Discovery method for distributed denial-of-service attack behavior in SDNs using a feature-pattern graph model[*]

Ya XIAO[†1], Zhi-jie FAN[†‡1,2,3], Amiya NAYAK[†2], Cheng-xiang TAN[1]

[1]College of Electronics and Information Engineering, Tongji University, Shanghai 201804, China

[2]School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa K1N 6N5, Canada

[3]The Third Research Institute of the Ministry of Public Security, Shanghai 200120, China

[†]E-mail: 1710053@tongji.edu.cn; aaronzfan@126.com; nayak@uottawa.ca

Abstract: The security threats to software-defined networks (SDNs) have become a significant problem, generally because of the open framework of SDNs. Among all the threats, distributed denial-of-service (DDoS) attacks can have a devastating impact on the network. We propose a method to discover DDoS attack behaviors in SDNs using a feature-pattern graph model. The feature-pattern graph model presented employs network patterns as nodes and similarity as weighted links; it can demonstrate not only the traffic header information but also the relationships among all the network patterns. The similarity between nodes is modeled by metric learning and the Mahalanobis distance. The proposed method can discover DDoS attacks using a graph-based neighborhood classification method; it is capable of automatically finding unknown attacks and is scalable by inserting new nodes to the graph model via local or global updates. Experiments on two datasets prove the feasibility of the proposed method for attack behavior discovery and graph update tasks, and demonstrate that the graph-based method to discover DDoS attack behaviors substantially outperforms the methods compared herein.

Key words: Software-defined network; Distributed denial-of-service (DDoS); Behavior discovery; Distance metric learning; Feature-pattern graph

CLC number: TP39

1 Introduction

A software-defined network (SDN) provides a newly structured network with decoupled data, control, and application planes. The data plane consists of OpenFlow switches and other network equipment such as a switch, which receives commands from the controller and treats them as the programmed rules. The controller in the control plane can command the forwarding actions of switches through programming, which simplifies the network management. The application plane provides application programming interfaces (APIs) for the network management applications. With this open, centralized, and programmable architecture, SDNs afford developers a convenient way to experiment and deploy new ideas.

However, SDNs have many vulnerabilities (Scott-Hayward et al., 2013). Attackers can launch denial-of-service attacks in the communication between the controller and switch to paralyze the network. Once an SDN switch is compromised by an attacker, the flow table can be easily modified. Malicious applications and unauthorized application access may appear on the application plane. Among the well-known vulnerabilities of SDNs, distributed denial-of-service (DDoS) attacks can have a devastating impact on the whole network, so more attention is required when building the network security mechanism for an SDN.

Creating a security mechanism against DDoS attacks on an SDN presents several challenges. First, the traditional statistics- and signature-based techniques need to be updated and adapted to DDoS attacks in an SDN environment. Since the size of ternary content-addressable memory (TCAM) that stores the flow rules is limited, the challenge rests in how to collaboratively use TCAM and reduce the usage rate when the mechanism is deployed in an SDN. Second, because the traffic in SDNs is dynamic and attack approaches are complex nowadays, it is difficult to use the attack detection technology with only the traffic header information to discover different types of attacks. Third, there are an increasing number of unknown attacks that SDNs cannot prevent; thus, it is challenging to automatically detect the unknown attacks.

Considering these challenges, we propose a discovery method for DDoS attack behaviors in SDNs using a feature-pattern graph (FPG) model. The proposed model is scalable to updates and can be extended to other attack scenarios. We use the DDoS attack dataset in the traditional network and simulate attack traffic in an SDN environment as signatures to build an FPG for known attacks. The non-flow data in the traditional network are transferred into the flow types which fit the SDN environment. To cope with the limited TCAM problem, we directly monitor and collect the packets and parse them into the flow table format instead of picking the flow table data from TCAM. Metric learning and the Mahalanobis distance are used to model the similarity of the different nodes in FPG. With each node representing a network pattern and the weighted edges indicating the feature-based similarity of nodes, the FPG model can demonstrate the node relationships properly. A graph-based attack detection method is proposed, which uses not only the traffic header based features, but also the relational context of network patterns. The FPG model is also scalable to updates for the unknown attacks which are not in the predefined graph. Our contributions can be summarized as follows:

1. We propose a method for modeling an FPG with link weight learning. The nodes represent various network patterns and the links between nodes denote the similarity. The graph model is scalable to updates and can be used in other attack scenarios.

2. We propose a DDoS detection method based on the FPG.

3. A graph update model with both local and global updates is proposed to extend FPG and help with the detection of new attacks.

4. We demonstrate that our DDoS detection method performs better than the compared methods and also show the effectiveness of the graph update model with two datasets.

2 Related work

2.1 SDNs

The switch and controller are two essential parts of an SDN. The network switches in the data plane act as the forwarding devices, and the controller in the control plane implements the functionality and control logic. The OpenFlow protocol (Scott-Hayward et al., 2013) is the most widely deployed protocol in SDNs, and it defines the communication approach between an SDN controller and the switches. The flow table proposed in the OpenFlow protocol is the foundation for querying and forwarding a package. The structure of the flow table in the OpenFlow protocol consists of six parts: match fields, priority, counters, instructions, timeouts, and cookies. When a data packet arrives, the packet header is compared with the match fields, and if there is a match, the switch will update the counters and execute the actions; otherwise, the packet is sent to the controller through the secure channel. The match fields contain the source and destination addresses of the Ethernet, source and destination IP addresses, source and destination TCP/UDP ports, etc. The priority defines the order of matching flow items. Counters count the relevant information such as packet count and byte count. Instructions define the actions such as forward, drop, or modify. Timeouts define the longest time during which a flow exists and the longest time during which a flow exists in a flow table if no packet matches the flow. Numerous open-source SDN controllers, such as Floodlight, OpenDaylight, POX, and Ryu (Kreutz et al., 2015), can be deployed in SDNs.

2.2 Security threats in SDNs

SDNs have attracted considerable attention from the attackers because of the open framework. We use the taxonomy in Fan et al. (2019) to illustrate

the security threats in SDNs.

1. Application plane security

The application plane consists mainly of different types of applications, some of which will play an essential part in creating the flow rules. Attacking these applications can cause significant damage to the SDN. Attackers can inject malicious code into the applications, rewrite the code or use a fake identity, and access the SDN illegally.

2. Protocol security

The protocols in SDN refer mainly to the southbound and northbound interface protocols. OpenFlow is a typical protocol for the southbound interface; although it is widely used in SDN frameworks, it still has some vulnerabilities (Klöti et al., 2013). For example, the communication between the switch and controller has no identification or access control. Although OpenFlow can establish a security channel with secure sockets layer/transport layer security (SSL/TLS) to encrypt the data, it is still an option rather than being mandatory. When the switch is establishing a connection with the OpenFlow protocol, both the switch and controller have to send the OF_HELLO message to each other. The attacker can cut off the connection by intercepting the message.

3. Control plane security

Because the controller is the most significant part in an SDN, an attacker who can successfully hack into the controller will have full control over the entire network. The vulnerabilities on the application plane or controller itself can be used by the attackers to control the controller illegally. In addition, the attacker can launch a flooding attack by using the vulnerabilities of the switch or OpenFlow protocol, and by using the compromised switch (Antikainen et al., 2014) to send a large number of packets to the controller. The attacker can send packets that do not match the flow table, so the switch will send the packets to the controller; as a result, too many packets will lead to DoS/DDoS attacks and disable the controller.

