Collaborative Defense Against Hybrid Network Attacks by SDN Controllers and P4 Switches

You-Chiun Wang and Pin-Yu Su

Abstract—*Software-defined networking (SDN)* uses a controller to manage the network. Applying SDN to resist *distributed denial-of-service flood (DDoS-F)* attacks receives attention. A controller identifies attack flows and gives rules to switches to discard attack packets. Doing so may cause the controller to be busy and impact SDN performance. P4 switches, on the other hand, can recognize DDoS-F attacks without controller involvement. However, some non-DDoS attacks like keylogging and data theft cannot be well identified by P4 switches due to their local views. Thus, the paper makes the controller and P4 switches cooperate to defend against *hybrid network attacks* that include both DDoS-F attacks and non-DDoS attacks. To this end, we propose a *collaborative defense by control and data planes (CD2P)* framework. P4 switches (i.e., data plane) find DDoS-F packets by using an entropy-aware detection scheme that can adjust thresholds based on the network status. They also report flow information (excluding DDoS-F flows) to the controller. With the deep learning technique, the controller (i.e., control plane) analyzes these reports to discover non-DDoS attacks. Hence, the controller can focus on detecting these attacks without the disturbance of many DDoS-F packets. Experimental results reveal that CD2P can quickly block DDoS-F attacks and better identify keylogging and data theft. Our contribution is to propose a novel framework for the controller and P4 switches to collaborate to defend against hybrid network attacks efficiently.

Index Terms—DDoS flood, deep neural network (DNN), hybrid network attack, P4, software-defined networking (SDN).

1 Introduction

DISTRIBUTED denial-of-service (DDoS) has always been a thorny network attack problem, and the frequency of its occurrence increases year by year. In a *DDoS flood (DDoS-F)* attack, a victim (i.e., host or server) is deluged with numerous requests in an attempt to exhaust the victim's resources and prevent legitimate requests from being served. DDoS-F attacks are usually performed by a botnet with compromised devices like computers and IoT devices whose security is breached [1]. This makes a conventional firewall hard-pressed to resist DDoS-F attacks, as an attack's packets originate from many sources that could be irrelevant. Even worse, the IP addresses and ports of packets can be forged to deceive the firewall.

The *software-defined networking (SDN)* technique displaces the control plane from switches to a central entity, namely the *controller*, to facilitate network management. More concretely, users can write programs on the controller to implement their policies or algorithms, and the controller then guides switches to process packets accordingly by installing flow rules [2]. In addition, the controller can query switches about the number and types of packets handled by them to monitor the network status. Due to its flexibility, using the SDN technique to defend against DDoS-F attacks attracts considerable attention [3].

However, DDoS-F attacks could degrade SDN performance. In particular, when an attack is in progress, some switches receive numerous packets from different flows. Since switches have no appropriate flow rules to process these packets, they need to ask the controller for instructions [4]. Besides, some countermeasures (discussed in Section 3) make the controller check packets to reduce false alarms. Inevitably, the controller is busy replying to switches or identifying DDoS-F packets. Since the controller takes charge of coordinating the network, the performance of SDN may be degraded.

On the other hand, P4 (standing for *programming protocol-independent packet processors*) is a domain-specific language developed for network devices (e.g., switches) [5]. P4 allows users to specify how these devices cope with packets, thereby giving them the ability to independently judge packets. Several studies indicate that DDoS-F packets can be efficiently identified by P4 switches without the controller's help. However, some cunning attacks (e.g., keylogging and data theft) cannot be well recognized by each P4 switch due to its local view. Such attacks need to be identified by the controller, which can obtain information from all switches.

With the above motive, this paper lets the controller and P4 switches work together in the defense mechanism. We consider *hybrid network attacks* that include DDoS-F attacks and non-DDoS attacks. P4 switches act as the first-line defense against DDoS-F attacks and filter out their malicious packets. Without the disturbance from DDoS-F flows, the controller can enable a more efficient analysis of non-DDoS attacks. Suggesting using the controller and P4 switches together in a hybrid setup is not simply an observation. Instead, there will be three challenges:

1. There is variability in the network, and some P4 switches may see only parts of the attack packets. Without coordinating switches (e.g., using the controller), how can each switch accurately identify attack packets by itself?
2. The P4 switches store flow information and report it to the controller for analysis. Due to the bandwidth and resource constraints of switches, how to save their communication, computation, and storage overheads is critical.
3. To detect non-DDoS attacks, the *machine learning (ML)* technique is applied to the controller to analyze reports sent by P4 switches. How can we improve the analysis

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, 80424 Kaohsiung, Taiwan (e-mail: ycwang@cse.nsysu.edu.tw; turningpoint1125@gmail.com).

efficiency, for example, using one ML model to recognize multiple non-DDoS attacks?

To address the three challenges, we develop a *collaborative defense by control and data planes (CD2P)* framework. For challenge 1, we propose an *entropy-aware detection (EAD)* scheme in CD2P for P4 switches to identify DDoS-F packets, where the thresholds on IP entropies will be adaptively adjusted depending on the network status. Compared to existing entropy-based methods that use fixed thresholds, EAD helps switches adapt themselves to changes in the network. Hence, even though some switches see just parts of the attack packets, they can identify attack packets accurately. For challenge 2, P4 switches weed out DDoS-F packets using the EAD scheme, which has low computation and memory complexity. Then, switches select features of non-DDoS-F flows to be recorded (and reported) to diminish communication, computation, and storage overheads. Moreover, switches report flow information only when the controller is not busy. Doing so avoids overloading the controller. For challenge 3, the controller performs data processing on the reports of switches to expedite analysis. Afterward, we construct a shared *deep neural network (DNN)* model for the controller to conduct analysis to recognize two non-DDoS attacks, including keylogging and data theft. With the above designs, our CD2P framework can not only facilitate the cooperation of P4 switches and the controller but also help them better perform their tasks.

The rest of this paper is organized as follows: Section 2 gives background knowledge, Section 3 surveys related work, and Section 4 offers the system model. We detail the CD2P framework in Section 5 and discuss some issues in Section 6. The performance evaluation is presented in Section 7. Section 8 concludes this paper and discusses future work.

2 Preliminary

2.1 SDN Architecture

SDN logically divides the network into application, control, and data planes. The application plane contains user applications, where they can interact with a controller in the control plane via the northbound *application programming interface (API)*. This API obeys the representational state transfer style and offers a programmable interface to help users monitor and change network states. The controller communicates with the switches in the data plane through the southbound API, where OpenFlow is the dominant protocol [6].

In OpenFlow, each switch maintains flow tables to store the instructions issued by the controller. The controller constructs a secure connection via TLS[1] with the switch and adds flow entries to its flow tables. Each flow entry has match fields to let the switch check whether a packet meets specific conditions. If so, the switch follows the entry's instructions to process the packet. OpenFlow also defines group tables and meter tables to provide more sophisticated management of traffic flows [7].

2.2 P4 Language

P4 is a programming language used to control the behavior of switches. With P4, users can define their parsers, protocols, and operations to process packet headers (known as protocol independence) [5]. P4 is also target-independent, meaning that the compiler takes the switch's capability into consideration as it converts a P4 program into a target-dependent binary. Thus, users need not know the underlying hardware of switches. In addition, P4 supports reconfigurability, where both the parser and processing logic can be redefined at runtime.

A P4 program usually contains three components: 1) *Header definitions* delineate packet formats and name the fields within a packet. Customized header names and arbitrary-length fields are allowed in P4. 2) *Parsers* are finite-state machines that extract headers from the incoming byte streams. For example, a parser extracts the source, destination, and type fields. Then, it does further extraction based on the value of the type field (e.g., ipv4 or ipv6). 3) *Match-action tables* are similar to flow tables in OpenFlow. P4 treats the tables as generic, which lets users add their match-action rules through the control plane.

2.3 DDoS-F Attacks

DDoS attacks target the network, transport, and application layers, where they drain a victim's bandwidth or resources. There are various types of DDoS attacks, and DDoS-F attacks are the predominant type [8].

Three kinds of DDoS-F attacks are widely used. In a *TCP SYN flood*, the attacker sends myriads of SYN packets to the victim. This forces the victim to be busy replying (i.e., sending SYN-ACK packets), thereby consuming bandwidth and TCP ports. In a *UDP flood*, a mass of UDP packets are sent to the victim. The victim will spend most of its computing resource on looking over small packets (with 64 bytes) or reassembling large packets (with 1500 bytes). In an *HTTP flood*, the victim is overwhelmed with HTTP GET or POST requests. Eventually, the victim will be saturated with requests and cannot respond to normal traffic from legitimate users.

In addition to generating numerous packets, DDoS-F attacks have some characteristics in terms of IP addresses [3]. Specifically, a few hosts are selected as victims to enhance the attack's strength. Hence, the destination IP addresses of attack packets are typically convergent. Regarding source IP addresses, there are two cases. First, since an attack could be carried out by a botnet of many compromised devices or the attacker may forge many IP addresses, attack packets will reveal source IP address diversity. Second, an attack may originate from merely a few sources, or the attacker could spoof a few IP addresses. This makes the source IP addresses of attack packets convergent. So, a few destination IP addresses along with too many or too few source IP addresses is a symptom of a DDoS-F attack.

2.4 Non-DDoS Attacks

Unlike DDoS-F attacks that breach a victim's availability by flooding it with packets, non-DDoS attacks may invade the victim for some purposes, like making it a botnet member [1] or stealing user data [9]. Non-DDoS attack flows are usually disguised as legitimate ones, and they send a similar number of packets as normal flows. As discussed in Section 3, most DDoS-F countermeasures are based on checking flow anomalies (e.g., sending many packets or diverse IP addresses), so they cannot be applied to identify non-DDoS attacks.

In this work, we consider two common non-DDoS attacks: keylogging and data theft. Keylogging (also called keystroke logging) secretly records input signals from a keyboard into

1. [Protocol acronyms] ACK: acknowledgement, DNS: domain name system, FTP: file transfer protocol, HTTP: hypertext transfer protocol, IP: Internet protocol, SSH: secure shell, SYN: synchronize, TCP: transmission control protocol, TLS: transport layer security, UDP: user datagram protocol.

a computer so that the user does not know [10]. Keylogging capabilities are usually added to different pieces of malware, such as the Zeus Trojan, to hijack personal or financial data. On the other hand, data theft (or information theft) involves unauthorized transfers of personal or financial data. In practice, an attacker can tunnel other protocols (e.g., FTP and SSH) via DNS queries and responses to pilfer data from the victim [9].

3 Related Work

Various SDN-based methods are proposed to counter DDoS-F attacks. In [11], when a host sends over a predefined number of SYN packets but does not yet complete their handshaking procedures, the host is considered an attacker of the TCP SYN flood. Tung et al. [12] let the controller monitor UDP packets passing through the ports of each switch. Once a port has much more received packets than sent packets, they judge that a UDP flood is underway. The work [13] installs an intrusion prevention system in the controller to analyze packets and detect TCP SYN floods. On identifying attacks, the controller updates the firewall rules in switches to drop malicious packets. In [14], the controller conducts the statistical analysis of packets, scores each packet by the relationship between its attributes, and discards low-score packets. The study [15] checks whether DDoS-F attacks occur according to flow size, IP variability, and duration. It distinguishes legitimate elephant and impulse flows, which also carry many packets, from DDoS-F flows to decrease false alarms. Both [16] and [17] apply principal component analysis to examine network traffic for detecting attacks, where a large traffic dataset is converted into a smaller dataset that can keep the most information. Gao et al. [18] detect attacks using the flow entry frequency, which depicts the number of packets in each flow received by a switch. They also adopt table-miss engineering to save link bandwidth and reduce useless flow entries for switches. The above studies ask the controller to identify DDoS-F packets, which may burden it with a heavy load and raise the detection time.

