JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 483-496 (2021)
DOI: 10.6688/JISE.202103_37(2).0013

Employing On-Line Training in SDN Intrusion Detection

PO-JEN CHUANG AND KUAN-LIN WU
Department of Electrical and Computer Engineering
Tamkang University
Tamsui, New Taipei City, 25137 Taiwan
E-mail: pjchuang@ee.tku.edu.tw

In SDN anomaly detection systems, when a training mechanism adopts semi-supervised learning (consisting of self-training and self-learning) to attain the classifiers of on-line training, it may cause the accumulation of identification errors – to degrade the performance. This paper presents a new training and learning mechanism which involves the operations of self-training and active learning to solve the problem. The proposed mechanism first adds samples with "high confidence weights" and classified as "malicious" to the training set by random selection. It then practices active learning to label those samples with "low confidence weights" and add them to the training set for training, to further lift up identification accuracy. A faster clustering method has been brought in to reduce the operation time of active learning. In classifier retraining, parallel training is applied to keep the classifier in constant service without interruption. Simulation results show that, in contrast to existing active learning IDS (ALIDS), our new mechanism performs better in identifying unknown attacks, without occupying the operation time of detection as it processes both training and detection in parallel.

Keywords: software defined networks (SDNs), intrusion detection system (IDS), machine learning, anomaly detection, on-line training, network security

1. INTRODUCTION

As software-defined networks (SDN) [1, 2] are susceptible to most of the attacks in traditional networks, it is essential to build proper intrusion detection systems (IDS) into the SDN structure to provide effective intrusion countermeasures. SDN anomaly detection systems used to involve *supervised learning* to train the classifiers. In application, IDS must identify unknown attacks through the old attack mode given that there is no tag for the received data. It is hence infeasible to apply the training mechanism of *supervised learning* to practical SDN anomaly detection systems because *supervised learning* uses only labeled samples for training – it cannot use on-line data for retraining, *i.e.*, it cannot apply on-line data to adapt the system to new types of attacks.

To solve the problem, some researchers introduce *semi-supervised learning* [3-7], a revised training method, to attain the classifiers of on-line training in anomaly detection systems. *Semi-supervised learning*, as the name carries, is an adaptive method between supervised learning and unsupervised learning, consisting mainly of *self-learning* and *self-training*. The concept of *self-training* is to conduct supervised learning practice in each round of training, add the samples with the best classification results from the previous round into the current sample set, and train itself again with the results generated by itself. There exists one major problem with the *self-training* practice: It will lead to the accumulation of errors. *Self-learning* attempts to improve the situation. The key concept of *self-*

Received March 12, 2020; revised June 16, 2020; accepted August 5, 2020.
Communicated by Fu-Hau Hsu.

learning is to increase the number of malicious samples in the training set by repeatedly identifying the unlabeled sample set. In exercise, we can add malicious unlabeled samples into the new training set to train the new classifier and to classify the data set until the new classifier does not recognize the attack sample.

Besides *self-training* and *self-learning*, we observe a more recent introduction of *active learning* (AL) in [6]. The new study applies *active learning* into IDS (to become ALIDS) and proves that such a practice can reduce the workload for network security analysts and is adaptive to data changes – *i.e.*, able to solve the problems related to data tampering and avoidance.

To pursue more satisfactory intrusion countermeasures for the SDN structure, we propose in this paper a new training and learning mechanism which involves both the operations of *self-training* and *active learning*. Our proposed mechanism operates as follows. It first adds samples with "high confidence weights" and classified as "malicious" to the training set by random selection and then puts the training practice of *active learning* into work. By *active learning*, it can label those samples with low confidence weights and add them to the training set for training, to pursue higher accuracy for the classifier. To speed up the training mechanism, we use a faster clustering method to reduce the operation time of the *active learning* practice. We also adopt *parallel training* in classifier retraining to help keep the classifier in service and replace it in an uninterrupted way. By doing so, we can immediately defend an attack when it is detected.

Extensive simulation runs are carried out to evaluate and compare the performance of our new training mechanism and existing ALIDS. As the results exhibit, our mechanism is able to identify unknown attacks in a more effective way than ALIDS. Even if an attack is not identified in the very beginning, we can constantly update the classifier according to the obtained training results, to achieve real-time attack detection – *i.e.*, to learn how to identify unknown attacks and successfully prevent them. Our training mechanism is also favorable in terms of time cost. It does not occupy the operation time of the detection system because both training and detection are processed in parallel. That is, the classifier will not stop functioning when it is getting replaced.

