Hindawi
Security and Communication Networks
Volume 2018, Article ID 9649643, 15 pages
https://doi.org/10.1155/2018/9649643

Research Article

OverWatch: A Cross-Plane DDoS Attack Defense Framework with Collaborative Intelligence in SDN

Biao Han, Xiangrui Yang, Zhigang Sun, Jinfeng Huang, and Jinshu Su

College of Computer, National University of Defense Technology, Changsha, China

Correspondence should be addressed to Xiangrui Yang; yangxiangrui11@nudt.edu.cn

Received 28 September 2017; Accepted 18 December 2017; Published 24 January 2018

Academic Editor: Chengchen Hu

Distributed Denial of Service (DDoS) attacks are one of the biggest concerns for security professionals. Traditional middle-box based DDoS attack defense is lack of network-wide monitoring flexibility. With the development of software-defined networking (SDN), it becomes prevalent to exploit centralized controllers to defend against DDoS attacks. However, current solutions suffer with serious southbound communication overhead and detection delay. In this paper, we propose a cross-plane DDoS attack defense framework in SDN, called OverWatch, which exploits collaborative intelligence between data plane and control plane with high defense efficiency. Attack detection and reaction are two key procedures of the proposed framework. We develop a collaborative DDoS attack detection mechanism, which consists of a coarse-grained flow monitoring algorithm on the data plane and a fine-grained machine learning based attack classification algorithm on the control plane. We propose a novel defense strategy offloading mechanism to dynamically deploy defense applications across the controller and switches, by which rapid attack reaction and accurate botnet location can be achieved. We conduct extensive experiments on a real-world SDN network. Experimental results validate the efficiency of our proposed OverWatch framework with high detection accuracy and real-time DDoS attack reaction, as well as reduced communication overhead on SDN southbound interface.

1. Introduction

Distributed Denial of Service (DDoS) attacks in TCP/IP networks are typically explicit attempts to disrupt legitimate users access to services, which are often launched by botnet computers that are simultaneously and continuously sending a large number of service requests to the victims [1]. The victims either respond so slowly as to be unusable or crash completely. According to Arbor Networks, which offers services to protect against DDoS attacks, they observed over 124,000 DDoS attacks per week since 2016, and they believe this number is growing rapidly [2]. Besides, since breaking the 100 Gbps barrier in 2010, DDoS attacks are also increasing in size, making them more and more difficult to defend against. Therefore, protecting network-wide resources from these frequent and large volume DDoS attacks necessitates the research community to focus on developing high-efficient defense frameworks that can be appropriately deployed in time.

Former DDoS attack defense in traditional networks involves the use of middle-box devices, which are generally complicated integration of customized hardware and software [3–6]. Although they are superior in defense performance, it is found that middle-box based DDoS attack detection is inflexible with network evolution, for example, hard to support new network architectures or protocols. Moreover, these devices are usually independently deployed in a network and have different communication interfaces. This hinders them from a holistic perception of network status, which is becoming extremely critical for network-wide defense against increasingly frequent and large volume DDoS attacks [7].

Recently, extensive research efforts have been conducted to apply software-defined networking (SDN) in diagnosing and defending DDoS attacks in a global point of view [8–12]. Different from traditional networks and information-centric networks (ICN) [13], in which the forwarding and routing decision can only be made locally, the centralized controller in SDN can quickly install reaction rules on switches and run DDoS attack defense applications without additional cost of middle-box devices. In this context, DDoS attacks

can be detected and defeated in an early stage. However, existing approaches which build DDoS attack defense applications upon the control plane are challenging to provide high defense performance. On the one hand, DDoS attack detection methods often require analysis techniques that are more advanced than de facto SDN data plane allows. Thus, the controller needs to poll flow statistics or packets from data plane switches frequently for attack detection and botnet location [8, 10, 14, 15], which increases southbound overhead and detection delay significantly. On the other hand, the potential advantages in exploiting collaborative intelligence of SDN have not been well investigated as effective DDoS attack defense requires extremely accurate detection and rapid reaction in both. Otherwise, it may result in SDN controller saturation attack in the worst case, as discussed in [16, 17].

In this paper, in order to protect hosts and servers from high volume DDoS attacks inside a particular network (e.g., autonomous system), we design and implement a high-efficient cross-plane DDoS attack defense framework with collaborative intelligence in a pure SDN environment, called OverWatch. OverWatch overcomes the aforementioned problems of existing SDN-based methods by collaboratively splitting defense functionalities across data plane and control plane and enabling both planes with abilities to intelligently detect and react to DDoS attacks in cooperation. The main difference between traditional frameworks in SDN and OverWatch is that we take the data plane into consideration for cross-plane optimization. In the proposed OverWatch framework, defense procedure is divided into two phases: detection phase and reaction phase.

In the detection phase, a lightweight flow monitoring algorithm is proposed to serve the data plane as DDoS attack sensor. We focus on two key features of DDoS attack traffic: volume feature and asymmetry feature. The proposed flow monitoring algorithm captures DDoS attack traffic in a coarse-grained manner by polling the values of SDN switch counters. On the control plane, a machine learning based DDoS attack classifier and a botnet tracking algorithm are utilized to locate a DDoS attack in finer granularity, for example, attack type and botnet locations. Specifically, features extracted from attack traffic and holistic information of the network are fed into DDoS attack classifier and botnet tracker, respectively, to determine the attack type and botnet locations.

In the reaction phase, based on the results obtained from the detection phase, a novel defense strategy offloading mechanism is proposed to enable DDoS attack defense actuators to be executed on the SDN switches automatically. Thus, SDN controller can be free from conducting specific defensive actions, resulting in a dynamic attack reaction efficiency. More specifically, we concentrate on exploiting the computational resources of switch CPUs and the flexibility of southbound interface, in order to deploy defense actuators on the switches which are closest to the botnet.

The main contributions of this paper can be summarized as follows:

(i) We design a cross-plane DDoS attack defense framework in SDN that exploits collaborative intelligence between data plane and control plane with high defense efficiency.

(ii) We develop a collaborative DDoS attack detection mechanism, which consists of a coarse-grained flow monitoring algorithm on the data plane and a fine-grained machine learning based attack classification algorithm on the control plane.

(iii) We propose a novel defense strategy offloading mechanism to dynamically deploy defense applications across the controller and switches, by which rapid attack reaction and accurate botnet location can be achieved.

(iv) We conduct extensive experiments on a real-world network with a FPGA-based OpenFlow switch prototype, a Ryu controller, and laptops generating DDoS attack traffic. Experimental results validate the efficiency of our proposed OverWatch framework with high detection accuracy and real-time DDoS attack defending reaction, as well as reduced communication overhead on SDN southbound interface.

The rest of this paper is organized as follows. Section 2 covers background and motivation of this paper. Section 3 presents the architecture of our proposed OverWatch framework. Sections 4 and 5 are mechanisms of two phases (detection phase and reaction phase) in OverWatch. Section 6 presents the experimental results. Finally, this paper is concluded in Section 7.

2. Background and Motivation

2.1. Middle-Boxes Based Defense Mechanisms. Characteristics of DDoS attacks have been widely studied, and researchers have proposed various methods to detect/defend DDoS attacks. Traditionally, DDoS attack defense applications are deployed on middle-box devices [3, 4, 19], which are specialized equipment or software that detects and reacts to DDoS attacks from a single spot on the network. The middle-boxes can provide high DDoS attack detection performance. However, due to various interfaces provided by them, different middle-boxes seldom share information, causing them and network operators to lack holistic view of DDoS attacks [20]. Mahimkar et al. in [19] proposed a method to deploy middle-boxes dynamically in the protected network, but without the global perspective of a network, where deploying them could be a nerve-racking problem. For example, assuming an attack is detected on multiple middle-boxes alone the attack trace, the most effective defense strategy would be processing malicious packets on the devices close to the source side. However, attack trace-back could not be accomplished without global information of network.

2.2. SDN-Based Defense Mechanisms. SDN provides a standard interface for a centralized controller to manage each switch under control remotely. This enables the SDN controller to obtain the entire network information and to make defense strategies in holistic views [8–11, 15, 18, 21]. In the field of network monitoring, many researches focus on how

to reduce monitoring overhead while ensuring accuracy [11, 14, 22–24]. Among them, Braga et al. in [8] proposes using Support Vector Machine (SVM) to detect DDoS attacks on the SDN controller. Chowdhury et al. in [14] proposes Payless, which includes an OpenFlow monitoring method based on an adaptive statistics collection algorithm. It can reduce the bandwidth of southbound channel but the accuracy is also reduced. Xu and Liu in [9] proposes a method to control granularity of flow by utilizing prefix masks to reduce the consumption of Ternary Content-Addressable Memory (TCAM) [25] resources while detecting DDoS attacks. Zhang in [11] proposes a prediction-based method to control the granularity of measurement while detecting abnormal traffic in order to reduce the monitoring overhead.

