Griffin: An Ensemble of AutoEncoders for Anomaly Traffic Detection in SDN

Liyan Yang[†*§], Yubo Song[†*§], Shang Gao[‡], Bin Xiao[‡], Aiqun Hu[†*§]
[†]*School of Cyber Science and Engineering, Southeast University*
[*]*Key Laboratory of Computer Network Techonology of Jiangsu Province*
[§]*Purple Mountain Laboratories*
[‡]*Department of Computing, The Hong Kong Polytechnic University*
yangliyan@stu.xidian.edu.cn, songyubo@seu.edu.cn, goldensaintgao@qq.com, csbxiao@polyu.edu.hk, aqhu@seu.edu.cn

***Abstract*—The Network Intrusion Detection Systems (NIDS) with machine learning in SDN become increasingly popular solutions. NIDS uses abnormal traffic detection to identify unknown network attacks. Most of today's abnormal traffic detection systems are supposed to continuously update the recognition model in time based on the features from newly collected packets to accurately identify unknown network attack behaviors. However, those existing solutions always require a large number of packets to train the recognition model offline. That means it is impossible to accurately detect the emergence of new cyber-attacks immediately. This paper proposes Griffin, a per-packet anomaly detection system that can dynamically update the training model based on neural networks. The Griffin is executed in SDN environment, utilizing a novel ensemble of autoencoders to collectively filter out abnormal traffic from normal traffic. Meanwhile, the autoencoders are updated based on the root mean square error to adjust the training model. The adjustment is done in an unsupervised manner, which needs no expert to label the network traffic or update the model from time to time. Our evaluations, with the open Datasets provided by Yisroel Mirsky, show that Griffin's time delay is around 0.1s and its accuracy is 98%. Moreover, we also compare Griffin with other four similar NIDSs and find that Griffin performs the best in terms of Matthews Correlation Coefficient and complexity.**

***Index Terms*—Software-defined Network, network intrusion detection system, anomaly detection, autoencoder, ensemble learning**

I. Introduction

Recently, the explosive growth of software-defined network (SDN) brings new ideas to design the Network Intrusion Detections Systems (NIDS). Intrusion detection is supposed to collect information from several key points in the computer network system and analyze the information to see if there are any violations of security policies and signs of attacks on the network. Intrusion detection techniques generally fall into one of two categories, namely misuse detection and anomaly detection. One of the drawbacks of misuse detection is its inability to detect novel, previously unseen attacks. Conversely, anomaly detection can detect new types of intrusions. It finds patterns in data that do not conform to expected behavior [1]. That means it can detect novel, previously unseen attacks.

In recent years, knowledge in the fields of machine learning is used for anomaly detection. Many papers detect attacks with machine learning, such as Naive Bayes (NB) [2], Support Vector Machine (SVM) [3],and unsupervised fuzzy C-means clustering [4]. Braga et. al analyze statistical information about DDoS, putting forward stream-based feature tuples [5]. After that, he uses Self-Organizing Map (SOM) to make normal traffic distinguish with anomaly traffic. However the SOM has slow convergence and needs long training time. Barki et. al. extract some features of traffic packets: response time, source address and destination address, using NB, KNN and K-means to classify the features to detect anomaly respectively [6]. But this paper analyzes different performances of different algorithms insufficiently and lacks a simulation environment for SDN. With the development and high popularity of deep learning, many researchers [7]–[9] try to use deep learning to detect attacks for higher accuracy. Granted, Tang et. al use a five-layer neural network to train the model, but due to poor feature selection, the accuracy rate is only 75% [8]. Niyaz et. al use autoencoders to classify the traffic [7]. However high accuracy of 95.65%, their algorithm is limited to detecting DDoS attack, having the same shortcoming with Z. Liu et. al [9]. The above-mentioned papers have shortcomings below: 1) training the model offline and can not detect a new type of network attack at the moment it appears; 2) using a supervision model and requiring a large number of labeled training datasets; 3) low accuracy; 4) high system time delay that cannot accommodate the requirements of online monitoring.