2. BACKGROUND STUDY

2.1 Software Defined Networks (SDN)

SDN is a centralized new generation network architecture proposed by Stanford University in 2008 [8]. The introduction of SDN is of special significance in that it makes possible to transfer the control of network policy from the supplier to the user. The new architecture is distinctive from traditional networks in being able to replace the "closed" software on a current network switch and allow the user to control the entire network through controller software execution. An SDN is composed of separated control and data layers, *i.e.*, its control layer is separate from the data layer. The data platform is responsible for transmitting data only, whereas the control platform is in charge of all decision-making. Such features may help a controller with correct logic to make early detection on suspicious traffic and, as a result, obtain faster responses in SDN-enabled switches. More specifically, when an SDN is assaulted, switches can instantly start the procedure of dynamic traffic management and attack blocking, to limit the attack traffic and isolate the attacker.

An SDN needs an appropriate agreement, such as the OpenFlow (OF) protocol [9], to contact the control layer and the data layer. The OF protocol includes the OF switch and OF controller. It is the first agreement implemented in accordance with the features of SDN and is used to maintain communication between both the OF switch and controller. The OF switch will first establish a connection with the OF controller through the transport layer security and the transmission control protocol; the controller then communicates with the switch by the OF protocol. An OF switch contains one or more flow tables and group tables, as Fig. 1 shows. Each of the tables holds multiple flow entries (rules) to perform packet routing. The elements in a flow entry include match fields, priority, counters, instructions and others (the interpretation of each element is shown in Table 1) [9]. The routing table on the switch gives the OF rule in the flow table, and the switch will perform the specified action based on the flow entry in the flow table.

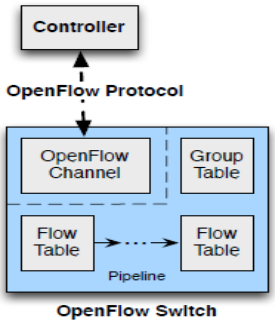

Fig. 1. The OpenFlow Switch architecture.

Table 1. The flow entry description.

Match fields	matching rules of the flow entry, consisting of the ingress port, packet headers, and optional metadata specified by the previous table
Priority	matching precedence of the flow entry
Counters	counts of the matching packets
Instructions	used to modify the action set or pipeline processing
Timeouts	maximum amount of time or idle time before the flow entry expires
Cookie	flow entry identifier specified by the controller
Flags	used to alter the way flow entries are managed

As mentioned, an SDN is vulnerable to most of the attacks in traditional networks. To monitor the network security of SDN, we must bring in proper IDS to the structure to work out desirable intrusion countermeasures. An IDS is a network security device which can be in the form of hardware or software. By monitoring the transmission of packets or the behavior of systems in the network (including system logs in the host, network traffic or traffic flow), an IDS is able to detect suspectful intrusion. When detecting an intrusion activity or suspectful intrusion, it will issue a notification to ensure the integrity, privacy and availability of the system in the network. To fully prevent the risk of network intrusion, more recent IDS tend to scan and detect all packets transmitted in the network, and then involve appropriate machine learning algorithms to attain more accurate traffic classification: intrusion or non-intrusion.

2.2 Machine Learning

Machine learning mechanisms may work by supervised or unsupervised learning. A *supervised learning* mechanism will predict the output values of input samples according to the input-output mapping in the training data of existing labels. *Unsupervised learning*, on the other hand, will cluster data with no labels and use only the eigenvalues of the data for prediction. *Semi-supervised learning* [3-7] is an adaptive method between *supervised learning* and *unsupervised learning*. It covers a number of machine learning algorithms, including the *self-training* and *self-learning* algorithms. The basic concept of *self-training*, as we have specified, is to apply the supervised learning practice in each round of training, add the samples with the best classification results from the previous round into the current sample set, and then train again with the self-generated results. It should be noted that such a *self-training* practice is likely to build up identification errors. As an extended deviation of *self-training*, the *self-learning* algorithm will identify unlabeled sample sets repeatedly so as to build up malicious samples in the training set. It can practically add malicious unlabeled samples into the new training set to train the new classifier and then classify the data set until the new classifier does not recognize the attack sample [5].

