Real-Time Malicious Traffic Detection With Online Isolation Forest Over SD-WAN

Pei Zhang, Fangzhou He, Han Zhang, *Member, IEEE*, Jiankun Hu, *Senior Member, IEEE*, Xiaohong Huang, Jilong Wang, Xia Yin, *Senior Member, IEEE*, Huahong Zhu, and Yahui Li

***Abstract*— Software Defined Network (SDN) has been widely used in modern network architecture. The SD-WAN is considered as a technology that has a potential to revolutionize the WAN service usage by utilizing the SDN philosophy. Attacks within SD-WAN can affect the network and block the entire services. In this paper, we propose a machine learning based anomalous traffic detection framework named OADSD over SD-WAN that can achieve task independently and has the ability of adapting to the environment. The OADSD adopts Distributed Dynamic Feature Extraction (DDFE) to extract representative features directly from the raw traffic, and proposes the On-demand Evolving Isolation Forest (OEIF) to make the system adapt to an environment. We provide a theoretical analysis of the performance of the OADSD. We also conduct comprehensive experiments to evaluate the performance of the OADSD with real world public datasets as well as a small real testbed. Our experiments under real world public datasets show that, the OADSD can accurately detect various kinds of attacks with a high performance. Compared with the state-of-the-art systems, the OADSD can achieve up to 60% accuracy improvement.**

***Index Terms*— Malicious traffic detection, traffic feature extraction, online learning.**

I. Introduction

SOFTWARE-DEFINED Wide Area Networking (SD-WAN) is an emerging paradigm that introduces the advantages of Software Defined Networking (SDN) into enterprise networking [7], [46]. By dynamically changing the forwarding rules according to the status of networks, the Quality of Service (QoS) can be significantly improved. Although the SD-WAN technology has already shown its excellent performance in many service providers' global WAN (e.g., Google [12], [17], [18], Microsoft [16], [26]), the threat activities (e.g., DDoS attacks) can occur and they can severely disrupt the user experience and reduce revenues finally. Therefore, real-time network malicious traffic detection is becoming an essential step, which can allow network administrators monitor the healthy status of the infrastructure.

Network malicious traffic detection technology is now widely used in SD-WAN scenarios [16], [18], [47] to provide network security. When anomaly events are detected, alerts are generated and sent to the network administrators to prevent further destruction from attackers.

Machine learning technologies are now becoming a new powerful tool in the network anomaly detection field. According to model complexity, we can divide the machine learning based anomaly detection methods into two categories, i.e., traditional machine learning methods, and the deep learning methods [43]. Traditional machine learning technologies based anomaly detection methods can be further divided into two types, i.e., supervised learning and unsupervised learning, where the supervised learning methods always train the detection models with labeled traffic and the unsupervised methods will separate the unlabeled traffic into clusters. Deep learning methods always train a complex neural network to improve the accuracy. Although machine learning techniques have already shown benefits in network anomaly detection, they still have following *limitations* over the SD-WAN scenarios:

Firstly, classical supervised machine learning based schemes (e.g., SVM [8], Decision Tree [39]) have a high accuracy but they need a large number of labeled traffic from experts for training. However, the internet traffic is now in an exploding growth stage, and *labeling traffic is always a time-consuming, error-prone, and even impossible work* for most network administrators. In comparison, classical unsupervised machine learning algorithms (e.g., Kmeans [20]) are easy to deploy but have a low accuracy.

Secondly, most of existing mainstream schemes (e.g., [11], [24], [48]) rely on expertly designed features to detect the anomalies. To efficiently detect the malicious traffic in SD-WAN, most methods need to extract specific features (e.g., flow size) from the raw traffic, which renders it less adaptable to the changing environment. Indeed, these methods are only effective for the unchanging environment but they might have low detection accuracy for the unknown attacks in a varying environment. Also, having too many features in a feature set requires a high volume of memory and computation, and a

Manuscript received 4 June 2022; revised 3 November 2022, 30 January 2023, and 11 March 2023; accepted 13 March 2023. Date of publication 27 March 2023; date of current version 5 April 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62002009 and in part by the Tsinghua University-China Telecom Joint Research Institute for Next Generation Internet Technology. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ghassan Karame. *(Corresponding author: Han Zhang.)*

Pei Zhang, Fangzhou He, and Xiaohong Huang are with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China.

Han Zhang, Jilong Wang, and Yahui Li are with the Institute for Network Sciences and Cyberspace (INSC), Tsinghua University, Beijing 100190, China, and also with the Zhongguancun Laboratory, Beijing 100194, China (e-mail: zhhan@tsinghua.edu.cn).

Xia Yin is with the Department of Computer Science and Technology (DCST), Tsinghua University, Beijing 100190, China, and also with the Zhongguancun Laboratory, Beijing 100190, China.

Jiankun Hu is with the School of Engineering and Information Technology, University of New South Wales at the Australian Defence Force Academy (UNSW@ADFA), Canberra, ACT 2612, Australia.

Huahong Zhu is with the Research Institute, China Telecom Corporation Ltd., Beijing 100033, China.

Digital Object Identifier 10.1109/TIFS.2023.3262121

long training time, which could affect the model accuracy at the end. In reality, the emerging SD-WAN provides a built-in inherent programmatic framework, where the edge routers always have a strong programming ability [18], [22], [40], [47]. This capability can be explored a scheme for dynamic feature extraction adaptable to the environment.

Thirdly, a model is always trained to gain a good performance over the training data. However, the network environment could change to a certain extent such that its operational data could be significantly different from the characteristics of the training data, leading to severe intrusion detection accuracy degradation. Therefore, the machine learning models should be smart enough to evolve with the network environment.

In addressing these challenges, we make the following **contributions** in this paper:

Firstly, we perform a deep analysis of malicious traffic detection algorithms and show the drawbacks of state-of-the-art machine learning based methods (See § II).

Secondly, we design and implement the OADSD, a system that aims for online malicious traffic detection for SD-WAN. The OADSD contains two parts, i.e., Distributed Dynamic Feature Extraction (DDFE), and On-demand Evolving Isolation Forest (OEIF), where DDFE is deployed over each edge device (e.g., VNF, router) and OEIF works at the controller and edge device. Instead of relying on domain expert extracted features, the edge devices in the DDFE take the raw network traffic as the input and adopt an autoencoder-based method to generate the optimal representation of the raw traffic. The OEIF constructs an Online Isolation Forest model to detect the malicious traffic in the on-demand update manner, where the parameter update is triggered when the observed false alarm rate is high. We analyze the theoretic performance of the proposed OADSD and prove that the performance gap between the proposed OADSD and the best static offline method tends to zero as iterations goes to infinity (See § III).

Thirdly, we conduct extensive experiments with real-world public datasets as well as a small testbed (see § IV). Our results show that our proposed OADSD can detect different types of attacks with AUC ranging between 0.85 and 0.99. Compared with domain expert features, our DDFE can extract more representative features, and could greatly help to improve the intrusion detection performance. We show that the OADSD can achieve a good performance when environment is not consistent with the training environment.

The rest of the paper is organized as follows: Section II shows the background and motivation. Section III introduces the design details of the framework. Section IV evaluates the performance of the framework. Section V introduces the related work. The last section summarizes the paper.

II. Background and Motivation

In this section, we will introduce the background about SD-WAN and show the limitations of state-of-the-art machine learning based anomaly detection methods.

A. Background

1) SD-WAN: SD-WAN is an overlay virtual solution for existing WAN that enables enterprises to securely connect

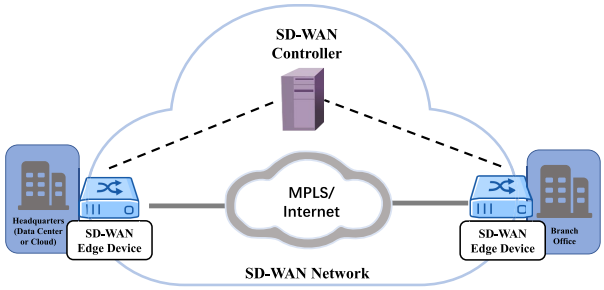


Fig. 1. The architecture of SD-WAN.

their branches using public internet resources. It incorporates the key principles of SDN into the wide area network, which can provide enhanced quality of service over unreliable links in a lower-cost way. Its primary focus is to connect an organization's different branches located in various *geographic locations* through the internet. As shown in Fig. 1, the SD-WAN can be used by companies to establish interconnections between their headquarters and external sites. The edge devices located at different locations are purchased and managed in a unified manner. After the edge devices are authenticated, a controller will establish an encrypted channel with the edge devices to transmit control messages, and a dedicated communication tunnel is established between the edge devices to transmit data.

2) Network Anomaly Detection Framework: Traditionally, a network anomaly detection framework usually contains three parts, i.e., traffic pre-processing, feature extraction and anomaly detection. The traffic pre-processing part will gather traffic from the network middleboxes. Then it will clean up the irrelevant network traffic (e.g., ICMP packets) which are not applicable and can affect the performance of detection. Sometimes the collected traffic might be imbalanced. Data normalization or augmentation technologies [45] can be used to improve the quality of data. The feature extraction part will derive the feature vectors from the captured traffic data. Traditionally, there are two types of features, i.e., packet-level features and flow level features [43]. The packet-level features (e.g., payload size, packet size, and payload ratio) are often used to find application anomalies, but they are not effective in providing a clear distinction in values between malicious and legitimate traffic flows, which might affect the detection accuracy. The flow-level features (e.g., mean packet length) will focus on multiple packets sequence rather than the contents of packets. Compared with the packet-level features, the flow-level features are more accurate but they often fail to detect the application anomalies. Finally, the anomaly detection part will perform detection algorithms to check whether the traffic contains suspicious behaviors.