To make up for the shortcomings of the methods above, in this paper, we introduce Griffin: a per-packet anomaly detection system that can dynamically update the training model according to each packet in SDN, with online manner, unsupervised learning, and efficient inspection. Griffin is used to protect the Internet, just like the ancient Griffin guarding treasure in Greek mythology.

For the variability of network attack types, it is necessary to update the model in real-time according to the features of each packet. Therefore we propose the per-packet anomaly detection that can dynamically update the training model according to each packet in SDN. It is established with four parts: packet capture, feature extractor, feature mapper, and anomaly detector. To meet the requirement for detecting new types of attacks in an online manner, we propose a per-packet feature extraction window with a damping parameter in feature

extractor and real-time anomaly detector. To address the drawback of the supervision model, we utilize an ensemble of autoencoders in anomaly detector, without supervision and labeled training datasets. As for the shortcoming of the high system time delay of other schemes before, we use the Hierarchical Clustering Algorithm in feature mapper to reduce the dimension of the feature set. It reduces the complexity of Griffin and provides delay guarantees.

In summary, the contributions of this paper are as follows:

- We propose Griffin: a novel per-packet anomaly detection system in SDN. It can dynamically update the training model according to each packet.
- We propose online technology and resource detector in Griffin. Moreover, in Griffin, an ensemble autoencoder is built automatically in an unsupervised manner. This method has a slight time delay and implement online monitoring, without a large number of labeled training data sets.
- We perform the experience with the open Datasets provided by Yisroel Mirsky to demonstrate the detection efficiency of Griffin. From our experiments, we find that Griffin has a high accuracy of 98% and a slight system time delay of around 0.1s. Furthermore, Griffin performs the best in terms of Matthews Correlation Coefficient and complexity, compared with the other four similar methods.

The rest of the paper is organized as follows: Section II presents Griffin's framework and its entire machine learning pipeline. Section III presents experimental results in terms of detection performance and run-time performance. Finally, in section IV we present our conclusion.

II. Griffin

In this section, the implementation of Griffin is revealed by steps, and the key problem to be addressed by the system is also shown in detail.

A. The architecture of Griffin

The Griffin can be divided into 4 parts: packet capture, feature extractor, feature mapper and anomaly detector, which is displayed in Fig. 1. The figure illustrates that the switches in the data plane of SDN take responsibility to capture packets, meanwhile every switches are deployed feature extractor and feature mapper. Additionally, switches in data plane execute concurrently, which contribute to reducing the time required for feature extradition and the time delay of our NIDS greatly.

The network control technology of the controller in the control plane mainly includes strategy selection and flow table issues through the southbound interface protocol. For the sake of fulfilling the anomaly detection's potential, we deploy the anomaly detection on the control plane. As shown in Fig. 1, this mode utilizes the features after feature mapper to establish the autoencoders in the ensemble layer and output layer. To be specific, in the training period, a backpropagation algorithm is used to train the neural networks. In the execution period, the positive propagation algorithm

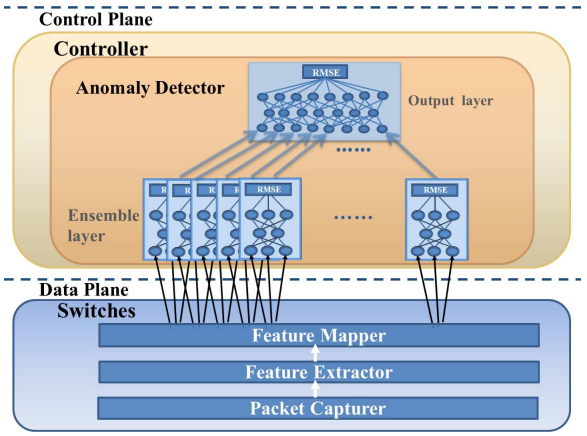

Fig. 1. Diagram of Griffin architecture

is used to execute neural networks and calculate the final root-mean-square error (RMSE) and score. We consider this score as a basis for network traffic anomaly detection. When the anomaly detection detects malicious traffic, the controller would message switches through the southbound interface to block this malicious traffic.