Some studies measure the entropy of IP addresses in packets, and an anomaly in the entropy (e.g., too high) is viewed as a symptom of attacks. Specifically, both [19] and [20] adopt the Shannon entropy:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i, \tag{1}$$

where $X = \{x_1, x_2, \cdots, x_n\}$ is an event set, and p_i $(i = 1...n)$ gives the probability of each event $x_i \in X$. Here, x_i can be treated as the event that packets with a certain IP address are sent to the victim. Li et al. [21] employ the φ-entropy:

$$H_\varphi(X) = -\frac{1}{\sinh(\varphi)} \sum_{i=1}^{n} p_i \sinh(\varphi \log_2 p_i), \tag{2}$$

where $\sinh(\cdot)$ is the hyperbolic sine function. The work [22] monitors a subsection of flows in a time window (called partial flows). The entropy of partial flows is then compared against a threshold to detect anomalies. However, these studies use a fixed threshold on the entropy, which cannot reflect changes in the network. Moreover, they require the controller to calculate the entropy, thereby increasing its workload and the reaction time to block attacks.

How to combat DDoS-F attacks by using P4 switches is also discussed. Shen et al. [23] implement source authentication and anomaly detection on P4 switches to discover TCP SYN flood attacks. In [24], each P4 switch calculates source and destination IP entropies (using constant thresholds) for attack detection. The study [25] performs traffic characterization and anomaly detection via P4 switches, where their computation and memory constraints are considered. Ding et al. [26] use P4 switches to estimate the number of distinct flows that contact the same destination to identify DDoS-F flows. The study [27] allows users to customize defense methods using defense primitives and then maps these primitives to run on P4 switches. In [28], defense primitives are developed for programmable switches (e.g., P4). Besides, a compiler is designed to generate switch programs to extract a panoramic view of attack signals. To decrease the storage and communication overheads, Zhou et al. [29] propose a distributed storage protocol, cooperative APIs, and a memory access proxy for P4 switches to identify attacks. Compared to controller-based methods, P4 switches can find DDoS-F attacks in a distributed manner and reduce detection time. Nevertheless, the above studies do not consider letting the controller also partake in defense mechanisms.

As compared with existing solutions, our paper develops a framework to help the controller efficiently team up with P4 switches against hybrid network attacks. P4 switches identify DDoS-F attacks using the EAD scheme, whose thresholds can be adjusted based on the network status. They also submit flow information to the controller, which performs in-depth analysis via DNN. Thus, the controller can focus on recognizing non-DDoS attacks (e.g., keylogging and data theft) without having to worry about the disturbance from DDoS-F flows.

In the literature, a few methods are proposed to deal with keylogging. The work [30] detects keyloggers on an infected host according to the correlation between user behavior (e.g., keystrokes), file access, and network communication. Nyang et al. [10] propose two visual authentication protocols to avoid keylogging: the one-time-password protocol and the password-based authentication protocol, and implement them using QR codes. Based on human vision properties like motion perception and visual interpolation, the study [31] develops a virtual keyboard to address keylogging. For data theft, the work [32] depicts the implementation of access and identity management for endpoint protection from USB devices to avoid data theft in a corporate environment. In [33], a stochastic forensic method using fuzzy inference is applied in the context of detecting data theft. The work [34] designs a database security framework to protect data privacy and prevent data theft. As can be seen, none of them adopt ML models to detect keylogging or data theft. This motivates us to build a shared DNN model for the controller to recognize these two non-DDoS attacks.

4 System Model

We consider an SDN-based network with a controller and P4 switches, as Fig. 1 shows. The controller coordinates the network, and switches obey their P4 programs and flow rules (issued by the controller) to process packets. More specifically, switches make preliminary judgments about attacks and drop attack packets based on P4 programs. They also record necessary flow information, which will be sent to the controller for in-depth analysis. On detecting other attacks, the controller gives flow rules to switches to drop malicious packets.

DDoS-F attacks and non-DDoS attacks (i.e., hybrid network attacks) would occur at any time and select some hosts in the SDN-based network to be targets (called *victims*). These

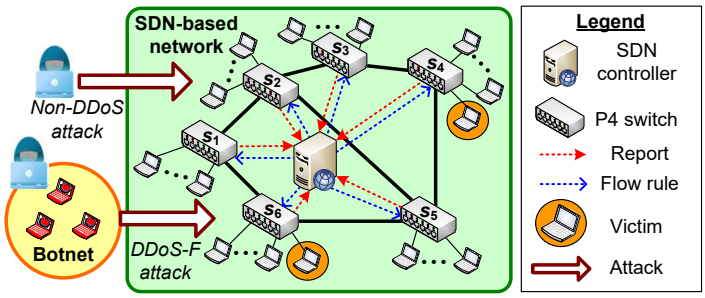


Fig. 1: Network model and attack schematic.

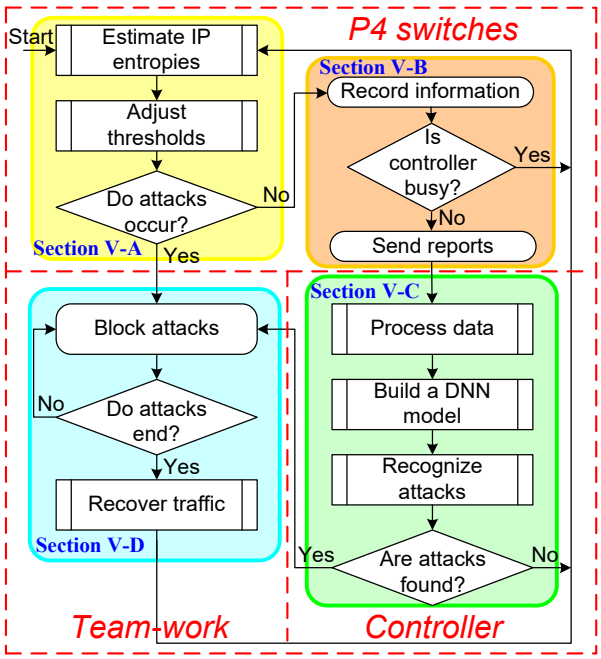


Fig. 2: Flowchart of the CD2P framework.

attacks may not necessarily be launched by the same attacker. A DDoS-F attacker could manipulate a botnet to send many packets to the victim. The botnet contains compromised hosts that may reside in different network domains. Regarding non-DDoS attacks, we consider both keylogging and data theft, as discussed in Section 2.4.

Our goal is to let the controller and P4 switches collaborate to quickly and accurately identify hybrid network attacks and block malicious packets in time. To do so, P4 switches shall take charge of detecting and stopping DDoS-F attacks to avoid the controller busily checking these packets. In this way, the controller can focus on identifying non-DDoS attacks.

5 The Proposed CD2P Framework

Fig. 2 shows CD2P's flowchart. When a P4 switch finds that the amount of traffic rises abnormally, it checks if a DDoS-F attack occurs via the EAD scheme in Section 5.1. The switch calculates IP entropies and adjusts thresholds (if necessary). Based on the relationship between IP entropies and thresholds, the switch judges whether there is a DDoS-F attack.

Even if P4 switches deduce that no DDoS-F attack occurs, since there could be non-DDoS attacks, the network may not necessarily be safe. Hence, switches record information about non-DDoS-F flows and send their reports to the controller for in-depth analysis. Besides the analysis task, the controller may have other tasks. To avoid overloading the controller, switches submit reports when the controller is not busy. The details will be given in Section 5.2.

The controller analyzes these reports to discover non-DDoS attacks. To facilitate the analysis, the controller performs data processing on the reports. Afterward, it builds a DNN model for analysis to find potential attacks. In Section 5.3, we will elaborate on the DNN-based analysis handled by the controller.

On detecting DDoS-F attacks, P4 switches stop attack flows on their own. On the other hand, when the controller figures out non-DDoS attacks, it gives switches flow rules to discard attack packets. Then, the controller or switches judge whether the attack ends (based on its type) and perform traffic recovery, as discussed later in Section 5.4.

5.1 EAD Scheme

P4 switches use the EAD scheme to detect DDoS-F attacks. Each switch calculates the IP entropies of packets that pass its ports and assesses whether a DDoS-F attack comes about based on the relationship between IP entropies and thresholds. Unlike the entropy-based methods discussed in Section 3 that adopt fixed thresholds, our EAD scheme can adaptively adjust thresholds according to the network status.

5.1.1 Entropy Estimation

As mentioned in Section 2.3, when a DDoS-F attack occurs, numerous packets are sent to a few victims (i.e., convergent destination IP addresses). These attack packets may have many or just a few source IP addresses. So, a low destination IP entropy along with a high (or low) source IP entropy will be an evident sign of DDoS-F attacks.

Let $\hat{\mathcal{A}}_{\mathtt{dst}}$ and $\hat{\mathcal{A}}_{\mathtt{src}}$ signify the sets of destination and source IP addresses of the packets processed by a switch in a period (denoted by $\hat{\mathcal{P}}$), respectively. Based on Eq. (1), we calculate the destination IP entropy by

$$H(\hat{\mathcal{A}}_{\mathtt{dst}}) = -\sum\nolimits_{a_i \in \hat{\mathcal{A}}_{\mathtt{dst}}} P_{\mathtt{dst}}(a_i) \times \log_2 P_{\mathtt{dst}}(a_i), \quad (3)$$

and the source IP entropy by

$$H(\hat{\mathcal{A}}_{\mathtt{src}}) = -\sum\nolimits_{a_i \in \hat{\mathcal{A}}_{\mathtt{src}}} P_{\mathtt{src}}(a_i) \times \log_2 P_{\mathtt{src}}(a_i), \quad (4)$$

where $P_{\mathtt{dst}}(a_i)$ and $P_{\mathtt{src}}(a_i)$ are the chances of occurrence for IP address a_i, which are defined by the number of packets in $\hat{\mathcal{P}}$ whose destination and source IP addresses are a_i, respectively, divided by the number of packets in $\hat{\mathcal{P}}$. The values of $H(\hat{\mathcal{A}}_{\mathtt{dst}})$ and $H(\hat{\mathcal{A}}_{\mathtt{src}})$ must be within $[0, \log_2 |\hat{\mathcal{P}}|]$.

Three adjustable thresholds $\delta_{\mathtt{dst}}$, $\delta^{\mathbf{H}}_{\mathtt{src}}$, and $\delta^{\mathbf{L}}_{\mathtt{src}}$ are defined, where $\delta_{\mathtt{dst}}, \delta^{\mathbf{H}}_{\mathtt{src}}, \delta^{\mathbf{L}}_{\mathtt{src}} \in (0, \log_2 |\hat{\mathcal{P}}|)$ and $\delta^{\mathbf{H}}_{\mathtt{src}} > \delta^{\mathbf{L}}_{\mathtt{src}}$. How to adjust these thresholds will be discussed in Section 5.1.2. Then, the switch infers that a DDoS-F attack occurs when either of the two conditions is met:

- $H(\hat{\mathcal{A}}_{\mathtt{dst}}) \leq \delta_{\mathtt{dst}}$ and $H(\hat{\mathcal{A}}_{\mathtt{src}}) \geq \delta^{\mathbf{H}}_{\mathtt{src}}$.
- $H(\hat{\mathcal{A}}_{\mathtt{dst}}) \leq \delta_{\mathtt{dst}}$ and $H(\hat{\mathcal{A}}_{\mathtt{src}}) \leq \delta^{\mathbf{L}}_{\mathtt{src}}$.

If so, the switch blocks DDoS-F packets, as discussed later in Section 5.4.

To find $H(\hat{\mathcal{A}}_{\mathtt{dst}})$ and $H(\hat{\mathcal{A}}_{\mathtt{src}})$, the switch gathers statistics on the number of packets in terms of their IP addresses. One intuitive method is to use a counter for each address a_i. However, this method may be memory-consuming. Hence, we borrow the notion of the *count-min sketch (CMS)* approach [35].

	0	1	2	3	4	5	6	7
h_0	0	1	0	0	0	0	0	0
h_1	0	0	0	0	0	0	1	0
h_2	0	0	0	1	0	0	0	0
h_3	0	1	0	0	0	0	0	0

(a) adding IP 140.117.172.88

	0	1	2	3	4	5	6	7
h_0	0	2	0	0	0	0	0	0
h_1	0	0	1	0	0	0	1	0
h_2	0	0	0	1	1	0	0	0
h_3	0	1	0	0	0	0	1	0

(b) adding IP 10.80.70.130

	0	1	2	3	4	5	6	7
h_0	0	3	0	0	0	0	0	0
h_1	0	0	1	0	0	0	2	0
h_2	0	0	0	2	1	0	0	0
h_3	0	2	0	0	0	0	1	0

(c) adding IP 140.117.172.88

	0	1	2	3	4	5	6	7
h_0	0	3	0	1	0	0	0	0
h_1	0	0	1	0	1	0	2	0
h_2	0	1	0	2	1	0	0	0
h_3	0	2	0	0	0	0	2	0

(d) adding IP 10.80.72.221

Fig. 3: An example of the CMS approach.