4. Data plane security

The switch is the crucial part of the data plane, as it can parse and forward packets, learn the media access control (MAC) address table, learn the address resolution protocol (ARP) request, etc. Different kinds of attacks can occur on this plane, like the DoS/DDoS attack. An attacker can use the fa-

cilities on the data plane to send a number of new and unknown flows targeting the controller. He/She can also potentially overload the switch memory, i.e., TCAM, thus making it difficult for the switch to install flow rules for normal use.

Among all the threats on the different planes in an SDN, the DDoS attack is the most threatening one since it can be launched on various planes and can have a devastating impact on the whole SDN environment. Therefore, we concentrate mainly on the security mechanism on DDoS attacks in this study.

2.3 Discovery of DDoS attack behaviors in SDNs

DDoS attacks aim at bringing down target services by distributed multiple sources, and have been critical threats in cloud security. The distinctive features of SDNs offer new opportunities to detect and mitigate DDoS attacks. Yan et al. (2016) discussed the potential DDoS vulnerabilities on different planes of the SDN platform, provided a survey of defense mechanisms against DDoS attacks using SDNs, and also discussed the open issues in dealing with DDoS attacks in SDNs.

The popular methods in DDoS attack detection can generally be divided into four categories (Bawany et al., 2017), i.e., entropy, machine learning, traffic analysis, and intrusion detection system/ intrusion prevention system (IDS/IPS) based methods. Table 1 depicts related research for each category.

Entropy is used to measure the randomness of an SDN environment; when a DDoS attack occurs, the randomness is reduced, leading to a lower value of entropy. Entropy-based methods calculate entropy and then detect the DDoS attack by filtering the entropy smaller than a threshold. Giotis et al. (2014) defined four flow-related traffic features to calculate entropy: source IP address (srcIP), destination IP address (dstIP), source port (srcPort), and destination port (dstPort). When a DDoS attack occurs, the entropies on dstIP and dstPort show a significant decrease. Based on this phenomenon, entropy-based calculations are periodically conducted on the NOX controller to implement the anomaly detection algorithm. Wang R et al. (2015) proposed an entropy-based lightweight DDoS flooding attack detection model running in the OpenFlow switch. The information entropy of an edge switch was

Table 1 Global view of methods for DDoS attack detection in SDNs

Method	Research	Description
Entropy	Giotis et al. (2014)	DDoS detection by entropy on dstIP and dstPort
	Wang R et al. (2015)	DDoS detection by entropy of edge switches
Machine learning	Braga et al. (2010); Xu and Liu (2016)	SOM-based normal or attack classification with predefined flow features
	Niyaz et al. (2017)	SAE-based deep learning approach for classifying different kinds of DDoS attacks
Traffic analysis	Yu et al. (2012)	Traffic similarity based botnet attack detection
	Wang B et al. (2015); AlEroud and Alsmadi (2017)	Traffic pattern graph based DoS/DDoS inference
IDS/IPS based	Chung et al. (2013); AlEroud and Alsmadi (2017)	Use Snort to classify traffic and raise anomaly alerts
	Aziz and Okamura (2017)	Leverage Suricata to detect the SMTP flood attack traffic

dstIP: destination IP address; dstPort: destination port; SOM: self-organizing map; SAE: stacked auto encoder; SMTP: Simple Mail Transfer Protocol

calculated, where each probability was estimated by the frequency of each IP address. When the difference in value between average entropy and real-time entropy was larger than a threshold value, DDoS alert was triggered. The entropy-based methods have a relatively low computation load and few limitations; however, when the values of the features are continuous, relevant information about the feature distribution can be lost, which will lead entropy-based methods to be prone to errors in some scenarios (Fiadino et al., 2015).

Machine learning based methods have been widely used in DDoS attack detection in both traditional networks and SDNs. This kind of method treats attack detection as a classification problem, by learning the feature patterns to classify a newly coming flow as an attack or benign flow. Braga et al. (2010) introduced a lightweight method for DDoS attack detection based on traffic flow features. A six-tuple of features was proposed as the most likely six features to influence the decision of whether there was a DDoS attack in the SDN. A self-organizing map (SOM) was used to classify the network traffic. The six features include average of packets per flow, average of bytes per flow, average of duration per flow, percentage of pair-flows, growth of single-flows, and growth of different ports. Niyaz et al. (2017) developed a deep learning based multi-vector DDoS detection system. They used the stacked auto encoder (SAE) to classify the flows into types such as a Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP) DDoS attack, or benign flow with 68 extracted features. Xu and Liu (2016) pro-

posed DDoS attack defense methods for victim detection and post-detection, leveraging the SDN's flow monitoring capability. For each detection mission, four features were selected and the SOM was used to classify the data by the similarity of the statistical features. Although machine learning based methods are widely used with a great success, their performance typically depends on the training datasets and selection of features.

In traffic pattern based attack detection an assumption is made that there is a significant difference between the patterns in attack traffic and benign traffic. Yu et al. (2012) used traffic similarity to detect attacks from botnets, and they hypothesized that attack traffic from a botnet has a high similarity which is approximately equal to one, and the benign patterns coming from different users thus have a low similarity close to zero. Wang B et al. (2015) built a relational graph with known traffic patterns and their labels, in which the maximum spanning tree was used to select features and maximum a posterior query was applied to detect an attack. This method is effective, inexpensive, and has low overhead. AlEroud and Alsmadi (2017) proposed a Markov-based graph model with DoS attack pattern as nodes and their relationships as edges, in which the traffic features were used to describe the nodes. The traffic feature based similarity was used to find the nearest nodes as the traffic class. However, both Wang B et al. (2015) and AlEroud and Alsmadi (2017) considered only the features that can be extracted directly from a packet or flow content, which may not be sufficient in most circumstances.

IDS/IPS based methods refer to those attack

detection methods relying on popular open-source network intrusion detection systems (IDSs) or intrusion prevention systems (IPSs), like Snort (Roesch, 1999) or Suricata (Albin and Rowe, 2012). The mirroring-based network intrusion detection agent in NICE (Chung et al., 2013) used Snort to sniff the mirroring port on each virtual bridge in Open vSwitch, and sent alerts when suspicious or anomalous traffic was detected. A scenario attack graph and an alert correlation graph were constructed to analyze the vulnerabilities, alerts, and traffic to select a proper countermeasure. To reduce the false alerts raised by Snort, NICE provided an approach to match if the alert was related to a vulnerability. AlEroud and Alsmadi (2017) employed Snort to label suspicious or benign types of flows. Aziz and Okamura (2017) leveraged Suricata to detect the Simple Mail Transfer Protocol (SMTP) flood attack traffic and proposed FlowIDS as a supplement to collect the anomaly traffic omitted by Suricata. This method can help with the labeling of traffic, but relies highly on third-party tools which may cause an error propagation problem.

In this study, we combine the entropy and traffic pattern analysis based methods and leverage the merits of each method in the design of our security mechanism. We consider not only the directly extracted features but also the latent features in contrast with Wang B et al. (2015) and AlEroud and Alsmadi (2017).

3 Proposed model

Fig. 1 shows the procedures in the proposed automated feature-pattern graph based DDoS attack detection mechanism for SDNs.

