Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks

Tuan A Tang*, Syed Ali Raza Zaidi*, Des McLernon*, Lotfi Mhamdi* and Mounir Ghogho†
*School of Electronic and Electrical Engineering, The University of Leeds, Leeds, UK.
†International University of Rabat, Morocco.
Email: eltat@leeds.ac.uk, s.a.zaidi@leeds.ac.uk, d.c.mclernon@leeds.ac.uk, l.mhamdi@leeds.ac.uk and m.ghogho@leeds.ac.uk.

***Abstract*—Software Defined Networking (SDN) has emerged as a key enabler for future agile Internet architecture. Nevertheless, the flexibility provided by SDN architecture manifests several new design issues in terms of network security. These issues must be addressed in a unified way to strengthen overall network security for future SDN deployments. Consequently, in this paper, we propose a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) enabled intrusion detection systems for SDNs. The proposed approach is tested using the NSL-KDD dataset, and we achieve an accuracy of 89% with only six raw features. Our experiment results also show that the proposed GRU-RNN does not deteriorate the network performance. Through extensive experiments, we conclude that the proposed approach exhibits a strong potential for intrusion detection in the SDN environments.**

***Index Terms*—software defined networking; SDN; intrusion detection; deep learning; recurrent neural network; gated recurrent unit; GRU; network security**

I. Introduction

A. Motivation

The current Internet architecture has existed for nearly three decades and is now becoming an increasingly complex system. Consequently, the legacy Internet lacks agility to respond to ever changing demands and dynamic nature of modern day applications. Software Defined Networking (SDN) [1] is introduced as a promising architecture, enabling scalability and unprecedented flexibility in the configuration and deployment of network services. The separation of control plane and data plane provides more flexibility and greater control over the traffic flows. The flow-based nature of SDNs enables network information acquistion in real-time via the OpenFlow [2] protocol. Nevertheless, as highlighted in [3], the SDN architecture also introduces various security issues pertaining to the control plane, the control-data interface and the control-application interface. Recently, SDN security has become a serious concern and has attracted significant interest (For instance, see [4] and [5] and references there in).

An intrusion detection system (IDS) is one of the most important network security tools. The Anomaly-based IDS tries to identify observations that deviates from a baseline model. Various approaches have been proposed for the Anomaly-based IDS like artificial neural network (ANN), support vector machine (SVM), and Bayesian Network. However, these techniques have a high False Alarm Rate (FAR) and associated computational cost as mentioned in [6]. Recently, Deep Learning (DL) has emerged as a new approach that delivers higher accuracy than traditional machine learning techniques. DL has the ability to process raw data and learn the high-level features on its own, and so DL has a strong case for its adaptability in resource constrained networks like SDNs.

B. Contribution

Following the current trajectory of research, we believe that deep recurrent neural networks (RNNs) can potentially offer better solution for implemention of IDS for SDN. A Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) is proposed for anomaly detection. The GRU-RNN is a powerful technique that can represent the relationship between current and previous events and enhance the anomaly detection rate.

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

- We introduce an IDS in the SDN paradigm using GRU-RNN. To the best of our knowledge, this is the first attempt to use GRU-RNN for an IDS in the SDN environment.
- Our GRU-RNN approach yields a detection rate of 89% using a minimum number of features compared to other state-of-the-art approaches.
- We also evaluate the network performance of the proposed approach in the SDN. The test results show that our approach is significantly potential for real time detection.

The rest of this paper is organized as follows. Section II shows the related work. In Section III, we present the system description. Section IV presents the detection performance analysis. The network performance analysis is described in Section V. Finally, Section VI concludes the paper and presents future work.

II. Related Work

In past several studies (see [7], [8], [9] and [10]), researchers have employed classical machine learning mechanism such as SVM, K-Nearest Neighbour (KNN), ANN, Random Forest etc. for developing an IDS. These proposed methods have achieved various degrees of success while rendering some inherent limitations. These work focus on tradintional network with a large set of features that cannot be applied to SDNs.

Early work on the flow-based anomaly detection approach using SDN include [11] and [12]. Braga et al. [11] present

a lightweight approach using a Self Organizing Map (SOM) to detect DDoS attacks in the SDN. This approach based on six traffic flow features gives quite high detection accuracy. In [12], the author use four traffic anomaly detection algorithms (threshold random walk with credit based rate limiting, rate limiting, maximum entropy and NETAD) in the SDN. The experiments indicate that these algorithms perform better in the SOHO (Small Office/Home Office) network than in the ISP (Internet Service Provider).