Active learning (AL) is a more recently introduced machine learning algorithm. It has been applied to IDS to form the active learning intrusion detection system (ALIDS) in [6]. Similar to self-training, from the results of identification, ALIDS conducts binary classification of data according to data confidence. Based on the value of data confidence, it first classifies data into high/low confidence data and then adds data with high confidence into the training set. ALIDS does not discard data with low confidence. It instead handles them by *active learning* to ensure more correct labels. To process the data with low confidence, ALIDS employs K-means [10] in AL to do clustering first. After clustering, the expert or the so-called oracle classifier in AL will take over to label the data in each cluster and add the processed low confidence data into the training set. It then trains the classifier with the new training set, to replace the current classifier. In this way, ALIDS actually realizes a better on-line training method – with certain drawbacks. For instance, by adding all data samples with high confidence into the training set, it may generate a fast-growing training set and also extremely unbalanced classes of samples in the training set, which in turn may incur unfavorable consequences. Possible consequences include that the trained classifier may yield unsatisfactory accuracy for low proportional classes of samples and also the probability of accumulated identification errors may hike up.

3. OUR PROPOSED LEARNING AND TRAINING

A traditional data training mechanism trains the classifiers in advance, solely by the adopted training set. An on-line training mechanism is relatively more sophisticated as, during the system functioning time, it can progressively enhance the classifier's ability to identify unknown attacks. In this investigation, we intend to pursue more satisfying on-line training by means of proper *semi-supervised* learning. As noted above, a *semi-supervised* learning approach which is helpful in attaining the classifiers of on-line training tends to produce accumulated identification errors. How to reduce the probability of accumulating identification errors turns out to be a major challenge for related studies. We also observe from existing *semi-supervised* learning practices that adding suitable data samples to

the on-line training set can be a feasible way to reduce the buildup of identification errors. Based on these findings, we decide to set up a new on-line training and learning mechanism which can work better than previous mechanisms, especially in avoiding potential defects, to realize more competent SDN intrusion detection.

Fig. 2 gives our SDN intrusion detection architecture which aims to enhance anomaly detection systems in SDN, *i.e.*, to engage on-line training and handle abnormal traffic in a more effective way. The proposed architecture contains three parts: **data feature detection**, **the learning mechanism** and **abnormal result processing**. In the figure, we can shift the on-line training job (the right portion of the flow chart) from the controller to another server, such as a flow analyzer, so as to maintain the controller's required real-time quick responses to flow control.

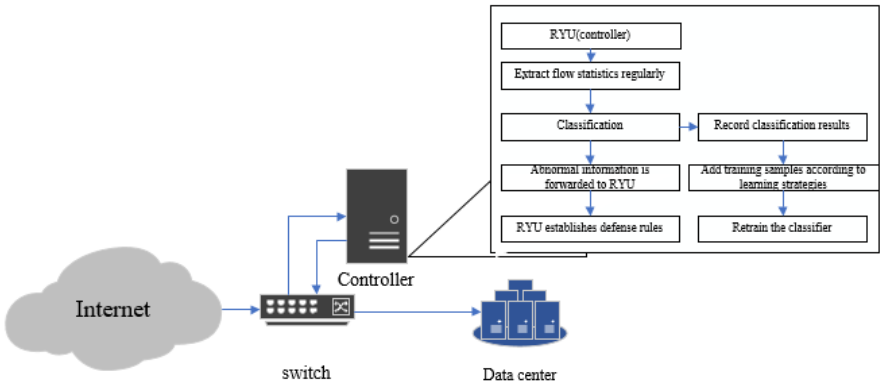

Fig. 2. Our SDN intrusion detection architecture.

In **data feature detection**, we use flow entry statistics to determine if the flow is abnormal or not – according to the corresponding rule. In practice, the controller will file requests to the switch for statistics once per second. **The learning mechanism** will process and analyze the recorded results when they are accumulated to a certain pre-set amount. It then adds the analyzed results to the training set to re-train the classifier. In **abnormal result processing**, the controller will be notified of the abnormal flow in the classified results after statistical information classification. It then adds the flow entries with defense rules into the flow table according to the corresponding attack types in the flow.

The learning mechanism will take the hybrid of *self-training* and *active learning* as the algorithm infrastructure. Fig. 3 gives the flowchart. It shows that we use labeled training samples to train the basic classifier and use this classifier to identify the actual flow data, *i.e.*, the unlabeled data. From identification results, we then set binary classification of data according to data confidence, to attain high and low confidence data. Note that we take the value of the highest classification category probability in the classifier as the value of confidence. A data with a value lower than 0.95 will be considered a low confidence data; otherwise, it is a high confidence data. The probability calculation for a decision tree is to divide (the number of samples of the same category in leaf nodes) by (the number of all samples in leaf nodes) to obtain the sample classification category probability in the tree. As for a random forest, it is to divide (the classification category probability of all trees) by (the number of trees) as the final classification category probability.