CMS uses d different hash functions $h_0(\cdot), h_1(\cdot), \cdots, h_{d-1}(\cdot)$, where the output size for every hash function is w. Let us use $\hat{\mathcal{A}}_{\texttt{dst}}$ as an illustration. Specifically, we construct a $d \times w$ array $\tilde{C}$ whose entries are initially set to zero. For each address $a_i \in \hat{\mathcal{A}}_{\texttt{dst}}$, the corresponding d entries in $\tilde{C}$ are updated by

$$\tilde{C}[k][h_k(a_i)] \leftarrow \tilde{C}[k][h_k(a_i)] + 1, \quad k = 0, 1, \cdots, d-1 \tag{5}$$

On the other hand, the occurring frequency of an address $a_j \in \hat{\mathcal{A}}_{\texttt{dst}}$ can be estimated by

$$\min_{k=0,1,\cdots,d-1} \tilde{C}[k][h_k(a_j)]. \tag{6}$$

Fig. 3 gives an example with 4×8 array. Suppose that hash functions h_0, h_1, h_2, and h_3 output 1, 6, 3, and 1 for address 140.117.172.88. Thus, we add entries $\tilde{C}[0][1]$, $\tilde{C}[1][6]$, $\tilde{C}[2][3]$, and $\tilde{C}[3][1]$ by one, as shown in Fig. 3(a). Fig. 3(b), (c), and (d) give the results by sequentially adding addresses 10.80.70.130, 140.117.172.88, and 10.80.72.221. Afterward, let us query the frequency of address 140.117.172.88. By Eq. (6), the answer is $\min\{\tilde{C}[0][1], \tilde{C}[1][6], \tilde{C}[2][3], \tilde{C}[3][1]\} = \min\{3, 2, 2, 2\} = 2$. Similarly, we can employ a $d \times w$ array to record the occurring frequencies of source IP addresses (i.e., $\hat{\mathcal{A}}_{\texttt{src}}$).

We analyze the amount of memory used in the two methods. Suppose that a counter requires b bits. Therefore, the intuitive method and the CMS approach will spend $(|\hat{\mathcal{A}}_{\texttt{dst}}| + |\hat{\mathcal{A}}_{\texttt{src}}|) \times b$ and $2dwb$ bits, respectively. Since d and w are small constants [35], but $\hat{\mathcal{A}}_{\texttt{dst}}$ and $\hat{\mathcal{A}}_{\texttt{src}}$ may cover many addresses, in effect we have $2dw \ll |\hat{\mathcal{A}}_{\texttt{dst}}| + |\hat{\mathcal{A}}_{\texttt{src}}|$. Hence, the CMS approach can reduce the memory usage of P4 switches efficiently.

Theorem 1 analyzes the time complexity of entropy estimation in our EAD scheme. Evidently, d is a small value, so the computational cost to check for DDoS-F attacks will be linearly proportional to the number of packets (i.e., $|\hat{\mathcal{P}}|$). In other words, checking DDoS-F attacks by using the EAD scheme will not put much burden on a P4 switch.

Theorem 1. Suppose that $\hat{\mathcal{P}}$ has n packets. With the CMS approach, the worst-case time complexity of entropy estimation in the EAD scheme is $O(nd)$.

Proof: In EAD, a P4 switch uses two $d \times w$ arrays to store the occurring frequencies of destination and source addresses (i.e., $\hat{\mathcal{A}}_{\texttt{dst}}$ and $\hat{\mathcal{A}}_{\texttt{src}}$) of the packets in $\hat{\mathcal{P}}$. According to the analysis in [35], the updating time for an array entry is $O(1)$. From Eq. (5), d entries need to be updated for each address. Hence, it takes time of $2nd \times O(1) = O(nd)$ to update the two arrays for all packets in $\hat{\mathcal{P}}$ for recording their occurring frequencies. Then, to retrieve the occurring frequency of every address in $\hat{\mathcal{A}}_{\texttt{dst}}$ and $\hat{\mathcal{A}}_{\texttt{src}}$ through Eq. (6), it will spend time of $2nd \times O(1) = O(nd)$ in total. Finally, using Eq. (3) and Eq. (4) to find the destination and source IP entropies requires $O(|\hat{\mathcal{A}}_{\texttt{dst}}|)$ and $O(|\hat{\mathcal{A}}_{\texttt{src}}|)$ time, respectively. Thus, the overall time complexity is $O(nd) + O(nd) + O(|\hat{\mathcal{A}}_{\texttt{dst}}|) + O(|\hat{\mathcal{A}}_{\texttt{src}}|)$. Since each packet has only one destination address and one source address, we have $\max\{|\hat{\mathcal{A}}_{\texttt{dst}}|, |\hat{\mathcal{A}}_{\texttt{src}}|\} \leq n$. Therefore, the time complexity can be simplified to $O(nd)$. □

5.1.2 Threshold Adjustment

As discussed in Section 3, most entropy-based methods adopt fixed thresholds. In practice, the thresholds should be adjusted depending on the network status, more concretely, the entropies of IP addresses of packets that the switch handled in the past. For example, the destination IP entropy of elephant flows (which carry a great deal of data) will be much lower [36]. If one uses a fixed threshold on the destination IP entropy, these legitimate elephant flows may be misjudged as DDoS-F flows, causing many false positives.

To this end, we modify the *exponentially weighted moving average (EWMA)* model, which is widely used to process serial data and forecast the value of the current time series according to the observed value [37]. For destination IP addresses (i.e., $\hat{\mathcal{A}}_{\texttt{dst}}$), the destination IP entropy at period t (i.e., the current period) can be estimated by

$$E^t_{\texttt{dst}} = \beta E^{t-1}_{\texttt{dst}} + (1-\beta)\bar{E}_{\texttt{dst}}, \tag{7}$$

where $\bar{E}_{\texttt{dst}}$ denotes the average destination IP entropy in the past (excluding the value at period $t-1$) and β is a coefficient to reveal the degree of weighting decrease ($0 \leq \beta \leq 1$). Unlike the original EWMA model, we make two modifications. First, we count the average entropy $\bar{E}_{\texttt{dst}}$ of no more than τ periods, where $\tau \in \mathbb{Z}^+$. Doing so can save the switch's computational cost. Besides, older information of traffic actually has a smaller influence on $\bar{E}_{\texttt{dst}}$. Second, if the switch inferred that a DDoS-F attack occurred at period $t-1$, we set $\beta \approx 0$. In this way, we can greatly reduce the impact of DDoS-F attacks on $\bar{E}_{\texttt{dst}}$.

Then, the threshold for destination IP entropy is defined by

$$\delta_{\texttt{dst}} = \max\{(1-\alpha_{\texttt{dst}})E^t_{\texttt{dst}}, \zeta_{\texttt{dst}}\}, \tag{8}$$

where $0 < \alpha_{\texttt{dst}} < 1$, $\zeta_{\texttt{dst}} > 0$, and $\zeta_{\texttt{dst}} \approx 0$. Based on Eq. (8), once the current destination IP entropy is below $(1-\alpha_{\texttt{dst}}) \times 100\%$ of the entropy $E^t_{\texttt{dst}}$ calculated by Eq. (7), the destination IP addresses are convergent. Here, we put a lower bound $\zeta_{\texttt{dst}}$ on threshold $\delta_{\texttt{dst}}$ to make sure that $\delta_{\texttt{dst}} > 0$.

For source IP addresses (i.e., $\hat{\mathcal{A}}_{\texttt{src}}$), the source IP entropy at period t is calculated by

$$E^t_{\texttt{src}} = \beta E^{t-1}_{\texttt{src}} + (1-\beta)\bar{E}_{\texttt{src}}, \tag{9}$$

where $\bar{E}_{\texttt{src}}$ indicates the average source IP entropy in the past (without the value at period $t-1$). The above two modifications are also applied to Eq. (9).

There are two thresholds for source IP entropy. Specifically, threshold $\delta^{\mathbf{H}}_{\texttt{src}}$ is defined by

$$\delta^{\mathbf{H}}_{\texttt{src}} = \min\{(1+\alpha^{\mathbf{H}}_{\texttt{src}})E^t_{\texttt{src}}, \zeta^{\mathbf{H}}_{\texttt{src}}\}, \tag{10}$$

where $0 < \alpha^{\mathbf{H}}_{\texttt{src}} < 1$. Besides, we have $\zeta^{\mathbf{H}}_{\texttt{src}} < \log_2|\hat{\mathcal{P}}|$ and $\zeta^{\mathbf{H}}_{\texttt{src}} \approx \log_2|\hat{\mathcal{P}}|$. Based on Eq. (10), if the current source IP

entropy is above $(1+\alpha^{\mathbf{H}}_{\mathtt{src}})\times 100\%$ of the entropy derived by EWMA (i.e., $E^t_{\mathtt{src}}$), the source IP addresses are divergent. To avoid $(1+\alpha^{\mathbf{H}}_{\mathtt{src}})E^t_{\mathtt{src}} > \log_2|\hat{\mathcal{P}}|$, we impose an upper bound $\zeta^{\mathbf{H}}_{\mathtt{src}}$ on threshold $\delta^{\mathbf{H}}_{\mathtt{src}}$. On the other hand, threshold $\delta^{\mathbf{L}}_{\mathtt{src}}$ is defined by

$$\delta^{\mathbf{L}}_{\mathtt{src}} = \max\{(1-\alpha^{\mathbf{L}}_{\mathtt{src}})E^t_{\mathtt{src}}, \zeta^{\mathbf{L}}_{\mathtt{src}}\}, \tag{11}$$

where $0 < \alpha^{\mathbf{L}}_{\mathtt{src}} < 1$. Also, we have $\zeta^{\mathbf{L}}_{\mathtt{src}} > 0$ and $\zeta^{\mathbf{L}}_{\mathtt{src}} \approx 0$. From Eq. (11), if the current source IP entropy is lower than $(1-\alpha^{\mathbf{L}}_{\mathtt{src}})\times 100\%$ of the entropy $E^t_{\mathtt{src}}$ estimated by Eq. (9), the source IP addresses are convergent. To ensure that $\delta^{\mathbf{L}}_{\mathtt{src}} > 0$, we give a lower bound $\zeta^{\mathbf{L}}_{\mathtt{src}}$ in Eq. (11).

Theorem 2. Threshold adjustment in EAD takes $O(1)$ time.

Proof: In EAD, we estimate threshold $\delta_{\mathtt{dst}}$ on the destination IP entropy by using EWMA to find $E^t_{\mathtt{dst}}$. On getting an entropy, it spends $O(1)$ time updating the average destination IP entropy $\bar{E}_{\mathtt{dst}}$. Thus, computing $E^t_{\mathtt{dst}}$ by Eq. (7) takes $O(1)$ time. Moreover, $\zeta_{\mathtt{dst}}$ is a constant, so using Eq. (8) to calculate $\delta_{\mathtt{dst}}$ takes $O(1)$ time. Since thresholds $\delta^{\mathbf{H}}_{\mathtt{src}}$ and $\delta^{\mathbf{L}}_{\mathtt{src}}$ on the source IP entropy are computed similarly, threshold adjustment takes $O(1)$ time. □

All thresholds are adjusted period by period, not packet by packet. Moreover, Theorem 2 shows that adjusting thresholds is easy. Thus, threshold adjustment in EAD has little effect on the computational burden of a P4 switch.

5.2 Information Record and Report Submission

P4 switches check if DDoS-F attacks occur and drop DDoS-F packets on their own (as discussed later in Section 5.4). Excluding DDoS-F flows, each switch stores information about flows that they handle, which will be sent to the controller for in-depth analysis.

5.2.1 Storing Flow Information

There are two formats widely used to store flow information: *packet capture (PCAP)* and *comma-separated values (CSV)*. PCAP is an API used to get detailed information about network traffic. A CSV file is a delimited text file using commas to separate values. Each line in the file is a data record, where each record has one or more fields separated by commas. In CD2P, we choose CSV for two reasons. First, a CSV file is smaller than a PCAP file. Second, if the switch sends a PCAP file to the controller, it needs to convert the file into a CSV file before conducting analysis. So, using CSV helps reduce the cost for the controller to perform data processing on the switch's report.