In [13] and [14], SVM is used to detect DDoS attacks quite efficiently. K-Nearest Neighbor and graph theory are combined to classify DDoS attacks from benign flows in SDNs by AlEroud et al. in [15]. Mousavi et al. [16] propose an early DDoS attack detection method against the SDN controller based on the variation of the entropy of the flow's destination IP addresses. Their detection rate is 96% with just first 250 packets. In [17], the authors propose a DL based approach using a stacked autoencoder (SAE) for detecting DDoS attacks in the SDN. They achieve a quite high accuracy and low FAR on their own dataset.

In 2016, we applied Deep Neural Network (DNN) under the context of SDNs with the NSL-KDD dataset [18]. We obtained a potential accuracy of 75.75% with just six basic features. In this paper, we continue this trend by using GRU-RNN to improve the detection accuracy and reduce the FAR.

III. Methodology/System Description

In this section, the RNN and GRUs are briefly reviewed. The architecture of the SDN-based IDS is described in detail. The NSL-KDD dataset are also discussed in this section.

A. Recurrent Neural Networks

A RNN, which is an extension of a conventional feed forward neural network, makes use of the sequential information. The RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.

The hidden states of the RNN are computed as:

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

where σ is a nonlinearity function, $\mathbf{x}_t$ is an input vector at time t, $\mathbf{h}_t$ is a hidden state vector at time t, W is an input to hidden weight matrix, U is a hidden to hidden weight matrix, and b_h is a bias term.

The Backpropagation Through Time (BPTT) algorithm is used for training the RNN. However, the traditional RNN encounters vanishing/exploding gradient problems [19]. Long Short Term Memory (LSTM) [20] networks and Gated Recurrent Units (GRUs) [21] were proposed to solve this problem.

B. Gated Recurrent Unit

GRUs are selected in our research because of their simplicity and faster training phase compared to LSTMs [22]. Fig. 1 shows architectural detail of a single GRU cell. The relationship in Fig. 1 is given by

$$\mathbf{r}_t = \sigma(\mathbf{x}_t W_r + \mathbf{h}_{t-1} U_r), \quad (2)$$

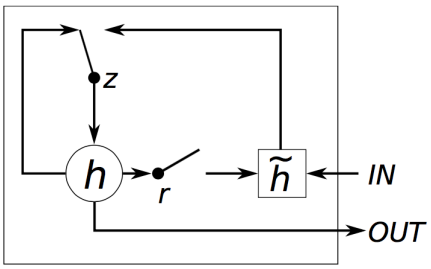

Fig. 1. Gated Recurrent Unit Structure [22]

$$\mathbf{z}_t = \sigma(\mathbf{x}_t W_z + \mathbf{h}_{t-1} U_z), \quad (3)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t \tilde{\mathbf{h}}_t, \quad (4)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{x}_t W_h + (\mathbf{h}_{t-1} \odot \mathbf{r}_t) U_h), \quad (5)$$

where $\mathbf{r}_t$ is the reset gate, $\mathbf{z}_t$ is the update gate, $\mathbf{h}_t$ is the activation function and $\tilde{\mathbf{h}}_t$ is the candidate activation. $\odot$ is an element-wise multiplication, and σ is the logistic sigmoid function. W_* and U_* are denoted as learned weight matrices.

C. System Architecture

The IDS is implemented as an application on the SDN controller. This paper focuses on the use of SDN paradigm as a network infrastructure for the IDS. The SDN-based IDS architecture is described in Fig. 2 with three main components: Flow Collector, Anomaly Detector and Anomaly Mitigator.

- **Flow Collector**: This module is triggered by a *packet-in* message or a timer function to aggregate all the flow statistics such as protocol, source and destination IP and source and destination port. All the aggregated features will be sent to the Anomaly Detector module.
- **Anomaly Detector**: We choose the GRU-DNN as the core of the Anomaly Detector module in this paper. This module loads a trained model, receives the network statistics and decides if a flow is an anomaly or not.
- **Anomaly Mitigator**: Through the Anomaly Detector's results, the Anomaly Mitigator module can make decisions on the flow (e.g., drop or forward the flow).