Since AlEroud and Alsmadi (2017) have proved the feasibility of using the existing attack signature to identify attacks on SDNs, the proposed model takes the advantage of this assumption and jointly uses the attack signatures extracted from traditional datasets and the data from the SDN environment to build the attack signatures in SDN.

A feature-pattern graph is built with the attack signatures and is also used to identify whether an incoming flow is malicious. The patterns are the network status such as TCP DDoS attack or benign, and the features are those who can properly describe the status of network patterns, like fraction of TCP flows. The number ① lines in Fig. 1 are

Fig. 1 Procedures in the proposed model

the procedures of graph model generation. The traditional datasets and data in an SDN environment are required as the input for this part. We directly monitor and collect the traffic packets in the SDN environment and parse them into the flow format. The packet data in a traditional network are transferred into the flow type by the flow creator. If the data are not classified, we need to label the data into different types to build the graph model. Number ② lines are the attack detection processes. The real-time traffic packets are monitored, collected, and parsed into flow tables and then fed into the graph-based attack detection model. If the newly coming flow is malicious, a countermeasure is selected to prevent the SDN from a further attack. When a traffic cannot be detected by graph-based attack detection, but the traffic tends to be malicious, the old graph needs to be updated and used for attack detection of subsequent flows (number ③ lines).

3.1 Creation of attack signatures in an SDN

To create attack signatures in an SDN, i.e., creating flows for the packets that are not in the flow format, we first need to install the flow rules based on the packet data from the SDN or traditional network, and then label the flow-based data into different patterns. Since TCP, UDP, and ICMP are the three most widely used protocols in network communication and DDoS attacks, we typically consider these three protocols in our study. A flow in TCP/UDP/ICMP is a group of packets having the same header fields. For TCP flows, the header fields contain protocol, source and destination MAC addresses, source and destination IP addresses, duration time, source and destination ports, flag, and packet size. For UDP flows, the header fields are

similar to TCP flows except that UDP flows have no flag. The ICMP flows do not have source and destination ports but ICMP type and code.

Algorithm 1 shows the procedures for creating the attack signatures in an SDN. "packets" is a list of packets that need to be processed, "header_fields" is the list of header fields that is predefined manually for each protocol. "flows" is a hashmap with "flow_id" (an auto increasing number from 0) as the key and list of packets' indices as the value. "flow$_{\text{fields}}$" is also a hashmap, where the key is "flow_id" and the value is the list of values of header fields. For each packet in the "packets" list, we extract the header fields and check if the field is already in the existing flow list. If so, update the corresponding list in "flows;" if not, create a new "flow_id" and add the extracted fields in "flow$_{\text{fields}}$." After traversing all the packets, the packets are classified by the flows and stored in "flows," and the fields of each flow are stored in "flow$_{\text{fields}}$." The flow rules will be installed to the switch according to the fields in "flow$_{\text{fields}}$." "label_flow(flows)" is a function to label the flows as different patterns if the data are not classified. The patterns of flows depend on the pattern of corresponding packets; for example, if the majority of packets in a flow are TCP flooding attack, then the flow is labeled as a TCP flooding attack.

3.2 Feature-pattern graph based DDoS attack detection

After the creation of DDoS attack signatures in the SDN, we propose our attack detection module, which is built on neighborhood classification in a feature-pattern graph model. The module is divided into two sub-modules, i.e., graph creation and detection engine.

3.2.1 Feature-pattern graph

Each node in our proposed graph model represents a type of DDoS attack or benign, i.e., network patterns. The DDoS attack patterns can be divided into three types on the basis of the protocol type, i.e., ICMP, UDP, and TCP. Patterns can also be classified by experience or according to the IDS alerts. The links among the nodes are weighted, where the weight represents the similarity between two network patterns.

The DDoS feature-pattern graph in an SDN is denoted as FPG = $\{V, E, \boldsymbol{F}\}$, where $V = \{V_1, V_2, \ldots, V_n\}$ is the set of nodes. Each node V_i represents a network pattern, and n is the total number of network patterns. Each node is specified by features, which are denoted as $\boldsymbol{F} = \{\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_n\}$. $\boldsymbol{F}_i$ $(i = 1, 2, \ldots, n)$ is a feature matrix of node V_i with dimension $l \times k$, where l is the number of samples whose pattern is V_i and k the number of different features. E is the set of links, where e_{ij} is the link weight between nodes V_i and V_j.

3.2.2 Link weight learning

The link weight between two nodes in an FPG represents the feature-based similarity. The Pearson correlation is a widely used method in similarity measurement in intrusion detection systems (Wu et al., 2009; AlEroud and Alsmadi, 2017); however, the Pearson correlation may not be suitable for all the datasets. We use the Mahalanobis distance and distance metric learning (Shen et al., 2010) to learn the proper distance formula given a set of DDoS attack signatures in an SDN.

The Mahalanobis distance between two feature vectors is denoted by

$$\text{dist}_{\text{mah}}^2(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^{\text{T}} \boldsymbol{M}(\boldsymbol{x}_i - \boldsymbol{x}_j)$$
$$= \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2, \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are two feature samples, and $\boldsymbol{M}$ is a symmetric positive semi-definite $d \times d$ matrix which is positive semi-definite and can be factorized as $\boldsymbol{P}^{\text{T}} \boldsymbol{P}$. $\boldsymbol{M}$ is called the metric matrix, and metric learning is to learn matrix $\boldsymbol{M}$ to improve the performance of

---

Algorithm 1 Procedures for creating attack signatures in an SDN

1: **Input:** packets and header_fields
2: Initialize flows = ∅
3: Initialize flow$_{\text{fields}}$ = ∅
4: **for** each packet in packets **do**
5: 　　Extract header fields and store them in field_list
6: 　　**if** field_list in flow$_{\text{fields}}$.values() **then**
7: 　　　　Obtain flow_id from flow$_{\text{fields}}$
8: 　　　　Insert index of packet into flows[flow_id]
9: 　　**else**
10: 　　　　Insert field_list into flow$_{\text{fields}}$
11: 　　　　flows[new flow_id]=[index of packet]
12: 　　**end if**
13: **end for**
14: Install flow rules for flows in flow$_{\text{fields}}$
15: label_flow(flows)

neighbor classification for DDoS attacks.

Neighborhood component analysis is a probabilistic-based metric learning, proposed by Goldberger et al. (2004). For each DDoS training sample $\boldsymbol{x}_i$, the probability of other samples being a neighbor of $\boldsymbol{x}_i$ can be calculated by

$$p_{ij} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2)}{\sum_m \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_m\|_{\boldsymbol{M}}^2)}. \tag{2}$$

The longer the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the lower the probability of $\boldsymbol{x}_i$ being the neighbor of $\boldsymbol{x}_j$. The probability of $\boldsymbol{x}_i$ being correctly classified by other samples is denoted by

$$p_i = \sum_{j \in \Omega_i} p_{ij}, \tag{3}$$

where Ω_i is the set of all samples that have the same network pattern as $\boldsymbol{x}_i$.