Moreover, many DDoS attack defense methods and systems are proposed based on SDN [26, 27]. Fresco [26] is a typical SDN-based security framework. It can poll statistics from data plane to detect different attacks. Once malicious behaviors are detected, it pushes the defense logic by installing forwarding rules on data plane switches. For example, if SYN flood attacks are detected by the defense application, the controller modifies switch rules to redirect suspected flow onto control plane to filter out malicious packets from normal ones.

2.3. Motivation of Cross-Plane Collaborative Defense. Although significant achievements have been made along this line, for example, Shin et al. in [16] enable SDN switches with more functionalities in detecting and defending SYN floods to eliminate the bottleneck between data plane and control plane, two critical problems of the existing SDN-based DDoS attack defense methods need to be pointed out here. First, both of detection and reaction process for DDoS attacks require to upload malicious traffic to the centralized controller, which introduces large amount of overhead for southbound interface and workload to the controller. Second, the controller-based DDoS attack defense mechanism breaks the initial idea of the separation of the control plane from data plane devices, as it requires the controller to process malicious packets directly. Such issues prevent SDN controller to be an intelligent centre while conducting DDoS attack defense.

Ideally, the controller in a well-defined DDoS attack defense framework should concentrate on attack analysis (e.g., attack classification and traffic trace-back). Thus, instead of implementing certain defense applications, the controller should be responsible for conducting fine-grained attack detection and making high level defense strategies, leveraging its global view of the whole network and abundant computational resources. Moreover, as data plane is where packets are proceeded, the switches in SDN should be enabled with new packets processing functions for DDoS attack detection and reaction, which are not implemented by current SDN-based DDoS attack defense approaches so far. Fortunately, most SDN switches (e.g., OpenFlow switches) consist of one or more CPUs running an operating system with abundant computational resource that is currently far from utilized [28, 29]. This inspires us to liberate the controller from heavy traffic and exploit the underutilized computing capabilities in switches to perform specific DDoS attack defense functions. Therefore, this goal could be achieved by modifying existing SDN devices. Sonchack et al. in [30] proposed a framework enabling security mechanisms to be loaded from controller to switches dynamically. Motivated by their work of leveraging computational resources on SDN switch CPUs to conduct defense mechanisms, we aim to design a collaborative DDoS attack defense framework. By exploiting the intelligence of the central controller, we focus on designing fine-grained attack detection mechanisms and automatic defense reaction by analyzing the detected DDoS attack traffic.

3. Proposed OverWatch Framework

3.1. Architecture Overview. As illustrated in Figure 1, the architecture of our proposed OverWatch framework is inspired by Knowledge Plane [31]. In OverWatch, DDoS attack sensors and defense actuators run on data plane switches. Meanwhile, DDoS attack classifier (responsible for classifying attacks), a botnet tracker (responsible for locating sources of attack traffic), and the library for defense actuators are located over the control plane.

Once OverWatch starts running, DDoS attack sensors keep monitoring every flow on the data plane constantly. If any abnormal flow is captured (i.e., DDoS attack traffic), the specific switch notifies control plane with information including an alert message and occurring attack features. Over the control plane, OverWatch leverages uploaded attack features to find out DDoS attack types and its global perspective to locate the attack sources. Next, the controller requests defense actuator library to implement specific defense actuators on the switches that are close to botnet. Last but not least, the loaded actuator will be executed on the specific switch, defending against certain DDoS attacks in the first place. After the attack is eliminated, the defense actuators used for defense will be removed from certain switches.

Our ultimate objective is that the network can detect DDoS attack threats accurately and react to them automatically. To achieve this, defense ability needs to be introduced both into control plane and data plane. We introduce design of the data plane and control plane in the following subsections, respectively.

3.2. Data Plane Design. The data plane in OverWatch is not just a group of forwarding entities, inside which there is a group of software sensors and actuators to detect and react to DDoS attacks. This leads to our three key functionalities to enhance the SDN switches: First, the data plane switches should be capable of capturing main features of DDoS attacks. Second, after the reaction strategies are made by the control plane, reaction functionalities should be loaded from control plane to data plane dynamically. Finally, these actuators can be executed on data plane to filter out attack packets. We introduce these key functionalities below.

3.2.1. DDoS Attack Sensor. We propose an algorithm which runs on the data plane devices to detect DDoS attacks as a

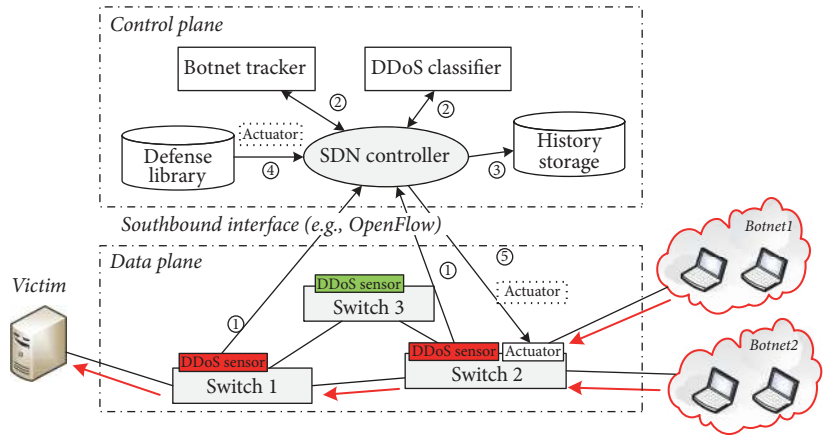

Figure 1: The architecture and workflow of OverWatch.

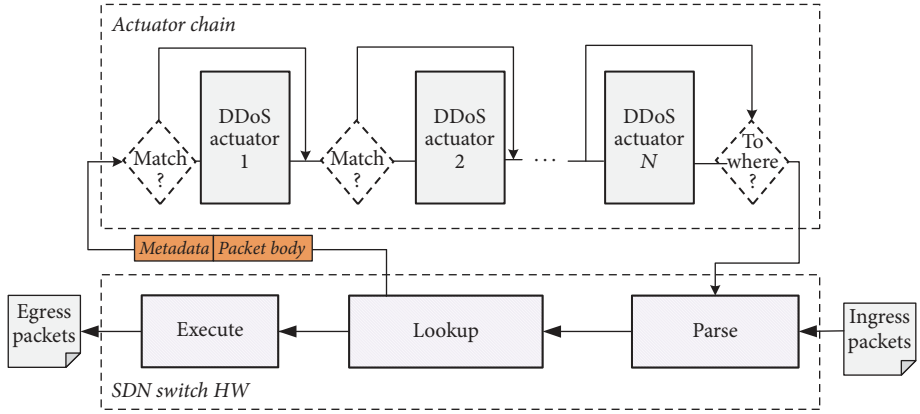

Figure 2: The process procedure of actuator chain when multiple actuators are loaded on the same switch.

DDoS attack sensor. Generally, there are plenty of approaches that are able to capture key features of DDoS attacks, for example, large volume of traffic and asymmetry in two-way traffic. However, in the context of SDN, the limitation of low-end CPU on SDN switch compared with middle-boxes causes most of them to be inappropriately deployed.

Thus, we propose a lightweight flow monitoring algorithm which recognizes DDoS attacks through reading hardware counters to server as DDoS attack sensors in OverWatch. The details of this algorithm are illustrated in Section 4. Here, we only list the two functions that can be completed by DDoS attack sensors: (1) capturing changes in flow characteristics without inspecting packets and (2) recording the DDoS attack traffic by physical port or flow ID. By this means, the sensors on the data plane are capable of capturing DDoS attacks in the first place.

3.2.2. DDoS Attack Defense Actuator. The DDoS defense actuators, to be specific, are a set of switch software tools to conduct various defense mechanisms independently. Different from traditional SDN switches, which are only capable of performing basic match-action processes, switches in OverWatch are enabled with different packet processing mechanisms by using software resources.

Various types of DDoS attacks may be conducted simultaneously to maximize the attack effect. On this occasion, multiple actuators are required to run on a single switch. Thus, we design the actuator chain, which aims to enable multiactuators to work independently, as shown in Figure 2. When a SDN switch startup, the agent process initializes an empty linked list. Once an recently loaded actuator is compiled, it is added to the tail of the list. When multiple actuators link into the list, a chain of actuators forms. Packets from hardware firstly enter the head of the chain. If the metadata contains a packet that matches the specific ID of the current actuator, this actuator will process the packet and send it back to hardware with a modified metadata (revealing the specific hardware module sent to). Otherwise, the packet will bypass the current actuator until a matched actuator ID is found. By this means, defense actuators in the same switch are able to work independently.

3.2.3. Defense Strategy Enabling. Once attacks are identified, the control plane makes a set of strategies to react. Enabling defense actuators on the data plane dynamically is a key step for executing these strategies.

