EC-NTD: Efficient Countermeasure Against DrDoS Attacks with NAPT and Two-Stage Detection in SDN-Based Networks

You-Chiun Wang and Cheng-Yan Wu

Abstract—*Distributed reflection denial of service (DrDoS)* attacks are a prevalent and troublesome form of DDoS attack. Fake service requests trigger a flood of services responses, typically in large packet sizes, that are sent to targeted hosts via public servers. As public servers are legitimate, and their services are necessary for targeted hosts, DrDoS attacks cannot be effectively blocked through firewalls or most solutions to DDoS floods. This paper leverages the *software-defined networking (SDN)* technique and proposes an *efficient countermeasure with NAPT and two-stage detection (EC-NTD)* scheme to safeguard against DrDoS attacks, where NAPT refers to network address port translation. We consider that attack sources (i.e., botnet members that send fake requests to public servers) may be outside or inside an SDN-based network where DrDoS targeted hosts reside. To guard against external attacks, filtering rules and NAPT are used in gateways to distinguish normal responses from those caused by attacks. To recognize internal attacks, the controller detects anomalies in the quantity of requests and their source IP addresses. Additionally, the adaptive adjustment of the attack detection period length helps alleviate the controller's load while maintaining effective defense. Simulation results reveal that the EC-NTD scheme can efficiently safeguard against DrDoS attacks with different services.

Index Terms—DrDoS attack, NAPT, SDN.

1 Introduction

Due to their low costs and high destructiveness, *distributed denial of service (DDoS)* attacks continue to be serious threats on the Internet [1]. Instead of intruding into systems, such attacks take advantage of properties of network protocols like sending specific packets to exhaust resources (e.g., computing resources and bandwidth) of targeted hosts. DDoS attacks are usually performed using a botnet, which is composed of compromised computers or Internet-of-Things devices whose security has been breached. To launch an attack, the attacker instructs these devices to send numerous packets to a targeted host to ask for connections or processing. This substantially reduces the performance of the targeted host, or even renders it incapable of providing services [2].

DDoS floods are a fundamental form of DDoS attack, in which a targeted host is overwhelmed with packets sent by botnet members. For example, in a TCP-SYN flood, the targeted host is engaged in answering SYN-ACK packets due to receiving many SYN packets. In a UDP flood, the targeted host is forced to keep reassembling large UDP packets or looking over small UDP packets, thereby consuming a lot of computing resources. In an HTTP flood, the targeted host is flooded with malicious GET or POST requests, which makes it unable to respond to legitimate requests. There have been various methods proposed to resist DDoS floods, from packet statistics [3] to principal component analysis [4], IP entropy [5], and deep learning [6].

In this paper, we aim at another common variant of DDoS attacks, namely *distributed reflection denial of service (DrDoS)* attacks. Unlike DDoS floods, botnet members do not directly transmit packets to a targeted host. Instead, they send many service requests whose source IP addresses are forged as the IP address of the targeted host to public servers. Thus, public servers return corresponding service responses to the targeted host. Because of the substantial volume of service responses and their large packet sizes, the targeted host will be inevitably busy processing them[1], thereby consuming resources. DrDoS attacks are more challenging. Traditional firewalls (using IP and port filtering) and the above solutions to DDoS floods cannot be applied to counter DrDoS attacks due to three reasons. First, as discussed later in Section 2.1, various network services can be used to carry out DrDoS attacks. Some of them are even indispensable for network operations. Second, public servers are legal and necessary, so they cannot be blacklisted (e.g., by a firewall). In addition, high IP variability is a symptom of DDoS floods [7], but the IP variability of responses sent by public servers is generally low. Third, the targeted host may also send requests to public servers. Dropping responses from public servers (during a DrDoS attack) prevents the targeted host from getting the services that it requests. This can be viewed as a form of denial-of-service.

Software-defined networking (SDN) is a promising technique to bring programmability and flexibility to networks. SDN decouples control and data planes and displaces the control plane from switches to a central entity called a *controller*. Since the controller can query switches about their statuses, it becomes easy to monitor a network. Furthermore, users can write programs on the controller to implement their methods, and the controller instructs switches to process packets accordingly by installing flow rules (e.g., using OpenFlow). In this way, SDN can facilitate network management and help accelerate the development of network technology [8].

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan (e-mail: ycwang@cse.nsysu.edu.tw; aa01912001@gmail.com).