Based on [38], we select the following fields of flow information to be stored in each record: start time, flags, protocol, source/destination IP addresses, source/destination ports, packets, bytes, state, sequence number, source/destination packets, source/destination bytes, and rate. The size of one record is 121 bytes. In Section 6.3, we will analyze the overhead of storing flow information using CSV.

5.2.2 Sending Reports

Besides attack analysis, the controller could also have other tasks to perform. Switches should send reports when the controller is not busy to avoid overloading it. Hence, we propose two mechanisms for report submission.

In the passive mechanism, each switch regularly submits its CSV report. The switch sends a *submission-request* message to the controller. If the controller is not busy, it replies with a *submission-grant* message, so the switch can send the report. Otherwise, the controller replies with a *submission-declining* message, and the switch will try later. The passive mechanism is suitable for a small network with a few switches.

In the active mechanism, the controller periodically broadcasts a beacon that reveals its status (i.e., busy or non-busy) to all switches. If the status is non-busy, switches are allowed to submit CSV reports. To avoid many switches sending reports at the same time, which causes network congestion, switches are divided into groups. Afterward, the controller indicates in the beacon which group of switches can send their reports at each given interval. In this way, we can mitigate congestion.

5.3 DNN-based Analysis

After getting reports from P4 switches, the controller carries out data processing on these reports to facilitate analysis. Then, it builds a DNN model to check whether there are non-DDoS attacks (i.e., keylogging and data theft).

5.3.1 Data Processing

This procedure contains three steps.

Step 1: According to the well-known KDD-Cup99 dataset, we pick 24 features from the reports (in CSV format)[2]:

[Connection] wrong_fragment, duration, urgent, land, protocol_type, service, flag, src_bytes, dst_bytes.

[Time] diff_srv_rate, srv_diff_host_rate, srv_count, count, rerror_rate, srv_rerror_rate, same_srv_rate.

[Host] dst_host_same_src_port_rate, dst_host_rerror_rate, dst_host_srv_diff_host_rate, dst_host_srv_rerror_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate.

Step 2: Convert text to numbers. We use the *one-hot encoding* method [40], which represents each word in the vocabulary through a numerical positional vector whose elements are all zeros except for the position of that word in the vocabulary list. For example, we employ vectors [1, 0, 0], [0, 1, 0], and [0, 0, 1] to represent the three protocols TCP, UDP, and HTTP.

Step 3: For each value v, we adopt a z-score to normalize it: $(v-\mu)/\sigma$, where μ is the average and σ is the standard deviation. Using the z-score avoids some excessively large values affecting the analytic result, making the analysis inaccurate.

5.3.2 Model Building

In DNN, a logistic regression is known as a *neuron*, and a neural network is composed of many neurons interconnected with each other. A DNN model comprises one *input layer*, multiple *hidden layers*, and one *output layer*. The neurons in each layer are fully connected by the neurons in the previous layer (called the *fully connected feed-forward network*). Data is transferred from the input layer to the output layer. Fig. 4 shows the architecture of a DNN model.

The input layer handles features and feeds a feature vector $\vec{X} = [x_1, x_2, \cdots, x_n]$ to the first hidden layer, which calculates an output vector $\vec{Y} = [y_1, y_2, \cdots, y_m]$ by

$$\vec{Y} = f_{\mathrm{A}}(\tilde{W} \times \vec{X} + \vec{B}). \tag{12}$$

2. KDD-Cup99 is widely used to recognize intrusions in computer networks and defines 42 features in total. A detailed description of each feature can be found in [39]. We choose 24 features for detecting keylogging and data theft. It may require additional features to detect other types of attacks.

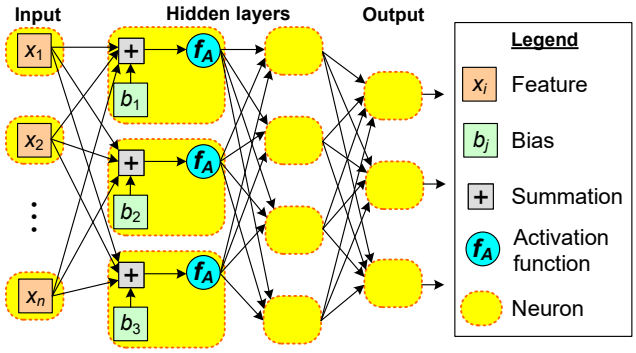


Fig. 4: Architecture of a DNN model.

TABLE 1: The DNN model used by the controller.

(a) Parameters:

layer	# of neurons	activation function
input	10	ReLU
1st hidden	50	ReLU
2nd hidden	10	ReLU
output	2	Softmax

(b) Effect of the number of hidden layers:

# of layers	train loss	validation loss
1	0.0516	0.0476
2	0.0495	0.0466
3	0.0571	0.0475

$\tilde{W} = [w_{i,j}]_{n\times m}$ is a weight matrix, and $\vec{B} = [b_1, b_2, \cdots, b_m]$ is a bias vector. Furthermore, $f_{\mathrm{A}}(\cdot)$ is an activation function, where *rectified linear unit (ReLU)* and *Softmax* are two common functions. Then, $\vec{Y}$ is employed as the input vector of the next layer, which uses Eq. (12) to find an output vector. The above procedure is repeated until we reach the output layer. A loss function is used to compare the target and predicted output values, which can be used to measure how well DNN models the training data. The objective is to obtain suitable parameters $\tilde{W}$ and $\vec{B}$ to minimize the loss between the predicted and target outputs. This can be carried out by *gradient descent*, an iterative optimization algorithm used to find a local minimum of the differentiable function (the details can be found in [41]).

For implementation, we build a four-layer DNN model, as shown in Table 1(a). The input layer uses ten neurons to handle 24 flow features. Because the controller needs to differentiate between attack packets and legitimate packets, two neurons are used in the output layer (i.e., binary classification). Moreover, we employ *binary cross-entropy* as the loss function:

$$\text{Loss} = -\frac{1}{K}\sum_{i=1}^{K} \xi_i \log \hat{\xi}_i + (1-\xi_i)\log(1-\hat{\xi}_i), \qquad (13)$$

where $\hat{\xi}_i$ is the i-th scalar value in the model output, ξ_i is the target value, and K is the number of scalar values in the model output. To avoid overfitting, an early stopping mechanism [42] is applied, where we stop training the DNN model when the loss calculated by Eq. (13) no longer decreases.

5.3.3 Attack Recognition

We adopt the botnet dataset in [38] to train the DNN model. The training result (i.e., both weight matrix $\tilde{W}$ and bias vector $\vec{B}$) is saved in an HDF5 file with hierarchical data format. The controller processes the reports from P4 switches by the three-step procedure in Section 5.3.1 and feeds the processed data to the DNN model. Because the model uses binary classification, the output will be either 0 (normal) or 1 (abnormal). If the percent of 1's outputs exceeds a predefined threshold (e.g., 10%), the controller infers that an attack of keylogging or data theft has taken place. In this way, we can avoid the controller misjudging (i.e., causing false alarms) due to just a few abnormal outputs.

5.3.4 Discussion on Model Setting

The DNN model employs binary classification of packets, so the output layer uses Softmax as the activation function. Then, the activation functions in other layers are ReLU. ReLU has four advantages over other functions (e.g., Sigmoid and Tanh) [43]: addressing the vanishing gradient, mitigating overfitting, capturing better features, and conserving computing resources. Regarding the number of hidden layers, Table 1(b) gives the train and validation losses calculated by Eq. (13) if there are one, two, and three hidden layers. Both losses will increase when adding the third hidden layer. Hence, we use two hidden layers in the DNN model.

5.4 Attack Blocking and Traffic Recovery

Depending on the type of attack, we use different strategies to block the attack and recover traffic after the attack ends.

5.4.1 DDoS-F Attack

On detecting a DDoS-F attack, the P4 switch discards attack packets on its own without asking the controller for permission. However, the switch can notify the controller of the incident to warn the network administrator of the DDoS-F attack.

Let $\hat{\mathcal{P}}_i$ be the subset of packets in $\hat{\mathcal{P}}$ whose destination IP addresses are a_i, where $\hat{\mathcal{P}}$ is the set of packets processed by the switch in the last period. If $|\hat{\mathcal{P}}_i|/|\hat{\mathcal{P}}| \geq \varrho_1$, where $0 < \varrho_1 < 1$, the host with an IP address of a_i could be a potential victim[3]. Let $\hat{\mathcal{A}}_i$ denote the set of source IP addresses of the packets in $\hat{\mathcal{P}}_i$. Then, there are two cases to be considered.

Case 1: The switch detects a DDoS-F attack due to diverse source IP addresses. We compute the source IP entropy $H(\hat{\mathcal{A}}_i)$ using Eq. (4). If $H(\hat{\mathcal{A}}_i) \geq \varrho_2 \times \delta^{\mathbf{H}}_{\mathtt{src}}$, where $0 < \varrho_2 < 1$, there is a good possibility that the attack targets the host whose IP address is a_i. Hence, the switch discards subsequent packets whose source IP addresses belong to $\hat{\mathcal{A}}_i$ and whose destination IP addresses are a_i. Since $\hat{\mathcal{A}}_i \subseteq \hat{\mathcal{A}}_{\mathtt{src}}$, we derive that $H(\hat{\mathcal{A}}_i) \leq H(\hat{\mathcal{A}}_{\mathtt{src}})$. That is why we add a coefficient ϱ_2.

Case 2: The switch detects a DDoS-F attack due to convergent source IP addresses. We check the proportion of packets sent by each source. Let $\hat{\mathcal{P}}_{i,j}$ be the subset of packets in $\hat{\mathcal{P}}_i$ whose source IP addresses are a_j. If $|\hat{\mathcal{P}}_{i,j}|/|\hat{\mathcal{P}}_i| \geq \varrho_3$ $(0 < \varrho_3 < 1)$, it implies that many packets originate from a source a_j (or their source addresses are spoofed to a_j by the attacker). The switch drops subsequent packets with source IP addresses of a_j and destination IP addresses of a_i. Notice that if the attacker changes the spoofed address to, say, a_k later, since the condition of $|\hat{\mathcal{P}}_{i,k}|/|\hat{\mathcal{P}}_i| \geq \varrho_3$ will hold, the switch can still block the attack packets whose source addresses are a_k.

Then, when no attack packets can be found (based on the above checking conditions), it means that the DDoS-F attack has terminated. As no packets will be dropped by the switch in this situation, traffic recovery is automatically carried out.

3. As mentioned in Section 5, the switch checks if a DDoS-F attack occurs when it finds that the amount of traffic rises abnormally (that is, $\hat{\mathcal{P}}$ has many packets). Thus, when the packets in $\hat{\mathcal{P}}$ that overtake the proportion of ϱ_1 are sent to a single host, the host would be a potential victim.

5.4.2 Non-DDoS Attack

Suppose that after analyzing a switch s_j's report, the controller discovers an attack (e.g., keylogging or data theft) whose source IP address is a_s, destination IP address is a_v (i.e., the victim's address), and protocol number is p_k. Then, the controller gives s_j a flow rule to drop attack packets. The matching rule is "eth_type=0x0800, ipv4_src=a_s, ipv4_dst=a_v, ip_proto=b_k", where 'eth_type=0x0800' means IPv4 packets. For IPv6, we set 'eth_type=0x86dd' and replace 'ipv4_src' and 'ipv4_dst' with 'ipv6_src' and 'ipv6_dst'. The term 'ip_proto=b_k' avoids dropping legitimate packets.

Besides, we set the 'hard_timeout' field to an amount T of time in the flow rule. Thus, after timeout T, s_j performs traffic recovery (since the flow rule is automatically removed). Doing so has two benefits. First, the controller does not need to keep checking if the attack is over[4], which saves its computational cost. Second, it incurs no extra message overhead to recover traffic, as there is no need for the controller to send messages to notify the switch of the removal of the flow rule.