B. Motivation

As a promising security method, machine learning based malicious traffic detection algorithms have been proposed as complements of the traditional fixed rule based methods [36]. Table I shows a the comparison between the machine learning based anomaly detection methods and the rule based anomaly detection methods. Although machine learning based methods

TABLE I
COMPARISON OF THE EXISTING MALICIOUS TRAFFIC DETECTION METHODS

Category of Detection Systems		Feature Extraction Method	Zero-Day Detection	Realtime Detection	High Accuracy	Task Independent	Environment Adaptive
Rule based		Preconfigured Fixed rules[4], [36]	✗	✓	✗	✗	✗
ML based	Domain Expert Features	Packet statistics [28]	✓	✓	✗	✗	✓
		Payload statistics [6]	✓	✗	✓	✗	✓
		Flow level statistics [37]	✓	✗	✗	✗	✓
	Automatic Feature Extraction	Packet header [25], [34]	✓	✗	✓	✓	✗
		Packet interval [44], [32]	✓	✗	✓	✓	✗
		OADSD	✓	✓	✓	✓	✓

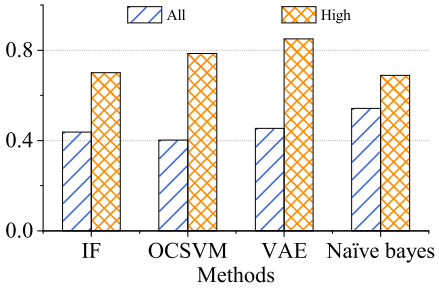


Fig. 2. F1-score of different feature set.

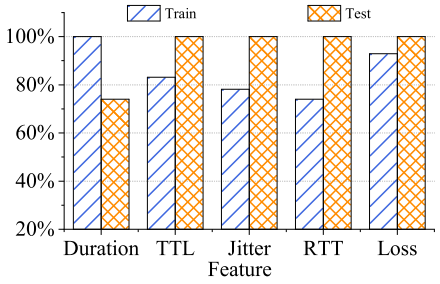


Fig. 3. Training and testing dataset.

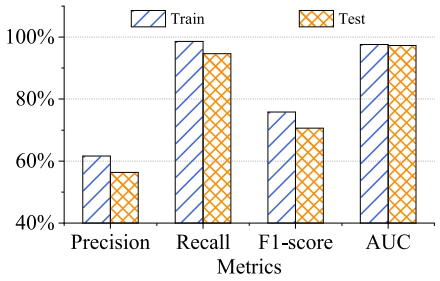


Fig. 4. Performance have large gaps.

can better detect zero-day attacks and have a higher detection accuracy, they still have the following two **limitations**:

Firstly, most of them fully rely on expertly designed specific features, which is costly and non-adaptable. As shown in Table I, machine learning based anomaly detection methods could be further divided into subtypes, i.e., domain expert feature based methods and automatic feature extraction based methods [6], [10], [28], [37], where the domain expert features based methods rely on pre-defined expert-designed features and they vary across attacks. In reality, more features do not necessarily imply better results. Indeed, too many features require a high volume of memory and computation and can extend the model training time which may affect the model accuracy at the end. The F1-score of different algorithms (e.g., Isolation Forest (IF) [49], One-class SVM (OCSVM) [9], Varational Autoencoder (VAE) [13] and Naive Bayes (NB) [21]) over CICIDS dataset [37] demonstrated in Fig. 2 is also consistent with this observation, where the top 17 features selected by Person score perform about 20% better than the all features provided by the CICIDS2017 dataset.

Secondly, most of them only perform well over training dataset but fail to perform well over varying testing datasets. Most machine learning algorithms always train their best parameters over the training dataset. However, the characteristics of the training dataset cannot always be the same as the testing dataset. The performance indicators over UNSW dataset [29] shown in Fig. 3 also prove this. For example, the jitter feature between the training dataset and test dataset could even be as large as 30%. The different characteristics of the training dataset and testing dataset also make the algorithms only perform well on the training dataset. We further take the Isolation Forest [49] as an example to demonstrate the performance gap between training and testing set provided by UNSW dataset. As Fig. 4 shows, the performance gap between the training dataset and the testing dataset can even be up to 10%, which demonstrates that using a static training model built over the training dataset would be problematic over a changing environment.

III. OADSD FRAMEWORK

In this section, we will present the OADSD, a system that aims for Online Anomaly Detection over SD-WAN. The architecture shown in Fig. 5 illustrates that OADSD contains two parts, i.e., Distributed Dynamic Feature Extraction (DDFE) and On-demand Evolving Isolation Forest (OEIF), where DDFE is deployed over each edge device (e.g., VNF, router) and OEIF works at the controller and edge device. The DDFE proposes to extract features by using a machine learning model deployed over the edge devices. The OEIF is able to update the model according to the feedback from network administrators to improve the detection accuracy when the false alarm is too high. In the following section, we will introduce the procedure as well as the analysis in details. Table II shows the main notations used in this paper.

A. Distributed Dynamic Feature Extraction

For the traffic passing through the edge devices, we need to extract the features for further detection. Different from

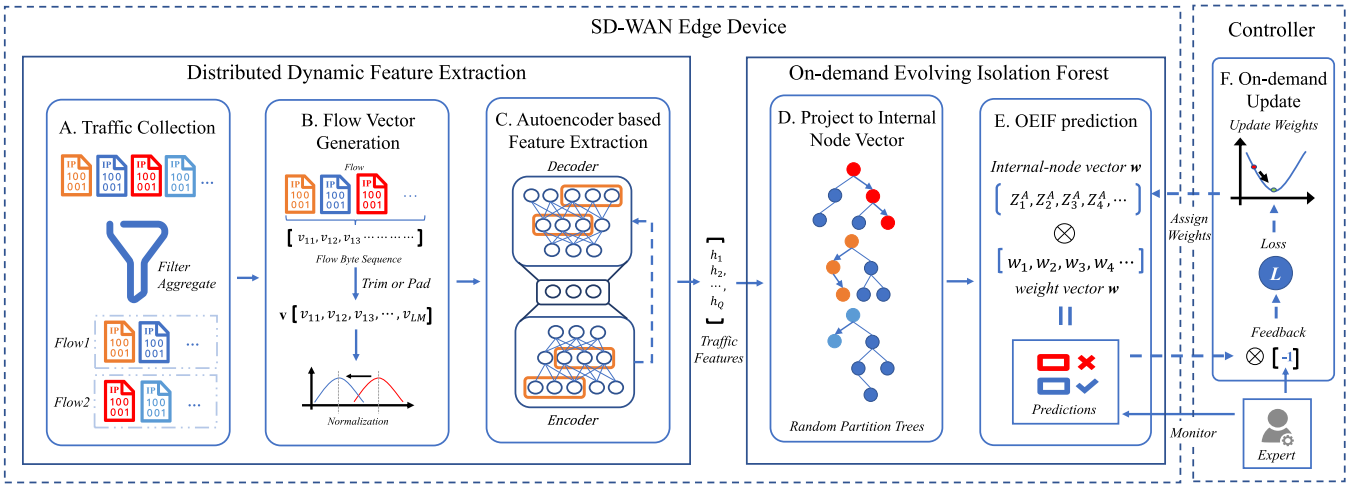


Fig. 5. The overall architecture of the OADSD, which contains two parts: Distributed Dynamic Feature Extraction (DDFE) and On-demand Evolving Isolation Forest (OEIF).

TABLE II
KEY NOTATIONS FOR OADSD

Symbol	Notation
μ	Packets are captured every μ time interval.
L	Number of packets used to construct a flow.
M	Number of features to present a packet.
$\mathbf{v}$	A flow vector.
$\mathbf{h}$	A flow feature vector, whose length is Q.
B	An node with two child nodes (B_l, B_r). A test consists of an feature B_q and a split value B_c.
F,N	Root node set and the Internal-node set.
$Z_B^{\mathbf{A}}$	Whether internal node B is visited for $\mathbf{A}$.
$\boldsymbol{w}$	Path weight vector.
λ	A pre-defined threshold for detection.
$y_{\mathbf{A}}$	Whether $\mathbf{A}$ is malicious or not.
Ψ	The observed false alarm rate
Φ	The pre-defined false alarm rate threshold.
Δ	The gap threshold.
η_i	The learning rate.
T	The flow feature vector which have feedbacks.
G	The upper bound of weight vector value.
δ	A constant for regularization.

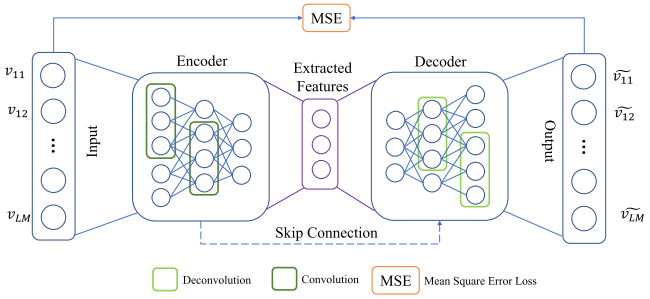


Fig. 6. Autoencoder based feature extraction. The input is a flow vector $\mathbf{v}$ and the encoder part will output the flow feature vector.

the traditional feature extraction methods which advocate to use the expert-designed features, OADSD proposes a dynamic feature extraction. As shown in Fig. 5, the SD-WAN edge devices will perform an autoencoder based feature extraction method for the captured raw traffic to derive the feature vectors for every μ time interval. Overall, the feature extraction procedure contains the following three main steps.

1) Step 1: Traffic Collection and Traffic Cleaning: Each edge device has a continuously running monitor process (e.g., tcpdump) to capture the raw network traffic going through the edge devices continuously. At the end of each μ time (e.g., 10s), each edge device will run a packet filter process to clean the irrelevant packets (e.g. ARP packet) based on the configuration of the controller, where μ is fixed and configured by the controller according to the ratio of edge resource (e.g., disk volume) and line speed. After the traffic cleaning step, packet number as well as the detection overheads are significantly reduced. In reality, the attacks can happen over many μ time intervals. We will detect these flows each μ time interval and regard them as the malicious ones even if only one time slot contains anomaly.