1. For example, the targeted host has to check these responses and determine whether to drop them. In addition, numerous responses will occupy bandwidth of the link between the targeted host and its switch.

The focus and scope of this paper is to leverage the SDN technique to safeguard against DrDoS attacks. In particular, we consider that DrDoS targeted hosts reside in an SDN-based network. Depending on the locations of attack sources, DrDoS attacks are divided into two categories: 1) *external botnet member (EBM)* attacks and 2) *internal botnet member (IBM)* attacks. EBM attacks are DrDoS attacks coming from external networks, while IBM attacks originate from botnet members inside the SDN-based network. Then, we propose an *efficient countermeasure with NAPT and two-stage detection (EC-NTD)* scheme, where NAPT stands for network address port translation. To resist EBM attacks, the controller sets filtering rules on gateways to block service responses by default. Then, it employs the NAPT protocol to allow legitimate responses whose corresponding requests were sent by targeted hosts to pass through gateways. To identify IBM attacks, the controller checks the number of service requests and their source IP addresses. On detecting attacks, the controller directs switches to discard malicious requests. To reduce the controller's load and maintain effective defense, the period length for attack detection is adaptively adjusted based on the frequency of attacks. Through simulations, we show that the proposed EC-NTD scheme can quickly and correctly block DrDoS attacks with different services, as compared with existing solutions also using SDN.

The rest of this paper is organized as follows: Section 2 provides background knowledge, and Section 3 discusses related work. The system model is presented in Section 4. We detail the EC-NTD scheme in Section 5 and evaluate performance in Section 6. Finally, Section 7 concludes this paper and gives future work.

2 PRELIMINARY

In this section, we briefly introduce DrDoS attacks, SDN and OpenFlow, and NAPT.

2.1 DrDoS Attacks

A DrDoS attack is a variant of a DDoS attack. The attacker sends numerous service requests to public servers, typically through a botnet. The source IP addresses of these requests are falsified as the IP addresses of targeted hosts. This way, public servers will send back many service responses to targeted hosts, thereby draining their resources.

The attacker usually exploits services with larger response packets than request ones to boost the efficacy of a DrDoS attack. Hence, small traffic can be exchanged for large attack traffic, which achieves several times the traffic amplification effect. In particular, five network services are widely used to carry out DrDoS attacks:

- *Domain name system (DNS):* DNS converts easily memorized domain names into IP addresses, which are required to locate and identify services and hosts on the Internet. With DNS, the locations of services and hosts can be changed without affecting users who continue to employ the same hostnames.
- *Network time protocol (NTP):* NTP takes charge of synchronizing the clocks of hosts over a network to a common time base. It belongs to the TCP/IP suite and typically runs in a client-server model.
- *Lightweight directory access protocol (LDAP):* LDAP allows users to remotely lookup directory data, which contains information about users, servers, or other equipment. *Connectionless LDAP (CLDAP)* is an enhancement to reduce the burden of creating a connection and performing session binding operations in connection-oriented LDAP.
- *Simple network management protocol (SNMP):* SNMP is widely used in monitoring devices for network management. It collects and organizes information about managed devices on IP networks. SNMP can change the behavior of devices by modifying the information.
- *Simple service discovery protocol (SSDP):* SSDP advertises and discovers network services supported by the universal plug-and-play protocol in a small network (e.g., a home network). SSDP is an HTTP-like protocol that exchanges messages using UDP.

TABLE 1: Bandwidth amplification factor of a DrDoS attack using different services.

service	bandwidth amplification factor
DNS	$28 \sim 54$
NTP	556.9
LDAP	$46 \sim 55$ (CLDAP: $56 \sim 70$)
SNMP	6.3
SSDP	30.8

Since public servers that offer these services have the effect of increasing small request traffic into large response traffic, they are also known as *amplifiers* or *reflectors*. The *bandwidth amplification factor* is a metric used to measure the effect of a DrDoS attack, which is defined by

$$\text{BAF} = \frac{\text{average payload size of responses}}{\text{average payload size of requests}}. \quad (1)$$

Table 1 gives the bandwidth amplification factor of a DrDoS attack using different services [9].