The enabling procedure is inspired by the work of OFX but there are some key differences between them. First, there

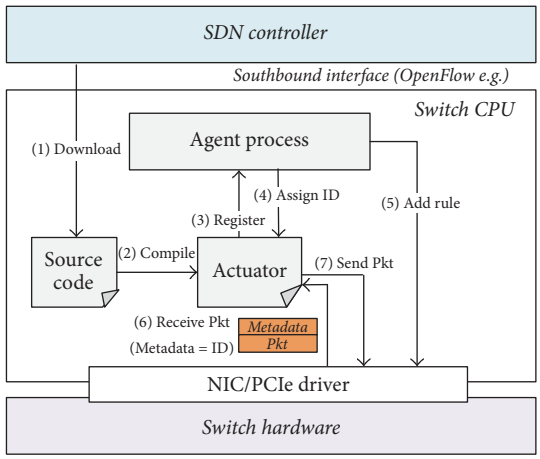

Figure 3: The procedure of defense enabling in typical SDN switches.

is no flow table maintained on the software so not all the packets need to be redirected to the software. Second, the packets redirected to the software will be processed by a specific actuator according to a metadata. The defense strategy enabling mechanism can be illustrated as Figure 3. Firstly, source code of a defense actuator is loaded from controller to local memory of a designated switch. Then, the code is compiled in the embedded operating system (usually Linux-based). Afterwards, the actuator registers to the agent process, during which an exclusive ID is assigned to the actuator. Furthermore, before the startup function of the actuator is executed, it sends a standard message to the agent process, indicating the specific packet types it processes. The agent process adds a high priority rule to the hardware match table, accordingly. In this way, packets that match such rule are polled from hardware with metadata navigating to the specific actuator. Once the above steps are completed, the actuator is executed to perform specific DDoS attack defense function.

3.3. Control Plane Design. The control plane is brain to OverWatch. It inspects the current DDoS attack (e.g., attack types and its traces) and makes proper strategy to defend it. According to our overall objective, the heart of the control plane is its intelligence ability to inspect current DDoS attack and reason proper strategies, which means the control plane should be able to (1) classify DDoS attacks and (2) track the botnets. To be specific, on the one hand, the control plane should determine exact attack types (SYN flood, UDP flood, DNS flood, etc.). On the other hand, it should also locate sources of occurring attack so as to defend DDoS attacks from the source, which proves to be more effective than defending from the destination. This argues that the control plane is responsible for the following three functionalities.

3.3.1. Attack Classification. As we use different defense actuators according to particular attack types, in order to perform defense mechanism more effectively, OverWatch is required to identify attack types firstly.

To achieve this, when DDoS attack traffic is firstly captured by data plane sensors, the abnormal traffic matching a specific rule is mirrored (by sampling) for traffic feature extraction. In order to reduce overhead of southbound interface in OverWatch to a greater extent, feature extraction is conducted on the switch software, rather than on the controller. Then the features are polled to the controller for classification. On the controller, there runs a DDoS attack classification module that leverages the extracted traffic features as input to verify the attack type. To guarantee the accuracy and reduce the false-positive rate during classification, a machine learning method is utilized in this module, which we will demonstrate in Section 4.

3.3.2. Botnet Tracking. Botnet tracking is another key issue in DDoS attack defense as it determines where the defense actuators should be deployed. Generally, DDoS attack is conducted by several botnets. As attack flows travel closer to the victim, the malicious traffic becomes larger due to traffic merger. This prevents us from effective defense measurements. In order to process attack traffic effectively, actuators should be deployed at positions close to botnets. This argues that the controller in OverWatch is ought to locate switches that are close to botnets.

Fortunately, this can be accomplished by leveraging holistic info of the network topology maintained by the controller and data from malicious packets received from data plane switches. In the next section, we propose a flexible botnet tracking algorithm suitable to be deployed on the SDN controller, which is able to locate the group of switches that are in the upstream of attack traffic.

3.3.3. Attack Reaction. When attack types and botnet locations of a DDoS attack are both determined, the control plane needs to perform highly automatic defense reaction immediately. In OverWatch, the control plane reacts by loading specific defense actuators onto directed data plane devices.

As shown in Figure 4, the reaction procedure for attacks in a typical SDN controller is quite straightforward. The defense library module, which contains various source codes of DDoS attack defense actuators, firstly registers to event listener. Once an attack is determined, a 2-tuple {`DPID, AttackType`} (DPID [32] is used to represent Data Path IDentity in the context of SDN) is sent to the event manager, indicating the defense strategy made by the built-in applications. This tuple is then received by the defense library. The defense library loads source code of specific actuator which matches AttackType in the received 2-tuple and notifies defense enabler. Finally, the actuator is loaded to designated switches through southbound channel. In order to support such mechanisms, certain modifications need to be done for existing SDN controllers, which we will describe in Section 5.

4. Detection Phase of OverWatch

As OpenFlow [32] is the leading reference implementation of the SDN paradigm, it is reasonable to implement OverWatch

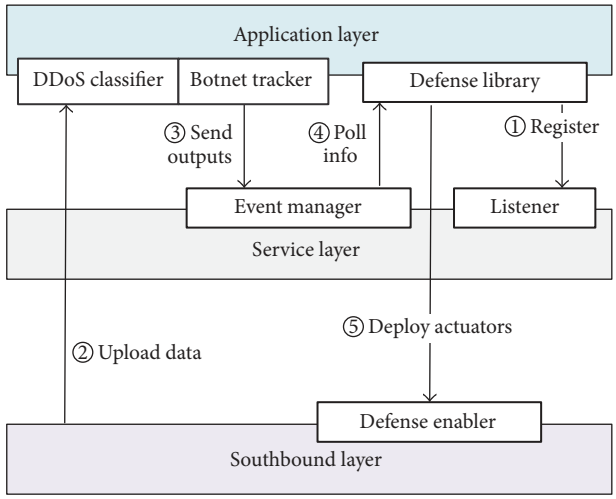

Figure 4: Workflow of attack reaction in a general SDN controller (i.e., Ryu controller). Be noted that this is only a sketch map for three-layer structured SDN controller.

in such an environment. Therefore, in this section, we introduce how we implement OverWatch into a typical SDN controller (Ryu controller [33]) and FPGA-based OpenFlow switches [18]. To describe the workflow of our OverWatch prototype clearly, we divide the working process of OverWatch into two phases: detection phase and reaction phase. The main goal in detection phase is to classify attack types as well as locate the botnets. We describe this phase in detail as follows.

4.1. Cross-Plane Attack Detection. The workflow in detection phase is shown in Figure 5. Firstly, the DDoS attack sensor constantly monitors data plane traffic by reading countervalues of each flow periodically. If attack flows are captured, the sensor notifies the OpenFlow switch agent of the specific flow ID to indicate the abnormal flow. Then, the switch agent modifies the action of hardware lookup table by a `OFPT_FLOW_MOD` message (defined in OpenFlow specification since 1.0) to mirror the sample packets from the abnormal flows onto local memory. The buffered packets have two uses: First, they are copied by the software-defined feature extraction module (SDFE), which extracts key features of different packets for attack classification on the control plane. Second, the packets themselves are also obtained by switch agent and sent to the controller for botnet tracking.

After the DDoS attack data (i.e., abnormal flow ID, sampled packets, and traffic features) is sent to the control plane encapsulated in a `OFPT_PKT_IN` message (also defined in OpenFlow specification since 1.0), it is firstly received by the OpenFlow control agent. Then, this agent extracts packet payload and passes it to the event manager. (*NB*. In Ryu, event manager is responsible for distributing messages received from data plane.) Afterward, the data is split into two parts, which are data related to traffic features and data related to botnet tracking (i.e., a {`DPID, FLOW_ID, Pkt_Buff`} three-tuple). The above two kinds of data are polled by DDoS attack classifier and botnet tracker, respectively, inside which the DDoS attack type and first-hop-switch of current DDoS attack are both determined.

From aforementioned, the detection phase is divided into two stages: a coarse-grained data plane detection stage and a fine-grained control plane detection stage. We discuss approaches we applied in both stages below.

4.2. Coarse-Grained Detection on Data Plane. As aforementioned, the data plane is where packets are forwarded; leveraging computing resources on the data plane to determine an attack coarsely and locally is quite reasonable. Therefore, on the data plane, we first present a lightweight flow monitoring algorithm that we utilize as a DDoS attack sensor on switches. It runs on the switch software as a monitor thread. Unlike many other monitoring methods, this algorithm aims to extract the key features of DDoS attack traffic by means of polling countervalues from an OpenFlow switch.

Generally, there are fundamental differences between a typical DDoS attack and normal network behaviors that we leverage to monitor DDoS attacks. Large traffic rate is one important feature for DDoS attacks. Moreover, during an attack, there is also huge rate difference between flows coming into a victim server and flows out of the server. They are defined as volume feature and asymmetry feature. Numerous researches have drawn the fact that typical DDoS attacks could be determined by verifying the above two features from traffic.