5.5 Rationale and Innovation Points

Our CD2P framework is designed to help both the controller and P4 switches cooperate in defense against hybrid network attacks composed of DDoS-F and non-DDoS attacks. Switches eliminate DDoS-F packets, and the controller finds non-DDoS attacks. To do so, we propose the EAD scheme for switches to check if DDoS-F attacks occur. Considering the computing power of switches, EAD adopts Shannon entropy rather than more complex entropy calculations (e.g., φ-entropy mentioned in Section 3). Except for DDoS-F flows, switches record flow information and submit reports to the controller. Afterward, the controller uses a DNN model to analyze the reports to discover non-DDoS attacks like keylogging and data theft.

As compared with the current studies, our CD2P framework has four innovation points. First, each P4 switch uses two CMS arrays to record the frequencies of source and destination IP addresses, which saves memory usage. Theorem 1 shows that the EAD scheme has a low time complexity, which means that computing IP entropies will not put much burden on a switch.

Second, unlike most entropy-based methods that use fixed thresholds, EAD lets P4 switches adjust the thresholds on IP entropies. Since each switch has merely a local view, dynamic thresholds can help switches adapt themselves to changes in network status and make more accurate judgments of DDoS-F packets. As discussed in Section 7.1, our CD2P framework (with EAD) outperforms other entropy-based methods in terms of F1-score, which implies that it can improve the detection of DDoS-F attacks and reduce false alarms.

Third, to help P4 switches store flow information and send reports more efficiently, we employ the CSV format and carefully select fields of flow information to be recorded. Doing so reduces the storage overhead for switches to record flow information in their memory, the communication overhead for switches to send reports to the controller, and the computation overhead for the controller to perform data processing on these reports. In Section 6.3, we will analyze these overheads.

4. If the attack persists after timeout T, the controller will be able to detect it through the method in Section 5.3. In this case, we increase the value of T to extend the attack blocking time. This can be done by using the exponential backoff method proposed in our previous work [36].

Fourth, the controller conducts data processing in the reports of switches to facilitate analysis. Instead of using an individual DNN model to analyze every type of non-DDoS attack, CD2P builds one shared DNN model in the controller to recognize multiple non-DDoS attacks (i.e., keylogging and data theft). In this way, we can make our DNN model scalable, allowing for detection of more types of non-DDoS attacks to be added.

With these innovation points, the controller and P4 switches can efficiently perform the defense mission and facilitate their collaboration on resisting hybrid network attacks.

6 Discussion

Three issues are discussed in this section: 1) differentiating the packets of DDoS-F and non-DDoS attacks, 2) the scenario of multiple P4 switches, and 3) communication, computation, and storage overheads for CSV reports.

6.1 Differentiation of Attack Packets

In the CD2P framework, P4 switches filter out most DDoS-F packets. Without disturbance by DDoS-F packets, the controller can focus on identifying non-DDoS attacks. In essence, DDoS-F attacks and the non-DDoS attacks handled by CD2P (i.e., keylogging and data theft) are different. DDoS-F attacks drain a victim's bandwidth and resources by sending numerous packets. Keylogging and data theft steal data from the victim, which produces a similar number of packets as a normal flow. This essential difference helps P4 switches cleanly and easily distinguish between DDoS-F and non-DDoS attack packets by using our EAD scheme in Section 5.1.

In effect, it will not have a significant impact even if a switch misjudges some attack packets. Suppose that the switch judges some packets containing keylogging or data theft as DDoS-F packets. Since these packets are discarded by the switch, they cannot cause any damage (that is, steal data) to the victim. On the other hand, if a few DDoS-F packets are missed by the switch and sent to the controller for inspection, the controller's burden will not significantly rise due to their small number.

Someone may suggest using a more sophisticated method to make a very clean distinction between DDoS-F and non-DDoS attack packets. However, doing so is unprofitable and inefficient. Despite spending a lot of computing resources (on P4 switches or the controller) on distinguishing attack packets, it does not help much in preventing the attack from harming the victim.

6.2 Scenario of Multiple Switches

Through the EAD scheme, each P4 switch can check if there are DDoS-F flows passing its ports and discard attack packets on its own. In general, data communication could go through multiple switches, and thus some switches may not see all the evidence of a DDoS-F attack. Despite this, there is no need to coordinate between switches in the CD2P framework to detect DDoS-F attacks due to the following three reasons:

First, a DDoS-F attack produces numerous packets that have convergent destination addresses and many (or a few) source addresses. If the DDoS-F flow goes through multiple switches, each switch will obtain a subset of the flow's packets that also reveal a low destination IP entropy and a high (or low) source IP entropy. Thus, the switch is capable of detecting

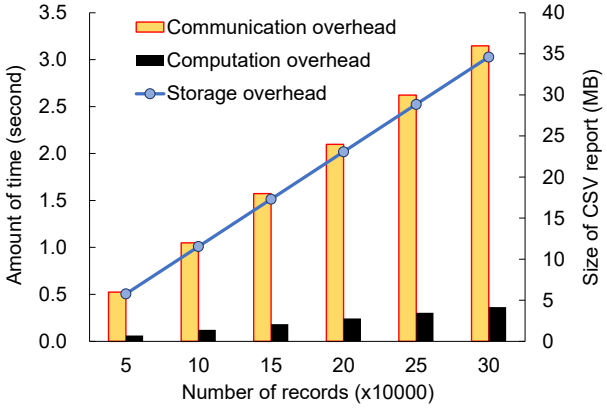


Fig. 5: Communication, computation, and storage overheads for a CSV report.

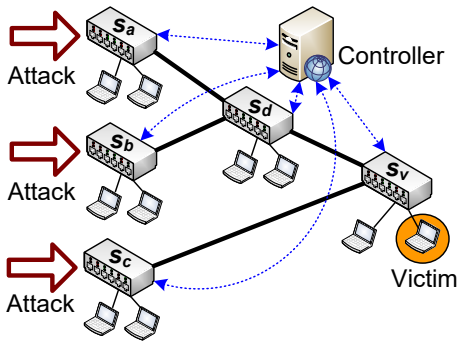


Fig. 6: Network topology used in Mininet experiments.

the DDoS-F flow even if it does not see all packets of the flow. Besides, EAD lets a switch adjust thresholds on IP entropies based on the network status. Compared to other entropy-based methods using fixed thresholds, EAD helps a switch faster discover the changes in IP entropies (caused by the DDoS-F flow), which is more adaptable to the scenario of multiple switches, where each switch may only get partial packets of a DDoS-F flow.

Second, there are two possible ways to coordinate between switches. One is to let switches exchange messages with each other. Evidently, this increases not only the message overhead but also the computational cost of switches (as switches need to consider how to cooperate with each other to check whether they detect the same DDoS-F attack). The other is to ask the controller to serve as the coordinator, which collects messages related to DDoS-F flows from switches and makes decisions. However, doing so violates the design intent of CD2P, which prevents the controller from being disturbed by DDoS-F flows to help it focus on finding non-DDoS attacks.

Third, the attack packets will be intercepted by the switches that they pass through during transmission. Let us take Fig. 1 as an example, where both s_1 and s_6 are gateway switches, and the DDoS-F attack targets a victim that connects to switch s_4. If the attack comes from exterior networks, gateway switches will first get attack packets. Hence, both s_1 and s_6 can weed out most DDoS-F packets. When the attack originates from the interior network (e.g., by some compromised hosts linking to switch s_2), s_2 can detect and discard DDoS-F packets. Even if some DDoS-F packets are not captured by s_2, the subsequent switches (i.e., s_3 and s_5) will receive these packets and discard them. As can be seen, most attack packets can be intercepted by switches. This is not too late, as DDoS will not occur at the victim. Moreover, as discussed in Section 7.1, experimental results also reveal that switches can quickly and accurately find DDoS-F flows using the EAD scheme.

6.3 Overheads for CSV Reports

As mentioned in Section 5.2, each P4 switch stores flow information in a CSV file and sends its report to the controller for analysis. Fig. 5 shows overheads for a CSV report, where the number of records in the report is increased from 50,000 to 300,000, and fast Ethernet links are used (whose bandwidth is 100 Mbps). Specifically, we consider three types of overheads. The *communication overhead* is the amount of time taken by the switch to send the report to the controller. The *computation overhead* is the amount of time that the controller consumes to conduct data processing (using the procedure in Section 5.3.1) on the report. The *storage overhead* is the report's size.

Evidently, as the number of records increases, all overheads grow merely linearly. The maximum communication and computation overheads are below 3.15 s and 0.37 s, respectively. This implies that the transmission delay (for sending a CSV report) is low, and the controller can quickly process the report. Moreover, even if the report contains 300,000 records, the file size (i.e., storage overhead) is no larger than 34.62 MB. Hence, using our method in Section 5.2 to store flow information will not consume much memory space of a P4 switch.

7 Performance Evaluation

Owing to the relatively recent development of the P4 language, there are merely a few high-priced commodity switches on the market that can support P4 (e.g., Intel Tofino switch [44]). Despite this, software solutions are capable of promoting the research progress on programmable networks. Hence, we conduct experiments by using software switches (e.g., Open vSwitch). Doing so will not significantly affect the evaluation of performance, since P4 is target-independent (as mentioned in Section 2.2) and the study [25] points out that hardware and software targets (i.e., switches) are functionally equivalent.

To implement software P4 switches, we use P4-Utils [45], a Python package for programmers to create virtual networks including P4 switches, in the Mininet environment. The capability of network creation is inherited from Mininet, and the P4 targets are taken from the behavioral model (i.e., a collection of software P4 switches). Compared with P4-programmable commodity switches that are expensive and difficult to operate, P4-Utils provides a convenient and cheap means of developing P4 data planes and control plane software written for them. In addition, Mininet is a powerful network emulation framework that can efficiently carry out virtualization of network nodes such as switches and hosts through the Linux kernel. Mininet and P4-Utils allow us to create an SDN environment in which P4 switches and hosts can be connected and tested.

Our testbed for Mininet experiments is installed on a virtual machine executing Ubuntu 16.04 with 4 processors and 32 GB of RAM. Fig. 6 illustrates the network topology used in the experiments, which is similar to that in [24]. More concretely, we consider one SDN-based network comprising a controller and five P4 switches. Each switch connects to two (legitimate) hosts. One host linking to switch s_v is chosen as the victim. Besides, we will inject attack flows into switches s_a, s_b, or s_c (depending on the attack scenario discussed in

Section 7.1). In addition to attack flows, some hosts generate normal flows whose destinations are also the victim. A normal flow makes five packets per second on average.

7.1 Performance on Blocking DDoS-F Attacks

We compare our CD2P framework with the three DDoS-F solutions mentioned in Section 3. The *principal component analysis (PrCA)* method [17] converts the traffic dataset to a smaller dataset to facilitate traffic examination for discovering DDoS-F attacks. Then, the *entropy-based DDoS-F prevention (EDP)* method [20] calculates the entropy of source IP addresses by Eq. (1) and checks if a DDoS-F attack occurs. Both PrCA and EDP are executed by the controller. The *BUNGEE* method [24] lets P4 switches analyze source and destination IP entropies for attack detection based on fixed thresholds.

Regarding the generation of DDoS-F flows, we employ the dataset in [46] and call it a *basic DDoS-F attack (BDA)* flow. A BDA flow is a mixture of TCP SYN, UDP, and HTTP flood attack flows. The BDA has a duration of 20 s and its attack target is the victim. Then, we consider the following five attack scenarios in the experiments (referring to Fig. 6):

A1. We study the effect of a DDoS-F attack on the victim's bandwidth consumption. One BDA flow is injected into switch s_c. However, we let s_c neglect attack packets, so all packets of the BDA flow will arrive at switch s_v. Thus, s_v has to block attack packets in P4-based methods (i.e., BUNGEE and CD2P) or ask the controller to identify attack packets in controller-based methods (i.e., PrCA and EDP). Doing so allows for a fair comparison of each method, since the attack is handled by the same switch (i.e., s_v).

A2. We study the effect of different attack rates. To do so, we halve and double the packet-generating rate of a BDA flow, called *1/2-rate BDA flow* and *2-rate BDA flow*, respectively. In different experiments, we inject a 1/2-rate BDA flow, a 1-rate BDA flow (i.e., the original BDA flow), and a 2-rate BDA flow into switch s_a.

A3. We study the effect of different attack durations. Without changing the packet-generating rate, we respectively inject a BDA flow whose duration is 20 s (i.e., the original duration), 30 s, and 40 s into switch s_a in each experiment.

A4. We study the effect of different numbers of attackers. The number of attackers is set to 1, 2, and 3, where the first, second, and third attackers inject one BDA flow into switches s_a, s_b, and s_c, respectively.