2.2 SDN and OpenFlow

A network can be logically divided into *control* and *data* planes, where the control plane takes charge of management and decision-making and the data plane deals with packet processing and forwarding. In the traditional network architecture, both planes are coupled up in each switch. Applying new policies or algorithms to large networks is usually tricky, as involved switches may require reconfiguration individually [10]. Hence, SDN displaces the control plane from switches to a controller to help users manipulate switches and monitor the network state more easily. Specifically, users can write programs on the controller to apply their policies or algorithms. Then, the controller sets flow rules into the switches spontaneously to perform the policies or algorithms. The controller can also query a switch about its status (e.g., the number and types of packets forwarded).

OpenFlow is a southbound protocol widely used to support SDN [11]. It provides an application program interface to let the controller communicate with switches. Specifically, each switch maintains *flow tables* that consist of flow entries. A *flow entry* has match fields for the switch to check if a packet meets certain conditions. If so, the switch forwards or discards the packet following instructions given in the flow entry. The controller builds a secure connection with each switch and adds flow entries to its flow table or removes existing ones. Hence, the switch's behavior on handling packets can be dynamically changed. Each flow entry is associated with

a priority. When a packet meets conditions of multiple flow entries, the one with the highest priority is chosen to execute its instruction.

2.3 NAPT

NAPT was first introduced in RFC 3022 [12], which is a protocol used to translate network addresses. NAPT maps the local IP addresses and port numbers of internal hosts to registered, public IP addresses (referred to as *alias IP addresses*) and corresponding ports. It enables connections of many hosts with external networks by employing just a few alias IP addresses.

NAPT is performed by a switch or router. When an internal host requests a connection with the external network, its packets have to pass through an NAPT switch/router. Doing so will translate the host's IP address and port number into an alias IP address and port number (owned by the NAPT switch/router). In practice, the NAPT switch/router is generally assigned one alias IP address. This means that multiple internal hosts have the same IP address with different port numbers to access external networks. For example, suppose that the local IP address of an internal host is 192.168.0.1 and its port number is 4323. Let an NAPT switch/router be configured with an alias IP address of 140.117.1.1. Then, when the host's packets leave the NAPT switch/router, the IP addresses and port numbers of these packets will be changed to 140.117.1.1 and 3612.

3 Related Work

In this section, we first survey DrDoS countermeasures in the traditional network architecture. Then, we discuss the defense methods against DrDoS attacks using the SDN technique.

3.1 DrDoS Countermeasures in a Traditional Network

Liu *et al*. [13] judge whether responses are far more than requests using IP entropy. If so, a DrDoS attack is presumed to occur in bilateral traffic. For a unilateral flow, they check if responses to or requests from the same IP address are beyond a threshold to identify attacks. However, they do not differentiate between normal and malicious responses. Yadav *et al*. [14] apply a filter to classify responses with large payload sizes and discard these responses based on a variable probability. Consequently, some legitimate responses would be dropped, resulting in a high number of false alarms. Fujinoki [15] assumes that there are many mirror servers in a cloud environment. If the load of incoming traffic exceeds a limit, it is split and forwarded recursively to two mirror servers until DrDoS reflectors can be identified. Nevertheless, doing so significantly raises the hardware cost.

Biagioni [16] proposes a weak authentication mechanism to deal with DrDoS attacks. When a public server gets a request, it sends a keepalive packet with a bit stream calculated by a hash function to the request's sender. If the response to this keepalive packet is sent back to the public server, the request is legitimate. Otherwise, the public server neglects the request. However, the protocols performed by public servers need to be modified to support weak authentication. Khooi *et al*. [17] consider a network architecture composed of access routers and border routers. Access routers record the number of response packets, while border routers store the number of request packets. If a border router finds that a flow has many requests, it negotiates with the corresponding access router to judge whether an attack has occurred and blocks malicious requests. However, access routers and border routers may frequently exchange messages for attack judgment, causing a high message overhead.

3.2 Defenses Against DrDoS Attacks Based on SDN

Some studies aim at DNS amplification attacks, a type of DrDoS attack. Ozdincer and Mantar [18] assume that DNS servers and targeted hosts coexist in an SDN-based network. To detect attacks, the controller monitors variations in the amplification factor (i.e., the ratio between the size of a request to a DNS server and the size of the corresponding response) and the time-to-live value (obtained from packet headers). Xing *et al*. [19] analyze the transmission speed of DNS request packets and the entropy of their source IP addresses to check if an attack is taking place. Gupta *et al*. [20] employ bloom filters as a data structure to build a one-to-one mapping of DNS requests and responses. If some DNS responses have no mapping requests, they are treated as malicious. Han *et al*. [21] propose a *DNS amplification attack defender (DAAD)* method. In DAAD, when a switch receives a new DNS query from an internal host, the controller sets a flow rule in the switch for upcoming DNS responses. Thus, some malicious DNS responses caused by EBMs can be filtered out by the switch.