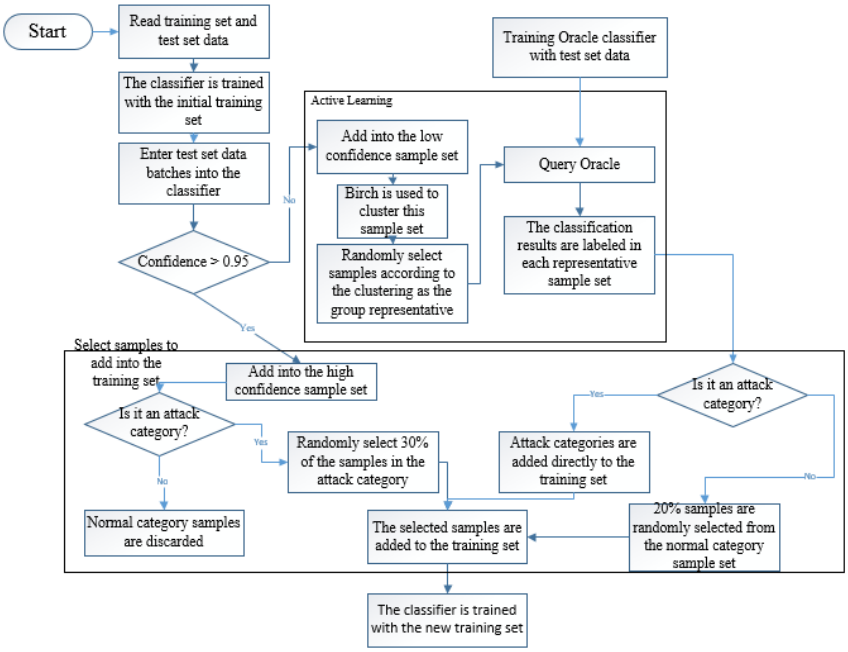

Fig. 3. The flowchart of our learning mechanism.

We move on to process low confidence data by *active learning* and add both high and low confidence data into the training set after sample selection, to train the classifier with the new training set and replace the classifier currently in use. Adopting *active learning* to process data with low confidence will help to attain more correct labels. In *active learning*, we employ the balanced iterative reducing and clustering using hierarchies (BIRCH) clustering algorithm [11], instead of K-means [10], because the BIRCH algorithm is faster without affecting the accuracy of clustering. To be more specific, BIRCH can achieve a good clustering effect by only one scanning without having to set the number of clusters in advance like K-means. That is, BIRCH can generate clustering feature trees by which we can save data compression to certain extent. Note that the expert in *active learning* will be taken over by the oracle classifier trained and learned from the test set data, *i.e.*, the accuracy of the oracle classifier will represent the ability of the expert. The classification results are then labeled to the samples of each cluster.

To preserve the identification ability of known attacks, we randomly select 30% of high confidence attack samples and add them into the training set. (Note that we do not include high confidence normal samples because they tend to produce *noise*). We nevertheless include all low confidence attack samples into the training set to help the re-trained model identify new unknown attacks more effectively. In practice, adding low confidence normal samples can further lift the normal category identification ability of the re-trained model, but, to avoid possible *noise*, we select only 20% of such samples into the training set.

We use the match fields in the flow entries to set up new forwarding rules which can help us better distinguish attack traffic from normal traffic. When detecting an abnormal

message, the IDS will send the anomaly classification result to the controller so that it can build the defense rule and block the attack traffic. After building the defense rule based on the received anomaly classification result, the controller will set the same match field parameter of the flow entry (for the attack) in a new flow entry and give it a higher priority to replace the original forwarding (by discarding) in order to block the attack traffic. Such a practice is favorable as it can effectively block attack traffic and meanwhile maintain ordinary network services. Fig. 4 displays the detailed flowchart of the proposed defense mechanism.