The accuracy of all samples is computed by

$$\sum_{i=1}^l p_i = \sum_{i=1}^l \sum_{j \in \Omega_i} p_{ij}. \tag{4}$$

To obtain matrix $\boldsymbol{M}$, the optimization goal is denoted as

$$\min_p \quad 1 - \sum_{i=1}^l \sum_{j \in \Omega_i} \frac{\exp(-\|\boldsymbol{P}^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{P}^{\mathrm{T}}\boldsymbol{x}_j\|_2^2)}{\sum_m \exp(-\|\boldsymbol{P}^{\mathrm{T}}\boldsymbol{x}_i - \boldsymbol{P}^{\mathrm{T}}\boldsymbol{x}_m\|_2^2)}. \tag{5}$$

The stochastic gradient descent can be used to obtain matrix $\boldsymbol{M}$.

After learning the metric matrix $\boldsymbol{M}$, the similarity of two attack patterns can be calculated by Eq. (1), where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are two k-dimensional feature-based vectors of two attack patterns V_i and V_j, denoted as $\boldsymbol{F}_i'$ and $\boldsymbol{F}_j'$ respectively. The method for calculating $\boldsymbol{F}_i'$ and $\boldsymbol{F}_j'$ is as follows. Taking $\boldsymbol{F}_i'$ as an example, for attack pattern V_i the feature matrix can be denoted as

$$\boldsymbol{F}_i = \begin{bmatrix} f_{11}^{(i)} & f_{12}^{(i)} & \cdots & f_{1k}^{(i)} \\ f_{21}^{(i)} & f_{22}^{(i)} & \cdots & f_{2k}^{(i)} \\ \vdots & \vdots & & \vdots \\ f_{l1}^{(i)} & f_{l2}^{(i)} & \cdots & f_{lk}^{(i)} \end{bmatrix}, \tag{6}$$

where each row represents a sample and each column is a particular feature, l is the number of samples, and k is the number of features.

The k-dimensional feature-based vector of attack pattern V_i can be represented as

$$\boldsymbol{F}_i' = \begin{bmatrix} f_1^{(i)'} & f_2^{(i)'} & \dots & f_k^{(i)'} \end{bmatrix}, \tag{7}$$

where the kth element $f_k^{(i)'}$ is the Shannon entropy of the kth feature on all samples that are classified as pattern V_i, i.e., the kth column of matrix $\boldsymbol{F}_i$ in Eq. (6).

3.2.3 Graph model update

In reality, the actual attack pattern may not follow the pattern distribution as in the signature dataset, which means that the FPG needs to be updated with new patterns. An FPG can be updated manually, to the effect that the administrator can manually insert new nodes in the FPG. Another type is to update the pattern nodes automatically, which can be divided into local and global update.

When a new node V_{new} is discovered, local update is to insert the new node V_{new} into the existing graph. First, we calculate the similarity between V_{new} and other existing nodes, and then find the γ nearest nodes whose distances are significantly smaller than others. If γ is larger than one, then we check whether the γ nearest nodes have links among themselves, and new weighted links will be generated between V_{new} and the nodes who connect with each other. The global update is to relearn the metric matrix $\boldsymbol{M}$ and regenerate the entire graph. This can consume much more time and computing resources than the local one, but can obtain a better performance on the subsequent attack detection.

To detect an unknown DDoS attack pattern, i.e., a new node, both the similarity- and entropy-based methods are used. If a new pattern of DDoS attack occurs, the randomness will decrease, and traffic will have a high similarity and a low entropy. When a traffic group cannot be classified into existing attack patterns, the Mahalanobis distance based similarities in the new traffics are higher than a threshold value, and entropy of the new traffic is lower than a threshold, then a new type of DDoS pattern needs to be added into the FPG.

3.2.4 Attack detection

The attack detection module is used to detect the pattern, i.e., DDoS type or benign, of an incoming OpenFlow-based flow according to the weighted

FPG at runtime.

Algorithm 2 shows the steps in detecting the pattern of a newly incoming flow. The algorithm needs two inputs: one is the weighted feature-pattern graph built in the previous part, containing the nodes, weighted links, and the feature matrix for each node; the other input is the $1 \times k$ feature vector of the newly incoming flow, denoted as $\boldsymbol{F}_{\text{new}} = \begin{bmatrix} f_1^{(\text{new})} & f_2^{(\text{new})} & \dots & f_k^{(\text{new})} \end{bmatrix}$. The distance between $\boldsymbol{F}_{\text{new}}$ and each node or distance between $\boldsymbol{F}_{\text{new}}$ and each feature is stored in hashmap "Dist." The keys for "Dist" are the indices of nodes, and values for "Dist" are the distance values or lists of distances (lines 8 and 20). First, we turn the $l \times k$ feature matrix $\boldsymbol{F}_i$ into a $1 \times k$ vector $\boldsymbol{F}_i'$ by calculating the entropy of each feature. By calculating the Mahalanobis distance between $\boldsymbol{F}_{\text{new}}$ and different nodes, we can find the minimum distance. If the minimum is significantly smaller than other distances, which means this method can classify the newly incoming flow into one attack pattern, the attack pattern node that has the minimum distance is the predicted pat-

---

Algorithm 2 FPG-based attack detection

1: **Input:** FPG: $\{V, E, \boldsymbol{F}\}$; $\boldsymbol{F}_{\text{new}} = [f_1, \ f_2, \dots, f_k]$
2: Initialize Dist $= \varnothing$
3: **for** each $\boldsymbol{F}_i$ in $\boldsymbol{F}$ **do**
4:     $\boldsymbol{F}_i' = \text{get\_entropy}(\boldsymbol{F}_i)$
5: **end for**
6: $\boldsymbol{F}' = \{\boldsymbol{F}_1', \boldsymbol{F}_2', \dots, \boldsymbol{F}_n'\}$
7: **for** each $\boldsymbol{F}_i'$ in $\boldsymbol{F}'$ **do**
8:     $\text{dist}(i, \text{new}) = \text{dist}_{\text{mah}}^2(\boldsymbol{F}_i', \boldsymbol{F}_{\text{new}})$
9:     $\text{Dist}[i] = \text{dist}(i, \text{new})$
10: **end for**
11: $i = \min(\text{Dist}).\text{index}$
12: **if** $\text{Dist}[i]$ is significantly smaller than other items in Dist **then**
13:     Return node V_i as the attack pattern
14: **else**
15:     $\text{nodes}_{\text{nearest}} = [\text{list of } \beta \text{ nearest nodes with } \boldsymbol{F}_{\text{new}}]$
16:     $\text{Dist} = \{\text{key:value}\}$
17:     **for** each node V_j in $\text{nodes}_{\text{nearest}}$ **do**
18:         **for** each row r in $\boldsymbol{F}_j$ **do**
19:             $\boldsymbol{f}_r^{(j)} = [f_{r1}^{(j)}, \ f_{r2}^{(j)}, \ \dots, \ f_{rk}^{(j)}]$
20:             $\text{dist}(f, \text{new}) = \text{dist}_{\text{mah}}^2(\boldsymbol{f}_r^{(j)}, \boldsymbol{F}_{\text{new}})$
21:         **end for**
22:         $\text{Dist}[j] = [\text{list of top } \alpha \text{ shortest } \text{dist}(f, \text{new})]$
23:     **end for**
24:     Return the smallest $\text{sum}(\text{Dist}[j])$, and V_j is the attack pattern
25: **end if**