Chen *et al*. [22] copy DNS and NTP responses passing through gateways to a detection agent and classify these responses using a support vector machine to detect both DNS and NTP amplification attacks. Zhauniarovich and Dodia [23] collect information about ongoing DrDoS attacks through a honeypot and then sets up flow rules to prevent malicious service requests from reaching public servers. Gupta *et al*. [24] make response packets be sent via the same path as request packets, so attack packets are diverted back to botnet members instead of victims. Lukaseder *et al*. [25] apply NAT (network address translation) to cope with DrDoS attacks. When an internal host sends a service request to a public server, the source IP address in the request will be replaced by an alias IP address. Afterward, the gateway checks if the service response uses the alias IP address. If not, the response is dropped.

Table 2 presents a comparison between prior studies based on SDN and our EC-NTD scheme. For network services used to launch DrDoS attacks, many studies consider DNS and NTP. Zhauniarovich and Dodia [23], Lukaseder *et al*. [25], and EC-NTD additionally take account of other services (e.g., CLDAP, SNMP, and SSDP). Regarding attack sources, the majority of studies focus solely on EBM attacks or solely on IBM attacks. Gupta *et al*. [24] and EC-NTD handle both EBM and IBM attacks. However, Gupta's method relies on symmetric routing, which lacks flexibility. Compared to prior studies, the proposed EC-NTD scheme provides more comprehensive defense against DrDoS attacks in terms of considering more types of services and locations of attack sources. Moreover, EC-NTD does not require extra components (e.g., detection agent, honeypot, and symmetric routing). This highlights the differences between our work and existing SDN-based solutions to DrDoS attacks.

4 System Model

This section introduces the network model, threat model, and problem description.

TABLE 2: Comparison between prior studies using SDN and our EC-NTD scheme.

	network services					attacks		
studies	DNS	NTP	CLDAP	SNMP	SSDP	EBM	IBM	extra components
Ozdincer & Mantar [18]	√	√					√	no need
Xing *et al.* [19]	√						√	no need
Gupta *et al.* [20]	√					√		no need
Han *et al.* [21]	√					√		no need
Chen *et al.* [22]	√	√				√		detection agent
Zhauniarovich & Dodia [23]	√	√		√	√	√		honeypot
Gupta *et al.* [24]	√					√	√	symmetric routing
Lukaseder *et al.* [25]	√	√	√	√	√	√		no need
EC-NTD	√	√	√	√	√	√	√	no need

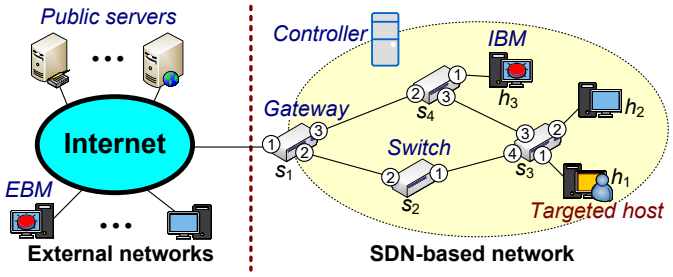
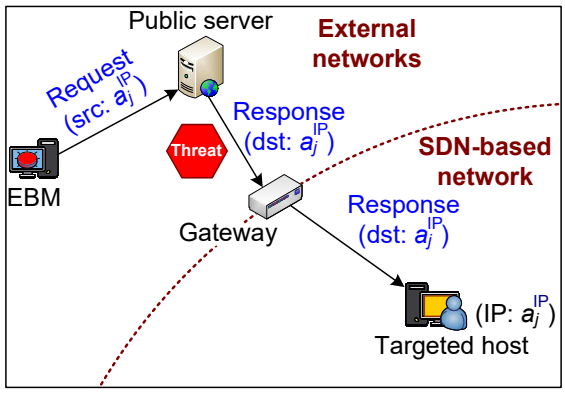
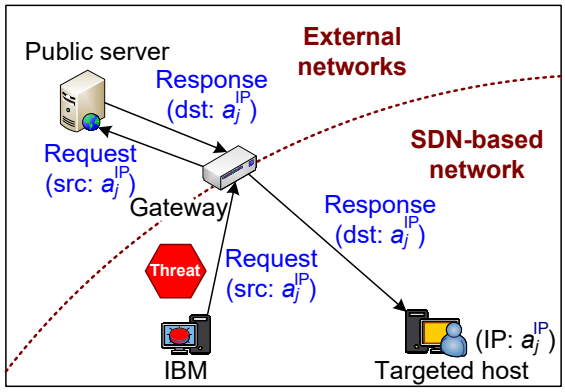


Fig. 1: Network model.