B. Packet Capture

According to the OpenFlow protocol, OpenFlow switches' table flows include three parts: domain information, calculator and action, among which domain information includes all information needed in the packet capture. Therefore, the nine original characteristics of the IP protocol version, source MAC address, destination MAC address, source IP address, destination IP address, source port number, destination port number, packet length, and time stamp can be obtained through layer-by-layer analysis of the data packets.

C. Feature Extractor

Given lacking memory space and eminent expansibility, we can't make statistical analysis to packet windows of every channel or calculate continuous statistical information in this packet windows. Therefore, a frame is created with an attenuation function to be the damping of windows to maintain increasing statistics. At first, we define a borderless data flow: $S = \{x_1, x_2, x_3, \ldots\}$, where $x_i \in R$, S can be uninterrupted feature set like time. When damping weight is 0, we let the former temporal incremental statistic be deleted, thus defining the attenuation function as [10]:

$$d_\lambda(t) = 2^{-\lambda t} \tag{1}$$

where $\lambda > 0$ is the attenuation coefficient and t denotes the duration time since observing S_i last time. Then we define a damping increment statistical tuple as: $TS_{i,\lambda} := (w, FS, SS, SK_{ij}, T_{last})$, where FS is the sum of features in feature set, SS is the sum of squares of features in feature set, w is the current weight and T_{last} denotes the time stamp when update the $TS_{i,\lambda}$ last time. Moving now on to consider how two temporal incremental statistics impact on result, we

define SK_{ij} as the sum of the remaining of flow i and flow j, which can be used to calculate two-dimensional statistical analysis, meanwhile, define attenuation coefficient as:

$$\gamma \leftarrow d_{\lambda}\left(t_{cur}-t_{last}\right) \tag{2}$$

In that case, the attenuation process can be updated as follows:

$$TS_{i,\lambda} \leftarrow (\gamma w, \gamma FS, \gamma SS, \gamma SK, T_{cur}) \tag{3}$$

$$TS_{i,\lambda} \leftarrow \left(w+1, FS+x_{cur}, SS+x_i^2, SK_{ij}+r_i r_j, T_{cur}\right) \tag{4}$$

To extract features, we are supposed to extract characteristics of the communication host and protocol when a packet arrived. To be specific, that information include: 1) Source MAC and IP address (denoted by Src-MAC-IP-Addr); 2) Source IP (denoted by Src-IP); 3) The channel between source and destination host (denoted by Channel-Src-Des); 4) TCP/UDP socket between source and destination host (denoted by TCP/UDP-Soc). According to those behavior information and statistical analysis indicators above, we could select 23 features displayed in Table. I, where $\mu_i = FS/w$, $\sigma_i = \sqrt{|SS/w-(FS/w)^2|}$, $\|S_i, S_j\| = \sqrt{\mu_{s_i}^2+\mu_{s_j}^2}$, $R_{S_i,S_j} = \sqrt{\left(\sigma_{S_i}^2\right)^2+\left(\sigma_{S_j}^2\right)^2}$, $Cov_{S_i,S_j} = \frac{SK_{ij}}{w_i+w_j}$ and $P_{S_i,S_j} = \frac{\mathrm{Cov}_{S_i,S_j}}{\sigma_{S_i}\sigma_{S_j}}$. After that one can extracts a set of features in tune with features in Table. I from a total of five time windows: 100ms, 500ms, 1.5s, 10s, and 1min, thus totaling 115 features.

D. Feature Mapper

In this part, $\overrightarrow{\mathrm{x}}$'s n features are mapped into k smaller sub-instances to reduce algorithm complexity. Let V denotes the ordered set of k sub-instances, where $V = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_k}\}$. For the purpose of efficient ensemble operation, we select a mapping: $f(\vec{x}) = V$ with conditions: guarantee every $\overrightarrow{v_i}$ has no more than m features.