Fortunately, the above two features can be determined by polling switch countervalues from hardware pipeline. We define $C_{t_n}^{\text{Byte}}$ and $C_{t_n}^{\text{Pkt}}$ as the byte and packet count of a specific flow at time t_n. The two features can be expressed as follows.

Byte Count per Second (B_{t_n}) *at Time* t_n. It describes the average byte rate of a flow, port, or switch between time t_{n-1} and t_n:

$$B_{t_n} = \frac{C_{t_n}^{\text{Byte}} - C_{t_{n-1}}^{\text{Byte}}}{t_n - t_{n-1}}. \tag{1}$$

Packet Count per Second (P_{t_n}) *at Time* t_n. It describes the average packet rate of a flow, port, or switch between time t_{n-1} and t_n:

$$P_{t_n} = \frac{C_{t_n}^{\text{Pkt}} - C_{t_{n-1}}^{\text{Pkt}}}{t_n - t_{n-1}}. \tag{2}$$

Byte Count Asymmetry ($A_{t_n}^{\text{Byte}}$) *at Time* t_n. It describes the average byte rate asymmetry of pair-flow or port between time t_{n-1} and t_n:

$$A_{t_n}^{\text{Byte}} = \frac{B_{t_n}^{\text{in}}}{B_{t_n}^{\text{out}}}. \tag{3}$$

Packet Count Asymmetry ($A_{t_n}^{\text{Pkt}}$) *at Time* t_n. It describes the average packet rate asymmetry of pair-flow, port, or switch between time t_{n-1} and t_n:

$$A_{t_n}^{\text{Pkt}} = \frac{P_{t_n}^{\text{in}}}{P_{t_n}^{\text{out}}}. \tag{4}$$

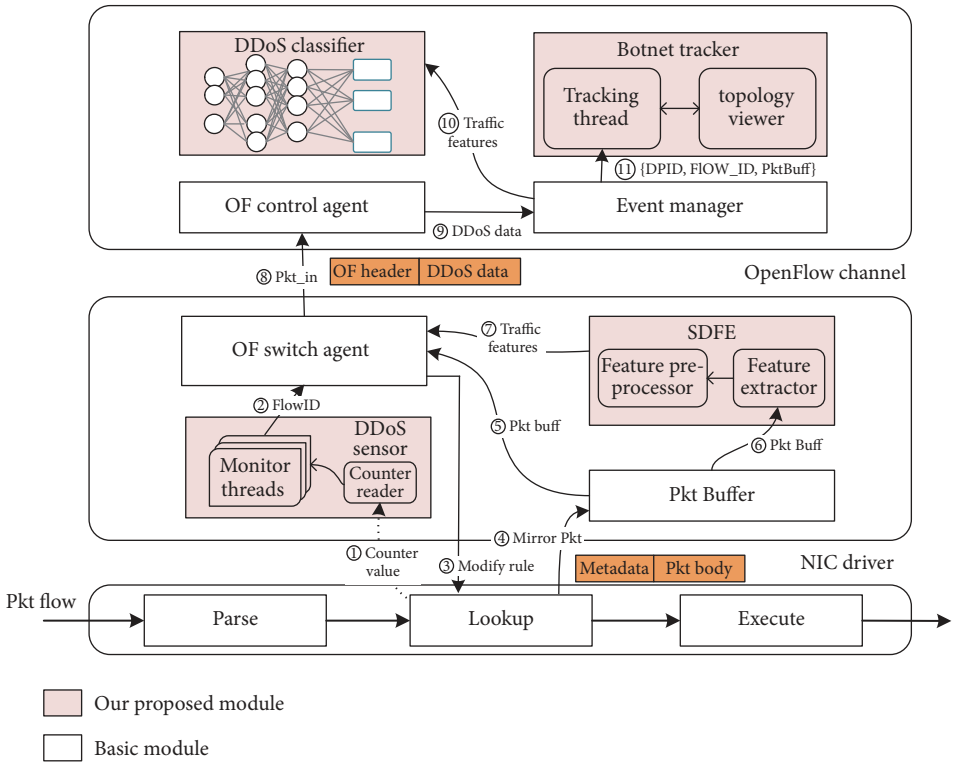

Figure 5: Implementation and workflow of OverWatch in detection phase.

We present our prediction-based algorithm to capture great changes of the above four metrics. This algorithm leverages previous metric samples from a specific flow to estimate a future value range. If the actual values of the four metrics fall into the range we predict, this indicates that the current flow is normal. Otherwise, the deviation between the predicted values and observed values indicates an abnormal flow caused by a DDoS attack.

Specifically, we leverage WMA (Weighted Moving-Average) to calculate prediction value for each metric. And Pauta criterion in Gaussian distribution is also utilized to get a reasonable prediction range. The pseudocode of this algorithm is shown in Algorithm 1.

4.3. Fine-Grained Detection on Control Plane. As a centralized and often high-performance platform, the control plane holds advantages of abundant computing resources and holistic info of the whole network. Thus, on the control plane, two functionalities are developed: DDoS attack classification and botnet tracking. Both of them are essential for attack reaction as they determine which actuator is to be deployed and on which data plane switch it is to be deployed. We introduce our machine learning based classification model as well as a lightweight botnet tracking algorithm below.

4.3.1. Autoencoder-Based Attack Classification. Machine learning has gained much attention in the community of network security as it improves the accuracy and reduces

globals: $\mathbf{V}[n]$ //Vector List of DDoS attack feature metrics
t_n //Current time
n //Number of Vectors in the list
while 1 **do**
if $t_n = t_{n-1} + \Delta t$ **then**
Update the list $\mathbf{V}[n]$ of history records.
for all $i \in \{1, 2, 3, 4\}$ **do**
Use WMA to calculate the prediction value $\mathbf{V}_{n+1}^{\text{predict}}$ for next time interval:

$$\mathbf{V}_{n+1}^{\text{predict}} = \sum_{i=1}^{n} \lambda_i \mathbf{V}_i^{\text{actual}} \quad \sum_{i=1}^{n} \lambda_i = 1$$

Use ratio metric R^{predict} to compare prediction value and actual value.
Calculate ideal value v_i^{ideal} and standard deviation σ for ratio metric.
Use Pauta criterion to calculate the prediction range:

$$R_u \longleftarrow v_i^{\text{ideal}} + 3 \times \sigma$$
$$R_l \longleftarrow v_i^{\text{ideal}} - 3 \times \sigma$$

end for
if each R^{predict} is out of the (R_l, R_u) range **then**
Trigger alert to controller;
else
Continue;
end if
end if
end while

Algorithm 1: Lightweight flow monitoring algorithm.

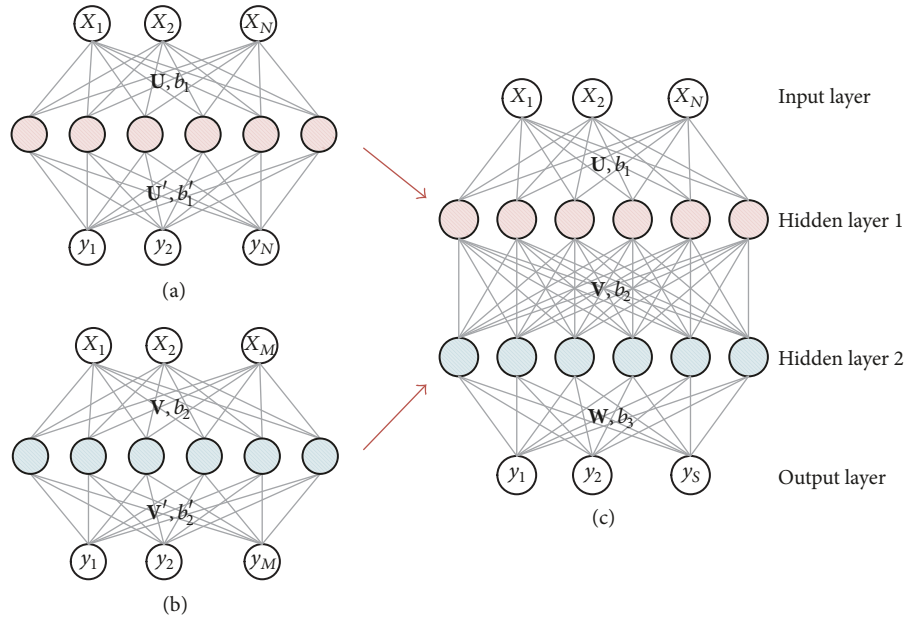

Figure 6: Schematic representation of autoencoder and DDoS attack classifier model: (a) autoencoder for A_1; (b) autoencoder for A_2; (c) the model of DDoS attack classifier.

false-positive rate while classifying different types of abnormal traffic. To determine the attack type from real-time extracted traffic features, a machine learning method, combined with autoencoder [34] and softmax classifier [35], is utilized in the module of DDoS attack classifier.