A5. We study the effect of mixed DDoS-F and non-DDoS attacks. One BDA flow is injected into switch s_a and a non-DDoS attack flow is injected into switch s_b. The non-DDoS attack flow has the same duration (i.e., 20 s) and target (i.e., the victim) with the BDA flow, and we use the dataset mentioned in Section 7.2 to generate its packets.

If a switch receives suspicious packets, it conducts recognition or notifies the controller[5]. The experiment time is 60 s in scenario A1 and 100 s in other scenarios. The purpose of shortening the experiment time in scenario A1 is to provide a better observation of the victim's bandwidth consumption. Each experiment has attacks that take place halfway through it. Hence, in scenario A1, the attack starts at the 20th second. In other scenarios, attacks start at the 40th second.

In scenarios A2–A5, we measure the *detection time*, *F1-score*, and *message overhead* of each method. The detection time is the amount of time required by a method to recognize DDoS-F attacks. The F1-score is computed by Eq. (17), where a higher F1-score implies a better detection rate and fewer false positives. For PrCA and EDP, the message overhead is defined by the number of Packet_In messages that switches submit to the controller for identifying DDoS-F packets. For BUNGEE and CD2P, the message overhead is defined by the number of Packet_In messages that switches use to notify the controller of an incident of DDoS-F attacks (since switches can detect DDoS-F attacks on their own).

5. The only exception is switch s_c in scenario A1.

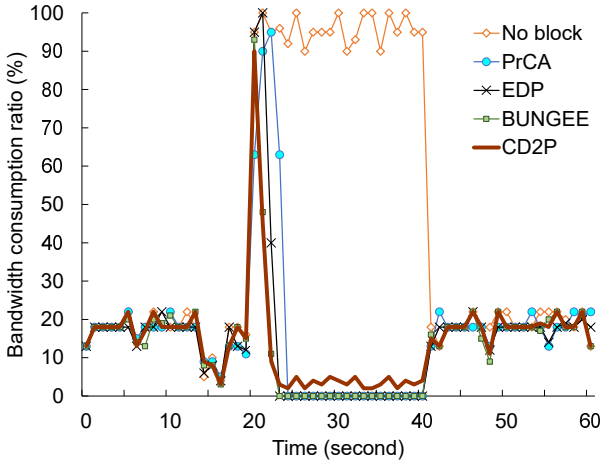
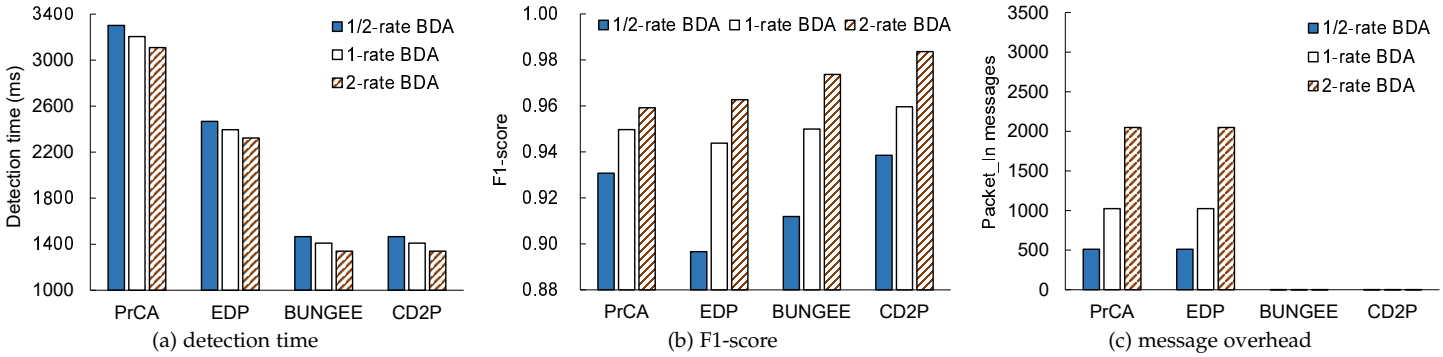


Fig. 7: Victim's bandwidth consumption in scenario A1.

7.1.1 Attack Scenario A1

In Fig. 7, we present the bandwidth consumption ratio of the victim over time (measured by switch s_v in Fig. 6). Moreover, we also show the result without any defense (i.e., *no block*) as a reference. Evidently, all methods (including no block) have little difference in results without any attack (i.e., 0th–19th seconds and 41st–60th seconds). At the 20th second, the attack is launched. Thus, the bandwidth consumption ratio rises drastically (in particular, above 90%), which means that the victim is flooded with numerous requests and runs out of bandwidth. After analyzing packets, each method can recognize the attack and discard malicious packets. Hence, the bandwidth consumption ratio drops swiftly. Despite this, there are still differences in the detection speed of each method. Specifically, the PrCA method is the slowest, as it may take more time to do the transformation of the dataset and then conduct analysis. Though EDP also estimates the IP entropy, it is slower than BUNGEE and CD2P, as EDP requires switches to send data to the controller for entropy estimation. On the contrary, BUNGEE and CD2P let P4 switches detect DDoS-F flows on their own, so they can quickly discover attacks. This result reveals the superiority of P4-based methods (i.e., BUNGEE and CD2P) over controller-based methods (i.e., PrCA and EDP) on the detection speed for DDoS-F attacks.

During the attack (i.e., 20th–40th seconds), the bandwidth consumption ratios in PrCA, EDP, and BUNGEE are down to zero. This implies that all flows to the victim (including legitimate flows) are actually blocked by these three methods. On the other hand, CD2P discards packets according to their IP entropies (as discussed in Section 5.4). In this way, our CD2P framework can allow some legitimate packets to still be sent to the victim and thus reduce false positives, as compared with the PrCA, EDP, and BUNGEE methods.



(a) detection time (b) F1-score (c) message overhead

Fig. 8: Performance comparison between different methods in scenario A2 (i.e., different attack rates).

7.1.2 Attack Scenario A2

Fig. 8(a) shows the detection time of each method with different attack rates. As the attack rate grows, the speed for switches to collect evidence (i.e., attack packets) increases. Hence, each method can recognize the attack faster, thereby reducing the detection time. EDP computes the source IP entropy of packets while PrCA conducts dataset transformation, so EDP's detection time is lower than that of PrCA. Since both BUNGEE and CD2P let switches forthright detect attacks (by using IP entropies), they have a much lower detection time than PrCA and EDP.

Fig. 8(b) presents the F1-score of each method with different attack rates. As the attack rate increases, more attack packets are generated. Since the number of normal packets (produced by legitimate hosts) does not change, the percentage of attack packets increases. Thus, the detection rate improves, and false positives (due to misjudgment of normal packets) reduce. That explains why the F1-score of each method can increase as the attack rate grows. Interestingly, entropy-based methods (i.e., EDP and BUNGEE) have lower F1-scores than PrCA (using principal component analysis) in the case of 1/2-rate BDA. That is because sometimes attack packets may not be enough to make IP entropies overtake (fixed) thresholds. This problem can be mitigated by adjusting thresholds based on the network status in CD2P (referring to Section 5.1.2). In fact, our CD2P framework keeps the highest F1-score, which verifies that it can detect DDoS-F attacks more accurately.

Fig. 8(c) gives the message overhead of each method with different attack rates. In PrCA and EDP, since switches have to send packet information to the controller to identify DDoS-F attacks, their message overheads significantly rise when the attack rate increases. For BUNGEE and CD2P, switches detect DDoS-F attacks by themselves and only notify the controller of the incident of DDoS-F attacks. In this case, two Packet_In messages are enough for notification (i.e., one indicates when the attack starts, and the other indicates when the attack ends).

7.1.3 Attack Scenario A3

In this scenario, we vary the attack duration. Figs. 9(a) and 9(b) show the detection time and F1-score. Evidently, increasing the attack duration has almost no effect on the detection time and F1-score of each method. The reason is that during the attack, legitimate hosts also send packets to the victim. Without changing the packet-generating rates of attack and normal flows, the ratio of attack packets to normal packets will not change. Thus, both the detection time and F1-score of each method remain the same, no matter how the attack duration changes. As compared with other methods, our CD2P framework can efficiently save detection time and raise the F1-score.

On the other hand, the attack duration affects the message overheads of controller-based methods, as shown in Fig. 9(c). Since more attack packets are generated when the attack lasts longer, switches need to transmit more Packet_In messages to the controller in both PrCA and EDP, which increases their message overheads significantly.

7.1.4 Attack Scenario A4

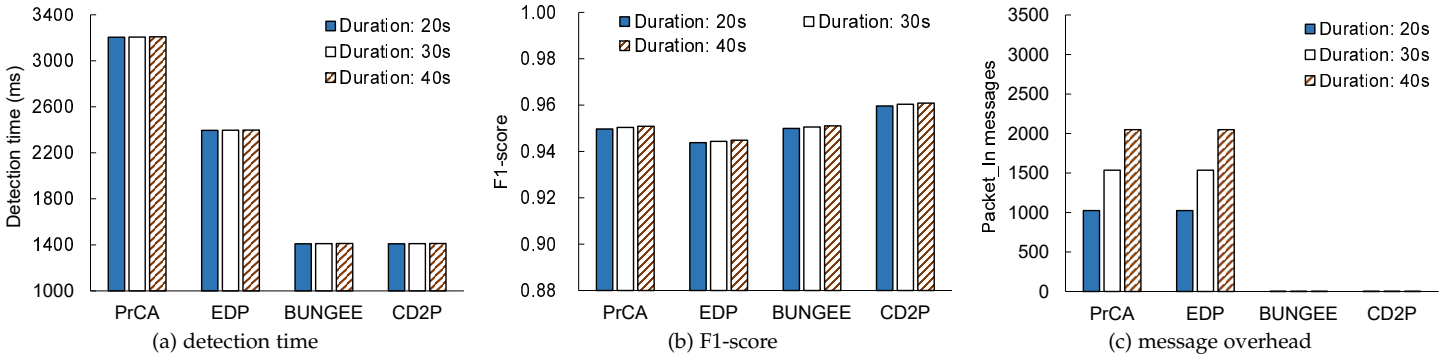
In Fig. 10(a), we show the detection time with 1–3 attackers. These attackers inject their BDA flows into different switches in Fig. 6, causing different effects on controller-based (i.e., PrCA and EDP) and P4-based methods (i.e., BUNGEE and CD2P). Since PrCA and EDP require the controller to analyze data from all switches, more attackers imply that the controller can collect more evidence of attacks, which facilitates the analysis. So, the detection time of PrCA and EDP can be reduced if there are more attackers. This phenomenon is especially evident when the number of attackers increases from 1 to 2. On the other hand, BUNGEE and CD2P let switches detect DDoS-F attacks independently. In this case, the number of attackers has no impact on their detection time. Despite this, P4-based methods are still superior to controller-based methods in terms of detection time.

The reasons above also explain how different numbers of attackers affect F1-scores, as Fig. 10(b) shows. The F1-scores of PrCA and EDP rise when there are more attackers, but those of BUNGEE and CD2P will not change. Thanks to the threshold adjustment mechanism in Section 5.1.2 and the attack blocking mechanism in Section 5.4, our CD2P framework can achieve the highest F1-score among all methods.

Fig. 10(c) presents the message overhead with 1–3 attackers. For PrCA and EDP, the number of Packet_In messages greatly increases as there are more attackers. By allowing P4 switches to detect DDoS-F attacks on their own, BUNGEE and CD2P incur almost no message overhead.