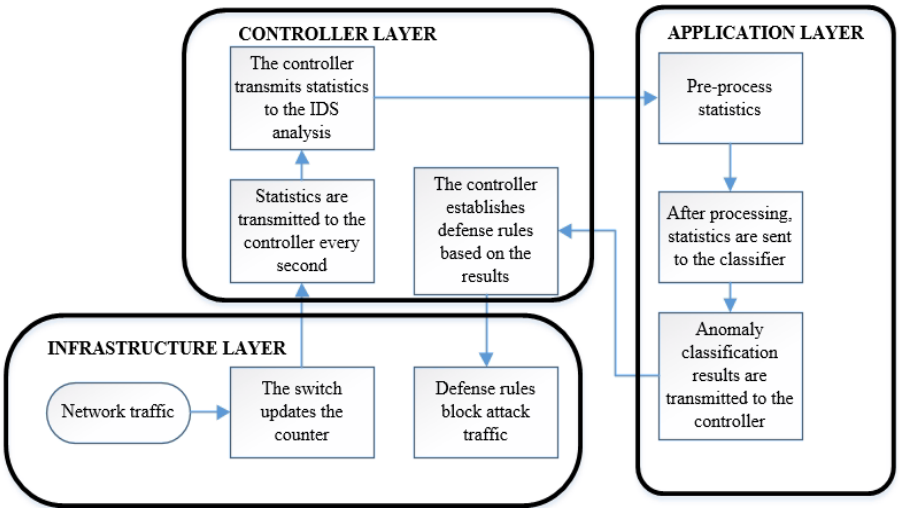

Fig. 4. The flowchart of the proposed defense mechanism.

4. PERFORMANCE EVALUTION

Extended simulation has been conducted to evaluate and compare the performance of ALIDS [6] and our new training mechanism. In the simulation, we use Python to write the classifier and learning programs under the execution environment of the Intel Xeon E3 v3 @3.40GHz CPU, 32GB Memory, and WINDOWS 10 operating system. Fig. 5 exhibits the architecture of our experimental simulation. We involve various classifiers in the simulation (along with their performance comparison) and use the NSL-KDD dataset [12] to train and test the classifiers. Six classification algorithms are employed, including the decision tree (DT) [13], k-nearest neighbor (KNN) [14], random forest (RF) [15], random tree (RT), bagged trees (BT) [16] and deep neural networks (DNN)) [17].

The adopted data sets vary with environmental assumptions. We use KDDTest+ as the test set, dividing it into a total of 226 100-sample test subsets which are input into the classifier, one by one, for identification. The obtained results will be stored in and processed by the trainer. Each time when a classifier finishes identifying a test subset, we set a one second delay to keep the classifier from identifying the 226 test subsets too rapidly (so as to lift the training update times of the classifier during the test). The training times vary for different classification algorithms, falling between 10 and 30.

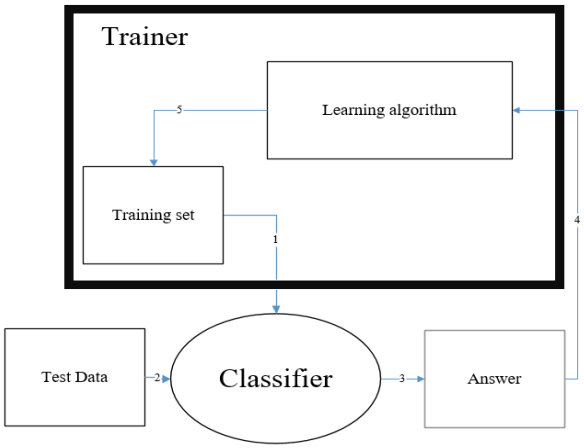

Fig. 5. Our experimental architecture.

Considering that the NSL-KDD dataset is a general intrusion detection dataset, not specially built for the SDN, we then select a subset of its features, as Table 2 lists, all of which can be more easily recorded from the SDN controller. We then test the traffic information closest to the SDN, to demonstrate the fact that our training and learning mechanism obtains higher accuracy and lower time cost than existing ALIDS. All classification algorithms adopt the subsets of the NSL-KDD dataset (with KDDTrain+_20Percent being the basic training set), except DNN which adopts the complete training set KDDTrain+ because it requires a large amount of data for training.

Table 2. The selected SDN features for the NSL-KDD dataset.

Feature Name	Description
duration	length (number of seconds) of the connection
service	network service on the destination, such as HTTP, telnet, ssh, *etc.*
protocol_type	type of the protocol, *e.g.* tcp, udp, *etc.*
src_bytes	number of data bytes from source to destination
dst_bytes	number of data bytes from destination to source
count	number of connections to the same host as the current connection in the past two seconds
srv_count	number of connections to the same service as the current connection in the past two seconds

The number of training samples tends to affect the accuracy of a classifier. Theoretically speaking, more training samples will result in better classification accuracy. The contents of training samples are also influential. For both SDN and general network environments with the same number of samples, the SDN environment usually turns over better classification accuracy mainly because of the difference in selected features. When operating training and learning by a small number of features, we are more likely to generate noise and be influenced by it. To reduce such influence, we must remove noise as much as possible to maintain the correct performance of training samples. Take ALIDS as an

example. ALIDS has a learning mechanism which unconditionally joins samples into the training set. It may hence add noise into the training set during the training and learning process, causing the trained model unable to enhance its classification ability or, even worse, to show signs of decline. In contrast, our mechanism adds only selected samples to the training set, which enables it to reduce the probability of generating noise and hence improves the performance of the classification model even with fewer sample features.