2) Step 2: Flow Vector Generation: Next, the OADSD will aggregate the packets with the same identifier (i.e., source IP address, destination IP address, source port, destination port and transport layer protocol) to form a flow. In reality, each flow has a diverse packet number, and this makes our feature extraction challenging. For simplicity, OADSD will first convert the flows with different length into the same length. OADSD will fills the short flows with zero elements to a fixed length size (e.g., L). For each packet, we adopt M features (e.g., packet size, payload size, packet head, payload) to present it. Therefore, the flow vector at the i-th time slot v_t can be denoted as $\{v_{11}, v_{12}, v_{13}, v_{14}, \cdots, v_{ij}, \cdots, v_{LM}\}$, where $v_i j$ is the i-th feature of packet i. When generating flow vectors, we first distinguish between short flows and long flows by checking the identifiers, i.e., if a flow's identifier does not appear in the previous time slot, we will identify it as a short flow, otherwise, it is a long flow. For the short flows, we only sample packets within the current time slot. For these *long flows*, we first perform a sample operation within the current time slot, and then perform another sample operation for the packets sampled in previous time slots together with the the newly sampled packets to form the flow vector. Therefore, long flow vector will contain information of all time slots.

3) Step 3: Autoencoder Based Flow Feature Extraction: As discussed in the previous section, the expert based feature

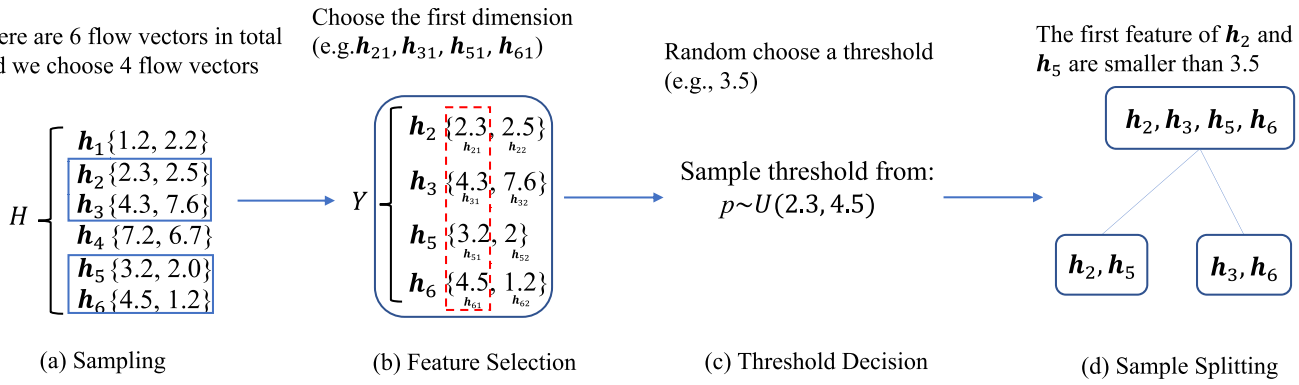


Fig. 7. The splitting procedure of an isolation tree.

extraction methods can incorporate human errors. To overcome this, we design an autoencoder based feature selection method to discover the non-intuitive features. Autoencoder is a special artificial neural network which tries to reconstruct its inputs. Autoencoder often consists of an encoder and a decoder. The encoder first compresses the input data into lower dimension, then the decoder tries to reconstruct the input by the output of the encoder. As shown in Fig. 6, each element in $\mathbf{v}$ will pass an encoder part which is constructed by several convolution layers with many neurons. The fabric of the decoder net is similar to the encoder net and it will output a vector $\widetilde{\mathbf{v}}$. The output of the encoder part can be regarded as the *flow feature vector* (i.e., $\mathbf{h}$). We also add skip connections to the original structure to accelerate the training speed, whose connecting strategy follows the pre-activation version of the deep residual network [15]. We train the model in an unsupervised manner and do not need any labels. We aim to minimize the Mean Square Error (MSE) between $\mathbf{v}$ and $\widetilde{\mathbf{v}}$:

$$\min \sum_{i=1}^{L} \sum_{j=1}^{M} (v_{ij} - \widetilde{v}_{ij})^2 \tag{1}$$

B. On-Demand Evolving Isolation Forest

After flow feature extraction, we now construct an On-demand Evolving Isolation Forest (OEIF) to distinguish malicious traffic. The OEIF contains two parts, i.e., Isolation Forest Construction and On-demand Update, where an Isolation Forest (IF) is trained beforehand and the On-demand Parameter Update works at the operational phase.

1) Isolation Forest Construction: We begin with the definition of Isolation Tree.

Definition 1: **Isolation Tree**. Let B denote a node of an isolation tree. B is either an external-node with no child, or an internal-node with one test and exactly two child nodes (B_l, B_r). A test consists of an feature B_q and a split value B_c.

The flow feature samples used for training can be presented as $H = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_i, \ldots\}$, where $\mathbf{h}_i = \{h_{i1}, h_{i2}, \ldots, h_{iQ}\}$ is the i-th flow feature set containing Q features. We need 5 steps to obtain a random isolation tree. Fig. 7 shows an example.

a) Step 1: Sampling: We need to randomly select some samples from H to train the model. The selected samples are presented as Y. All the elements in Y construct the root node. Fig. 7(a) shows a simple example, where we use a 6 sample in H to train a tree. After randomly sampling, we derive a sub-sample Y which contains 4 samples for the following steps.

b) Step 2: Feature selection: Next, we randomly select a dimension from the flow features to split the selected samples and build the isolation tree. Take Fig. 7(b) as the example, we choose the first dimension (i.e., $h_{11}, h_{21}, h_{31}, h_{41}, h_{51}$) for sample splitting.

c) Step 3: Threshold decision: For each tree node including the root node that contains more than one flow feature vector, we need to choose a threshold for sample splitting. We randomly choose a value between the minimum and maximum value of the corresponding feature chosen in Step 2 as the threshold. For example, in Fig. 7(c), the minimum and maximum value of the first feature among the 4 samples in Y are 2.3 and 4.5, and we choose 3.5 as the threshold. Since different device has different input data, the selected thresholds will also be different.

d) Step 4: Sample splitting: We need to separate each tree nodes (i.e., external nodes) which contains more than one feature vector into two child nodes. The left node (i.e., B_l) contains the samples whose split value is equal or smaller than the threshold decided in Step 3, and the right node (i.e., B_r) contains the remaining ones. Then we set the selected feature's index to B_q and set the split value to B_c. For example, as Fig. 7(d) shows, the left node (B_l) contains flow feature vector $\mathbf{h}_2$ and $\mathbf{h}_5$, while the right node (B_r) contains $\mathbf{h}_3$ and $\mathbf{h}_6$. The feature of the node is the label of the first feature dimension and the split value is 3.5 in our example.

e) Step 5: Recursively building: Finally, we will perform Step 2 to Step 4, until all tree nodes contain one flow feature vector and are unable to be split any more. Then we will return the root node of the isolation tree.

Repeating the above steps, we can obtain an IF (Isolated Forest). Let F denote the root node set and N denote the Internal-node set. Considering an arrival flow feature vector $\mathbf{A}$, $\mathbf{Z}^{\mathbf{A}}$ denotes a visited-internal-node vector, where $\mathbf{Z}^{\mathbf{A}}_B \in \{0, 1\}$ denotes whether an internal node B is visited ($\mathbf{Z}^{\mathbf{A}}_B = 1$) or not ($\mathbf{Z}^{\mathbf{A}}_B = 0$). In the detection stage, the IF is used to map the input feature vector $\mathbf{A}$ into $\mathbf{Z}^{\mathbf{A}}$. Algorithm 1 describes how the map procedure works. The algorithm will loop each internal-node. The loop will continue until the leaf node is reached. If the split value of the internal-node is larger than

Algorithm 1 Visited Internal-Node

Input: Arrival flow feature vector $\mathbf{A}$, root node set F.
Output: Internal-node path vector $\mathbf{Z^A}$

1: $Z_B = 0, \quad \forall B \in N$;
2: **for** $B \in F$ **do**
3: **while** B is an internal-node **do**
4: $Z_B^{\mathbf{A}} = 1$;
5: $T = \begin{cases} B_l & A_{B_q} \leq B_c \\ B_r & A_{B_q} > B_c \end{cases}$
6: **end while**
7: **end for**
8: **return** $\mathbf{Z^A}$

$\mathbf{A}$'s corresponding feature value, then we will loop its right child, otherwise, we will loop its left child (Line 5). If the B-th internal node has been visited, the B-th element in $\mathbf{Z^A}$ will be set to 1. Therefore, $\sum_{B\in N} \mathbf{Z}_B^{\mathbf{A}}$ can present the total path length $\mathbf{A}$ traverses through the forest.

2) On-Demand Parameter Update: The on-demand parameter update scheme works at the operational phase. As an environment changes, the static model weight might lead to a high false alarm rate. If users find the gap between the observed false alarm rate (e.g., Ψ) and the pre-defined false alarm rate threshold (e.g.,Φ) is higher than the pre-defined threshold Δ, the parameter update process will be triggered:

$$\Psi - \Phi \geq \Delta \tag{2}$$

There are also other update signal in practice, e.g., zero-day attack. When zero-day attacks happen, it would be observed by the local site operators. Then the results are reported to the controller and trigger the parameter update process. Therefore, the zero-day vulnerabilities can also be detected by our method. To update the parameters, we consider incorporating feedback from IT administrators to tune the weight (i.e., $\boldsymbol{w}$). Let T present the indices set of the feedback that received from operators. For each $t \in T$, $\boldsymbol{w}_t$ denotes the value of weight vector after receiving the feedback t. We now give the domain S for all possible weights:

$$S = \left\{ \boldsymbol{w}_t | \boldsymbol{w}_t \in \mathbb{R}^N, ||\boldsymbol{w}_t|| \leq G \right\} \tag{3}$$