Given that, we find the mapping f by hierarchically clustering the features of X into k groups which are no larger than m. Moreover, correlation is defined as the distance between two dimensions through the process of clustering. The correlation distance d_{cor} that is used to calculate the distance between vector u and v defined as:

$$d_{\mathrm{cor}}(u,v) = 1-\frac{(u-\bar{u})\cdot(v-\bar{v})}{\|(u-\bar{u})\|_2\|(v-\bar{v})\|_2} \tag{5}$$

where $\bar{u}$ is the mean of elements in vector u, $\bar{v}$ is the mean of elements in vector v, $(u-\bar{u})\cdot(v-\bar{v})$ is dot product. When it comes to the solution of correlation between features in hierarchical clustering, one can perform agglomerative hierarchal clustering on the correlation distance matrix between each two features of X (denoted by D) to find f. This algorithm starts from n clustered features and each one is expressed by D. Then, search for the two nearest points and connect their associated clusters. This search and join process is repeated until there is a large cluster containing all n points.

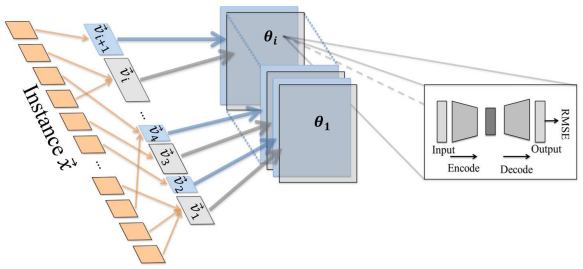

Fig. 2. An illustration of Griffin's anomaly detection algorithm.

E. Anomaly detector

This part is deployed in the control plane, which is comprised of two layers: ensemble larger and output larger. Ensemble larger is an k' ordered autoencoders, which are mapped into k' sub-instances in V. Output larger is an autoencoder, which learns the RMSE from the ensemble layer. Two modes of abnormal detector–namely training mode and monitoring mode will be shown below.

1) Training mode: When the anomaly detector receives the first set of mapped k' sub-instances of V from the feature mapper, it uses them as a blueprint initialization architecture. To be specific, let θ denotes a complete autoencoder, let $L^{(1)}$ and $L^{(2)}$ be the ensemble layer and output layer separately, where $L^{(1)}$ is the ordered set:$L^{(1)} = \{\theta_1, \theta_2, \ldots, \theta_{k'}\}$.

When it comes to the autoencoders $\theta_i \in L^{(1)}$ in ensemble layer, they have three-layer neural network, namely $\dim(\overrightarrow{v_i})$ neural network of input layer and output layer, also the $[\beta \cdot \dim(\overrightarrow{v_i})]$ neural network of hidden layer ($\beta \in (0,1]$). The Fig. 2 displays the mapping between $\overrightarrow{v_i} \in V$ and $\overrightarrow{\theta_i} \in L^{(1)}$.

We define $L^{(2)}$ as autoencoder θ_0, which has k' input and output neurons and $\lceil k' \cdot \beta \rceil$ internal neutrons. Its input comes from 0-1 normalized RMSE error signal of every autoencoder in $L^{(1)}$, reducing the complexity of the network. Meanwhile, we utilize the evenly distributed random value: $\mathcal{U}\left(\frac{-1}{\dim(\vec{v}_i)}, \frac{1}{\dim(\vec{v}_i)}\right)$ to initialize the weight of autoencoder θ_i.

For every normal packet, the neural network executes forward propagation firstly. We use the activation function called $sigmoid$: $f(\vec{x}) = \frac{1}{1+e^{\vec{x}}}$ and define the last output as: $\overrightarrow{y'} = a^{(L)}$. Let h be the forward propagation from input $\vec{v}_i$ to output $\overrightarrow{y'}$, which is defined as: $h_\theta(x_i) = \overrightarrow{y'}$.