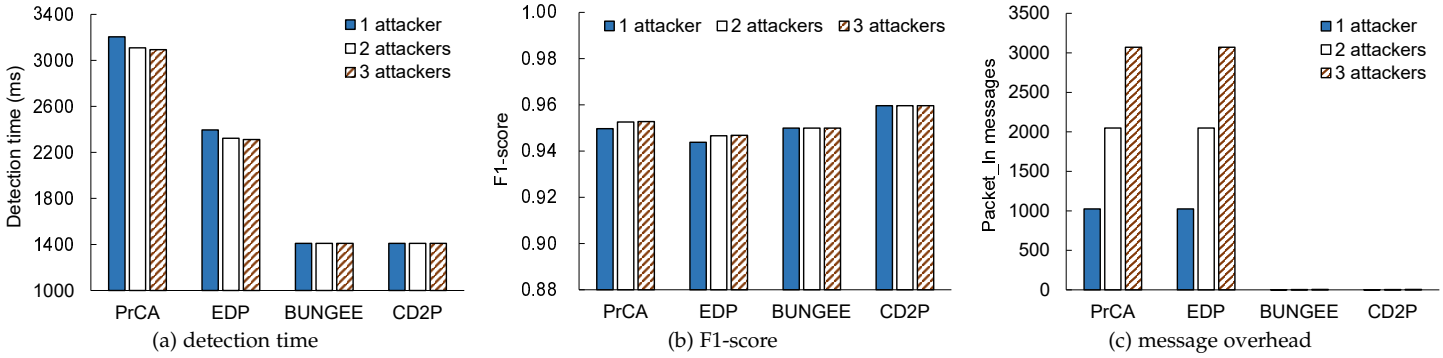
7.1.5 Attack Scenario A5

Fig. 11(a) gives the detection time with hybrid network attacks (i.e., mixing DDoS-F and non-DDoS attacks). In PrCA and EDP, the controller additionally analyzes packets of the non-DDoS attack (to check if a DDoS-F attack launches), so the detection time of PrCA and EDP slightly increases in the case of hybrid network attacks. For BUNGEE and CD2P, the DDoS-F packets will be blocked by switch s_a. On the other hand, switch s_b



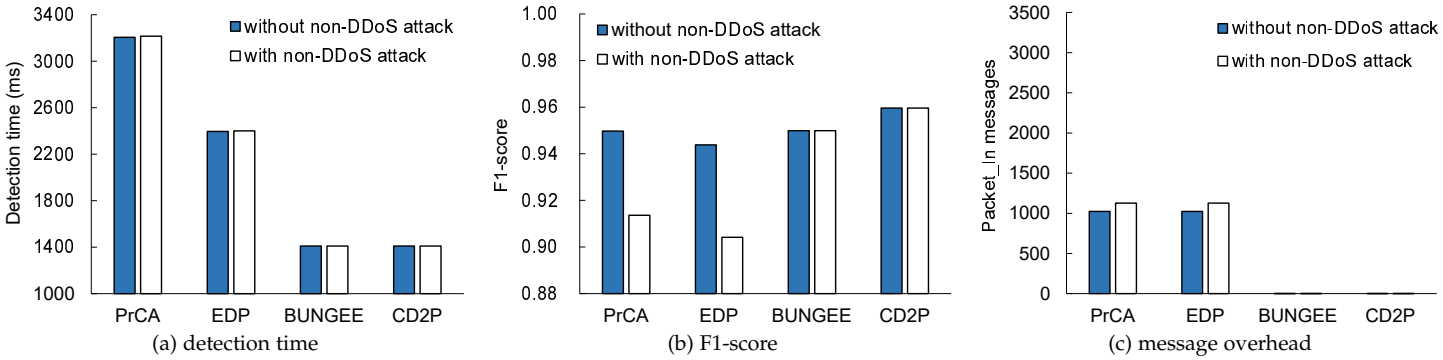
(a) detection time (b) F1-score (c) message overhead

Fig. 9: Performance comparison between different methods in scenario A3 (i.e., different attack durations).



(a) detection time (b) F1-score (c) message overhead

Fig. 10: Performance comparison between different methods in scenario A4 (i.e., different numbers of attackers).



(a) detection time (b) F1-score (c) message overhead

Fig. 11: Performance comparison between different methods in scenario A5 (i.e., hybrid network attacks).

records the packets of non-DDoS attacks and sends reports to the controller later for analysis. Therefore, the detection time of BUNGEE and CD2P will not change in this case.

Fig. 11(b) presents the F1-score with hybrid network attacks. Regarding PrCA and EDP, since the controller obtains the packets of DDoS-F and non-DDoS attacks, some packets of the non-DDoS attack could be mistaken for the packets of the DDoS-F attack, which lowers their F1-scores. Since DDoS-F and non-DDoS attacks are handled by different switches, the F1-scores of both BUNGEE and CD2P remain the same even if there are hybrid network attacks. This result also shows the benefit of using P4 switches to detect DDoS-F attacks.

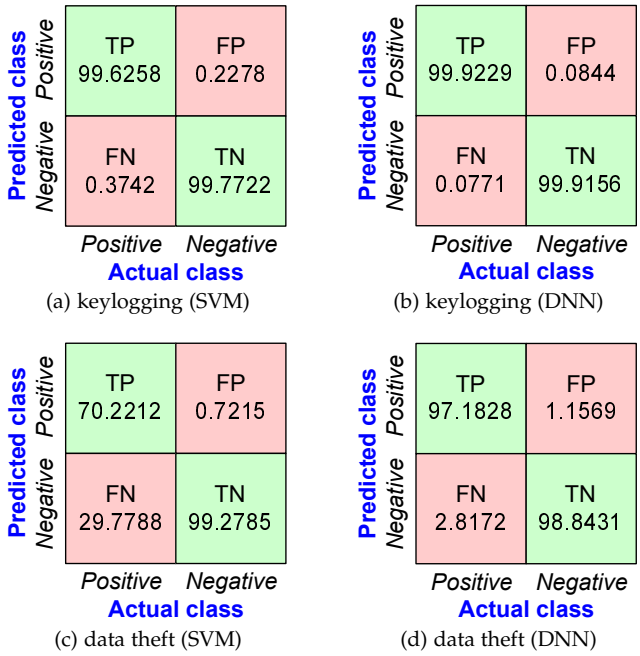
Fig. 11(c) gives the message overhead with hybrid network attacks. As the packet-generating rate of the non-DDoS attack is identical with a normal flow (i.e., around 5 packets/s), the message overheads of PrCA and EDP slightly increase when adding a non-DDoS attack. In BUNGEE and CD2P, switch s_b is actually aware that the packets do not belong to a DDoS-F attack (but s_b cannot know whether these packets belong to a non-DDoS attack). According to the definition of message overhead, the reports sent by s_b will not be counted in the message overheads of BUNGEE and CD2P[6].

7.2 Performance on Detecting Non-DDoS Attacks

As discussed in Section 3, none of the existing methods build ML models to detect keylogging or data theft (i.e., non-DDoS attacks). Hence, we compare the DNN model (in CD2P) with the *support vector machine (SVM)*, which is one of the most commonly used classification methods.

SVM is a kernel-based ML model for data classification and regression analysis. Owing to its generalization capability and

6. In fact, even if we count in s_b's reports (below 100 Packet_In messages), CD2P still has a much lower message overhead than PrCA and EDP.



(a) keylogging (SVM) (b) keylogging (DNN)

(c) data theft (SVM) (d) data theft (DNN)

Fig. 12: Confusion matrices of SVM and DNN models.

discriminative power, SVM has attracted much attention from the pattern recognition, data mining, and ML communities. SVM is a powerful tool for addressing many binary classification problems (including attack identification). Besides, SVM has been shown to be superior to other supervised ML methods [47]. In effect, many studies apply SVM to recognize various attacks, such as eavesdropping attacks [48], false data injection attacks [49], and DDoS attacks [50]. Hence, we use SVM for performance comparisons.

To train the SVM model, we set hyperparameters C and γ, where C gives penalties for misclassified data points. A small C value makes SVM choose a decision boundary with a large margin, thereby increasing misclassifications. When the kernel function is RBF (radial basis function), γ adjusts the distance of influence of a training point. A small γ value results in a large similarity radius and more points being grouped together. If γ is large, the points need to be pretty close to each other to be included in the same group, which may cause overfitting. To tune hyperparameters, we use GridSearchCV [51] with the instruction: "tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [0.1, 1, 10, 100]}, {'kernel': ['linear'], 'C': [0.1, 1, 10, 100]}]." GridSearchCV returns the score for each combination of γ (= 0.001 and 0.0001) and C (= 0.1, 1, 10, 100) using the RBF kernel and each value of C (= 0.1, 1, 10, 100) using the linear kernel. Then, we choose the one that has the highest score, which sets $C = 10$ and $\gamma = 0.001$ (with RBF as the kernel function).

For performance evaluation, we fetch 600,432 records from the botnet dataset [38], where the training set contains 540,388 records and the validation set has 60,044 records. The attack flow is injected into switch s_c in Fig. 6. Then, Fig. 12 gives the confusion matrices of SVM and DNN. In a confusion matrix, each row represents an instance (i.e., positive or negative) in the predicted class, and each column displays one instance in the actual class. The matrix is composed of four entries: 1) the *true positive (TP)* gives the percentage of packets where the model predicts that they are attack packets, and these packets actually are, 2) the *false positive (FP)* gives the percentage of packets where the model predicts that they belong to attack packets, but these packets actually are not, 3) the *false negative (FN)* gives the percentage of packets where the model predicts that they are not attack packets, yet these packets actually are, and 4) the *true negative (TN)* gives the percentage of packets where the model predicts that they are not attack packets, and the packets actually are not. TP and TN are the correct cases, while FP and FN are the wrong cases.

TABLE 2: Performance of SVM and DNN on detecting attacks.

(a) keylogging

model	accuracy	precision	recall	F1-score
SVM	0.996990	0.997719	0.996258	0.996988
DNN	0.999193	0.999156	0.999229	0.999193

(b) data theft

model	accuracy	precision	recall	F1-score
SVM	0.847499	0.989830	0.702212	0.821576
DNN	0.980130	0.988236	0.971828	0.979963

TABLE 3: Amount of time taken to detect non-DDoS attacks.

procedure	amount of time spent (in seconds)
report submission	6.299 (link bandwidth: 100 Mbps)
data processing	0.729
model training	SVM: 1080.366, DNN: 139.041
attack identification	SVM: 61.631, DNN: 3.481

Regarding keylogging, since its features are relatively apparent, both SVM and DNN can detect keylogging efficiently. More specifically, the TPs of SVM and DNN (for detecting keylogging) exceed 99.6 and 99.9, respectively. On the other hand, data theft is a more crafty attack. In this case, SVM's TP drops below 70.3, while DNN's TP still stays above 97.1. The result demonstrates that the DNN model outperforms the SVM model, especially in detecting data theft.

Table 2 lists the accuracy, precision, recall, and F1-score of SVM and DNN on detecting keylogging and data theft, which are calculated as follows:

$$\text{accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}), \quad (14)$$

$$\text{precision} = \text{TP}/(\text{TP} + \text{FP}), \quad (15)$$

$$\text{recall} = \text{TP}/(\text{TP} + \text{FN}), \quad (16)$$

$$\text{F1-score} = 2(\text{precision} \times \text{recall})/(\text{precision} + \text{recall}). \quad (17)$$

As can be seen, the DNN model (used in CD2P) is superior to the SVM model, and the performance gap between them is especially evident when detecting data theft. This result shows the high effectiveness of using the DNN model to detect non-DDoS attacks, including keylogging and data theft.

Then, we study the amount of time taken by each procedure in our CD2P framework for detecting non-DDoS attacks, as shown in Table 3. Regarding the report submission procedure, the amount of time taken by a P4 switch to send its report to the controller will depend on the link bandwidth. In particular, this procedure takes 6.299 s when the bandwidth is 100 Mbps (i.e., fast Ethernet). The above result reveals that the additional traffic (i.e., report submission) introduced by CD2P will not add much latency to normal communications. In addition, the data processing procedure (discussed in Section 5.3.1) spends just 0.729 s on computation, which shows its low complexity. Since the training set is larger than the validation set, model training takes more time than attack identification. As can be seen, DNN requires much less time than SVM. Specifically, on model training and attack identification, SVM consumes nearly 7.77 and 17.71 times as much time

as DNN, respectively. This result demonstrates the efficiency of our DNN model.

8 Conclusion and Future Work

Unlike previous studies using either an SDN controller or P4 switches solely to find DDoS-F attacks, this paper lets the controller and P4 switches collaborate to resist hybrid network attacks. In particular, we develop the CD2P framework, where P4 switches drop DDoS-F packets and the controller identifies non-DDoS attacks. To recognize DDoS-F attacks, each switch exploits the proposed EAD scheme that can adjust thresholds on IP entropies based on the network status. Except for DDoS-F flows, switches store flow information in CSV format and send the CSV reports to the controller for in-depth analysis. To reduce the burden on the controller and switches, our design in CD2P takes account of communication, computation, and storage overheads for CSV reports. Moreover, switches send reports when the controller is not busy to avoid overloading it. Then, the controller builds a DNN model to analyze CSV reports to discover keylogging and data theft. Simulation results reveal that CD2P can detect DDoS-F attacks more quickly than PrCA and EDP and more accurately than BUNGEE. Besides, the DNN model used in CD2P outperforms the SVM model in recognizing keylogging and data theft, in terms of detection and time efficiency.