(a) DrDoS attacks launched by EBMs



(b) DrDoS attacks launched by IBMs

Fig. 2: Threat models (src: source, dst: destination).

4.1 Network Model

Fig. 1 shows the network model. We consider an SDN-based network composed of 1) a controller, 2) multiple switches that support OpenFlow, and 3) a set $\hat{\mathcal{H}}$ of hosts. The controller takes charge of managing the network. As mentioned in Section 2.2, each switch notifies the controller of its status. On the other hand, the controller can issue instructions to switches by setting corresponding flow entries. When a switch is also connected to an external network (e.g., switch s_1), it is referred to as a *gateway*.

An attacker selects some hosts in $\hat{\mathcal{H}}$ as *targeted hosts* and manipulates a botnet to attack them. Botnet members take advantage of public servers outside the SDN-based network to perform DrDoS attacks. As discussed in Section 2.1, they send many service requests to public servers, where the source IP addresses of these requests are tampered with as the IP addresses of targeted hosts. Hence, public servers return numerous service responses to the targeted hosts, thereby exhausting their resources. Depending on locations, botnet members are classified into two categories: if a botnet member is outside the SDN-based network (i.e., not in $\hat{\mathcal{H}}$), it is known as an EBM; otherwise, we call it an IBM.

4.2 Threat Model

Fig. 2(a) presents the threat model for DrDoS attacks launched by EBMs. Suppose that the IP address of a targeted host in the SDN-based network is a_j^{IP}. EBMs send many service requests to public servers by setting their source IP addresses to a_j^{IP}. Thus, public servers send numerous service responses whose destination IP addresses are a_j^{IP}. These responses pass a gateway and eventually reach the targeted host to consume its resources. Since EBMs are located in external networks, we cannot prevent them from sending requests to public servers. However, we can avoid malicious responses entering the SDN-based network (e.g., blocking them by the gateway) and being sent to the targeted host. Consequently, the main threat will be responses (caused by attacks) sent from public servers.

Fig. 2(b) illustrates the threat model for DrDoS attacks launched by IBMs. Like EBMs, IBMs send service requests whose source IP addresses are a_j^{IP} to public servers. This leads to a significant number of service responses being returned to the targeted host by public servers. In fact, we have the opportunity to stop such an attack in its infancy, as IBMs are also located in the SDN-based network. Specifically, if we prevent IBMs' malicious requests from being sent to public servers, no subsequent responses will be generated by public servers. Hence, the main threat can be viewed as requests sent from IBMs.

4.3 Problem Description

Based on the above threat models, our problem asks how to quickly and correctly stop DrDoS attacks, which has two objectives. First, we shall prevent malicious service responses (caused by EBMs and IBMs) from entering the SDN-based network and thus consuming the targeted host's resources.

TABLE 3: Summary of acronyms.

acronym	full name
DAAD	DNS amplification attack defender
DDoS/DrDoS	distributed (reflection) denial of service
DNS	domain name system
EBM/IBM	external/internal botnet member
EC-NTD	efficient countermeasure with NAPT and two-stage detection
LDAP	lightweight directory access protocol (CLDAP: connectionless LDAP)
LNPF	lightweight NAT-based packet filtering
NAT/NAPT	network address (port) translation
NTP	network time protocol
SDN	software-defined networking
SNMP	simple network management protocol
SSDP	simple service discovery protocol

TABLE 4: Summary of notations.

notation	definition
$\hat{\mathcal{H}}$	set of hosts in the SDN-based network
$a_j^{\texttt{IP}}$, $a_j^{\texttt{MAC}}$	IP and MAC addresses of a host $h_j \in \hat{\mathcal{H}}$
ξ_k	port k of a switch s_i
p_y	UDP port for a network service
$\alpha_{\texttt{L}}$, $\alpha_{\texttt{H}}$	priorities for flow rules ($\alpha_{\texttt{L}} < \alpha_{\texttt{H}}$)
$T_{\texttt{poll}}$	period length for IBM attack detection
$\delta^t_{\texttt{REQ}}$, $\delta^t_{\texttt{EPT}}$	thresholds for the number of requests and their entropy in period t

Second, if the targeted host sent service requests to public servers, we need to ensure that its service responses will not be blocked (i.e., avoiding false alarms).