The algorithm employed to train the neural network is a backpropagation algorithm based on Stochastic Gradient Descent (SGD). In this process, the RMSE between the actual value and the expected value would be retained. And the RMSE can be computed as: $\mathrm{RESE}(\vec{x}, \vec{y}) = \sqrt{\frac{\sum_{i=1}^{n}(x_i-y_i)^2}{n}}$, where n is the dimension of the input vector. In this paper, the largest iteration number is 1. In that case, we only learn once from each instance to hold the online algorithm. In summary, we will train the model by the steps following:

- Use normal instance x_i in X to train autocoder.
- Execute: $\mathbf{s} = \mathrm{RMSE}(x_i, h_\theta(x_i))$.

TABLE I
FEATURES SELECTED

The packet's...	Statistics	Characteristic	Features
Size	μ_i, σ_i	Src-MAC-IP-Addr, Src-IP, Channel-Src-Des,TCP/UDP-Soc	8
Magnitude	$\|S_i, S_j\|$, R_{S_i,S_j}, Cov_{S_i,S_j}, P_{S_i,S_j}	Channel-Src-Des, TCP/UDP-Soc	8
Count	w_i	Src-MAC-IP-Addr, Src-IP, Channel-Src-Des,TCP/UDP-Soc	4
Jitter	w_i, μ_i, σ_i	Channel-Src-Des	3

- Update: if $(s \geq \varnothing)$ then $\varnothing \leftarrow s$ (let $\varnothing$ be threshold, whose initial value is -1).
- Update the θ through learning the features of x_i.

2) Monitoring mode: Generally, the autoencoder trained in X can recreate an instance with the same data distribution as X. If instance is inconformity to the features learned from X, the RMSE would be high between recreated features and original features. In that case, we can through executing the steps following:

- Execute: $s = RMSE\,(\vec{x}, h_\theta(\vec{x}))$.
- Judgment: if $(s \geq \varnothing\beta)$ then alert.

In the monitoring mode, the autoencoder does not update any internal parameters. Instead, it performs forward propagation through the entire network and returns the RMSE reconstruction error of $L^{(2)}$. The output of Griffin is the RMSE anomaly score $s \in [0, \infty)$. The larger the score s, the greater the anomaly.

III. EVALUATION

In this section, we provide an evaluation of Griffin in terms of its detection and runtime performance. We open by researching impact of key position to prediction accuracy, followed by feature selection, and finally, it shows the time delay, accuracy, CPU usage of our system as well as the comparison with other similar systems to show the superiority of our scheme.

A. Experimental Environment

Griffin requires operating in SDN environment, and this paper adopts Mininet platform to simulate the SDN environment. As for the hardware condition, we use the core of Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz and the memory of 8G.

B. Datasets

The datasets used in the evaluation is the open Datasets provided by Yisroel Mirsky [10]. Details of the datasets are shown in Table. II.

C. Impact of Key Position to Prediction Accuracy

Based on the autoencoder used in Griffin mentioned in the previous chapter, it is obvious that the maximum count of features *MaxFE* included in every sub-instance in feature mapper would impact on detection accuracy and time taken by the Griffin to process all sample traffic packets *TimePa*. The Fig. 3 shows the link between *TimePa* and *FalsePa* with the change of *MaxFE*.

