. Feature selection is considered as a prime component of this research work. For both the approaches we have prepared our dataset by applying some fruitful feature selection algorithm. Previously researcher from different domain have used NSL-KDD dataset for intrusion detection but none of the approaches have followed a proper feature selection approach for their experiment. In Machine Learning approach, we have used some well researched machine learning algorithm like J48, Random Forest, PART, Nave Bayes, Decision Tree, Radial Basis Function Network (RBFN) and Bayes Net. In order to eliminate the ambiguous data from the dataset, we also employed some feature selection algorithm like Info Gain, Gain Ratio, CfsSubset Evaluator, Symmetric Uncertainty, and Chi Square Test. After successful Experiment we found that, from Table 5 and 6, Random Forest classifier with Gain Ratio Feature selection approach generate 81.946% accuracy with a very low false alarm rate of 0.297%.

Apart from that, we also developed a GRU-LSTM based deep learning model with ANOVA F-test and Recursive Feature Elimination (RFE) selection approach. As our aim is to achieve a high detection accuracy in terms of different approach with feature selection methods, we have tested our model with different time steps with different learning rate. After successful experiment we observed that with a learning rate of 0.01 and time steps of 70 our model achieved a detection accuracy of approximately 88%. Complete results for 0.01 learning rate with time steps [10, 20, 30,……., 90, 100] is depicted in Table 8. At this point Deep learning approach shows much potential outcomes comparing with machine learning approach. In our research implementation of the SDN based intrusion detection system, the classification model is mainly 2-class based namely normal and another is anomaly. After evaluating both the results, we proposed a model of security architecture which detects flow-based anomaly in OpenFlow based Controller. From the detailed results above and from Table 9, we can derive to a decision that, the ANOVA F-Test and Recursive Feature Elimination (RFE) methods with GRU-LSTM classifier provides highest Accuracy of 87.911% with a very low false alarm rate of 0.0076%.

Furthermore, we generated the results using the selected features from complete NSL-KDD dataset. Some other approaches have been presented from different authors in order to show the accuracy of deep learning algorithm for NSL-KDD Dataset. Lack of preprocessing of dataset and appropriate feature selection for testing and training, however, are absent in some approaches.

5. Conclusions

In this research, we have presented two different approaches for predicting the flow based anomaly in Software Defined Networking. Deep learning based GRU-LSTM model and Machine learning based Random Forest (RF) model for detecting network intrusion in SDN have been designed. And, we have showed the best classifier model in terms of different evaluation metrics with ANOVA F-Test and RFE feature selection and Gain Ratio feature selection mechanism respectively. Although both the approaches produces significant experimental results comparing with others work therefore it still has some effective contribution in the field of appropriate feature selection from a dataset. In SDN environment, both the approaches has enormous potential to detect malicious activity. SDN supports the nature of centralized controller and a very flexible structure. Our proposed intrusion detection module is capable of easily extract the information about network traffic due to its centralized controller and flexible nature. From the experimental results, it's evident that GRU-LSTM and GAIN RATIO-RF shows a high-test accuracy comparing with all other algorithms. However, deep learning approach produce slightly better results than machine learning approach therefore for flow based anomaly detection use of GRU-LSTM model is entirely essential in order to achieve high accuracy and speeding up the process of intrusion detection in SDN. In near future, we plan to implement our proposed model in a real environment of SDN with real traffic of network.

References

1. Software defined networking definition, https://www.opennetworking. org/sdn-definition, accessed May 16, 2017.
2. Onf sdn evolution, http://3vf60mmveq1g8vzn48q2o71a-wpengine. netdna-ssl.com/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf, accessed February 25, 2018.
3. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: Enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74. doi:10.1145/1355734.1355746. URL http://doi.acm.org/10.1145/1355734.1355746.
4. S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. H¨olzle, S. Stuart, A. Vahdat, B4: Experience with a globally-deployed software defined wan, SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14. doi:10.1145/2534169.2486019. URL http://doi.acm.org/10.1145/2534169.2486019.
5. C. t. huawei press centre and h. unveil world's first commercial deployment of sdn in carrier networks, http:://pr.huawei.com/en/news/ hw-332209-sdn.htm, accessed February 28, 2018.
6. N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: Towards an operating system for networks, SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 105–110. doi:10.1145/1384609.1384625. URL http://doi.acm.org/10.1145/1384609.1384625.
7. Ryu, http://osrg.github.io/ryu, accessed March 11, 2018.
8. D. Erickson, The beacon openflow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 13–18. doi:10.1145/2491185.2491189. URL http://doi.acm.org/10.1145/2491185.2491189.
9. Opendaylight: a linux foundation collaborative project, http://www. opendaylight.org, accessed March 6, 2018.
10. Floodlight, http://www.projectfloodlight.org, accessed March 15, 2018.
11. D. Kreutz, F. M. Ramos, P. Verissimo, Towards secure and dependable software-defined networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 55–60. doi:10.1145/2491185.2491199. URL http://doi.acm.org/10.1145/2491185.2491199.
12. T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), 2016, pp. 258–263. doi: 10.1109/WINCOM.2016.7777224.
13. R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: 2010 IEEE Symposium on Security and Privacy, 2010, pp. 305–316. doi:10.1109/SP.2010.25.