Let λ represents the pre-defined threshold. Let y_t denote the detection result for t. If $\boldsymbol{w}_t \cdot \mathbf{Z}^t > \lambda$, we will regard it as a malicious flow ($y_t = -1$), otherwise, we regard it as a benign one ($y_t = 1$). And let $\hat{y}_t$ denote the true label of t. Mathematically, we should try to seek $\boldsymbol{w}$ that minimizes the incorrectly classified examples over the total feedbacks, i.e.,

$$\min_{\boldsymbol{w}\in S} \sum_{t\in T} \theta(sign(\boldsymbol{w}_t \cdot \mathbf{Z}^t - \lambda) \neq \hat{y}_t) \tag{4}$$

where $\theta(x) \in \{0, 1\}$ is the indicator function that takes the value 1 if x is satisfied and 0 otherwise. The $sign(x) \in \{-1, 1\}$ is the sign function. $\hat{y}_t$ is the true label of t, where $\hat{y}_t = -1$ when t is a malicious flow, otherwise ($\hat{y}_t = 1$), it is a benign one. However, (4) is hard to use in practice since it is 0/1 loss and needs to access the whole feedback set. In this case, we would like to replace it with a convex loss function whose parameter is $\boldsymbol{w}_t$ after receiving the feedback of t :

$$f_t(\boldsymbol{w}_t) = \max\{0, -\hat{y}_t \cdot (\boldsymbol{w}_t \cdot \mathbf{Z}^t - \lambda)\}. \tag{5}$$

To derive the best parameter when receiving the feedback of t, we aim to use any vector which has a minimal loss on all past rounds, i.e.,

$$\boldsymbol{w}_{t+1} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{t} \{f_i(\boldsymbol{w}) + R_i(\boldsymbol{w})\}. \tag{6}$$

where $R_i(\boldsymbol{w})$ is a regularized function (i.e., the regularizer) to stabilize the prediction. From (4), we can derive the best parameters. However, it is often hard to solve for the online application. In this case, we adopt the online mirror descent algorithm [14] and change (5) to its gradient function, i.e.,

$$f_i(\boldsymbol{w}_i) \simeq \bigtriangledown f_i(\boldsymbol{w}_i)\boldsymbol{w}_i, \tag{7}$$

where $\bigtriangledown f_i(\boldsymbol{w}_i)$ is the gradient of f_i at $\boldsymbol{w}_i$, which can be derived as:

$$\bigtriangledown f_i(\boldsymbol{w}_i) = \begin{cases} -\hat{y}_i \cdot \mathbf{Z}^i & \hat{y}_i \neq y_i \\ 0 & \hat{y}_i = y_i. \end{cases} \tag{8}$$

The regularization term $R_i(\boldsymbol{w})$ is defined as follows,

$$R_i(\boldsymbol{w}) = \frac{1}{2}\delta(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})||\boldsymbol{w} - \boldsymbol{w}_i||^2, \tag{9}$$

where δ is a constant and η_i is the learning rate. Take (7) and (9) into (6), we can derive a convex optimization problem with no constraints. According to KKT (Karush–Kuhn–Tucker) condition, we take the derivative of (6) equaling to 0 and get the optimal feature weight vector:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta_t}{\delta} \sum_{i=1}^{t} \bigtriangledown f_i(\boldsymbol{w}_i) \tag{10}$$

In the area of online learning, the regret metric [14] can measure the difference between the online algorithm and the best parameters. A practical online algorithm should have a sub-linear regret upper bound. This implies that on average the algorithm is performing as the best static strategy we can have in hindsight,. i.e.,

$$Regret_T = \max_{\boldsymbol{w}^*\in S} \left\{ \sum_{t=1}^{T} f_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{w}^*) \right\} = o(T), \tag{11}$$

where $f_t(\boldsymbol{w}_t)$ is the performance loss after receiving the feedback of flow feature vector t. To analyze the performance of on-demand parameter update scheme, we first give the following lemma:

Lemma 1: Compared with the best static decision $\boldsymbol{w}^*$, the regret of the online learning algorithm is bounded by:

$$Regret_T \leq R_t(\boldsymbol{w}^*) + \sum_{t=1}^{T} [f_t(\boldsymbol{w}_t) - f_t(\boldsymbol{w}_{t+1})], \tag{12}$$

where $R(\cdot)$ is the regularization term in the loss equation, and $\boldsymbol{w}^*$ is the optimal parameters in hindsight.

Proof: First, we consider the regret against the regularized loss function, i.e.,

$$l_t(\boldsymbol{w}_i) = f_t(\boldsymbol{w}_i) + R_t(\boldsymbol{w}^*) \tag{13}$$

The regularized loss can be computed as:

$$J_T = \sum_{t=1}^{T}\{\sum_{i=1}^{t} l_t(\boldsymbol{w}_i) - \sum_{i=1}^{t-1} l_t(\boldsymbol{w}_i)\} - \sum_{t=1}^{T} l_t(\boldsymbol{w}^*) \tag{14}$$

Since, the objective of online learning is to find the $\boldsymbol{w}_{t+1}$ which has the minimum accumulated loss over previous t steps. Therefore, $\boldsymbol{w}_{t+1}$ has the smallest accumulated loss over previous t steps.

$$\begin{aligned} J_T &\leq \sum_{t=1}^{T}\{\sum_{i=1}^{t} l_t(\boldsymbol{w}_i) - \sum_{i=1}^{t-1} l_t(\boldsymbol{w}_i)\} - \sum_{t=1}^{T} l_t(\boldsymbol{w}_{T+1}) \\ &= \sum_{t=1}^{T}\{\sum_{i=1}^{t} l_t(\boldsymbol{w}_i) - \sum_{i=1}^{t} l_{t+1}(\boldsymbol{w}_i)\} \end{aligned} \tag{15}$$

According to the definition of regret shown in (11), then taking (13) and (14) into (15), we could derive the following equation:

$$\begin{aligned} Regret_T \leq \sum_{t=1}^{T} R_t(\boldsymbol{w}^*) + \sum_{t=1}^{T}\{\sum_{i=1}^{t}[f_i(\boldsymbol{w}_t) + R_i(\boldsymbol{w}_t)] \\ - \sum_{i=1}^{t}[f_i(\boldsymbol{w}_{t+1}) + R_i(\boldsymbol{w}_{t+1})]\} - \sum_{t=1}^{T} R_t(\boldsymbol{w}_t) \end{aligned} \tag{16}$$

According to (6), $\boldsymbol{w}_t$ has the minimal accumulated loss of previous $t-1$ steps. And our regularization term $R(\cdot)$ is non-negative. Therefore, we have:

$$\sum_{t=1}^{t-1}[f_i(\boldsymbol{w}_{t+1}) + R_i(\boldsymbol{w}_{t+1})] \geq \sum_{t=1}^{t-1}[f_i(\boldsymbol{w}_t) + R_i(\boldsymbol{w}_t)] \tag{17}$$

With (17), we have:

$$\begin{aligned} &f_t(\boldsymbol{w}_t) - f_t(\boldsymbol{w}_{t+1}) + R_t(\boldsymbol{w}_t) - R_t(\boldsymbol{w}_{t+1}) \\ &\geq \sum_{i=1}^{t}[f_i(\boldsymbol{w}_t) + R_i(\boldsymbol{w}_t)] - \sum_{i=1}^{t}[f_i(\boldsymbol{w}_{t+1}) - R_i(\boldsymbol{w}_{t+1})] \end{aligned} \tag{18}$$

Combine(16) and (18) we have:

$$Regret_T \leq R_t(\boldsymbol{w}^*) + \sum_{t=1}^{T}[f_t(\boldsymbol{w}_t) - f_t(\boldsymbol{w}_{t+1})] \tag{19}$$

we can prove Lemma 1. ■

Theorem 1: Let G be the maximal value of the weight vector $\boldsymbol{w}$. N denotes the number of nodes in the forest. Let $\eta_t = \frac{G}{\sqrt{t}}$, we have:

$$Regret < (2\delta + \frac{1}{\delta})N^2 G\sqrt{T} \tag{20}$$

Proof: For every $u, v \in S$. $||u - v|| \leq ||u|| + ||v||$. According to 3, we have $||\boldsymbol{w}_t|| \leq NG$, thus, $||\boldsymbol{w}^* - \boldsymbol{w}_t|| \leq 2NG$. Note that $\bigtriangledown f(\boldsymbol{w}_t) = \bigtriangledown f(\boldsymbol{w}_t)\cdot \boldsymbol{w}_t$ and $||\boldsymbol{Z}^t|| < N$, so we have $||\bigtriangledown f(\boldsymbol{w}_t)|| < N$

$$\begin{aligned} Regret &\leq \frac{1}{2}\sum_{t=1}^{T}\delta(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}})||\boldsymbol{w}^* - \boldsymbol{w}_t||^2 \\ &\quad + \frac{1}{2\delta}\sum_{t=1}^{T}\eta_t||\bigtriangledown f(\boldsymbol{w}_t)||^2 \\ &\leq \frac{1}{2}\sum_{t=1}^{T}\delta(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}})||\boldsymbol{w}^* - \boldsymbol{w}_t||^2 + \frac{N^2}{\delta}\sum_{t=1}^{T}\eta_t \\ &< 2N^2 G\delta\sqrt{T} + \frac{N^2}{\delta}G\sqrt{T} \\ &= (2\delta + \frac{1}{\delta})N^2 G\sqrt{T} \end{aligned} \tag{21}$$

■

This implies that with the feedback number growing the difference between the best static strategy and the model goes to zero. In a real-life application, alarms generated by the IDS will often require the inspection from the operator in any system. So, using the feedback from such alarms do not increase any extra inspection cost. Setting up a feedback triggering threshold will further reduce the involvement of the operator.

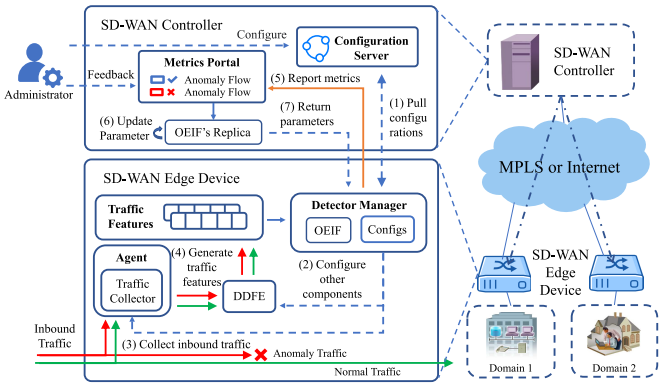


Fig. 8. The overall design of OADSD system.