TABLE II
THE DATASET USED TO EVALUATE GRIFFIN

Attack Type	Attack Name	Vector	Packets
Recon	Fuzzing	3	2244139
Man in the Middle	Active Wiretap	2	4554925
Denial of Service	SSDP Flood	1	4077266
	SYN DDoS	1	2771276
	SSL Renegotiation	1	6084492
Botnet Malware	Mirai	X	764137

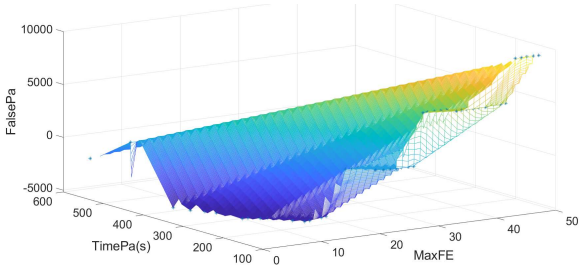

Fig. 3. The affect Griffin's *MaxFE* has on *FalsePa* and *TimePa*

Obviously, there is a clear trend of decreasing of *TimePa* with the increasing number of *MaxFE*. Moreover, when *MaxFE* $\geq$ 23 the *TimePa* reaches the bottom and changes tinily. Also, it is displayed from this figure that when *MaxFE* $\in [4, 19]$, the *FalsePa* is a negative value, which indicates many missing reports. What's more, when *MaxFE* $\in [20, 29]$, the absolute value of *FalsePa* reaches the bottom, therefore the accuracy reaches the top. Additionally, when *MaxFE* $\in [30, 49]$, the *FalsePa* is positive value indicating many false-negative instance.

In summary, for the informants in this study, let *MaxFE* $\in [23, 25]$ to make the *FalsePa* and the *TimePa* reach the bottom at the same time. In this paper, let *MaxFE* $= 25$ and count of sub-instance is 19.

To improve the accuracy of Griffin and reduce the time delay of the system, we are supposed to make importance evaluation to divided 19 sub-instances based on Random Forest and select the sub-instances. Sub-instance selection is a work, on which we rank all sub-instances according to their feature's importance. One can utilize a series of rules to get the relative relationship of the importance of sub-instances and we use the random forest. To be precise, when constructing a decision tree, get node segmentation effect metric by using the Gini Index and calculating the Gini Importance of feature to indicate the importance of the feature. And the Gini Index

can be defined as:

$$\text{Gini(p)} = \sum_{q=1}^{Q} p_q \left(1 - p_q\right) = 1 - \sum_{q=1}^{Q} p_q^2 \tag{6}$$

where q denotes q-th class and p_q denotes the sample weight of q-th class. In that case, the importance of feature X_j in node m, namely the variable quantity of Gini Index before and after branching the node m, denoted as $VIM_{jm}^{(Gini)}$, can be defined as:

$$VIM_{jm}^{(Gini)} = GI_m - GI_l - GI_r \tag{7}$$

where GI_l and GI_r denote Gini Index of new nodes after branching. If the node of feature X_j in decision tree i is in set M, then the importance of X_j in i-th tree $VIM_{ij}^{(Gini)}$ is defined as:

$$VIM_{ij}^{(Gini)} = \sum_{m \in M} VIM_{jm}^{(Gini)} \tag{8}$$

We make a assumption that there are n trees totally, then the sum of them, denoted as $VIM_{j}^{(Gini)}$, is defined as:

$$VIM_{j}^{(Gini)} = \sum_{i=1}^{n} VIM_{ij}^{(Gini)} \tag{9}$$

Finally, one can make a normalization process to importance scores as following:

$$VIM_j = \frac{VIM_j}{\sum_{i=1}^{c} VIM_i} \tag{10}$$

where the denominator is the sum of all feature gains, and the numerator is the Gini index of feature j. Utilizing the measure to evaluate the importance of every sub-instance, we get the Fig. 4. Obviously, the importance of sub-instance 1 and sub-instance 5 is little. Therefore we can reject them. After that, 17 autoencoders should be deployed in the ensemble layer in anomaly detector.