Four metrics are used to measure the performance of attack detection: *accuracy*, *recall*, *precision*, and *F1-score*. According to the judgment and the true answer, there are four combinations of the detection result for each service response x_i: (1) *True positive*: x_i is judged as an attack packet, and it indeed is. (2) *True negative*: x_i is judged not to be an attack packet, and it indeed is not. (3) *False positive*: x_i is judged as an attack packet, but it actually is not. (4) *False negative*: x_i is judged as not an attack packet, but it actually is. Let $f_{\texttt{TP}}$, $f_{\texttt{TN}}$, $f_{\texttt{FP}}$, and $f_{\texttt{FN}}$ be the number of occurrences for true positives, true negatives, false positives, and false negatives, respectively. These four metrics are calculated as follows:

$$\text{Accuracy} = (f_{\texttt{TP}} + f_{\texttt{TN}})/(f_{\texttt{TP}} + f_{\texttt{TN}} + f_{\texttt{FP}} + f_{\texttt{FN}}), \quad (2)$$

$$\text{Recall} = f_{\texttt{TP}}/(f_{\texttt{TP}} + f_{\texttt{FN}}), \quad (3)$$

$$\text{Precision} = f_{\texttt{TP}}/(f_{\texttt{TP}} + f_{\texttt{FP}}), \quad (4)$$

$$\text{F1-score} = 2(\text{Recall} \times \text{Precision})/(\text{Recall} + \text{Precision}). \quad (5)$$

Tables 3 and 4 summarize the acronyms and notations used in the paper.

5 The Proposed EC-NTD Scheme

Fig. 3 gives the architecture of our EC-NTD scheme, which consists of two modules. The *EBM attack resisting module* is used to defend against DrDoS attacks launched by botnet members outside the SDN-based network, which will be discussed in Section 5.1. The *IBM attack resisting module* protects targeted hosts when some botnet members also located in the SDN-based network launch DrDoS attacks, which will be detailed in Section 5.2. Then, Section 5.3 discusses the superiority and limitations of the EC-NTD scheme.

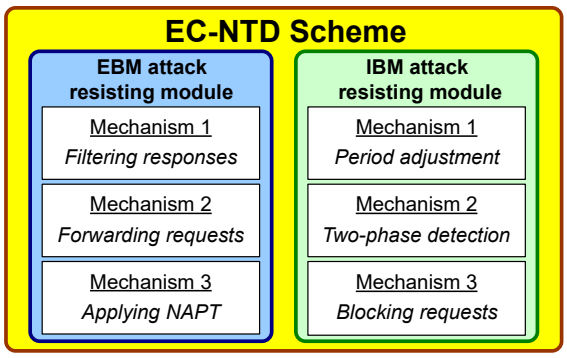


Fig. 3: Architecture of the EC-NTD scheme.

5.1 EBM Attack Resisting Module

This module is used primarily in gateways to drop malicious service responses caused by EBMs. It contains three mechanisms:

- *Filtering responses:* The controller sets filtering rules on gateways to ask them to block response packets sent from public servers to any host in $\hat{\mathcal{H}}$ by default.
- *Forwarding requests:* If some hosts in $\hat{\mathcal{H}}$ send requests to public servers, the controller finds routing paths from them to gateways and installs forwarding rules on the switches along these paths.
- *Applying NAPT:* We apply NAPT to gateways, enabling legitimate response packets to be returned to those hosts in $\hat{\mathcal{H}}$ that have ever sent requests to public servers.

Below, we elaborate on each mechanism and then remark on the EBM attack resisting module.

5.1.1 Filtering Responses

Since we cannot prevent EBMs from sending requests with fabricated IP addresses to public servers, a feasible solution to combat EBM attacks is to block malicious response packets (caused by attacks) via gateways. However, as mentioned in Section 2.1, various network services can be used to perform DrDoS attacks. If the controller is asked to execute protocol analysis and issue flow rules every time each gateway receives a response packet, it will become busy and thus significantly degrade SDN performance. Hence, instead of doing so, we let gateways block all response packets from public servers by default and then allow legitimate ones to pass so that the hosts in $\hat{\mathcal{H}}$ can utilize the services of public servers.