C. System Design

Fig. 8 shows the overall architecture of the system, which contains one controller and multiple edge devices. The controller is responsible for configuration management and updating models. The edge device is responsible for malicious traffic detection. The OADSD works as follows: The edge devices capture and filter the inbound traffic from the internet. Then they will extract features and perform malicious traffic detection. The edge devices timely report the detection results to the controller. If the detection results trigger the update policies (e.g., low accuracy), the operators will try to mend the model.

Controller is the coordinator of the whole system which contains three main components: (1) Configuration Server. This module maintains the configurations of the OADSD (e.g., μ). After an edge device connect to the network, it will pull the configurations to edge devices; (2) Controller Portal. If the

administrator decides to update the model, then the detection parameters will update and redistribute to the corresponding edge devices. In reality, the feedback from the administrator may not always be accurate. OADSD also provides a rollback mechanism which could rollback to last checkpoint. When the administrator gives incorrect feedback and lead to worse performance, the edge device could quickly rollback to its last checkpoint; (3) Communication Channel. This module is responsible for communicating with the edge devices, where we use long-lived TCP connections to avoid connection established costs.

Edge device is responsible for monitoring and detecting the inbound traffic. We first extract a set of features from each packet (e.g., packet size, payload size, packet head, payload) and form a flow vector to represents the flow. Then the sequence will be fed into an auto-encoder. The auto-encoder is 1d-CNN network with an average pooling in front of it. The average pooling will transform the input matrix into a unified lower dimension input. This allows the auto-encoder to handle network traffic with varied lengths. Then we send the traffic into the auto-encoder to get the traffic features. In details, it consists of four modules: (1) Agent. It runs a sniffer (e.g., tcpdump) to collect and filter the inbound traffic every μ time interval; (2) DDFE. This module is responsible for extracting the traffic feature from the collected packets; (3) Detector Manager. The detector manager is responsible for detecting malicious traffic based on the feature vectors extracted by DDFE. The detector will send a message to controller after detection. The messages sent by the edge device include the identifier (serial number and MAC address of the device). , the controller can differentiate messages from different edge devices. (4) Communication Channel. This module is responsible for communicate with the controller.

We introduced a feedback mechanism to improve the detection accuracy of malicious traffic using machine learning model in SD-WAN environment. By collecting the detection results of the edge devices in the controller, introducing expert feedback and training the machine learning model at the controller in an online fashion. Then sending the parameters to the edge devices for detection, the detection accuracy of machine learning model can be secured. This mechanism makes full use of SD-WAN centralized control and reduces the pressure on the training model of edge equipment.

We use a whitelist and certificate to guarantee the security of the system where the whitelist records each trusted device information (e.g., the serial number, MAC address, and authentication status). Edge devices and the controller have certificates to prove their identity. The whitelist and certificate are managed by the network operator and are preinstalled both in the controller and edge devices. When an edge device establishes control connections to SD-WAN controllers, it exchanges its device SSL certificate with the controller during the SSL handshake process for authentication. Then controller will check whether the edge device is whitelisted. If the above steps are passed, the authentication between controller and edge device is completed, and a two-way SSL connection will be established for secure communication. This will prevent malicious devices from joining SD-WAN domain and all devices are known, trusted, and authorized. Sometimes some edge devices might be compromised and become malicious, then, the network operator will remove the edge device from the whitelist in the controller. Whenever the controller receives a message from an edge device, it will check whether the edge device is whitelisted and authenticated. If it is not authenticated, the controller will terminate the SSL connection with the edge device. In this way, the controller is secured from malicious edge devices in multiple domains. Also, the controller might be compromised and the network operator will remove the controller from the whitelist in each edge device and set a backup controller. When the edge device receives each request from a controller, it will also check whether the controller is whitelisted and authenticated, and if not, the edge device will terminate the DTLS/TLS connection with the controller and switch to a backup controller.

IV. Evaluation

In this section, we will evaluate the performance of the OADSD by using real-world attacks (e.g., DDoS, DoS) with three public real-world datasets as well as a small testbed. Our main results are as follows:

- Compared with existing methods, OADSD could achieve the best performance on most of the datasets and could achieve a high detection accuracy on all attack types.
- The proposed OADSD could adapt to the environment with only a few iterations of updates. We test OADSD on a two-dimensional synthesize dataset. The result shows OADSD could adapt to environment changes. But the baseline IF could not adapt to the environment change, with a significant performance drop being observed.
- Classical Semi-supervised and unsupervised detection algorithms achieve a lower detection accuracy compared to the supervised detection algorithms. Hence, we further compare the detection accuracy with other supervised detection algorithms [19], [21]. The results demonstrate that OADSD could achieve compatible or better performance than those supervised algorithms.

A. Setup

We use the PyTorch to implement the autoencoder described in Section III and the variational autoencoder in the baseline methods. We use C++ (version 5.4.0) to implement the isolation forest model and also the online update algorithm. For all our traffic trace, we use 80% of traffic (i.e. both normal and anomaly) to train the machine learning algorithms and the rest of the 20% traffic are used to test the performance. OADSD has some parameters. L is set to 900 in default. The maximal value of weight vector (i.e., G) is set to 1. The default value of δ is 2. For a flow, when its weighted path length is larger than 0.6 (i.e., λ), we will regard it as a malicious flow. The default pre-defined false alarm rate threshold (e.g.,Φ) is 10% and default update trigger threshold Δ is 5% in our evaluations. For an arriving traffic, if the parameter update condition is satisfied, we will give the true label to our system to mimic the feedback of network operators.

Baselines. We compare the performance of OADSD with the following methods:

- *Packet-level Detection.* We use the state-of-the-art machine learning based detection method, Kitsune [28], which extracts 23 features via flow state variables and feeds the features to an autoencoder based classifier. We use the open source Kitsune implementation [28], and detect attacks with the same hardware resource as the OADSD.
- *SVM with Flow-level Statistics Classification (SFSC).* Flow-level statistics features have already been used by many anomaly detection works [21], [27], [37]. We choose the flow-level statistics features described in [29] and [37], including the maximum, minimum and the mean of packet size. Then we perform normalization for the flow-level statistics features. For the classification algorithm, we choose SVM, because it has been widely used and has a good detection performance [9], [19] in network anomaly detection. The implementation of SVM is shown in [5].
- *Static Isolation Forest with highest correlation features (SIFD).* We perform correlation analysis over all the features adopted by the previous works [21], [27], [28], [37]. Then SIFD chooses the top 10 features and uses the static isolation forest model to detect the anomalies. Compared with OADSD, SIFD detection algorithm is unable to adapt to environment. We use the open source isolation forest implementation [33] in our following evaluations.

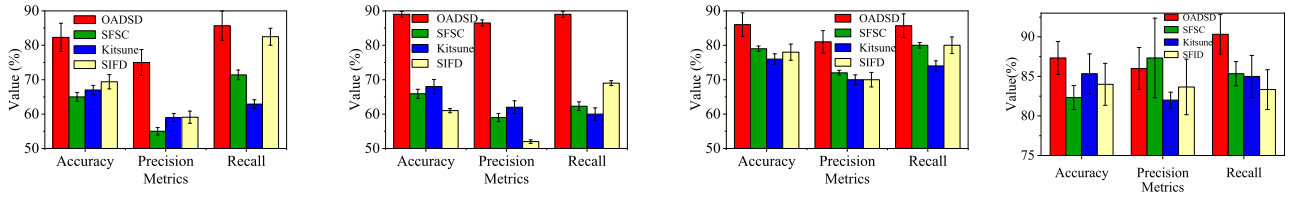
Datasets. We test OADSD and above baseline methods with the three following three public datasets:

- *CICIDS 2017 Dataset* [37]. The CICIDS 2017 dataset has been developed by University of New Brunswick. The dataset is generated in a simulated real world network architecture. It contains DoS, Brute Force, Infiltration, Heartbleed, DDoS, Botnet and Web attack. It is publicly available at CIC homepage which comprises millions of network records.
- *ISCX 2012 Dataset* [38]. The ISCX 2012 dataset has been developed by the University of New Brunswick. The entire ISCX dataset contains 20 features taking nominal, integer or float. The dataset covers roughly seven days of network activities (i.e. normal and attack). We separate four different attack types, called as Brute Force SSH, Infiltrating, HTTP DoS, and DDoS are conducted on different days.
- *UNSW-NB15 2015 Dataset* [29]. The UNSW-NB15 2015 dataset has been developed by the University of New South Wales. The entire UNSW-NB15 labeled dataset contains over two million records. Each record is described through 49 features taking nominal, integer or float values. UNSW-NB15 dataset contains seven anomaly classes, namely, Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic and Reconnaissance.
- *CERNET dataset*. The China Education and Research Network (CERNET) is the largest national education and research computer network in China, connecting thousands of colleges and campuses through CERNET backbone. Each campus network deploys a core router connected to the CERNET backbone network. CERNET is similar to SD-WAN, e.g., they both use intelligent schemes to establish new routes between campuses with tunnels, they both collect network performance indicators at the edge devices to monitor network performance, and they both switch to backup paths through centralized control to respond to failures. The CERNET dataset is generated from one core router, similar to the inbound traffic collected from edge devices. There are 875 attacks, including DoS, DDoS, and PortScan. The trace is about 26GB and contains approximately 190,000,000 flow information with raw packets from 2014-12-10 to 2014-12-13, whose sampling ratio is 1:1. We extract a set of features from each raw packet (e.g., packet size, payload size, packet head, payload) in a flow and form a flow feature vector to perform online malicious traffic detection.