	DT	RF	RT	BT	KNN	DNN
ALIDS trainset quantity	44992	45784	45289	39151	44200	130626
Proposed trainset quantity	30036	29671	31407	33154	28807	126806
ALIDS training time	0.225388947	0.239379941	0.189122498	0.544905517	0.841268003	14.87582721
Proposed training time	0.155405619	0.172320869	0.14136064	0.399740016	0.662806598	7.526128576

Fig. 6. The training time and final training set size.

Fig. 6 displays the training time and final training set size for both ALIDS and our proposed mechanism. As it shows, our mechanism takes less training time than ALIDS because we can filter noise and meanwhile reduce the number of samples. Take the classification algorithm DNN, whose training requires the longest time, as an example. By DNN, we see more than 7 seconds of training time difference between ALIDS and our mechanism, with the advantage to our mechanism. Such an advantage mainly comes from our effective noise filtering as well as ability to properly select samples into the training set (in contrast to ALIDS which includes all samples in the training set).

Fig. 7 gives the update times and system accuracy for both mechanisms. It reveals a fact that gradual increase in running time and update times may progressively enhance the performance of the classifier. For instance, in KNN classification, the accuracy of our mechanism becomes steady upon updating twice and then rises 2% higher when updating increases to 8 times (mainly because our filtering mechanism makes it possible to involve less training samples than ALIDS). For ALIDS, the accuracy decreases and gets steady upon updating twice. Its training samples keep growing at a very fast speed and, with the large number of training samples, its training set fails to yield good accuracy (because, without a filtering mechanism, ALIDS is susceptible to the impact of noise). Note that when the number of training samples grows significantly (as in ALIDS), the training set may likely lose learning ability. Our mechanism, on the other hand, uses proper sample selection (acting as a noise filter) to reduce noise and meanwhile maintain stable increase in training samples. It helps us to obtain more effective learning and a stronger classifier (*i.e.*, a classifier with better classification and identification ability).

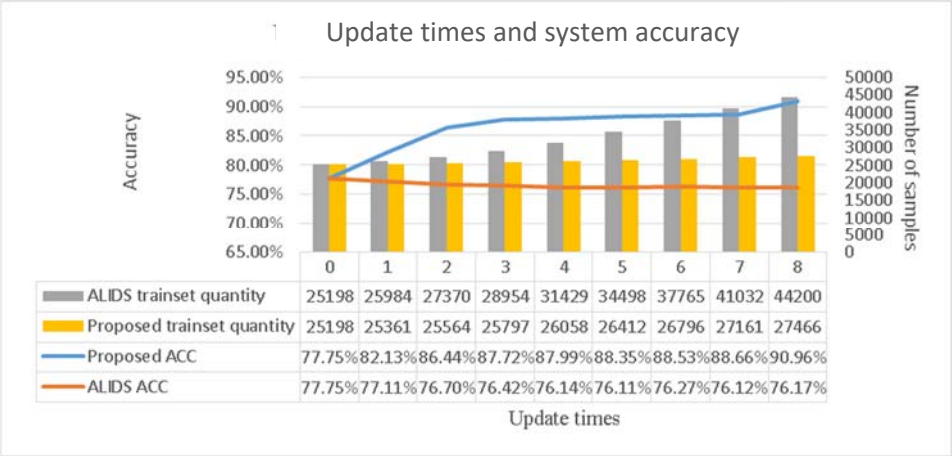

Fig. 7. Update times and system accuracy.