As shown in Figure 6, each autoencoder contains three layers: input layer, hidden layer, and output layer. Two autoencoders (A_1 and A_2) are stacked with each other in a way that the outputs of first hidden layer are fed into the inputs of the second. Then, the outputs of the second hidden layer are fed into a softmax classifier. Finally, all layers stacked together, forming the DDoS attack classifier. Generally, if the extracted traffic features are fed into the input layer, the output vector of the model indicates to which attack type the features belong. We introduce the structure below.

The autoencoder has three layers: an input layer of N nodes for a record of N features (i.e., $X = \{x_1, x_2, \ldots, x_N\}$), a hidden layer of M nodes for learning key patterns of input record, and a output layer of N nodes for reconstruction of the input (i.e., $\widehat{X} = X$). The network finds optimal values of weight matrix (i.e., $U \in R^{N\times M}$ and $U' \in R^{M\times N}$) and bias vector (i.e., $b_1 \in R^{N\times 1}$ and $b_1' \in R^{M\times 1}$) together, while trying to learn the key patterns of an input record (e.g., a DDoS attack record). The second autoencoder feeds the outputs of the first one as its input. It uses the same methods to calculate the optimal weight matrix $V \in R^{P\times N}$ and bias vector $b_2 \in R^{P\times 1}$.

Then, a softmax classifier builds a mapping relationship between the hidden layer of the former autoencoder (i.e., $H = \{h_1, h_2, \ldots, h_M\}$) and S types of DDoS attacks (i.e., $Y = \{y_1, y_2, \ldots, y_S\}$). Similarly, This network finds optimal values of weight matrix (i.e., $W \in R^{M\times S}$) and bias vector (i.e., $b_3 \in R^{S\times 1}$) too, while polling the output close to the label value, which indicates the real DDoS type. Before this model is able to classify any record collected from DDoS traffic, each layer needs to be trained with backpropagation algorithm [36], separately. Then, they are stacked together and fine-tuned to improve the performance of the entire model. The brief training process for the first autoencoder is shown in Algorithm 2, and the other two layers share similar training process.

After the training process with historical DDoS attack dataset, this model can be utilized to perform attack classification with run traffic records.

4.3.2. Collaborative Botnet Tracking. The collaborative botnet tracking aims to locate the switches close to the botnets, thus making the defense actuators more effective in defending DDoS attacks. We propose a botnet tracking algorithm based on the collaboration of the data and control plane and implement it in the controller as a built-in module, called botnet tracker.

Before diving into the details of the algorithm, two prerequisites need to be stressed out: The first one is that the whole network info (link state, topology info, forwarding rule, etc.) is maintained on the controller. The second prerequisite is that the info maintained by the controller can be accessed by other built-in applications on the controller. Fortunately, both prerequisites can be satisfied by leveraging an enhanced topology viewer [37], a built-in module in Ryu. More specifically, the key procedure in the algorithm is to obtain the last hop of a sampled packet. This could be achieved by extracting the source MAC address of the packet and leveraging the network info on the controller to locate the last hop switch.

Based on above, we propose our botnet tracking algorithm. Specifically, we consider a set A consisting a set of switches that have captured DDoS attacks on themselves $A = \{a_1, a_2, \ldots, a_m\}$. And S is the total set of data plane switches a

Require: Weight matrix of the hidden layer: U
Bias vector of the hidden layer: b_1
Ensure: Training dataset e
for all number of training iterations **do**
Train the single layer autoencoder using back-propagation:
$\{X^1, X^2, \ldots, X^m\} \leftarrow$ Sample minibatch of e //get batches of traffic records
Update U, U', b, b' by using gradient descent method to minimize the loss function:

$$\text{Loss} = \left(\frac{1}{2m}\sum_{i=1}^{m}\|X^i - \widehat{X}^i\|^2\right) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^l)^2$$

end for

Algorithm 2: DDoS attack classifier training process.

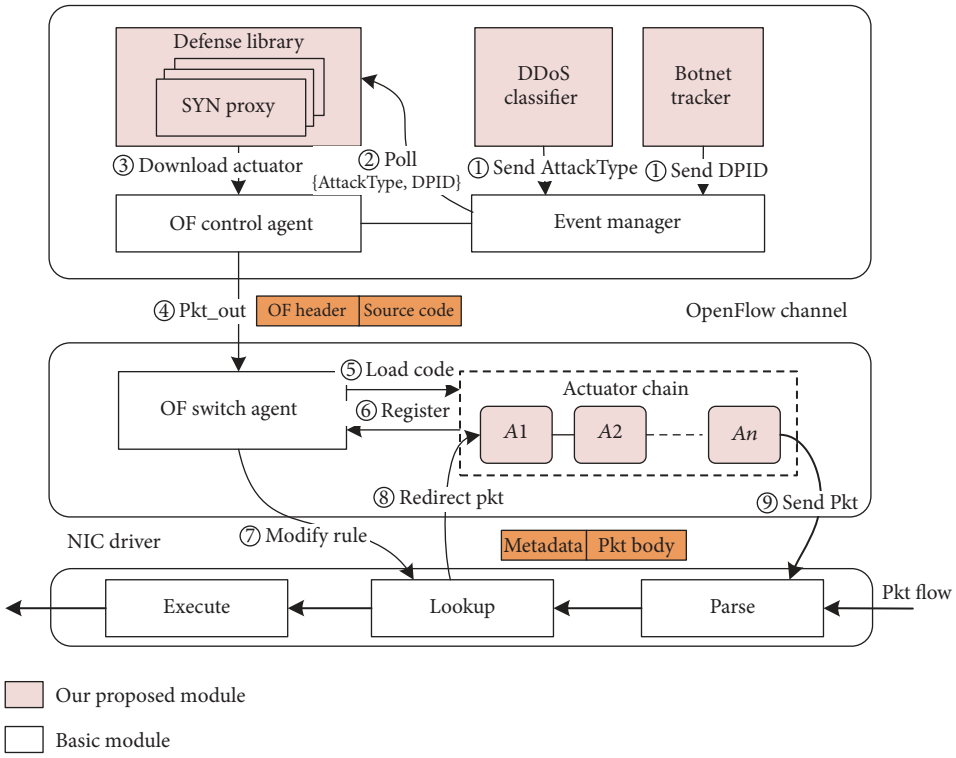

Figure 7: Implementation and workflow of OverWatch in response phase.

controller maintains $S = \{s_1, s_2, \ldots, s_n\}$. We take one element a_i from set A at a time, and use the sampled packets collected from a_i to determine the last hop switch s_k. If $s_k \in A$, then we eliminate a_i from set A. Otherwise, a_i is one of the switches we are searching for. We use this method to traverse set A and obtain a subset B ($B \subseteq A$) consisting of all a_i whose last hop is not included in A. In this way, we are able to locate all the switches which are likely close to botnets.

5. Reaction Phase of OverWatch

In this section, we express how we design and implement the reaction phase of OverWatch. After the attack type and switches that are close to botnets are addressed in the former phase, OverWatch is supposed to react to the occurring attack efficiently. Thus, first, we illustrate the workflow of the reaction phase in our prototype. Then, two reaction applications are introduced in the second part.

5.1. Attack Mitigation. The workflow of reaction phase in OverWatch is depicted in Figure 7. From aforementioned, the specific attack type and the most close-in switches can be determined in the detection phase, respectively. Then, both results are sent back to the event manager. This triggers the defense library, which has registered to the event manager, to poll up the messages. This module firstly leverages the attack

type to match a particular actuator so as to indicate the source code which is required to be loaded onto data plane. Then, it invokes functions provided by OpenFlow control agent to download the specific source code onto the switch whose DPID matches the received message from the event manager.

In OverWatch, the source code of designated actuator is encapsulated inside a `OFPT_EXPERIMENTER` message (defined in OpenFlow specification since 1.1) and sent to the specific switch through OpenFlow channel. The corresponding switch agent running on that switch receives the message and loads the codes of the actuator in the running space. Then, the source code is compiled to an executable file and then registered back to the switch agent, which later allocates an exclusive ID to the specific actuator so that it could be added to the actuator chain. Meanwhile, the actuator also notifies the switch agent of the packet type it processes. Once this is received by the agent, the agent generates a high priority flow rule and adds it to the flow table using a `OFPT_FLOW_MOD` message.

After the rule modification takes effect, packets that match the higher priority rule are redirected to the actuator chain with metadata that could match a certain ID of the actuator, in which they are going to be processed. In addition, packets sent back to the hardware are also allocated with metadata to match the lower priority rule so that they can be forwarded by the switch's original rules.

5.2. Intelligent Reaction Applications. We develop two sample applications of DDoS defense actuators to exemplify the feasibility of OverWatch. This includes SYN proxy and DNS reflection filter. These two actuators are motivated by (1) the functionality that Avant-Guard [16] proposed as a data plane extension to defend SYN flood and (2) the example DNS filter proposed in SDPA [38] to filter out DNS refection attack packets on OpenFlow switches.