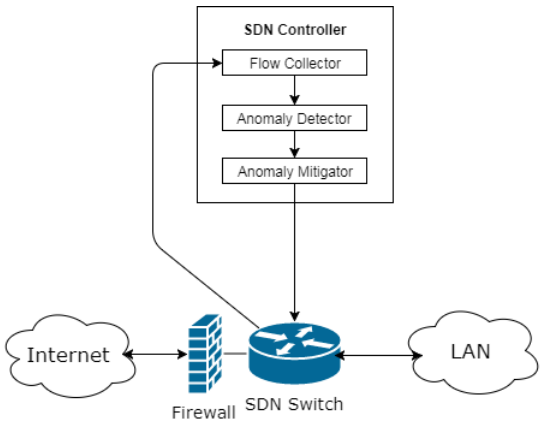

Fig. 2. SDN-based IDS Architecture

D. Dataset

The NSL-KDD dataset [23] is one of the state-of-the-art datasets for IDS evaluation. This dataset has 41 features which are categorised into three types of features: basic, content-based and traffic-based features. Our IDS is trained by the *KDDTrain+* dataset and tested by the *KDDTest+* dataset. In addition, the *KDDTest+* dataset contains 17 different types of attacks in addition to 22 attack types out of the *KDDTrain+* dataset. Thus, the *KDDTest+* dataset is a reliable indicator to the performance of the model on zero-day attacks as well.

Within the context of SDN, the packet content is not directly accessible in the current OpenFlow protocol. So we just focus on the basic and traffic-based features of the NSL-KDD dataset. In our research, a mixed feature set with six features is selected from these two feature set. These features are selected based on their SDN related nature without any feature selection algorithms. The selected features are <*duration, protocol_type, src_bytes, dst_bytes, srv_count, dst_host_same_src_port_rate*>. Details of these features can be seen in [23].

The NSL-KDD dataset contains both the numerical and symbolic features, consequently all the symbolic features are transformed into numerical values. After converting, the dataset is normalized into the range of [0-1] by Min-Max scaling. Its mathematical equation is given as:

$$x^{'} = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{6}$$

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

IV. Detection Performance Analysis

In this section, we firstly explain all the detection evaluation metrics. Secondly, we describe all the experiment setup. Finally, the results are given and compared with other works for a better overview.

A. Evaluation Metrics

For evaluation purpose, Precision (P), Recall (R), F-measure (F) and accuracy (ACC) metrics are used. These metrics are calculated by using four different measures, true positive (TP), true negative (TN), false positive (FP) and false negative (FN):

- 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.

Accuracy (AC): the percentage of true detection over total traffic records,

$$AC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{7}$$

Precision (P): the percentage of predicted anomalous instances predicted are actual anomalous instances,

$$P = \frac{TP}{TP + FP}. \tag{8}$$

Recall (R): the percentage of predicted anomalous instances versus all the anomalous instances presented,

$$R = \frac{TP}{TP + FN}. \tag{9}$$

F-measure (F): the harmonic of the precision and recall metrics to express the performance of the model,

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}}. \tag{10}$$

B. Experimental Setup

In our experiments, we use Keras [24] to implement our GRU-RNN, DNN, and VanilaRNN models. We use a Nadam optimizer [25] and a mean squared error (MSE) for the model. In addition, we added L_1-regularization to prevent over fitting during the training phase. The hyper-parameter configuration is 25, 10000 and 0.001 for the batch size, the epoch and the learning rate respectively. Scikit-learn library [26] is used to implement the SVM algorithm and measure all the evaluation metrics. The detail of our models can be seen in Table I.

TABLE I
NEURAL NETWORK MODEL STRUCTURE

Algorithm	Input Layer	Hidden Layer	Output Layer
GRU-RNN	6	6,4,2	1
DNN	6	6,4,2	1
VanilaRNN	6	4	1

C. Experimental Results

To start with, we present the detection performance of the GRU-RNN in terms of P, R, F and AC. We also compare the performance of the GRU-RNN with other algorithms like VanilaRNN, SVM and DNN using the same mixed feature set. Table II shows that the proposed GRU-RNN outperforms in all the evaluation metrics for all classes. The detection rate of the legitimate and anomaly traces is 89% and 90% respectively. The results also show that the GRU-RNN is good at detecting zero-day attacks with the anomaly detection AC of 90%. The GRU-RNN yields good results for all classes, while other algorithms just work well in only one class.