---

tern of $\boldsymbol{F}_{\text{new}}$. As shown in Fig. 2a, the distance between V_1 and $\boldsymbol{F}_{\text{new}}$ is significantly smaller than that between other nodes, so the pattern of V_1 is treated as the pattern of $\boldsymbol{F}_{\text{new}}$.
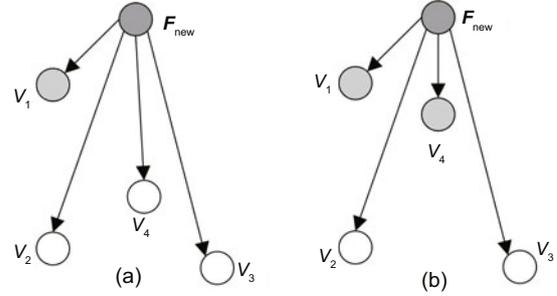

Fig. 2 Discovering the nearest neighbor using the Mahalanobis distance: (a) discovering the nearest neighbor; (b) discovering the two nearest neighbors

On the other hand, if the distance between $\boldsymbol{F}_{\text{new}}$ and nodes cannot properly detect the pattern, the algorithm calculates the distance between $\boldsymbol{F}_{\text{new}}$ and the feature vector of each sample in the β nearest nodes. On finding the β nearest nodes, we first find the nearest node V_i, and then find $(\beta - 1)$ nodes by sorting the node similarity with V_i. For example, in Fig. 2b, nodes V_1 and V_4 are two nearest neighbors of $\boldsymbol{F}_{\text{new}}$, both having a short distance to $\boldsymbol{F}_{\text{new}}$, and the two nodes have a high similarity. The calculation of distances between $\boldsymbol{F}_{\text{new}}$ and the feature of each sample in nodes V_1 and V_4 is described in Fig. 3, where $\boldsymbol{F}_1$ and $\boldsymbol{F}_4$ are feature matrices of nodes V_1 and V_4 respectively, and each row represents a feature sample. We use a list $\text{nodes}_{\text{nearest}}$ to store the β nearest nodes with $\boldsymbol{F}_{\text{new}}$, i.e., two nearest nodes V_1 and V_4 in Fig. 3. For each node in $\text{nodes}_{\text{nearest}}$, we compute the distance between $\boldsymbol{F}_{\text{new}}$ and the feature vector of each sample, and store the top α shortest distances into Dist. For example, if the samples $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ in V_1 and samples $\boldsymbol{f}_3$ and $\boldsymbol{f}_2$ in V_4 have the shortest two distances to $\boldsymbol{F}_{\text{new}}$, then

$$\text{Dist} = \{1: [\text{dist}_{\text{mah}}^2(\boldsymbol{f}_1^{(1)}, \boldsymbol{F}_{\text{new}}), \text{dist}_{\text{mah}}^2(\boldsymbol{f}_2^{(1)}, \boldsymbol{F}_{\text{new}})],$$
$$4: [\text{dist}_{\text{mah}}^2(\boldsymbol{f}_3^{(4)}, \boldsymbol{F}_{\text{new}}), \ \text{dist}_{\text{mah}}^2(\boldsymbol{f}_2^{(4)}, \boldsymbol{F}_{\text{new}})]\}.$$

We then obtain the summation of each list in Dist, and the node with the least summation, i.e., the highest similarity, is regarded as the predicted pattern. It can also work if we calculate the distances between $\boldsymbol{F}_{\text{new}}$ and the features of every node to find the most similar one, but when the nodes and samples in each

node are large enough, the detection will be slow, so we choose to compute the distances between only F_{new} and β nearest nodes to ensure the efficiency of the mechanism.
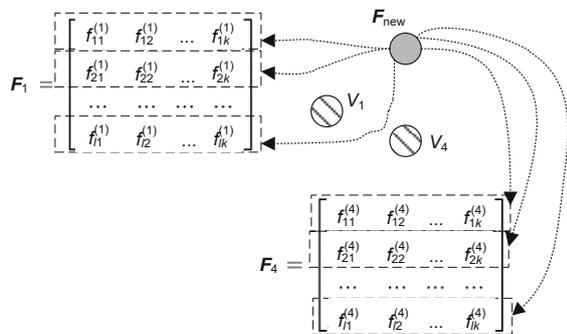

Fig. 3 Calculating the distances between F_{new} and each feature

After detecting the pattern of flows, the countermeasures should be taken. The countermeasures can be divided into forward, modify, or drop by the reaction method. For example, the benign flows will be forwarded to the target port; flows that have the same target port at the same time will be modified and redirected to other ports; flows that have no predefined countermeasures will be dropped. The countermeasures we define include: flow redirection, flow isolation, deep packet inspection, creating filtering rules, MAC address change, IP address change, block port, quarantine, network reconfiguration, and network topology change.

4 Evaluation

4.1 Network topology

The test scenarios are performed by simulation to prove the feasibility and effectiveness of the proposed method. Fig. 4 shows the network topology which is created to simulate the SDN environment. Mininet (de Oliveira et al., 2014) is used to create a realistic virtual SDN and run example topologies based on the Ryu controller and OpenFlow switch. Ryu (Shalimov et al., 2013) is a component-based SDN framework written in Python which supports OpenFlow. The well-defined APIs can be used for network prototyping, management, and monitoring. Open vSwitch is a widely used SDN OpenFlow switch which supports the OpenFlow protocol. The

network1 in Fig. 4 acts as a botnet, and the hosts simulate the DDoS attack traffic by sending packets to the OpenFlow switch and Ryu controller with a packet manipulation program Scapy (Kobayashi et al., 2007). Network2 simulates the normal traffic. We use the Wireshark connection monitoring system to capture the network traffic between hosts and the controller.
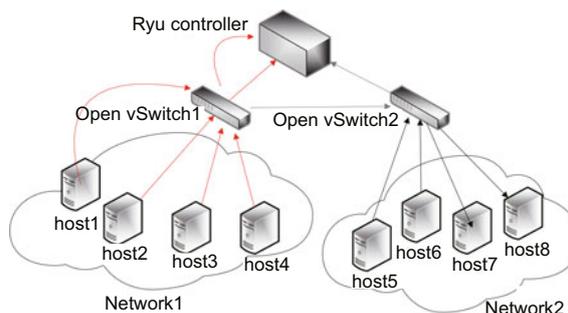
Fig. 4 Network topology for the test scenario

4.2 Dataset and implementation

The datasets we use for evaluation contain two parts. One is the intrusion detection evaluation dataset (ISCXIDS2012) (Shiravi et al., 2012), called dataset1, which contains the network traffic collected during the time periods with DDoS attack using an Internet relay chat (IRC) botnet. The traffic flows of the DDoS attack are labeled as attack or normal, which are separated into two sets for training and testing, respectively, in the experiment. Data in dataset1 are organized in the network packet format, so we need to transfer the packet-based data into flow-based data according to the procedure in Section 3.1. The other part of dataset, i.e., dataset2, is created with our test scenario shown in Fig. 4. We generate DDoS attack traffic and normal traffic at different time periods so that the traffic will be automatically classified as attack or benign. The dataset statistics used for training and testing are shown in Table 2. The implementation procedures are described as follows.