5.2.1. SYN Proxy. In a SYN flood, attackers send numerous SYN packets to exhaust memory resources of victim by enforcing it maintaining a large number of semiconnected states, so the victim will not respond to affirmed connection request. To filter out these malicious SYN requests, an OpenFlow rule is preloaded to lookup table so that all SYN packets will be polled onto an actuator called SYN proxy. If a SYN packet is received by it, firstly, to prevent multiple SYN requests sending to the victim, it calculates a cookie to record the request using harsh table, and then the actuator generates a response packet sent back to the source and drops the initial SYN request packet. If, during a certain time interval, an affirmed ACK packet is received from a source that has been recorded in the harsh table, the proxy will generate TCP handshake packets to build up validation between the legal source and its destination. Otherwise, the proxy simply drops the malicious packets. In this way, the proxy eliminates the threat of SYN flood.

5.2.2. DNS Filter. Botnets in a DNS reflection attack send DNS requests to name servers using the victim host's IP address. Thus, the victim will be flooded by these massive DNS responses. To filter out these unsolicited DNS response, once the actuator (DNS filter) is loaded on the data plane, two high priority OpenFlow lookup rules, which redirect the packets whose UDP source or destination port equals 53, are loaded as well. After the redirected DNS packets are received by the filter, each DNS request packet is recorded by a five-tuple (source IP, destination IP, source port, destination port, protocol) in memory. If the upcoming packet is a DNS response packet and matches one of the tuples of request, it is sent to hardware pipeline to match a lower priority rule for forwarding. On the contrary, the filter drops the packet to protect the victim.

Figure 8: Diagram of our testbed network.

6. Experiment and Evaluation

6.1. Experiment Setup. To evaluate the performance of our proposed OverWatch framework, we modified a FPGA-based (Altera EP4SGX180) OpenFlow switch [28] to support the aforementioned data plane functions. We also modified Ryu controller to enable proposed controller-based mechanisms (including the adding of three built-in applications: DDoS attack classifier, botnet tracker, and defense library).

Figure 8 illustrates the testbed of our experiment. It consists of a FPGA-based OpenFlow switch prototype with 8 Gigabit ports, a 1.99 GHz Intel Celeron J1900 CPU and a 2 GB memory that runs Ubuntu 14.04 on it, a control platform with a quad-core Intel i7 CPU and a dual NVIDIA-GTX1080 GPU (used for training machine learning based classifier) with 16 GB of RAM running Ryu controller, and up to eight laptop hosts, which represent DDoS attackers, victims, and normal traffic generators, respectively.

6.2. Efficiency of Coarse-Grained Detection. In order to evaluate the performance of the algorithm running as DDoS attack sensors, we firstly add two rules to enable traffic forwarding from H_1 and H_2 to H_8 ($H_1, H_2 \rightarrow H_8$). Then, Stacheldraht [39] is utilized to conduct DDoS attacks from H_1 and H_2 to H_8 within 20 seconds. The detection results in the DDoS attack sensor, together with volume and asymmetry features of attack flow ($H_1 \rightarrow H_8$), are depicted in Figure 9. These depicted results demonstrate that the four metrics in the DDoS attack sensor are able to capture great changes in volume and asymmetry features as soon as the attack occurs, which also evidently shows the effectiveness of the algorithm.

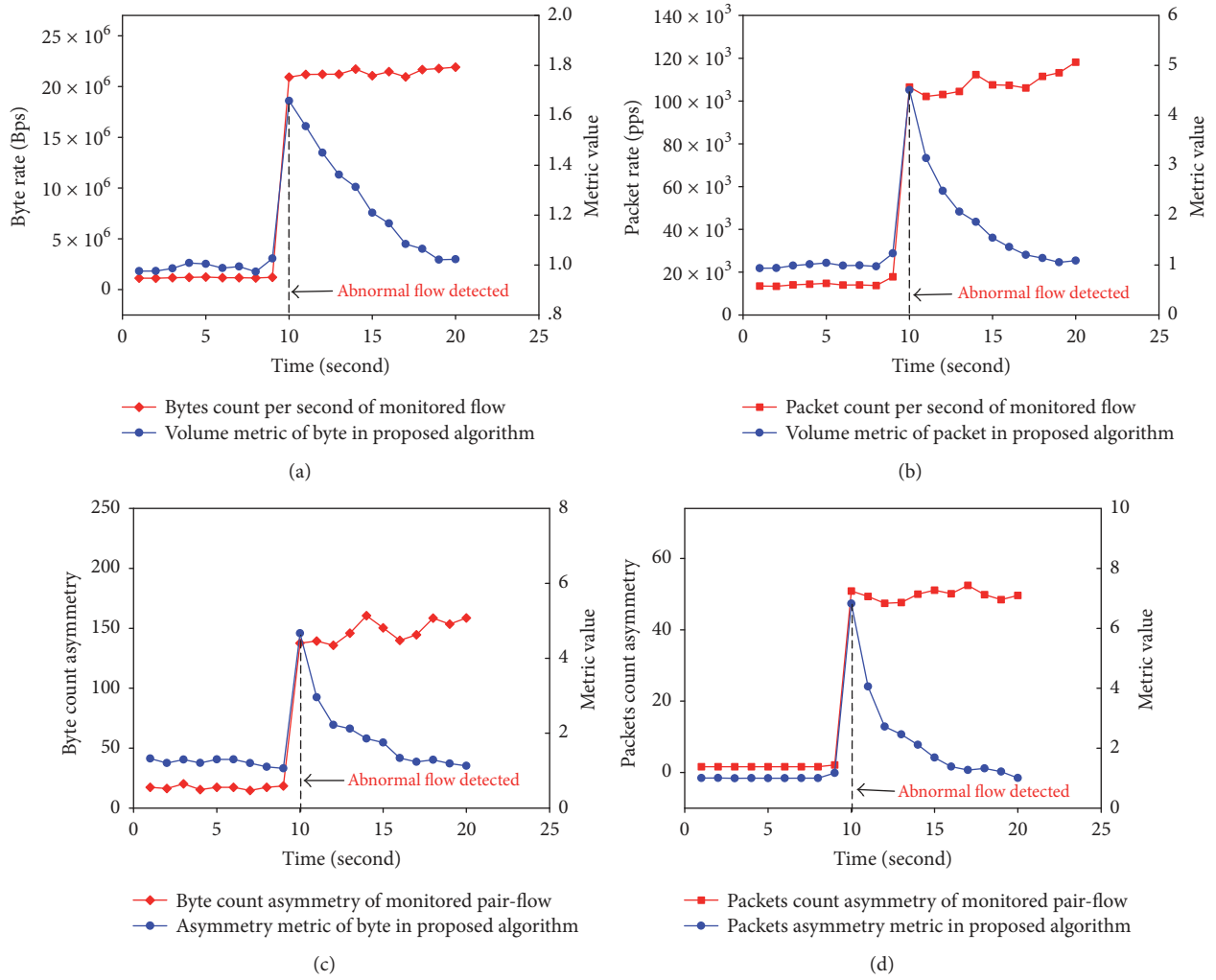

FIGURE 9: The coarse-grained detection results during a SYN flood attack [18]: (a) byte rate and volume metric of byte in monitored flow; (b) packet rate and volume metric of packet in monitored flow; (c) byte count asymmetry and asymmetry metric of byte in monitored pair-flow; (d) packet count asymmetry and asymmetry metric of packet in monitored pair-flow.

Next, we use FTP to transfer a 4 GB data block from H_1 to H_8. The detection results of four metrics are shown in Figure 10. It is found that even though three of the metrics dramatically change in the algorithm result, the asymmetry feature of packet count barely changes. This is because during the data transfer, the receiver H_8 keeps sending ACK packets to H_1, which ensures a rough equivalence of packet count in both directions. This explanation can be testified by using Wireshark to capture packets from H_1 or H_8. Therefore, such mechanisms in the proposed algorithm enable the DDoS attack sensor to reduce the misjudgment rate of DDoS attack detection.

To demonstrate the advantage of communication overhead reduction in OverWatch, we move the DDoS attack sensor to the controller side and use the same algorithm to poll switch countervalues through OpenFlow channel. We set this typical controller-based DDoS detection method as a baseline method, Besides, we also implement another mechanism, which optimizes the baseline method by utilizing an adaptive polling algorithm proposed in Payless [14] to reduce communication overhead of southbound interface. Then, we implement these three methods in a scenario where different times of DDoS attacks are conducted from random hosts within 1 minute. Wireshark is utilized to capture all the packets of southbound interface during the experiment. The total amount of southbound overhead is shown in Figure 11. It is found that OverWatch has orders of magnitude less overhead than the other two methods. This is achieved by offloading the DDoS attack sensor onto data plane. And compared with Avant-Guard, which can reach similar results, there is no modification introduced in the switch hardware.