TABLE II
THE DETECTION PERFORMANE COMPARISON

Algorithm	Legitimate Class			Anomaly Class		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
VanilaRNN	43	90	58	57	10	17
SVM	71	32	44	64	90	75
DNN	67	89	76	88	66	76
GRU-DNN	**87**	**89**	**88**	**91**	**90**	**90**

As seen in Table III, our approach outperforms other approaches dealing with low-dimension and raw features. The DNN, coming in second place, shows the potential of the DL approach in anomaly detection. The VanilaRNN gives the worst result compared with its counterpart GRU-RNN.

TABLE III
ACCURACY COMPARISON WITH OTHER ALGORITHMS

Algorithm	Accuracy
VanilaRNN	44.39%
SVM	65.67%
DNN	75.9%
GRU-RNN (Proposed Model)	**89%**

In the following, the Receiver Operating Characteristic (ROC) curve is presented as a standard measure for classifier comparison. The ROC curve is created by plotting the False Positive Rate (FPR) versus the True Positive Rate (TPR). The area under the curve (AUC) is used to determine which classifier predicts the classes best. The higher the AUC, then the better is the classifier. Fig. 3 shows that the proposed GRU-RNN achieves the highest AUC amongst all the tested algorithms as expected. The TPR of the GRU-DNN is about 90% and the FPR is about 10%. It has higher TPR and lower FPR compared to other algorithms. As we can see, the GRU-RNN helps reduce the FP which is an important factor of the IDS. The VanilaRNN gives the worst performance as expected.

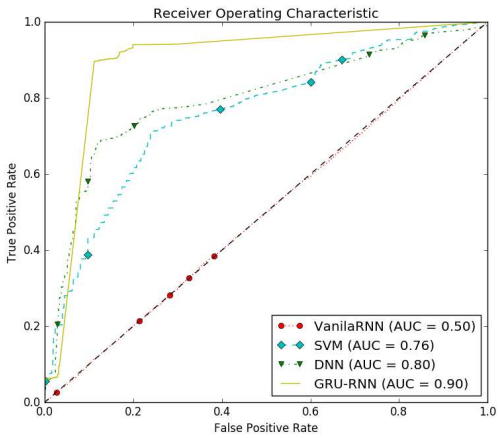

Fig. 3. ROC Curve Comparison for Different Algorithms

Furthermore, we also compare the performance of the proposed model with others in the literature. Our GRU-RNN is compared with SVM, DNN and NB Tree algorithms. Table IV contains the anomaly detection AC of the state-of-the-art results against our proposed model. The results show that our model outperforms all the previous methods. Our GRU-RNN performs better than the SVM and NB Tree algorithms that use a set of 41 features for training and testing. The GRU-RNN result also indicates a significant improvement in AC compared to the basic DNN in our previous work.

TABLE IV
ACCURACY COMPARISON WITH PREVIOUS STUDIES

Method	Accuracy
SVM [23]	69.52%
DNN [18]	75.75 %
NB Tree [23]	82.02%
GRU-RNN (Proposed Model)	**89%**

V. NETWORK PERFORMANCE ANALYSIS

In this section, we evaluate the effect of the GRU-RNN on the network performance. The evaluation testbed is described in the first part and then the network performance evaluation is presented.

A. Experimental Setup

The GRU-RNN is implemented as an application written in Python language in a POX [27] controller. Cbench [28] is a standard tool used for evaluating the SDN controller performance. Cbench runs in two modes: throughput and latency modes. In the throughput mode, it computes the maximum number of packets handled by the controller. In the latency mode, it computes the time needed to process a single flow by the controller.

We run our experiments on a virtual machine having an Intel Core i5-4460 3.2GHz with 3 cores available and 8GB of RAM. The operating system is Ubuntu 14.04 LTS-64bit.

The controller performance is tested with a different number of virtual OpenFlow switches emulated by Cbench. The performance of the stand-alone POX controller is considered as a baseline for our evaluation. We also compare the GRU-RNN with the DNN in our previous work [18].

B. Analysis of Results

Fig. 4 depicts the average response rate of the controller under three testing scenarios. As we can see, both the DNN and GRU-RNN cause the overhead on the controller. The DNN algorithm is simpler than the GRU-RNN, and so it gives a slightly better performance than that of the GRU-RNN. However, the GRU-RNN outperforms the DNN in terms of the detection accuracy. The affect of the GRU-RNN on the controller is predictable and unavoidable. The throughput decreases slightly when the network size increases from 32 to 64 switches. The network performance degrades by about 3.5% when the network size is under 32 switches. When we increase the size to over 64 switches, the throughput drops by about 4%.