Metrics. We use the following metrics to evaluate the detection performance: (1) the Area Under ROC Curve (AUC), the AUC score stands for the area under the ROC curve. The ROC curve is created by plotting the true positive rate against the false positive rate at different thresholds; (2) the average precision score (AP), the average precision score stands for the area under the precision recall curve (PR curve). The PR curve is created by plotting the precision score against the recall score at different thresholds;. (3) Precision, the precision for a class is the number of true positive (TP) items divided by the total number of elements labeled as the positive class. Precision score could be calculated by: $Precision = \frac{TP}{TP+FP}$; (4) Recall, the recall of a class is the number of true positive items divided by the sum of true positives and false negatives, which could be calculated by: $Recall = \frac{TP}{TP+FN}$; (5) F1-score (F1), combines the precision and recall into together. The F1-score could be calculated by: $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$.

B. Performance With Real World Traffic

1) Performance Comparison: In this part, we compare the performance of OADSD with the most widely used malicious traffic detection algorithms. Fig. 9 shows the performance comparison between OADSD and SFSC, Kitsune, SIFD under different data set. We can see that OADSD has consistently high performance under all datasets. For detection accuracy, OADSD performs up to 40% higher than the other methods. Next, we select the samples that contain the specific attacks to evaluate the performance, e.g., UNSW-Fuzzers denotes that traffic only contains Fuzzer attack traffic and benign traffic. Table III shows the results measured with AUC and AP score. We find that OADSD can detect all 15 attacks with AUC ranging from 0.932 to 0.99 and AP ranging from 0.75 to 0.98, which performs up to 2×, 10×, 50% better than SFID, Kitsune, SFSC, respectively. For the comparison of the detection algorithm accuracy, the OADSD could achieve a much higher detection accuracy than SIFD, especially for attacks which rely on payload information (e.g., WebAttacks or Infiltration). We can see that unsupervised (e.g. SIFD) and



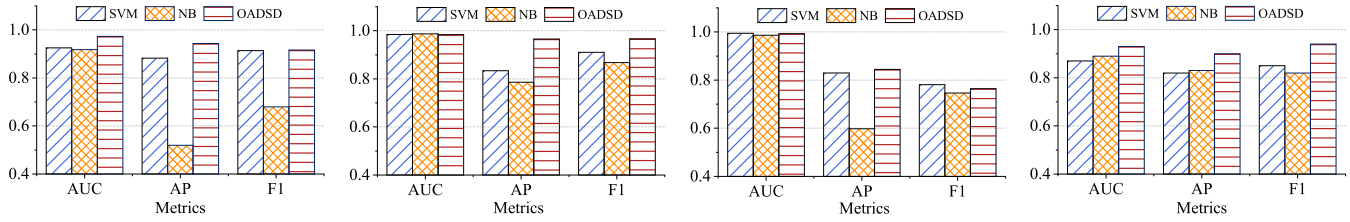
(a) CICIDS data set (b) ISCX data set (c) UNSW data set (d) CERNET data set

Fig. 9. Overall performance comparison.

TABLE III
THE DETECTION ACCURACY COMPARISON

Methods	SIFD		Kitsune		SFSC		OADSD	
Attack types	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
UNSW-Shellcode	0.98305	0.55933	0.941514203	0.195711603	0.999841117	0.99350985	0.997304	0.903297
UNSW-Reconnaissance	0.97481	0.48044	0.8772758	0.131257218	0.973835963	0.84532369	0.996164974	0.918044794
UNSW-Fuzzers	0.95719	0.39095	0.884176593	0.227484417	0.985009866	0.683967929	0.989730996	0.780245132
UNSW-Backdoor	0.96792	0.08603	0.930067843	0.109221492	0.998721167	0.901027363	0.997303922	0.915763533
CICIDS-DDos	0.53523	0.41787	0.961726889	0.93062217	0.8456	0.6435	0.99810457	0.997833224
CICIDS-WebAttacks	0.874474	0.12784	0.78626853	0.073393052	0.996059229	0.789351344	0.997332528	0.85222373
CICIDS-Boynet	0.86441	0.07224	0.727524929	0.027323437	0.792034207	0.652303598	0.924708417	0.69025
CICIDS-BruteForce	0.72786	0.10128	0.522442717	0.061372558	0.9159	0.2708	0.999998	0.99997
ISCX-BruteForce	0.91117	0.17114	0.99913614	0.922732293	0.999968029	0.998480866	0.999541049	0.98903316
ISCX-DDoS	0.79691	0.12789	0.421980582	0.047920694	0.988247439	0.81886349	0.985723162	0.867704955
ISCX-Infitration	0.900738	0.438688	0.982031856	0.972588616	0.98087659	0.963824803	0.980064351	0.952659378
ISCX-DoS	0.90467	0.44645	0.964587496	0.613145843	0.988247439	0.987174077	0.973391291	0.888651315
CERNET-DoS	0.823212	0.65143	0.934324	0.913434	0.82328	0.8743432	0.999541049	0.98903316
CERNET-DDoS	0.6323432	0.323353	0.323434	0.2324343	0.92343432	0.844534	0.97232444	0.88343243
CERNET-PortScan	0.734323	0.234349	0.62342343	0.53453434	0.983431234	0.844343432	0.90343234	0.9645435345

[1] We highlight the best in • and the worst in • and AUC <0.5 means no better than random guess.

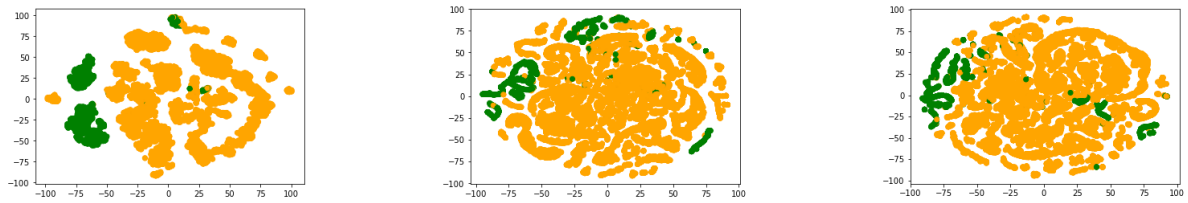


(a) CICIDS data set (b) ISCX data set (c) UNSW data set (d) CERNET data set

Fig. 10. Performance comparison between OADSD and supervised methods.

semi-supervised (e.g. Kitsune) methods achieve a very low AP score in most of the time. The reason for this is that the unsupervised and semi-supervised methods, like IF or VAE, could not utilize the label information, which makes them hard to give an unbiased classification result and leads to a higher false alarm rate. In comparison, OADSD could detect both brute force attacks like DDoS, SSH or FTP Patator and attacks highly correlated with payload information, like SQL injection, exploits and worms. The reason for this is that OADSD can efficiently capture the features of traffic and is able to adapt to network change. We further compare the performance of the supervised methods like SVM, Naive Bayes [21]. The results are shown in Fig. 10. We can see that OADSD could also performs better than those supervised methods. On CICIDS dataset, our OADSD could out perform the Bayes about 48% AP and about 10% AUC improvement over SVM.

OADSD adopts DDFE to derive the flow features dynamically without the knowledge of experts. We now evaluate the effectiveness of DDFE. We choose the CICIDS 2017 Dataset [37] with 20000 benign and 2000 malicious features and use T-SNE [41] to compress the features into two dimension. The visualization results show the center of the malicious traffic is far from the benign traffic. We also present the visualization results of SFSC and SIFD in Fig. 11(b) and Fig. 11(c). This implies DDFE could generate more separable features than other methods, which contributes much to the detection results.



(a) OADSD (b) SFSC (c) SIFD

Fig. 11. The T-SNE [41] visualization of different feature extraction method under CICIDS data set. The green dots represents malicious traffic, the yellow dots represents normal traffic.

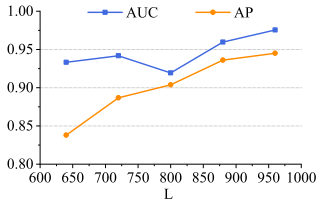


Fig. 12. Different L.

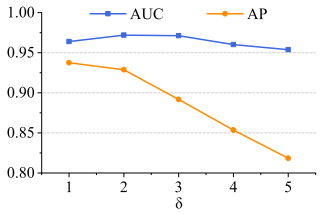


Fig. 13. Different δ.

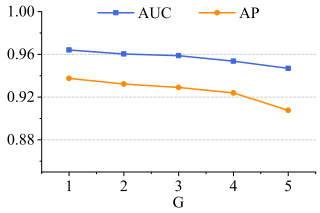


Fig. 14. Different G.

2) Performance Under Different Settings: In this part, we will test the performance of OADSD under different parameters settings on CICIDS dataset. L controls the length of flow used for feature extraction. Fig. 12 shows that with the increasing L, the performance also increases, since the feature extractor could use more flow data and derive better features. However, a larger L also leads to a longer training and processing time. Setting the L to 900 can ensure a good performance in most cases. Fig. 13 shows that the AUC and AP score under different δ which controls the regularization degree. A large δ will make the weight vector stay around its initial point. From the result, setting δ around 2 can produce the best performance, larger δ will greatly affect the AP score. G is defined as the maximum element of the weight vector. Fig. 14 shows the AUC and AP score under different G. The AUC and AP decrease with the increase of G. The reason for this is that a large G will lead to a large learning rate, which makes the weight vector fluctuate and converge slowly.

We add skip connections to the auto-encoder to accelerate the training speed in the DDFE (see § III-A). We will show its benefits in the following. We trained the IF on three different feature sets under CICIDS dataset: (1) feature extracted by autoencoder without skip connections (AE); (2) features extracted by autoencoder with skip connections(AE+skip); (3) expertly designed flow-level features provided by the CICIDS2017 dataset. The results are shown in Fig 15. Features extracted by the autoencoder with skip connections achieves the best performance and other two methods by a large margin. Also, compared to flow-level features, the features extracted by autoencoder without skip connections achieves lower performance, and has the largest deviation among these three feature sets. The reason is that on the one hand, autoencoder is hard to train and difficult to converge to global optimal weights; On the other hand, the training objective of autoencoder makes the model focus on the fine-grained information of the raw traffic, the patterns of different attacks are more on the high-level.