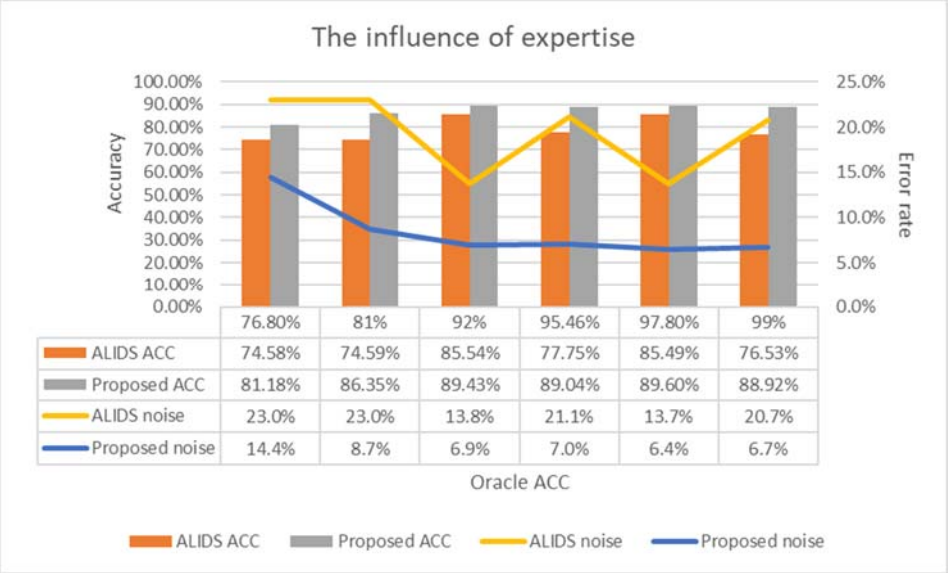

Fig. 8. The influence of the expert.

Fig. 8 depicts the expert influence in both mechanisms. As we can see, noise fluctuates between 13.7% and 23% for ALIDS and accuracy varies significantly with noise. Its expert ability grows from 76.8% to 99%, but the high expert ability and large number of training samples does not necessarily translate into high accuracy – if the noise in clustering and high confidence data remains (*i.e.*, not screened out as in our mechanism).

In our mechanism, the ratio of noise is inversely proportional to the expert ability, and both the ratio of noise and accuracy stay steady by approximately 90% expert ability. It indicates an effective noise filter will significantly help a learning mechanism to reduce and avoid the influence of noise. As noise is usually generated in high confidence data and

cluster labeling, ALIDS, which adopts direct random sampling without proper selection policy as our mechanism, hence cannot perform as well in screening out noise.

Fig. 9. Precision, TPR, accuracy and *F*1-score.

In addition to *Accuracy* (*ACC*), Fig. 9 also lists the *Precision*, *TPR* and *F*1-*score* results of each classifier for both mechanisms. Table 3 exhibits that, for a trained classifier, the classification result of a sample can be divided into four performance categories according to the True and Predicted classifications: True Positive (*TP*), False Positive (*FP*), False Negative (*FN*) and True Negative (*TN*). We can take the four categories to help illustrate *Precision*, *TPR* and *F*1-*score.*

Table 3. Sample classification performance categories.

True Classification	Predicted Classification	
	Predicted to be positive	Predicted to be negative
Truly positive	True Positive, *TP*	False Negative, *FN*
Truly negative	False Positive, *FP*	True Negative, *TN*

Note that *ACC* is referred to as the percentage of correct sample prediction for the classifier. Based on Table 3, we can get *ACC* by Eq. (4.1):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.1}$$

We can use *Precision*, indicated by Eq. (4.2), to judge if the positive data sample predicted by the classifier is truly positive, and then attain the so-called *False Alarm Rate* by 1 – *Precision.*

$$Precision = 1 - False\ Alarm\ Rate = \frac{TP}{TP + FP} \tag{4.2}$$

TPR (True Positive Rate) is the percentage of truly positive data samples which are predicted by the classifier as positive. It can be taken to indicate *Detection Rate* or *Recall*, as Eq. (4.3) shows.

$$TPR = Detection\ Rate = Recall = \frac{TP}{TP + FN} \tag{4.3}$$

To check the performance of a classifier by either *Precision* or *Recall* is insufficient. *F*1-*score* has been taken as a more fitting and comprehensive performance indicator for a trained classifier [18]. *F*1-*score*, as indicated in Eq. (4.4), considers both *Precision* and *Recall*: It is the harmonic mean of *Precision* and *Recall*. Note that high *Precision* does not necessarily indicate good performance unless it goes with high *Recall*. It is also true that, with low *Precision*, high *Recall* alone may not guarantee good performance. We can expect a classifier to yield good performance only when both *Precision* and *Recall* are significantly high or "balanced". In other words, we can use *F*1-*score* to judge if *Precision* and *Recall* are favorably balanced and to judge if the trained classifier yields good performance.