6.3. Efficiency of Fine-Grained Detection. To demonstrate the performance of the DDoS attack classifier in OverWatch, we collected network traffic from the testbed as training dataset. Specifically, we use H_1 and H_2 to send DDoS attack traffic to H_8. Hosts from H_3 to H_6 communicated with each other randomly for web browsing, video transferring, and

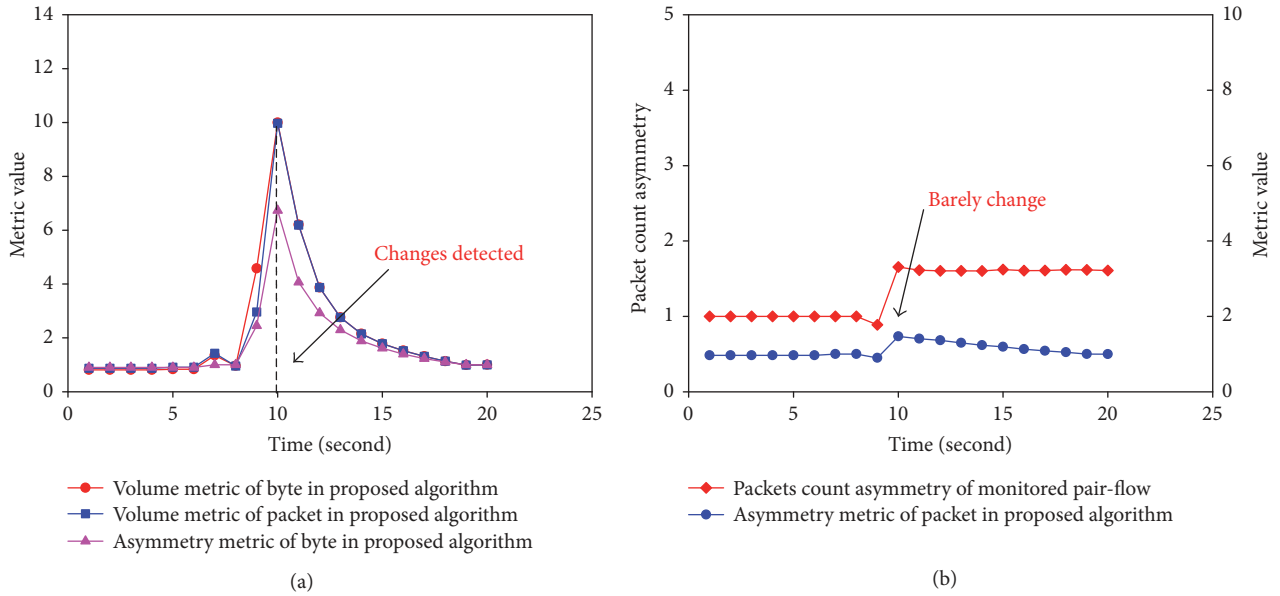

Figure 10: The coarse-grained detection results during big data block transferring using FTP [18]: (a) the changes of three metric values except asymmetry metric of packet; (b) the changes of packet count asymmetry and asymmetry metric of packet in monitored pair-flow.

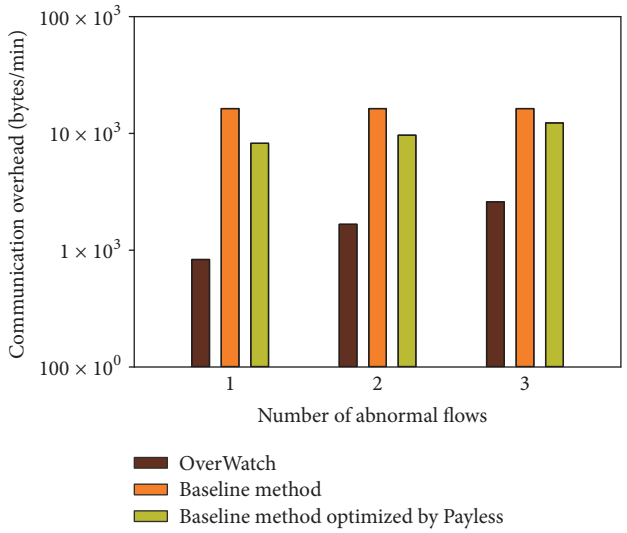

Figure 11: Overhead of southbound interface in OverWatch, baseline method, and baseline method optimized by Payless within 1 minute [18].

Table 1: Number of records in training dataset.

Traffic class	Records number	
	Training	Test
Normal traffic	17539	9824
Attack traffic		
SYN flood	2831	1566
UDP flood	2706	1591
ICMP flood	2519	1630
SYN & UDP flood	3054	1321
UDP & ICMP flood	2936	1729
SYN & ICMP flood	3144	1538

online gaming, which led to background traffic variation. We mirrored all the traffic to H_7 by tcpdump, so that the traffic can be leveraged as dataset. We collected 12 hours of traffic data in total, including 6 hours of normal traffic and 6 hours of attack traffic. DDoS attacks in the conducted experiment consist of 6 types: UDP flood, SYN flood, ICMP flood, and their permutations. They are all generated by Stacheldraht. And Table 1 shows the distribution of records in the dataset. The features we extracted from traffic flows during the classification are listed in Table 2, while Table 3 lists the parameters in our experiment. In the training and evaluation process, the features are real-valued positive numbers by digitization and max-min normalization to improve accuracy.

We evaluate the performance of the classifier using parameters including confusion matrix, precision, recall, and f-measure. In a confusion matrix, each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. In binary classification, precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both of them indicate the performance of the classifier. F-measure considers both parameters above to compute a score, which is the harmonic average of the parameters, where f-measure reaches its best value at 1 and worst at 0. Figure 12 illustrates the confusion matrix of our evaluation. It is observed that our DDoS attack classifier has fairly good accuracy for detecting single type of DDoS attacks, reaching about 96%. The detection accuracy for mixed attacks is lower but still reaches around 83%.

Table 2: Features extracted from different packets.

Packet type	#	Feature description
TCP	1	Fraction of TCP packets with SYN flag set
	2	Fraction of TCP packets with ACK flag set
	3	Entropy of src IP addresses
	4	Entropy of dst IP addresses
	5	Entropy of src ports
	6	Entropy of dst ports
	7	Entropy of TCP sequences
UDP	8	Fraction of dst port ≤ 1024 UDP packets
	9	Fraction of dst port > 1024 UDP packets
	10	Entropy of src IP addresses
	11	Entropy of dst IP addresses
	12	Entropy of length for UDP packets
ICMP	13	Entropy of src IP addresses
	14	Entropy of dst IP addresses
	15	Entropy of TTL values
	16	Fraction of ICMP packets in total

Table 3: Autoencoder training parameters.

Parameter	Value
Learning rate	0.1
Batch size	5
Epoch limit	3500

More specifically, we demonstrate the precision, recall and f-measure for 8 types of traffic in Figure 13. Except the mixed traffic of SYN and UDP flood as well as SYN and ICMP flood, the 3 parameters for classification of all types traffic reach above 90%, which is quite acceptable in classifying DDoS attacks in real network.

We claim that the performance of the autoencoder-based classifier is not necessarily better than other machine learning approaches, but these results demonstrate the great feasibility of leveraging machine learning approaches to serve OverWatch as a DDoS attack classifier, which is the main purpose of the evaluation above.

6.4. Performance of Attack Reaction. We also evaluated the performance of defense actuators on the data plane. We set a scenario where we redirected packets from a certain port to a defense actuator, which performs no operations to the packets, and then send the packets back to a designated port. We compare this forwarding path with another two baseline methods. The first one is direct hardware forwarding; the second one is redirecting the packets to the Ryu controller and then sending them back to a designated port. The main goal here is to evaluate how much performance gain is introduced by the defense actuator in OverWatch, compared with traditional SDN-based defense mechanisms. According

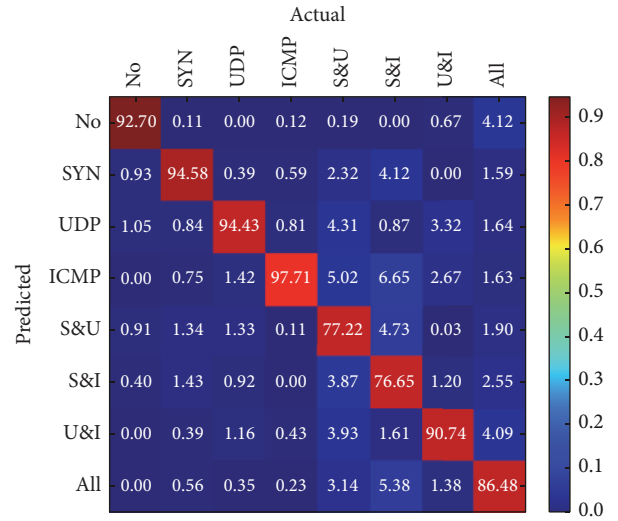

Figure 12: Confusion matrix for DDoS attack classifier in OverWatch.