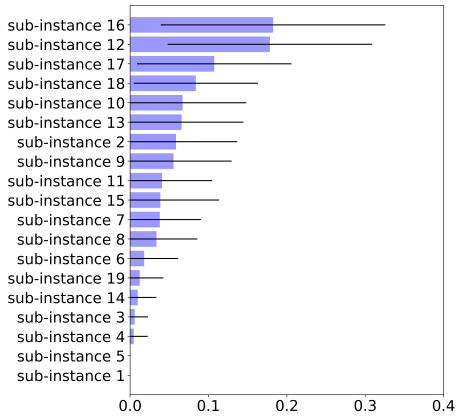

Fig. 4. Feature selection based on random forest

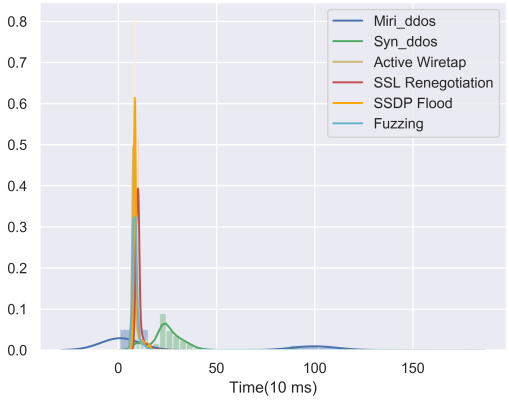

Fig. 5. Time delay of system with multiple test datasets

D. Evaluation Metrics

Move on to the system time-delay, namely the time it takes for the system to detect an attack from the moment of attack. Let t_{delay} be the system time-delay, $t_{feature}$ denotes the time it takes for the feature extractor and the feature mapper in switches of the data plane. What's more, t_{test} and $t_{message}$ denote the time taken by the anomaly detector and messaging to switches respectively. In that condition, the t_{delay} can be computed as: $t_{\text{delay}} = t_{\text{feature}} + t_{\text{test}} + t_{\text{message}}$. We utilize several attack datasets provided by Yisroel Mirsky to process the system time-delay test. Meanwhile we use Fig. 5, which is a destiny plot to indicate the result. It is apparent from this figure that the peaks of different datasets appear at the same point of 0.1s, which means the system's time-delay for all attack datasets participating in the test is concentrated at 0.1s. Further analysis shows that the system time-delay does not exceed 1.5s. These results indicate that the Griffin is blessed with high system stability and a short time-delay.

Moving on now to consider the accuracy of the Griffin. We utilize the cross-verified iterator to calculate the ROC curve (Receiver Operating Characteristic curve) of Griffin. The result is presented in the Fig. 6, where the abscissa indicates the false positive rate, the ordinate indicates the true positive rate. Additionally, AUC denotes the area enclosed by the curve, which is used to measure the manifestations of the ROC curve, namely larger AUC means better system performance. As shown in the Fig. 6, in each iteration, the AUC is close to 1, which expresses the good performance of the system.

Let us now consider the complexity of Griffin. CPU usage is introduced as an indicator to evaluate the complexity of Griffin. To highlight the low complexity of the algorithm used in Griffin, it is necessary to compare it with two other algorithms: network intrusion detection system based on artificial neural network and support vector machine (ANN-SVM), and network intrusion detection system based on artificial neural network and random forest (ANN-RF). It is reasonable to compare Griffin with those two NIDS, because the architecture of these two NIDS is similar to that of the Griffin – all using the ensemble learning. After that we get

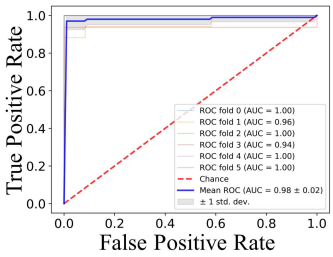

Fig. 6. The ROC curve of Griffin.

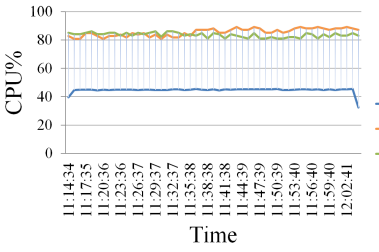

Fig. 7. Compare the Griffin's CPU usage with ANN-SVM NIDS and ANN-RF NIDS.