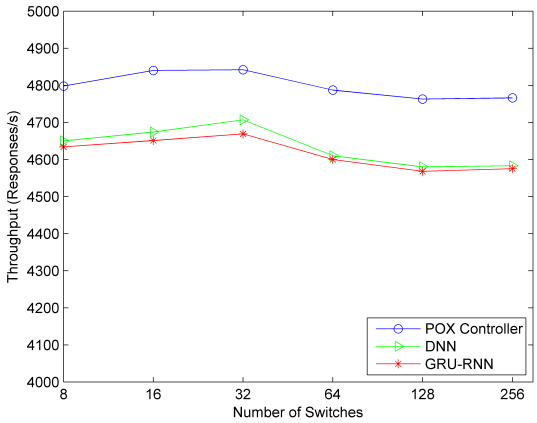

Fig. 4. Throughput Evaluation

As we can see in Fig. 5, the latency increases along with increasing the network size. When we increase the network

size, the load on the controller is increased as well and causes the overhead. The GRU-RNN still has the highest overhead amongst all.The overall degradation is about 7% in all cases.

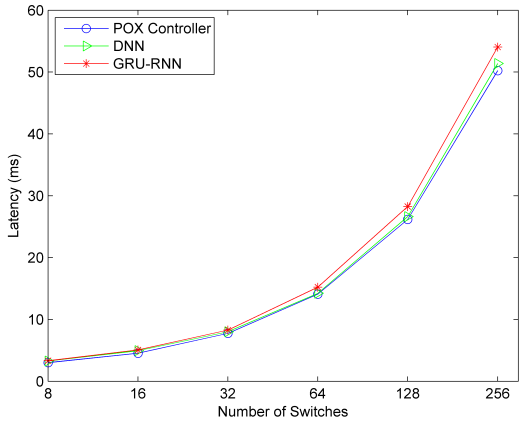

Fig. 5. Latency Evaluation

All in all, the overhead caused by the GRU-RNN on the POX controller is quite low, and so our proposed approach has significant potential for real-time anomaly detection in the SDN environments. From a network perspective this is a compromise between performance and security which all network administrators have to deal with.

VI. Conclusions and Future Work

In this paper, we present an Anomaly-based IDS in the SDN environment using the GRU-RNN algorithm. We show that the GRU-RNN outperforms other state-of-the-art algorithms with an accuracy of 89%. Our scheme uses a minimum number of features compared to other state-of-the-art approaches. This makes the model more computationally efficient for real time detection. In addition, the network performance evaluation showed that our proposed approach does not significant affect the controller performance. Therefore, it is practical for implementation under the context of SDN.

In the future, we will optimize our model and use other features to increase the accuracy. We will also try to implement our approach in a distributed manner to reduce the overhead on the controller.

References

[1] "Software Defined Networking Definition," Available: https://www.opennetworking.org/sdn-definition/, [Accessed 15 Sept. 2017].

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

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

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

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

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

[7] M. S. Parwez, D. Rawat, and M. Garuba, "Big data analytics for user activity analysis and user anomaly detection in mobile wireless network," *IEEE Transactions on Industrial Informatics*, 2017.

[8] L. Nie, D. Jiang, and Z. Lv, "Modeling network traffic for traffic matrix estimation and anomaly detection based on bayesian network in cloud computing networks," *Annals of Telecommunications*, vol. 72, no. 5-6, pp. 297–305, 2017.

[9] J.-h. Bang, Y.-J. Cho, and K. Kang, "Anomaly detection of network-initiated lte signaling traffic in wireless sensor and actuator networks based on a hidden semi-markov model," *Computers & Security*, vol. 65, pp. 108–120, 2017.

[10] S. T. Ikram and A. K. Cherukuri, "Improving accuracy of intrusion detection model using pca and optimized svm," *Journal of computing and information technology*, vol. 24, no. 2, pp. 133–148, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

[23] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009.

[24] F. Chollet, "keras," https://github.com/fchollet/keras, 2015.

[25] T. Dozat, "Incorporating nesterov momentum into adam." [Online]. Available: http://cs229.stanford.edu/proj2015/054_report.pdf

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] "POX," Available:https://github.com/noxrepo/pox.

[28] "Cbench," Available:https://github.com/mininet/oflops/tree/master/cbench.