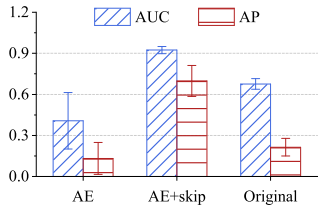


Fig. 15. Skip connection.

TABLE IV
THE HYPER-PARAMETERS OF THE SYNTHESIZE DATASET

Dataset	Center x	Center y	Variance	Proportion
Before Drift	3.0	4.0	0.5	0.85
	5.0	5.0	0.7	0.15
After Drift	3.6	4.6	0.6	0.85
	4.9	4.8	0.7	0.15

3) Adaptation to Environment Change: The OADSD can adapt to the environment change, as "concept drift", which could be defined as the joint distribution shift of input variable x and y. This is a common challenge in the traffic anomaly detection [6]. The offline training methods are not suitable for adapting to the changing environment, because of the long training time.

In this experiment, we test the performance of OADSD under a two-dimensional synthetic test dataset to evaluate the adaptability to environment change. We measure the AUC and AP of OADSD and Isolation Forest (IF), which could be seen as an offline version of our detection algorithm. The synthetic dataset is consisted by two major parts, and both of them are sampled from a two-dimensional normal distribution. Each of these parts is controlled by two main parameters, are the center of the mass and the variance of the distribution. The detailed parameters are shown in Table IV.

TABLE V
PERFORMANCE AFTER CONCEPT DRIFT HAPPENED

Description	Method	AUC	AP
Trained on *After Drift*, tested on *After Drift*.	OADSD	0.830	0.380
	SIFD	0.781	0.331
Trained on *Before Drift*, tested on *After Drift*.	OADSD	0.817	0.381
	SIFD	0.733	0.30

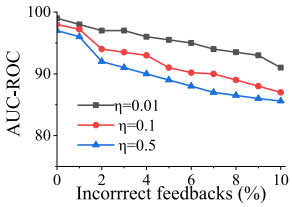


Fig. 16. Incorrect feedbacks.

The before and after dataset has a different distribution, which means the parameters learnt from *Before Drift* dataset cannot transfer smoothly to *After Drift* dataset. The experiment result shows in Table V also proves this. Because the distribution of the training and testing data is different, the AUC of baseline drops from 0.78 to 0.73 (i.e. 10% drops on AUC) and AP of baseline drops from 0.33 to 0.30 (i.e. 6% drops on AP), when trained the baseline on *Before Drift* and tested on *After Drift*.

After a few times of feedback, which only about 2% of the total training data, the AUC and AP score increase back to 0.82 and 0.38. Which is equivalent to the performance by training OADSD on the *After Drift* dataset (i.e. 0.83, 0.38). But in comparison, the IF could only achieve 0.73 AUC and 0.30 AP, which was 30% lower on AP and 21% lower on AUC compares to OADSD.

The above result shows OADSD could well adapt to the environment change. But the offline version (e.g. SIFD) could not adapt to environment change like OADSD. The reason is that OADSD could fully utilize the feedback of IT administrator and tunes its parameters based on the current context.

In our assumption, the feedback from IT administrator is always correct. However, this is not necessarily true, since humans are notoriously prone to mistakes. And this may deteriorate the performance of OEIF.

Therefore, we test the performance of OEIF under different ratio of incorrect feedback under a two-dimensional synthetic dataset. We measure the performance of OEIF by AUC score. How the synthetic dataset is constructed has been described in the previous section. We use the *Before Drift* dataset, mentioned in the Table IV, in this experiment to evaluate the performance.

In reality, the false feedback from operators could hurt the detection accuracy. We test the performance of OADSD under different ratio of incorrect feedback. Fig. 16 demonstrates the performance of OADSD. We can see that OADSD maintains steady when there are small number of incorrect feedbacks. Also, smaller learning rate can eliminate the side effect of incorrect feedbacks.

C. Testbed Evaluation

In this section, we will evaluate the performance of OADSD with a small testbed. The OADSD contains three parts, i.e., Distributed Dynamic Feature Extraction (DDFE), On-demand Evolving Isolation Forest (OEIF) and On-demand Update, where both the DDFE and the OEIF are deployed at the edge devices, and the On-demand Update module is deployed at the controller. We construct a small testbed with five dell servers, each is equipped with Intel i5-10210U@1.6HZ CPU (8 cores) and 8GB RAM. Three servers install software routers that realize routing functions by the X86 architecture and user-defined software. The servers connect each other with 1Gbps bottleneck link. One server performs as the controller. The controller will configure and communicate with the routers through the TCP connections. Each second, the last server will randomly choose one router and setup a TCP connection with it. For each connection, we randomly generate an integer p between 0 and 100 for each link. If $p/100$ is smaller than a pre-config threshold $D\%$, the TCP connection will contain malicious traffic, otherwise, it is benign. We also setup a tester to generate background traffic to each site to emulate the internet traffic. Each connection will last x seconds, where the default value of x is 10 in our evaluations. The default pre-defined false alarm rate threshold (i.e., Φ) is 10% and default update trigger threshold Δ is 5% in our evaluations. Each group of experiments will contain 1000 TCP connections and we repeat each group 10 times.

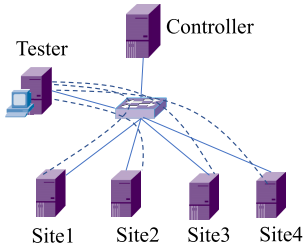


Fig. 17. Testbed topology.

Fig. 18 shows that the OADSD consistently keeps high performance under different threshold and the performance advantage is larger with D. The odds are up to 30% in our evaluations, which matches the results of the simulations. Fig. 19 demonstrates that OADSD can adjust its performance according to the feedback from the IT administrators, while other schemes have poor performance when the testing environment is different from the training environment. Fig. 20 depicts the resource utilization. We can see that both the CPU and memory utilization is below 10% even the arrival traffic approaches the line rate (e.g., 1Gbps). The small utilization indicates OADSD can work in an efficient manner. When the gap between the observed false alarm rate (e.g., Ψ) and default pre-defined false alarm rate threshold (i.e., Φ) is larger than a pre-defined threshold Δ, we will perform parameter update process to restore the parameters in our evaluations. We now show the number of feedbacks needed with different Ψ and Δ. Fig. 21 shows that OADSD only needs less than 60 feedbacks when the gap is larger than 20%, which implies some cost for the involvement of the IT operator

V. RELATED WORK

A. Feature Extraction

Feature extraction is the start point of utilizing machine learning based methods in traffic anomaly detection.

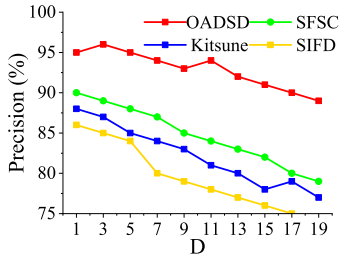


Fig. 18. Different D.

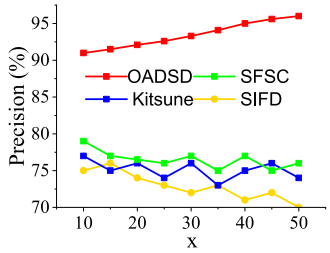


Fig. 19. Different x.

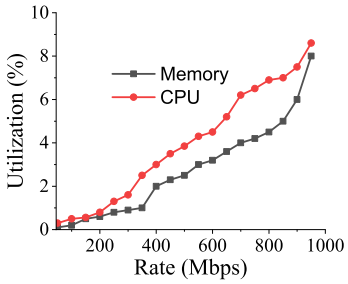


Fig. 20. Resource utilization.

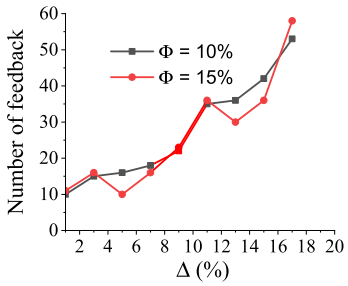


Fig. 21. Feedback number.

Reference [37] propose a traffic feature extractor based on the flow-level statistical features in network traffic. Reference [28] utilize the per-packet features and use a feature clustering method to evaluate the correlation between features. References [1], [3], and [35] used automatic feature compression methods on the statistical traffic features, which could preserve the most representative part of the feature set. Compared with the statistical features, these methods could achieve a better performance. Reference [32] leveraged the download and upload size interval to get the website fingerprint from network traffic. Reference [42] uses a CNN model to extract features from raw traffic data for traffic classification.

B. Machine Learning Based NIDS

Machine learning based Network Intrusion Detection Systems (NIDS) can detect zero-day attacks and achieve a higher detection accuracy, compared with traditional rule based methods. References [4] and [23] adopted statistical machine learning methods, i.e., Wavelet Analysis, PCA, to detect malware traffic in an unsupervised manner. Although, these methods have a small detection overhead. They often have a low detection accuracy and is sensitive to outliers. References [21] and [27] detected malicious traffic, i.e., DDoS, Black Holes, by using flow-level statistical features and Naive Bayes Classifiers. However, these methods could not achieve real-time detection. Reference [28] proposed Kitsune which aims to reduce the detection overhead by using a lightweight deep learning models, i.e., autoencoders.

C. Malicious Traffic Detection

Online Traffic Anomaly Detection methods are widely investigated [2], [28], [30], [31], [44]. [30], [31] combine the online and offline machine learning methods to make the methods easier to train and can identify new traffic based on the prediction result of online and offline algorithms. Reference [2] proposed a fast activation function that helps the deep neural network converge faster, which is then used to detect DDoS attacks. Reference [28] provide an online schema for training the traffic detection model, which achieves a low training overhead. Reference [44] leverage deep dictionary learning to achieve online anomaly detection for encrypted traffic. However, unlike OADSD, these methods do not guarantee that the model will converge.