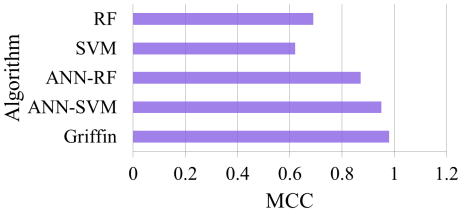

Fig. 8. Comparison results of the considered methods in terms of Mathews correlation coefficient.

the result shown in Fig. 7. Obviously, CPU usage of Griffin is around 45%, however CPU usage of other two systems is around 85%. Therefore, one can conclude that the complexity of Griffin is low.

For the sake of evaluating detection effect of Griffin, we compare its effect with other four models: ANN-RF, ANN-SVM, SVM and RF. Research shows that the Matthews correlation coefficient (MCC) is one of the best performance indicators for the classification problem of machine learning. Therefore one can evaluate the accuracy of the test by MCC between the prediction and practical results. MCC is a value between -1 and +1. The coefficient +1 means perfect prediction, 0 means no better than random prediction, and -1 means completely inconsistency between prediction and practical results. Obviously, the higher the MCC, the more accurate the prediction results are, where the MCC is defined as:

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{11}$$

where TP is the count of true positive instance, TN is the count of true negative instance, FP is the false positive instance and FN is the false negative instance. If any of the four sums in the denominator is zero, the denominator will be set to 1.

From the Fig. 8 above we can notice that the Griffin gains the top of MCC reaching 0.982, ANN-SVN and ANN-RF follow it reaches 0.95 and 0.87 respectively, however SVM and RF get relatively lower MCC are 0.62 and 0.69 respectively. Armed with this research, one can conclude that Griffin has higher accuracy and better detection effect.

IV. Conclusion

Griffin is an NIDS based on SDN and neural network, which has been designed to be efficient. It accomplishes anomaly detection in a novel per-packet detection way updating the detection model dynamically, effectively monitoring the behavior of all network connections, and utilizing ensemble learning with autocoders. In this paper, an online novel per-packet anomaly detection is first provided based on the SDN and its performance is investigated in terms of detection and runtime. Griffin, an online algorithm, is blessed with the accuracy of 98% and slight time delay – around 0.1s. Evaluation show that the proposed scheme outperforms other algorithms in terms of complexity and MCC.

Acknowledgment

This work is supported in part by National Key R&D Program of China under Grant Nos. 2018YFB2202200, 2018YFB2100403. Yubo Song is the corresponding author. This work is partially supported by Frontiers Science Center for Mobile Information Communication and Security, Southeast Univeristy, Nanjing, China.

References

[1] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the 2003 SIAM international conference on data mining*. SIAM, 2003, pp. 25–36.

[2] M. Zekri, S. El Kafhali, N. Aboutabit, and Y. Saadi, "Ddos attack detection using machine learning techniques in cloud computing environments," in *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*. IEEE, 2017, pp. 1–7.

[3] J. Liu, Y. Lai, and S. Zhang, "Fl-guard: A detection and defense system for ddos attack in sdn," in *Proceedings of the 2017 international conference on cryptography, security and privacy*, 2017, pp. 107–111.

[4] W. Shang, J. Cui, C. Song, J. Zhao, and P. Zeng, "Research on industrial control anomaly detection based on fcm and svm," in *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing And Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 218–222.

[5] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*. IEEE, 2010, pp. 408–415.

[6] L. Barki, A. Shidling, N. Meti, D. Narayan, and M. M. Mulla, "Detection of distributed denial of service attacks in software defined networks," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 2576–2581.

[7] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2017, pp. 21–26.

[8] T. A. 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)*. IEEE, 2016, pp. 258–263.

[9] Z. Liu, Y. He, W. Wang, and B. Zhang, "Ddos attack detection scheme based on entropy and pso-bp neural network in sdn," *China Communications*, vol. 16, no. 7, pp. 144–155, 2019.

[10] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.