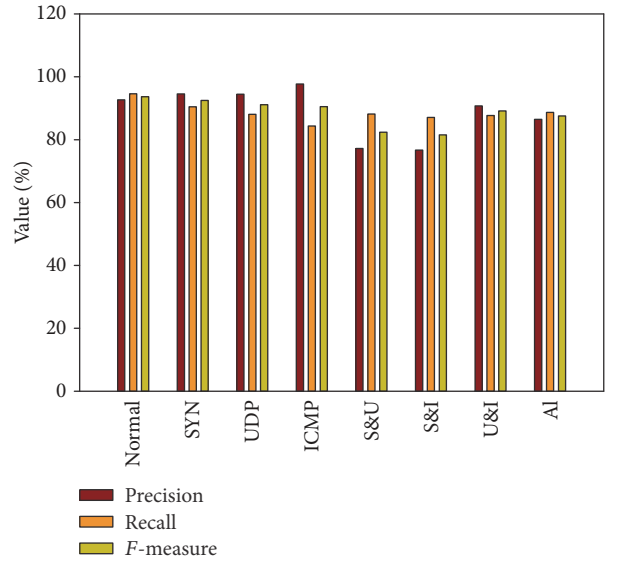

Figure 13: Precision, recall, and f-measure for the DDoS attack classifier in OverWatch.

to the evaluation result shown in Figure 14, defense actuators in OverWatch perform several orders of magnitude better than the controller-based reaction mechanisms. Moreover, though the forwarding performance for actuators is evidently lower compared with hardware path, this could be improved if high-efficient data path (e.g., DPDK [40]) is utilized for the communication between software and hardware on the switch.

7. Conclusion

To overcome the challenges of southbound bottleneck and the lack of collaborative intelligence in SDN-based DDoS attack defense mechanisms, in this paper, we introduced OverWatch, a SDN-based high-efficient cross-plane DDoS

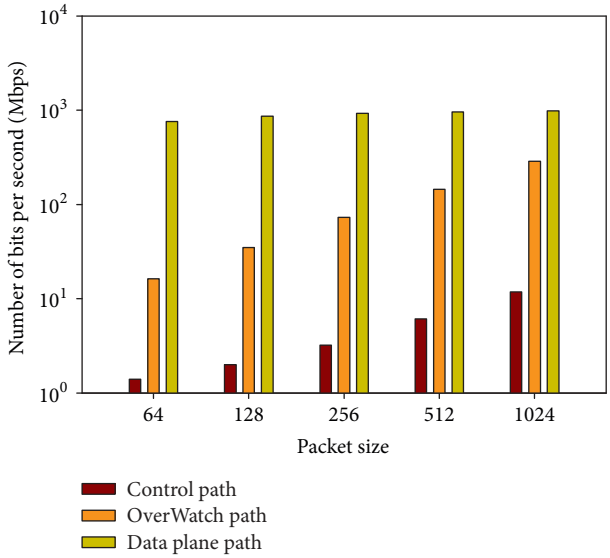

FIGURE 14: Different bits rate according to packet size using different paths in the testbed.

attack defense framework with collaborative intelligence. It is collaboratively splitting defense functionalities across data and control plane and enabling both planes to detect and defend against DDoS attacks on different levels. Through experiments, it can be concluded that OverWatch is capable of high accuracy detection and real-time defending reaction. Meanwhile, the communication overhead on SDN southbound interface is also greatly reduced. All these outcomes demonstrate the feasibility of OverWatch in large-scale networks. We anticipate that OverWatch becomes a building block in the SDN-based network security applications.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported in part by Chinese National Programs for High Technology Research and Development (863 Programs) (Grant no. 2015AA016103), the project of National Science Foundation of China (Grant no. 61601483), and Startup Research Grant of National University of Defense Technology (Grant no. JC15-06-01).

References

[1] S. Yu, Y. Tian, S. Guo, and D. O. Wu, "Can we beat DDoS attacks in clouds?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2245–2254, 2014.

[2] D. Bisson, "The 5 most significant ddos attacks of 2016," https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/5-significant-ddos-attacks-2016/.

[3] D. Geneiatakis, G. Portokalidis, and A. D. Keromytis, "A multilayer overlay network architecture for enhancing IP services availability against DoS," in *Proceedings of the International Conference on Information Systems Security*, vol. 7093, 2011.

[4] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer DoS defense against multimillion-node botnets," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM'08*, pp. 195–206, August 2008.

[5] P. Mittal, D. Kim, Y.-C. Hu, and M. Caesar, *Mirage: Towards Deployable Ddos Defense for Web Applications*, 2011.

[6] H. Wang, Q. Jia, D. Fleck, W. Powell, F. Li, and A. Stavrou, "A moving target DDoS defense mechanism," *Computer Communications*, vol. 46, pp. 10–21, 2014.

[7] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[8] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN '10)*, pp. 408–415, Denver, Colo, USA, October 2010.

[9] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM 2016*, pp. 1–9, April 2016.

[10] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *Proceedings of the 2015 International Conference on Computing, Networking and Communications, ICNC 2015*, pp. 77–81, February 2015.

[11] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the 2013 9th ACM International Conference on Emerging Networking Experiments and Technologies, CoNEXT 2013*, pp. 25–30, December 2013.

[12] Y. Cui, L. Yan, S. Li et al., "SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks," *Journal of Network and Computer Applications*, vol. 68, pp. 65–79, 2016.

[13] S. Oueslati, J. Roberts, and N. Sbihi, "Flow-aware traffic control for a content-centric network," in *Proceedings of the IEEE Conference on Computer Communications, INFOCOM 2012*, pp. 2417–2425, March 2012.

[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, NOMS 2014*, pp. 1–9, May 2014.

[15] J. Seo, C. Lee, T. Shon, K.-H. Cho, and J. Moon, "A new DDoS detection model using multiple SVMs and TRA," in *Proceedings of the EUC Workshops*, 2005.

[16] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pp. 413–424, November 2013.

[17] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1206–1219, 2017.

[18] X. Yang, B. Han, Z. Sun, and J. Huang, "Sdn-based ddos attack detection with cross-plane collaboration and lightweight flow monitoring," in *Proceedings of the Global Communications Conference*, 2017.

[19] A. Mahimkar, J. Dange, V. Shmatikov, H. M. Vin, and Y. Zhang, "dfence: Transparent network-based denial of service mitigation," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[20] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and Software-Defined Networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[21] K. Kalkan, G. Gur, and F. Alagoz, "Defense Mechanisms against DDoS Attacks in SDN Environment," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 175–179, 2017.

[22] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of the 2013 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013*, pp. 73–78, August 2013.

[23] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," RFC Editor RFC3176, 2001.

[24] M. Aslan and A. Matrawy, "On the impact of network state collection on the performance of SDN applications," *IEEE Communications Letters*, vol. 20, no. 1, pp. 5–8, 2016.

[25] Z. Cai, Z. Wang, K. Zheng, and J. Cao, "A Distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417–427, 2013.

[26] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *Proceedings of the Network and Distributed System Security*, 2013.

[27] "Defense4all:tutorial," https://wiki.opendaylight.org/view/Defense4All:Tutorial.

[28] J. Mao, B. Han, Z. Sun, X. Lu, and Z. Zhang, "Efficient mismatched packet buffer management with packet order-preserving for OpenFlow networks," *Computer Networks*, vol. 110, pp. 91–103, 2016.

[29] L. Boero, M. Cello, C. Garibotto, M. Marchese, and M. Mongelli, "BeaQoS: Load balancing and deadline management of queues in an OpenFlow SDN switch," *Computer Networks*, vol. 106, pp. 161–170, 2016.

[30] J. Sonchack, J. M. Smith, A. J. Aviv, and E. Keller, "Enabling practical software-defined networking security applications with ofx," in *Proceedings of the Network and Distributed System Security*, vol. 16, pp. 1–15, 2016.

[31] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 3–10, ACM, 2003.

[32] O. S. S. Version, *Openflow Switch Specification 1.5. 1 (Protocol Version 0x06)*, 2014.

[33] F. Tomonori, "Introduction to ryu sdn framework," in *Proceedings of the Open Networking Summit*, April 2013.

[34] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[35] R. Gens and P. Domingos, "Deep symmetry networks," in *Proceedings of the 28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014*, pp. 2537–2545, December 2014.

[36] L. Wang, Y. Zeng, and T. Chen, "Back propagation neural network with adaptive differential evolution algorithm for time series forecasting," *Expert Systems with Applications*, vol. 42, no. 2, pp. 855–863, 2015.

[37] Y. Hideki, "Topology viewer," https://github.com/osrg/ryu/blob/master/doc/source/gui.rst.

[38] S. Zhu, J. Bi, C. Sun, C. Wu, and H. Hu, "SDPA: Enhancing stateful forwarding for software-defined networking," in *Proceedings of the 23rd IEEE International Conference on Network Protocols, ICNP 2015*, pp. 323–333, November 2015.

[39] D. Dittrich, *The "Stacheldraht" Distributed Denial of Service Attack Tool*, 1999.

[40] D. Intel, *Data Plane Development Kit*, 2015.