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{4.4}$$

As Fig. 9 shows, in contrast to ALIDS, our mechanism obtains higher *TPR*, *Accuracy* and *F*1-*score* (but not *Precision*) for all classifiers. As for *Precision*, ALIDS produces slightly better results in most of the classifiers except KNN (where both mechanisms show very close results) and BT (where our mechanism yields a better value). Our mechanism nevertheless produces constantly higher *F*1-*score* in all classifiers because its *Precision* values are always accompanied by more balanced *Recall* values. The fact pinpoints again the importance of selecting proper samples into the training set. Compared with ALIDS which includes all samples in the training set, our mechanism involves significantly less training samples and realizes more effective noise filtering. It is able to shape a stronger classifier with enhanced learning ability and, as a result, to improve the performance of SDN intrusion detection effectively.

5. CONCLUSIONS

In this investigation, we present a new training and learning mechanism to attain better intrusion countermeasures for the SDN structure. Our proposed mechanism involves the hybrid operations of *self-training* and *active learning*, preserving advantages while eliminating disadvantages, to reach more satisfactory performance. The new training and learning mechanism starts by randomly selecting samples with high confidence weights and classified as malicious and adding such samples to the training set. It then puts the training mechanism of *active learning* into work. Through the operation of *active learning*, the proposed mechanism is able to label samples with low confidence weights and add them to the training set for training, to raise the accuracy of the classifier. It meanwhile adopts a faster clustering method to cut down the operation time of *active learning*, and involves parallel training in classifier retraining in order to keep the classifier in constant service (without service interruption even during replacement). As the obtained simulation results demonstrate, in comparison to the existing ALIDS training mechanism, our new

mechanism performs more effectively in lifting up the identification accuracy of classifiers as well as the identification ability of unknown attacks. Note that our mechanism attains the performance gain (better identification accuracy and effective prevention of unknown attacks) at a rather contained time cost – mainly because we practice both training and detection in parallel without occupying the operation time of the detection system.

REFERENCES

1. G. A. Ajaeiya, "Flow-based intrusion detection system for SDN," in *Proceedings of IEEE Symposium on Computers and Communications*, 2017, pp. 787-793.
2. A. Abubakar, "Machine learning based intrusion detection system for software defined networks," in *Proceedings of the 7th International Conference on Emerging Security Technologies*, 2017, pp. 138-143.
3. A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the 11th Annual Conference on Computational Learning Theory*, 1998, pp. 92-100.
4. J. Zhang, C. Chen, Y. Xiang, and W. Zhou, "Robust network traffic identification with unknown applications," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013, pp. 405-414.
5. P.-J. Tsai, "Hybrid intrusion detection system toward unknown attack classification," Master Thesis, Institutional Repository of National Chiao Tung University, Taiwan, 2014.
6. S. McElwee, "Active learning intrusion detection using *k*-means clustering selection," in *Proceedings of Southeast Conference*, 2017, pp. 1-7.
7. Y. Xue and P. Peauseroy, "Constant false alarm rate for online one class SVM learning," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 2821-2825.
8. N. McKeown, *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, Vol. 38, 2008, pp. 69-74.
9. "OpenFlow Switch Specification, Version 1.3.4.," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf.
10. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, Vol. 31, 1999, pp. 1-69.
11. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1996, pp. 103-114.
12. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2nd IEEE International Conference on Computational Intelligence for Security and Defense Applications*, 2009, pp. 53-58.
13. J. R. Quinlan, "Introduction of decision trees," *Machine Learning*, Vol. 1, 1986, pp. 81-106.
14. N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, Vol. 46, 1992, pp. 175-185.
15. L. Breiman, "Random forests," *Machine Learning*, Vol. 45, 2001, pp. 5-32.

16. L. Breiman, "Bagging predictors," *Machine Learning*, pp. 24, 1996, pp. 123-140.
17. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol. 521, 2015, pp. 436-444.
18. Precision and recall, https://en.wikipedia.org/wiki/Precision_and_recall, 2020.

Po-Jen Chuang (莊博任) received the B.S. degree from National Chiao Tung University, Taiwan, in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with the Department of Electrical and Computer Engineering, Tamkang University, Taiwan, where he is currently a Professor. He was the Department Chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, mobile computing, network security, cloud computing, software defined networking, virtualization and internet of things.

Kuan-Lin Wu (吳冠霖) received his B.S. and M.S. degrees in Electrical and Computer Engineering in 2016 and 2019 from Tamkang University, Taiwan. He is currently with CAMEO Communications, Inc. in Taiwan. His research interests include software defined networking and network security.