The EAD scheme is developed for DDoS-F attacks. There are other types of DDoS attacks, such as amplification attacks and slow attacks. A *DNS amplification attack* is a specimen of amplification attack. It makes use of public DNS servers to flood a victim with DNS responses [52]. A *slow HTTP attack* is representative of slow attacks. The attacker divides HTTP requests into parts and transmits them to the victim slowly to use up resources [53]. Due to their different properties, these DDoS attacks may not be detected by entropy-based methods (including EAD). Hence, how to help P4 switches identify the above DDoS attacks deserves further investigation. This may require cooperation among multiple switches.

In CD2P, we build a shared DNN model for the controller to detect two non-DDoS attacks, namely keylogging and data theft. This demonstrates the feasibility and scalability of using a single DNN model to recognize multiple attacks (by properly selecting features, layers, neurons, and activation functions). In effect, attacks are open-ended, and anticipating all of them in a general solution is difficult or even infeasible. Despite this, CD2P offers an efficient framework to make P4 switches filter out DDoS-F packets and send necessary flow information to the controller for analysis. If an attack X or a class of attacks X is important or challenging, we can develop a new detecting method for X (possibly tailoring the original DNN model or adopting other ML methods) and easily add the method to the controller. In this way, we can expand the practical coverage of our CD2P framework (i.e., detecting new non-DDoS attacks) without changing other components and the operation process (in Fig. 2) of the framework.

References

[1] N. Hoque, D.K. Bhattacharyya, and J.K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Comm. Surveys & Tutorials*, vol. 17, no. 4, pp. 2242–2270, 2015.

[2] W.K. Lai, Y.C. Wang, Y.C. Chen, and Z.T. Tsai, "TSSM: Time-sharing switch migration to balance loads of distributed SDN controllers," *IEEE Trans. Network and Service Management*, vol. 19, no. 2, pp. 1585–1597, 2022.

[3] Q. Yan, F.R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Comm. Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[4] P. Dong, X. Du, H. Zhang, and T. Xu, "A detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows," *Proc. IEEE Int'l Conf. Comm.*, 2016, pp. 1–6.

[5] E.F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[6] N. Anerousis, P. Chemouil, A.A. Lazar, N. Mihai, and S.B. Weinstein, "The origin and evolution of open programmable networks and SDN," *IEEE Comm. Surveys & Tutorials*, vol. 23, no. 3, pp. 1956–1971, 2021.

[7] Y.C. Wang and T.J. Hsiao, "URBM: User-rank-based management of flows in data center networks through SDN," *Proc. IEEE Int'l Conf. Computer Comm. and the Internet*, 2022, pp. 142–149.

[8] Nexusguard, "DDoS statistical report for 1HY 2022," http://blog.nexusguard.com/threat-report/ddos-statistical-report-for-1hy-2022.

[9] M. McCormick, "Data theft: A prototypical insider threat," in *Insider Attack and Cyber Security Beyond the Hacker*. Boston: Springer, 2008, ch. 4, pp. 53–68.

[10] D. Nyang, A. Mohaisen, and J. Kang, "Keylogging-resistant visual authentication protocols," *IEEE Trans. Mobile Computing*, vol. 13, no. 11, pp. 2566–2579, 2014.

[11] R. Mohammadi, R. Javidan, and M. Conti, "SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks," *IEEE Trans. Network and Service Management*, vol. 14, no. 2, pp. 487–97, 2017.

[12] Y.H. Tung, H.C. Wei, Y.W. Ti, Y.T. Tsou, N. Saxena, and C.M. Yu, "Counteracting UDP flooding attacks in SDN," *Electronics*, vol. 9, no. 8, pp. 1–27, 2020.

[13] P. Rengaraju, V.R. Ramanan, and C.H. Lung, "Detection and prevention of DoS attacks in software-defined cloud networks," *Proc. IEEE Conf. Dependable and Secure Computing*, 2017, pp. 217–223.

[14] K. Kalkan, G. Gur, and F. Alagoz, "SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment," *Proc. IEEE Symp. Computers and Comm.*, 2017, pp. 669–675.

[15] Y.C. Wang and Y.C. Wang, "Efficient and low-cost defense against distributed denial-of-service attacks in SDN-based networks," *Int'l J. Comm. Systems*, vol. 33, no. 14, pp. 1–24, 2020.

[16] D. Wu, J. Li, S.K. Das, J. Wu, Y. Ji, and Z. Li, "A novel distributed denial-of-service attack detection scheme for software defined networking environments," *Proc. IEEE Int'l Conf. Comm.*, 2018, pp. 1–6.

[17] S. Salaria, S. Arora, N. Goyal, P. Goyal, and S. Sharma, "Implementation and analysis of an improved PCA technique for DDoS detection," *Proc. IEEE Int'l Conf. Computing Comm. and Automation*, 2020, pp. 280–285.

[18] S. Gao, Z. Peng, B. Xiao, A. Hu, Y. Song, and K. Ren, "Detection and mitigation of DoS attacks in software defined networks," *IEEE/ACM Trans. Networking*, vol. 28, no. 3, pp. 1419–1433, 2020.

[19] A. Mishra, B.B. Gupta, D. Perakovic, S. Yamaguchi, and C.H. Hsu, "Entropy based defensive mechanism against DDoS attack in SDN-cloud enabled online social networks," *Proc. IEEE Int'l Conf. Consumer Electronics*, 2021, pp. 1–6.

[20] C.S. Whittle and H. Liu, "Effectiveness of entropy-based DDoS prevention for software defined networks," *Proc. IEEE Int'l Symp. Technologies for Homeland Security*, 2021, pp. 1–7.

[21] R. Li and B. Wu, "Early detection of DDoS based on φ-entropy in SDN networks," *Proc. IEEE Information Technology, Networking, Electronic and Automation Control Conf.*, 2020, pp. 731–735.

[22] H. Lotfalizadeh and D.S. Kim, "Investigating real-time entropy features of DDoS attack based on categorized partial-flows," *Proc. Int'l Conf. Ubiquitous Information Management and Communication*, 2020, pp. 1–6.

[23] Z.Y. Shen, M.W. Su, Y.Z. Cai, and M.H. Tasi, "Mitigating SYN flooding and UDP flooding in P4-based SDN," *Proc. Asia-Pacific Network Operations and Management Symp.*, 2021, pp. 374–377.

[24] L.A.Q. Gonzalez, L. Castanheira, J.A. Marques, A. Schaeffer-Filho, and L.P. Gaspary, "BUNGEE: An adaptive pushback mechanism for DDoS detection and mitigation in P4 data planes," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management*, 2021, pp. 393–401.

[25] A.S. Ilha, A.C. Lapolli, J.A. Marques, and L.P. Gaspary, "Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation," *IEEE Trans. Network and Service Management*, vol. 18, no. 3, pp. 3121– 3139, 2021.

[26] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric DDoS victim identification using programmable

commodity switches," *IEEE Trans. Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.
[27] G. Li, M. Zhang, S. Wang, C. Liu, M. Xu, A. Chen, H. Hu, G. Gu, Q. Li, and J. Wu, "Enabling performant, flexible and cost-efficient DDoS defense with programmable switches," *IEEE/ACM Trans. Networking*, vol. 29, no. 4, pp. 1509–1526, 2021.
[28] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," *Proc. USENIX Security Symp.*, 2021, pp. 3865–3881.
[29] H. Zhou, S. Hong, Y. Liu, X. Luo, W. Li, and G. Gu, "Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches," *Proc. IEEE Symp. Security and Privacy*, 2023, pp. 3178–3192.
[30] J. Fu, Y. Liang, C. Tan, and X. Xiong, "Detecting software keyloggers with dendritic cell algorithm," *Proc. Int'l Conf. Comm. and Mobile Computing*, 2010, pp. 111–115.
[31] C. Bacara, V. Lefils, J. Iguchi-Cartigny, G. Grimaud, and J.P. Wary, "Virtual keyboard logging counter-measures using human vision properties," *Proc. IEEE Int'l Conf. High Performance Computing and Comm.*, 2015, pp. 1230–1235.
[32] S. Verma and A. Singh, "Data theft prevention & endpoint protection from unauthorized USB devices," *Proc. Int'l Conf. Advanced Computing*, 2012, pp. 1–4.
[33] P.C. Patel and U. Singh, "Detection of data theft using fuzzy inference system," *Proc. IEEE Int'l Advance Computing Conf.*, 2013, pp. 702–707.
[34] J. C. Doshi and B. Trivedi, "Hybrid intelligent access control framework to protect data privacy and theft," in *International Conference on Advances in Computing, Communications and Informatics*, 2015, pp. 1766–1770.
[35] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[36] Y.C. Wang and S.Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks," *IEEE Trans. Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.
[37] S. Sukparungsee, Y. Areepong, and R. Taboran, "Exponentially weighted moving average: Moving average charts for monitoring the process mean," *PLoS ONE*, vol. 15, no. 2, pp. 1–24, 2020.
[38] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.
[39] S. Choudhary and N. Kesswani, "Analysis of KDD-Cup'99, NSL-KDD and UNSW-NB15 datasets using deep learning in IoT," *Procedia Computer Science*, vol. 167, pp. 1561–1573, 2020.
[40] J.D. Prusa and T.M. Khoshgoftaar, "Improving neural network design with new text data representation," *J. Big Data*, vol. 4, no. 7, pp. 1–16, 2017.
[41] S. Santra, J.W. Hsieh, and C.F. Lin, "Gradient descent effects on differential neural architecture search: A survey," *IEEE Access*, vol. 9, pp. 89602–89618, 2021.
[42] P. Lukac and P. Tarabek, "Improving DNN solution using repeated training," *Proc. Int'l Conf. Information and Digital Technologies*, 2019, pp. 311–315.
[43] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Proc. Int'l Conf. Artificial Intelligence and Statistics*, 2011, pp. 315–323.
[44] A. Agrawal and C. Kim, "Intel Tofino2 – A 12.9Tbps P4-programmable Ethernet switch,"*Proc. IEEE Hot Chips 32 Symp.*, 2020, pp. 1–32.
[45] P4-Utils. [Online]. Available: https://nsg-ethz.github.io/p4-utils/index.html
[46] A. Hamza, H.H. Gharakheili, T. Benson, and V. Sivaraman, "Detecting volumetric attacks on IoT devices via SDN-based monitoring of MUD activity," *Proc. ACM Symp. SDN Research*, 2019, pp. 36–48.
[47] J. Cervantes, F. Garcia-Lamont, L. Rodriguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp. 189–215, 2020.
[48] T.M. Hoang, T.Q. Duong, H.D. Tuan, S. Lambotharan, and L. Hanzo, "Physical layer security: Detection of active eavesdropping attacks by support vector machines," *IEEE Access*, vol. 9, pp. 31595–31607, 2021.
[49] Z. Zhang, J. Hu, J. Lu, J. Cao, and F.E. Alsaadi, "Preventing false data injection attacks in LFC system via the attack-detection evolutionary game model and KF algorithm," *IEEE Trans. Network Science and Engineering*, vol. 9, no. 6, pp. 4349–4362, 2022.
[50] G.O. Anyanwu, C.I. Nwakanma, J.M. Lee, and D.S. Kim, "Optimization of RBF-SVM kernel using grid search algorithm for DDoS attack detection in SDN-based VANET," *IEEE Internet of Things J.*, vol. 10, no. 10, pp. 8477–8490, 2023.
[51] GridSearchCV. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[52] Cybersecurity and Infrastructure Security Agency, "DNS amplification attacks," https://www.cisa.gov/uscert/ncas/alerts/TA13-088A.
[53] Y.C. Wang and R.X. Ye, "Credibility-based countermeasure against slow HTTP DoS attacks by using SDN," *Proc. IEEE Annual Computing and Comm. Workshop and Conf.*, 2021, pp. 890–895.