Table 2 Dataset statistics for training and testing

Type	Dataset1		Dataset2	
	Training	Testing	Training	Testing
Attack	650 785	383 306	73 575	8623
Benign	278 913	147 225	68 347	9678

1. Flow table creation

Considering the characteristics of DDoS attacks, the attack usually lasts a certain time period, so we need to analyze the features of traffic at different time periods. The creation of the flow table with flow-based data is necessary for feature extraction. A flow table is defined as a set of flows that are generated during the same time period, and the time interval is set to 10 s in this experiment. According to the generation time for packets in each flow, those flows whose packet timestamps located in one time slot are categorized into the same flow table.

2. Feature extraction

We define different features for DDoS attack patterns on TCP, UDP, and ICMP, respectively. Each flow table is turned into a 42-dimensional feature vector after feature extraction. The TCP-, UDP-, and ICMP-based DDoS attacks contain 21, 13, and 8 features, respectively (Table 3).

3. Link weight learning and graph generation

Based on the features extracted, we use the method proposed in Section 3.2.2 to calculate the distance matrix $\boldsymbol{M}$. By computing the Mahalanobis distance between each pair of nodes, a weighted graph is generated.

4. Attack detection

We evaluate the attack detection performance by launching a binary class prediction, i.e., attack or normal, and a multi-class prediction, i.e., normal or attack based on TCP, UDP, ICMP, or a combination of three. The binary classification uses the dataset

with binary labels to learn the metric matrix, while multiple classification uses the dataset with multiple labels.

Two methods are used for prediction to evaluate the prediction performance, as described in Section 3.2.4. One calculates the similarity between newly coming flow features and each node to find the smallest distance, abbreviated as SFN. The other computes the similarity between new flow features and each feature in β nearest nodes, abbreviated as SFF; this method is more time-consuming but has higher accuracy than the first one.

4.3 Results

4.3.1 Attack detection

We measure the effectiveness of the proposed attack detection approach in terms of Precision (P), Recall (R), and F-score (F). P is the fraction of the predicted pattern which matches the corresponding flow, R is the fraction of the target flow's pattern values which are correctly predicted, and F is the harmonic mean of P and R:

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

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

$$F = \frac{2PR}{P + R}, \tag{10}$$

where TP means true positive, FP false positive, and FN false negative.

Table 3 Features contained in the TCP-, UDP-, and ICMP-based DDoS attacks

Pattern	Features
TCP-based	Fraction of TCP flows, fraction of symmetric TCP flows, number of distinct source IP, entropy of source IP, median of bytes per flow, median of packets per flow, number of distinct window size, entropy of window size, number of distinct TTL values, entropy of TTL values, number of distinct source ports, entropy of source port, number of distinct destination ports, entropy of destination ports, fraction of destination ports less than 1024, fraction of flows with SYN flag, and fraction of flows with ACK flag, fraction of flows with URG flag, fraction of flows with FIN flag, fraction of flows with RST flag, and fraction of flows with PSH flag
UDP-based	Fraction of UDP flows, fraction of symmetric UDP flows, number of distinct source IP, entropy of source IP, median of bytes per flow, median of packets per flow, number of distinct source ports, entropy of source port, number of distinct destination ports, entropy of destination port, fraction of destination ports less than 1024, number of distinct TTL values, and entropy of TTL values
ICMP-based	Fraction of ICMP flows, fraction of symmetric ICMP flows, number of distinct source IP, entropy of source IP, median of bytes per flow, median of packets per flow, number of distinct TTL values, and entropy of TTL values

Binary class prediction and multi-class prediction are launched in the attack detection procedure. Binary classification predicts the attack or normal pattern of a newly incoming flow. Multi-class prediction represents five classes here, i.e., a normal pattern with a majority of TCP flows, a normal pattern with a majority of UDP flows, a normal pattern with both TCP and UDP flows, a TCP flooding DDoS attack, and a combination of TCP and UDP flooding DDoS attack.

We compare our proposed method (FPG) with three other methods on distance calculation, described as follows. Each method launches the binary class and multi-class prediction, while each prediction computes the similarity between the flow feature and node (SFN) and similarity between the flow feature and features of a node (SFF). Note that $\boldsymbol{x}$ and $\boldsymbol{y}$ in each equation represent the feature vector of a new flow and the entropy vector of each node $\boldsymbol{F}_i'$ respectively, when using the SFN method; they denote the feature vector of a new flow and each flow in a node respectively when using the SFF method.

1. FPG

Our proposed attack detection method is based on a feature-pattern graph by calculating the Mahalanobis distance with metric learning.

2. KLD

The Kullback-Leibler divergence (KLD), also called related entropy (van Erven and Harremos, 2014), is used to measure the similarity between two probability distributions and is calculated by

$$D_{\mathrm{KL}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_i x_i \log \frac{x_i}{y_i}. \qquad (11)$$

The lower the value of $D_{\mathrm{KL}}(\boldsymbol{x}, \boldsymbol{y})$, the higher the similarity of $\boldsymbol{x}$ and $\boldsymbol{y}$.

3. Pearson

Pearson correlation (Wu et al., 2009; AlEroud and Alsmadi, 2017) is a widely used similarity measure in intrusion detection, calculated by

$$\begin{aligned} D_{\mathrm{Pearson}}(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\mathrm{cov}(\boldsymbol{x}, \boldsymbol{y})}{\sigma_x \sigma_y} \\ &= \frac{E[(\boldsymbol{x} - \mu_x)(\boldsymbol{y} - \mu_y)]}{\sigma_x \sigma_y}, \end{aligned} \qquad (12)$$

where cov is the covariance, σ the standard deviation, E the expectation, and μ_x the mean of $\boldsymbol{x}$. A higher value of $D_{\mathrm{Pearson}}(\boldsymbol{x}, \boldsymbol{y})$ denotes a higher similarity of $\boldsymbol{x}$ and $\boldsymbol{y}$.

4. Cosine

Cosine similarity (Nguyen and Bai, 2010), calculated by

$$D_{\mathrm{cosine}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}, \qquad (13)$$

is a widely used measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. A lower value of $D_{\mathrm{cosine}}(\boldsymbol{x}, \boldsymbol{y})$ represents $\boldsymbol{x}$ and $\boldsymbol{y}$ as being more similar.

The precision, recall, and F-score of the attack detection are demonstrated in Figs. 5 and 6.

The results show that our proposed method generally outperforms the compared methods. The SFN gives an imperfect result in multi-class prediction but behaves well in binary class prediction. The reason may be that the entropy values of the features in the nodes belonging to the same category, i.e., attack or normal, are not sufficiently distinguishable. Almost all the compared methods with SFF have higher precision and recall than SFN; this is because the proposed SFF calculates the similarity of the feature vectors of different flows, which can properly distinguish the similarity of flows. The proposed FPG method outperforms the other compared methods considerably on multi-class prediction with SFF, which indicates that by using the combined feature-pattern graph with metric learning and similarity of feature vectors, our model can better predict the pattern of a new flow and can thus better detect DDoS attacks.

4.3.2 Graph update

The other part of the experiment is to measure the effectiveness of the updated graph model. We add a new node by local update and global update and measure the performance.