VI. Conclusion

In this paper, we developed the OADSD, an online malicious traffic detection system, which contains traffic collections, traffic feature Extraction and time evolving anomaly detection. Distributed Dynamic Feature Extraction (DDFE) can extract features directly from the raw traffic and On-demand Evolving Isolation Forest (OEIF) utilizes expert feedback to update parameters which makes the system have the ability of adapting to the environment. We prove that the OEIF has a sub-linear regret bound, which means the performance gap between OEIF and the best hypothesis goes to zero as update iterations go to infinity. Extensive experiments show that the OADSD can achieve an AUC score greater than 0.9 in most cases. The OADSD can still achieve good performance in the environment which has a different distribution between the training dataset with the help of the expert feedback.

References

[1] M. Abdallah, N. A. L. Khac, H. Jahromi, and A. D. Jurcut, "A hybrid CNN-LSTM based approach for anomaly detection systems in SDNs," in *Proc. 16th Int. Conf. Availability, Rel. Secur.*, New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1–7.

[2] K. Alrawashdeh and C. Purdy, "Fast activation function approach for deep learning based online anomaly intrusion detection," in *Proc. IEEE 4th Int. Conf. Big Data Secur. on Cloud (BigDataSecurity), IEEE Int. Conf. High Perform. Smart Comput., (HPSC)*, May 2018, pp. 5–13.

[3] S. Boukria and M. Guerroumi, "Intrusion detection system for SDN network using deep learning approach," in *Proc. Int. Conf. Theor. Applicative Aspects Comput. Sci. (ICTAACS)*, vol. 1, 2019, pp. 1–6.

[4] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Combining sketches and wavelet analysis for multi time-scale network anomaly detection," *Comput. Secur.*, vol. 30, no. 8, pp. 692–704, 2011.

[5] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, 2011.

[6] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," *IFAC Proc. Volumes*, vol. 46, no. 20, pp. 12–17, 2013.

[7] Z. Duliński, R. Stankiewicz, G. Rzym, and P. Wydrych, "Dynamic traffic management for SD-WAN inter-cloud communication," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1335–1351, 2020.

[8] H. S. Emadi and S. M. Mazinani, "A novel anomaly detection algorithm using DBSCAN and SVM in wireless sensor networks," *Wireless Pers. Commun.*, vol. 98, no. 2, pp. 2025–2035, 2018.

[9] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016.

[10] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 3431–3446.

[11] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in SDN: A social multimedia perspective," *IEEE Trans. Multimedia*, vol. 21, no. 3, pp. 566–578, Mar. 2019.

[12] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from Googles network infrastructure," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 1–15.

[13] K. Han, Y. Wang, C. Zhang, C. Li, and C. Xu, "Autoencoder inspired unsupervised feature selection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 2941–2945.

[14] S. Han et al., "Log-based anomaly detection with robust feature extraction and online learning," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2300–2311, 2021.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 770–778.

[16] C.-Y. Hong et al., "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf.*, Hong Kong, Aug. 2013, pp. 15–26.

[17] C.-Y. Hong et al., "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proc. Conf. ACM Special Interest Group Data Commun.*, New York, NY, USA: ACM, 2018, pp. 74–87.

[18] S. Jain et al., "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, New York, NY, USA: ACM, 2013, pp. 3–14.

[19] E. Kabir, J. Hu, H. Wang, and G. Zhuo, "A novel statistical technique for intrusion detection systems," *Future Gener. Comput. Syst.*, vol. 79, pp. 303–318, Feb. 2018.

[20] P. Karczmarek, A. Kiersztyn, W. Pedrycz, and E. Al, "K-means-based isolation forest," *Knowl.-Based Syst.*, vol. 195, May 2020, Art. no. 105659.

[21] M. Klassen and N. Yang, "Anomaly based intrusion detection in wireless networks using Bayesian classifier," in *Proc. IEEE 5th Int. Conf. Adv. Comput. Intell. (ICACI)*, Oct. 2012, pp. 257–264.

[22] A. Kumar et al., "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proc. ACM Conf. Special Interest Group Data Commun.*, New York, NY, USA, 2015, pp. 1–14.

[23] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun.* New York, NY, USA: Association for Computing Machinery, 2004, pp. 219–230.

[24] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 249–260.

[25] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net: A flow sequence network for encrypted traffic classification," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1171–1179.

[26] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM Conf. SIGCOMM*, New York, NY, USA, 2014, pp. 527–538.

[27] T. Liu, A. Qi, Y. Hou, and X. Chang, "Method for network anomaly detection based on Bayesian statistical model with time slicing," in *Proc. 7th World Congr. Intell. Control Automation*, 2008, pp. 3359–3362.

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

[29] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, 2015, pp. 1–6.

[30] M. Odiathevar, W. K. Seah, and M. Frean, "A hybrid online offline system for network anomaly detection," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2019, pp. 1–9.

[31] M. Odiathevar, W. K. Seah, M. Frean, and A. Valera, "An online offline framework for anomaly scoring and detecting new traffic in network streams," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5166–5181, Nov. 2021.

[32] A. Panchenko et al., "Website fingerprinting at internet scale," in *Proc. NDSS*, 2016, pp. 1–15.

[33] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 10, pp. 2825–2830, Jul. 2017.

[34] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," 2018, *arXiv:1803.10769*.

[35] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Network anomaly detection using LSTM based autoencoder," in *Proc. 16th ACM Symp. QoS Secur. Wireless Mobile Netw.* New York, NY, USA: Association for Computing Machinery, 2020, pp. 37–45.

[36] S. Kumar, "Survey of current network intrusion detection techniques," 2007. [Online]. Available: http://www.cse.wustl.edu/~jain/cse571-07/ftp/ids.pdf

[37] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, vol. 1, Jan. 2018, pp. 108–116.

[38] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2012.

[39] V. K. Singh and M. Govindarasu, "Decision tree based anomaly detection for remedial action scheme in smart grid using PMU data," in *Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM)*, Aug. 2018, pp. 1–5.

[40] S. Troia, F. Sapienza, L. Varé, and G. Maier, "On deep reinforcement learning for traffic engineering in SD-WAN," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2198–2212, Jul. 2020.

[41] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 1–27, 2008.

[42] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE Int. Conf. Intell. Secur. Inform. (ISI)*, Jul. 2017, pp. 43–48.

[43] Z. Wang, K. W. Fok, and V. L. L. Thing, "Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study," *Comput. Secur.*, vol. 113, Feb. 2022, Art. no. 102542.

[44] J. Xing and C. Wu, "Detecting anomalies in encrypted traffic via deep dictionary learning," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 734–739.

[45] B. Yan, G. Han, Y. Huang, and X. Wang, "New traffic classification method for imbalanced network data," *J. Comput. Appl.*, vol. 38, no. 1, pp. 20–25, 2018.

[46] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (SD-WAN): Architecture, advances and opportunities," in *Proc. 28th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2019, pp. 1–9.

[47] H. Zhang et al., *Boosting Bandwidth Availability Over Inter-DC WAN*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 297–312.

[48] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 25–30.

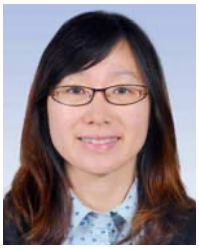
[49] J. Zhou, Y. Fu, Y. Wu, H. Xia, Y. Fang, and H. Lu, "Anomaly detection over concept drifting data streams," *J. Comput. Inf. Syst.*, vol. 5, no. 6, pp. 1697–1703, 2009.

Pei Zhang received the Ph.D. degree from the Beijing University of Posts and Telecommunications, in 2012. He is currently with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications. His research interests include computer networks, network security, and AI.

Fangzhou He received the B.S. degree in software engineering from Northwestern Polytechnical University. He is currently pursuing the M.S. degree with the Beijing University of Posts and Telecommunications. His research interests include network security and machine learning.

Han Zhang (Member, IEEE) received the B.S. degree in computer science and technology from Jilin University and the Ph.D. degree from Tsinghua University. He is currently working with Institute for Network Sciences and Cyberspace, Tsinghua University. He has published more than 50 papers in his research areas. His research interests include computer networks and network security.

Jiankun Hu (Senior Member, IEEE) received the Ph.D. degree in control engineering from the Harbin Institute of Technology, China, in 1993, and the master's degree in computer science and software engineering from Monash University, Australia, in 2000. He was a Research Fellow with the Delft University of Technology, The Netherlands, from 1997 to 1998, and Melbourne University, Australia, from 1998 to 1999. He was with Ruhr University, Bochum, Germany. He is currently a Full Professor and the Research Director of the Cyber Security Laboratory, School of Engineering and Information Technology, University of New South Wales, Australia. His main research interests include cyber security and biometrics security, where he has published many papers in high-quality conferences and journals, including the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. He received the prestigious German Alexander von Humboldt Fellowship from Ruhr University, from 1995 to 1996. He has received seven Australian Research Council (ARC) Grants and serves at the prestigious Panel of Mathematics, Information and Computing Sciences and the ARC Excellence in Research for Australia Evaluation Committee. He was the Security Symposium Chair of the IEEE ICC and the IEEE Globecom. He has served on the editorial board for seven international journals, including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY.



Xiaohong Huang received the B.E. degree from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2000, and the Ph.D. degree from the School of Electrical and Electronic Engineering (EEE), Nanyang Technological University, Singapore, in 2005. Since 2005, she has been with BUPT, where she is currently a Professor and the Director of the Network and Information Center, School of Computer Science (National Pilot Software Engineering School). She has published more than 50 academic papers in the area of WDM optical networks, IP networks, and other related fields. Her current research interests include the performance analysis of computer networks and service classification.

Jilong Wang received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2000. He is currently a Professor with Tsinghua University. His research interests include network measurement, location-oriented networks, SDN systems, and network security.

Xia Yin (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997, and 2000, respectively. She is a Full Professor with the Department of Computer Science and Technology, Tsinghua University. Her research interests include future internet architecture, formal method, protocol testing, and large-scale internet routing.

Huahong Zhu is a Senior Engineer. Her research interests include 5G& IP communication technology, big data, and AI.

Yahui Li received the B.S. degree from the School of Software Engineering, Jilin University, Changchun, China, and the Ph.D. degree from Tsinghua University, Beijing, China. She is currently with the Institute for Network Sciences and Cyberspace, Tsinghua University. Her research interests include computer network systems and network security.