14. Z. Jadidi, V. Muthukkumarasamy, E. Sithirasenan, M. Sheikhan, Flow-based anomaly detection using neural network optimized with gsa algorithm, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, 2013, pp. 76–81. doi:10.1109/ICDCSW.2013. 40.
15. P. Winter, E. Hermann, M. Zeilinger, Inductive intrusion detection in flow-based network data using one-class support vector machines, in: 2011 4th IFIP International Conference on New Technologies, Mobility and Security, 2011, pp. 1–5. doi:10.1109/NTMS.2011.5720582.
16. S. A. Mehdi, J. Khalid, S. A. Khayam, Revisiting traffic anomaly detection using software defined networking, in: Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, RAID'11, Springer-Verlag, , Berlin, Heidelberg, 2011, pp. 161–180. doi:10.1007/978-3-642-23644-0_9. URL http://dx.doi.org/10.1007/978-3-642-23644-0_9.
17. R. Braga, E. Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: IEEE Local Computer Network Conference, 2010, pp. 408–415. doi:10.1109/LCN.2010.5735752.
18. and S. Thamarai Selvi, K. Govindarajan, Ddos detection and analysis in sdn-based environment using support vector machine classifier, in: 2014 Sixth International Conference on Advanced Computing (ICoAC), 2014, pp. 205–210. doi:10.1109/ICoAC.2014.7229711.
19. S. M. Mousavi, M. St-Hilaire, Early detection of ddos attacks against sdn controllers, in: 2015 International Conference on Computing, Networking and Communications (ICNC), 2015, pp. 77–81. doi:10.1109/ICCNC. 2015.7069319.
20. A. AlEroud, I. Alsmadi, Identifying cyber-attacks on software defined networks, J. Netw. Comput. Appl. 80 (C) (2017) 152–164. doi:10.1016/j. jnca.2016.12.024. URL https://doi.org/10.1016/j.jnca.2016.12.024.
21. S. K. Dey, M. M. Rahman, Flow based anomaly detection in software de-fined networking: A deep learning approach with feature selection method, in: 2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT), 2018, pp. 630–635. doi: 10.1109/CEEICT.2018.8628069.
22. S. K. Dey, M. M. Rahman, M. R. Uddin, Detection of flow based anomaly in openflow controller: Machine learning approach in software defined networking, in: 2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT), 2018, pp. 416– 421. doi:10.1109/CEEICT.2018.8628105.
23. S. K. Dey, M. Raihan Uddin, M. Mahbubur Rahman, Performance analysis of sdn-based intrusion detection model with feature selection approach, in: M. S. Uddin, J. C. Bansal (Eds.), Proceedings of International Joint Conference on Computational Intelligence, Springer Singapore, Singapore, 2020, pp. 483–494.
24. P. Louridas, C. Ebert, Machine learning, IEEE Software 33 (5) (2016) 110-115. doi:10.1109/MS.2016.114.
25. G. M. Khan, S. Khan, F. Ullah, Short-term daily peak load forecasting using fast learning neural network, in: 2011 11th International Conference on Intelligent Systems Design and Applications, 2011, pp. 843–848. doi: 10.1109/ISDA.2011.6121762.
26. M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani, A detailed analysis of the kdd cup 99 data set, in: Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications, CISDA'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 53–58. URL http://dl.acm.org/citation.cfm?id=1736481.1736489.
27. Y. Meng, The practice on using machine learning for network anomaly intrusion detection, in: 2011 International Conference on Machine Learning and Cybernetics, Vol. 2, 2011, pp. 576–581. doi:10.1109/ICMLC.2011. 6016798.
28. Y. Yang, J. O. Pedersen, A comparative study on feature selection in text categorization, in: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 412–420. URL http://dl.acm.org/citation.cfm?id=645526.657137.
29. B. Ingre, A. Yadav, Performance analysis of nsl-kdd dataset using ann, in: 2015 International Conference on Signal Processing and Communication Engineering Systems, 2015, pp. 92–96. doi:10.1109/SPACES.2015. 7058223.
30. J. F. Kolen, S. C. Kremer, Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, IEEE, 2001. doi:10.1109/9780470544037.ch14. URL https://ieeexplore.ieee.org/document/5264952.
31. S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

32. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder– decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724-1734. doi:10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.
33. J. Chung, C¸. Gu¨lc¸ehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, CoRR abs/1412.3555. arXiv:1412.3555. URL http://arxiv.org/abs/1412.3555.
34. Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436-444. URL https://doi.org/10.1038/nature14539.
35. H. Nkiama, S. Zainudeen, M. Saidu, A subset feature elimination mechanism for intrusion detection system, International Journal of Advanced Computer Science and Applications 7. doi:10.14569/IJACSA.2016. 070419.
36. Weka, https://www.cs.waikato.ac.nz/ml/weka, accessed March 22, 2018.
37. Tensorflow, https://github.com/tensorflow, accessed March 30, 2018.