1. Graph before update

The feature-pattern graph generated before update is shown in Fig. 7a with black nodes and edges. Each node stands for a network pattern. Since the data used for training contain only five different patterns before graph update, the graph has five nodes, where nodes V_0, V_2, V_4 are in a normal pattern and nodes V_1, V_3 are in a DDoS pattern. The links between two nodes represent the feature-based Mahalanobis distance of two patterns. Node V_0 means a normal pattern whose flows are principally in TCP
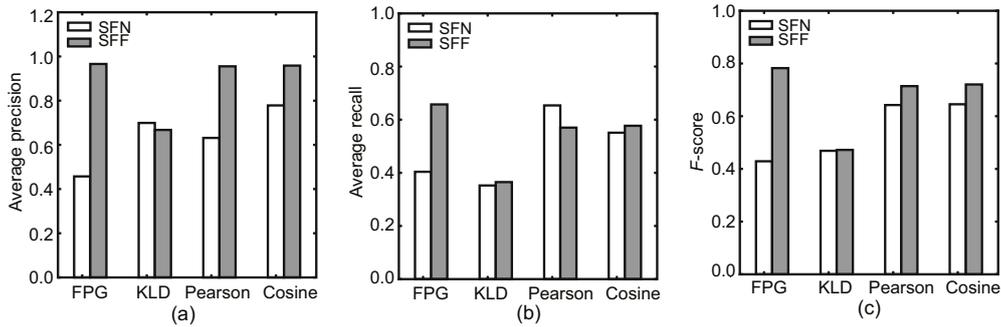
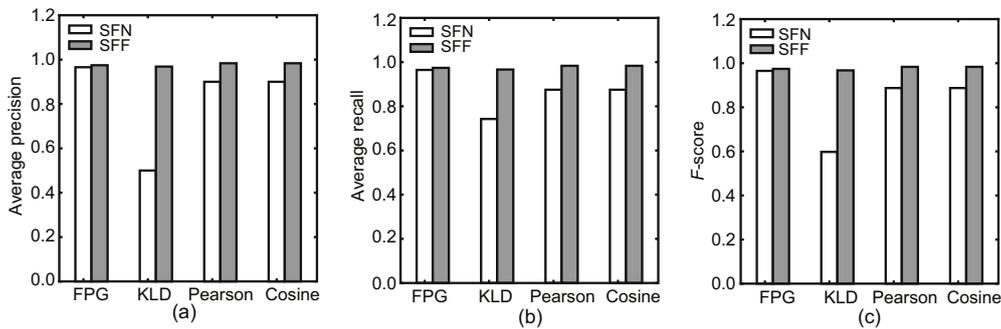**Fig. 5 Average precision (a), recall (b), and *F*-score (c) of multi-class prediction**

Fig. 6 Average precision (a), recall (b), and *F*-score (c) of binary class prediction
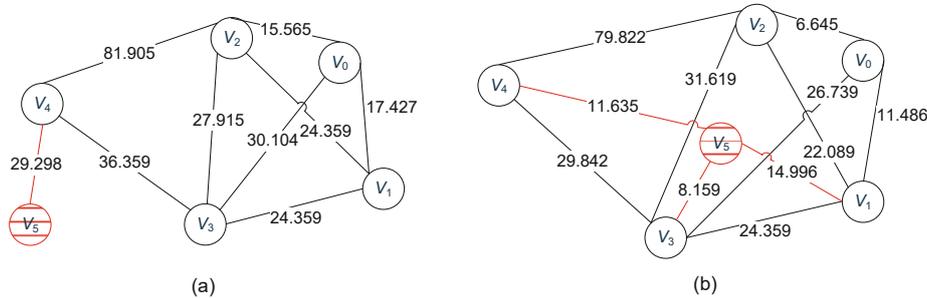

Fig. 7 Graph generated by local (a) and global (b) update

traffic, node V_1 is a DDoS attack pattern with TCP traffic, V_2 and V_3 represent flows with an approximately equal amount of TCP and UDP in normal and attack pattern respectively, and V_4 is the normal pattern with UDP traffic. The graph distinctly shows the similarity and relationships of the different nodes. The distance between nodes is noted in Fig. 7a; the shorter the distance between two nodes, the more similar they are. Nodes V_0 and V_2 are similar because they share similar features on TCP traffic and also feature values in normal status. Nodes with no link mean that they have pretty low similarity.

2. Graph with local update

Node V_5 in Fig. 7a is a new node added to the

graph. We generate a test dataset of DDoS attacks with UDP traffic, which is not in the existing feature-pattern graph; so, we add a new node, i.e., V_5, in the graph. The local update measures the traffic feature similarity of the new pattern and the existing patterns to find the γ nearest nodes and establish links. During the update, we find that only the distance between V_5 and V_4 is significantly smaller than other distances, so local update inserts only one link.

3. Graph with global update

Global update relearns matrix $\boldsymbol{M}$ in Section 3.2.2 and recalculates the Mahalanobis distance between nodes. The result of a global update is shown in Fig. 7b. The global update can find the

latent relationships and similarities between a new node and existing nodes more accurately.

The precision, recall, and F-score of the SFF method after various updates are shown in Fig. 8. The three nodes in each line represent no update, local update, and global update. We notice a minor decline in these three values after local update; this is because local update cannot precisely find the links between the new node and existing nodes, which leads to wrong prediction of a new pattern and a decline in precision and recall values. However, after global update, the values of precision and recall increase and go back to the values before update. Despite the decline in values of P, R, and F after local update, R and F are still higher than those of the compared methods. A global update can bring greater effectiveness, but needs more time and computation. The best way to combine local and global update in practical use is to launch local update in real time when finding a new pattern, and to implement a global update regularly during slack hours to guarantee effectiveness.
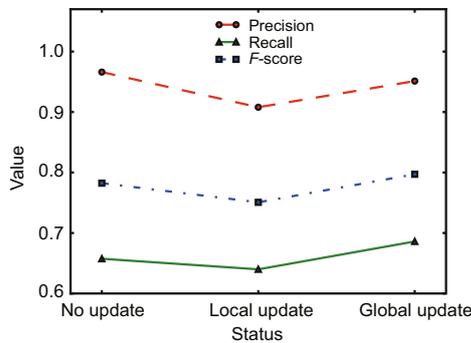

Fig. 8 Detection performance after graph update

5 Conclusions

In this paper, we have proposed a discovery method for distributed denial-of-service attack behaviors in SDNs using a feature-pattern graph model. We created attack signatures adapted to the SDN environment through flow creation with the dataset from the traditional network and SDN traffic. A feature-pattern graph has been modeled based on the attack signatures, where each node represents a network pattern and weighted links denote the feature-based similarity of the nodes. To better depict the relationships among network patterns, we introduced link weight learning. Distance met-

ric learning was used to learn the matrix $\boldsymbol{M}$ and the Mahalanobis distance calculated with $\boldsymbol{M}$ was treated as the similarity between nodes or feature vectors. The graph model is scalable and we proposed local and global update methods to insert new nodes. To optimize the performance of FPG-based DDoS attack detection, we also presented a model which is a combination of two methods. One calculates the distance between the newly incoming flow and nodes in the graph to find the nearest node; the other computes the distances between the new flow and features in each node. Generally, we paid more attention to the typical and common DDoS attacks in our work. Other types of DDoS attack such as IP address spoofing, certain mimicry attack, attacks based on complexity exploitation, and characteristics of link-flooding and target link-flooding attacks are not considered in this work.

Experiments have been conducted to evaluate the performance of the proposed method. A simulated SDN environment was established to monitor and collect SDN traffic. The evaluation results on two datasets demonstrated the feasibility of the proposed method. The results of the experiments demonstrated that the feature-pattern graph based discovery method for DDoS attack behaviors substantially outperforms the compared methods on precision, recall, and F-score. Results from graph update testified to the effectiveness of the local and global update approaches. Since flooding-based DDoS attacks are the most common kinds among all the DDoS attacks and also easy to simulate, we typically used the datasets on flooding-based attacks.

As part of our future work, we anticipate to improve the method in three ways to make it effective and serviceable in practical use. The first one is to equip the method with an automated countermeasure selector to facilitate the selection of countermeasures according to the traffic patterns. The second one is to reduce time consumption and computing resources in the metric learning process to make attack detection more efficient. Finally, considering the complexity of the practical environment, besides flooding-based DDoS attacks, various kinds of DDoS attacks should be considered.

References

Albin E, Rowe NC, 2012. A realistic experimental comparison of the Suricata and Snort intrusion-detection systems. Proc 26th Int Conf on Advanced Information Networking and Applications Workshops, p.122-127. https://doi.org/10.1109/WAINA.2012.29

AlEroud A, Alsmadi I, 2017. Identifying cyber-attacks on software defined networks: an inference-based intrusion detection approach. *J Netw Comput Appl*, 80:152-164. https://doi.org/10.1016/j.jnca.2016.12.024

Antikainen M, Aura T, Särelä M, 2014. Spook in your network: attacking an SDN with a compromised OpenFlow switch. Proc 19th Nordic Conf on Secure IT Systems, p.229-244.
https://doi.org/10.1007/978-3-319-11599-3_14

Aziz MZA, Okamura K, 2017. Leveraging SDN for detection and mitigation SMTP flood attack through deep learning analysis techniques. *Int J Comput Sci Netw Secur*, 17(10):166-172.

Bawany NZ, Shamsi JA, Salah K, 2017. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arab J Sci Eng*, 42(2):425-441. https://doi.org/10.1007/s13369-017-2414-5

Braga R, Mota E, Passito A, 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. Proc IEEE Local Computer Network Conf, p.408-415. https://doi.org/10.1109/LCN.2010.5735752

Chung CJ, Khatkar P, Xing TY, et al., 2013. NICE: network intrusion detection and countermeasure selection in virtual network systems. *IEEE Trans Depend Sec Comput*, 10(4):198-211.
https://doi.org/10.1109/TDSC.2013.8

de Oliveira RLS, Schweitzer CM, Shinoda AA, et al., 2014. Using Mininet for emulation and prototyping software-defined networks. Proc IEEE Colombian Conf on Communications and Computing, p.1-6. https://doi.org/10.1109/ColComCon.2014.6860404

Fan ZJ, Xiao Y, Nayak A, et al., 2019. An improved network security situation assessment approach in software defined networks. *Peer-to-Peer Netw Appl*, 12(2):295-309. https://doi.org/10.1007/s12083-017-0604-2

Fiadino P, D'Alconzo A, Schiavone M, et al., 2015. Challenging entropy-based anomaly detection and diagnosis in cellular networks. *ACM SIGCOMM Comput Commun Rev*, 45(4):87-88.
https://doi.org/10.1145/2829988.2790011

Giotis K, Argyropoulos C, Androulidakis G, et al., 2014. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Comput Netw*, 62:122-136. https://doi.org/10.1016/j.bjp.2013.10.014

Goldberger J, Roweis S, Hinton G, et al., 2004. Neighbourhood components analysis. Proc 17th Int Conf on Neural Information Processing Systems, p.513-520.

Klöti R, Kotronis V, Smith P, 2013. OpenFlow: a security analysis. Proc 21st IEEE Int Conf on Network Protocols, p.1-6.
https://doi.org/10.1109/ICNP.2013.6733671

Kobayashi TH, Batista AB, Brito AM, et al., 2007. Using a packet manipulation tool for security analysis of industrial network protocols. Proc IEEE Conf on Emerging Technologies and Factory Automation, p.744-747. https://doi.org/10.1109/EFTA.2007.4416847

Kreutz D, Ramos FM, Veríssimo PE, et al., 2015. Software-defined networking: a comprehensive survey. *Proc IEEE*, 103(1):14-76.
https://doi.org/10.1109/JPROC.2014.2371999

Nguyen HV, Bai L, 2010. Cosine similarity metric learning for face verification. Proc 10th Asian Conf on Computer Vision, p.709-720.
https://doi.org/10.1007/978-3-642-19309-5_55

Niyaz Q, Sun WQ, Javaid AY, 2017. A deep learning based DDoS detection system in software-defined networking (SDN). *EAI Endorsed Trans Secur Safety*, 4(12):e2. https://doi.org/10.4108/eai.28-12-2017.153515

Roesch M, 1999. Snort: lightweight intrusion detection for networks. Proc 13th USENIX Conf on System Administration, p.229-238.

Scott-Hayward S, O'Callaghan G, Sezer S, 2013. SDN security: a survey. IEEE SDN for Future Networks and Services, p.1-7.
https://doi.org/10.1109/SDN4FNS.2013.6702553

Shalimov A, Zuikov D, Zimarina D, et al., 2013. Advanced study of SDN/OpenFlow controllers. Proc 9th Central & Eastern European Software Engineering Conf in Russia, Article 1. https://doi.org/10.1145/2556610.2556621

Shen C, Kim J, Wang L, 2010. Scalable large-margin mahalanobis distance metric learning. *IEEE Trans Neur Netw*, 21(9):1524-1530.
https://doi.org/10.1109/TNN.2010.2052630

Shiravi A, Shiravi H, Tavallaee M, et al., 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput Secur*, 31(3):357-374.
https://doi.org/10.1016/j.cose.2011.12.012

van Erven T, Harremos P, 2014. Rényi divergence and Kullback-Leibler divergence. *IEEE Trans Inform Theory*, 60(7):3797-3820.
https://doi.org/10.1109/TIT.2014.2320500

Wang B, Zheng Y, Lou WJ, et al., 2015. DDoS attack protection in the era of cloud computing and software-defined networking. *Comput Netw*, 81:308-319. https://doi.org/10.1016/j.comnet.2015.02.026

Wang R, Jia ZP, Ju L, 2015. An entropy-based distributed DDoS detection mechanism in software-defined networking. Proc IEEE Trustcom/BigDataSE/ISPA, p.310-317. https://doi.org/10.1109/Trustcom.2015.389

Wu QS, Ferebee D, Lin YY, et al., 2009. An integrated cyber security monitoring system using correlation-based techniques. Proc IEEE Int Conf on System of Systems Engineering, p.1-6.

Xu Y, Liu Y, 2016. DDoS attack detection under SDN context. Proc 35th Annual IEEE Int Conf on Computer Communications, p.1-9.
https://doi.org/10.1109/INFOCOM.2016.7524500

Yan Q, Yu FR, Gong QX, et al., 2016. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: a survey, some research issues, and challenges. *IEEE Commun Surv Tutor*, 18(1):602-622.
https://doi.org/10.1109/COMST.2015.2487361

Yu S, Guo S, Stojmenovic I, 2012. Can we beat legitimate cyber behavior mimicking attacks from botnets? Proc IEEE INFOCOM, p.2851-2855.
https://doi.org/10.1109/INFCOM.2012.6195714