Based on OpenFlow, the controller issues the following filtering rule to each gateway to block the response packets from public servers:

[Match fields] in_port=ξ_e, eth_type=0x0800, ip_proto=0x11, udp_src=p_y
[Priority] $\alpha_{\texttt{L}}$ **[Action]** Drop

In the match fields, the term "in_port=ξ_e" means to handle packets from the gateway's port ξ_e, which links to an external network. Take Fig. 1 as an example. For gateway s_1, since it uses port 1 to connect with an external network, we have $\xi_e = 1$. The term "eth_type=0x0800" indicates IPv4 packets. When IPv6 is used, we can set the eth_type field to 0x86DD. The term "ip_proto=0x11" implies that we employ UDP, and the udp_src field gives the UDP source port. For example, to filter out DNS-based and NTP-based DrDoS packets, we can

TABLE 5: Three tables used to manage hosts and find packet routes in the SDN-based network.

(a) path table:

(source, destination)	path
(s_2, s_1)	$[s_2, s_1]$
(s_3, s_1)	$[s_3, s_2, s_1]$
(s_4, s_1)	$[s_4, s_1]$

(b) host table:

host	switch
$(h_1, 10.0.0.1, 00:00:00:00:00:01)$	$(s_3, 1)$
$(h_2, 10.0.0.2, 00:00:00:00:00:02)$	$(s_3, 2)$
$(h_3, 10.0.0.3, 00:00:00:00:00:03)$	$(s_4, 1)$

(c) link table:

switch pair	port pair
(s_1, s_2)	$(2, 2)$
(s_2, s_3)	$(1, 4)$
(s_3, s_4)	$(3, 3)$
(s_4, s_1)	$(2, 3)$

set udp_src to 53 and 123, respectively. Then, the priority of this flow rule is set to α_{L}, where $\alpha_{\text{L}} \in \mathbb{Z}^+$. When packets coming from an external network (via the gateway's port ξ_e) satisfy the conditions in the match fields, the gateway will drop these packets based on the instruction in the action field.

In this flow rule, we do not specify the IP address of any public server for two reasons. First, public servers employ well-known protocol ports in their packets to offer network services. Thus, the gateway can check if a packet comes from a public server by examining the packet's UDP source port (i.e., udp_src=p_y). Second, doing so can provide flexibility. When public servers are added or removed, there is no need to update filtering rules installed in gateways. In addition, the flow rule considers only UDP packets. In TCP, the source must build a connection with the destination. Suppose that an EBM sends fake requests using TCP. Since the targeted host did not build connections with a public server, due to TCP's property, it will neglect subsequent packets sent from the public server. Hence, we do not need to consider TCP packets in the flow rule.

5.1.2 Forwarding Requests

The controller adopts three tables to manage hosts and find packet routes in the SDN-based network. The *path table* gives the shortest path between any two switches. The entry format is $\langle (s_i, s_j), [s_i, s_{l_1}, \cdots, s_{l_n}, s_j] \rangle$, where s_i is the source switch, s_j is the destination switch, and the path is $s_i \rightarrow s_{l_1} \rightarrow \cdots \rightarrow s_{l_n} \rightarrow s_j$. The *host table* stores the mapping of IP and MAC addresses for each host and the host's connection relationship with a switch. Its entry format is $\langle (h_j, a_j^{\texttt{IP}}, a_j^{\texttt{MAC}}), (s_i, \xi_k) \rangle$, meaning that a host h_j whose IP address is $a_j^{\texttt{IP}}$ and MAC address is $a_j^{\texttt{MAC}}$ connects to a switch s_i via its port ξ_k. Here, the mapping information of IP and MAC addresses can be easily obtained using the *address resolution protocol (ARP)* [26]. The *link table* records how two adjacent switches connect with each other via their ports. The entry format is $\langle (s_i, s_j), (\xi_u, \xi_v) \rangle$, where switch s_i uses its port ξ_u to connect with the port ξ_v of another switch s_j. Given the network topology in Fig. 1, Table 5 shows an example of these three tables.

When a host $h_i \in \hat{\mathcal{H}}$ wants to send data to another host h_j, the controller uses the host table to find the source switch that h_i attaches to and the destination switch to which h_j links. If h_j is not in $\hat{\mathcal{H}}$ (i.e., an external host), the destination switch is a gateway. Based on the path table, the shortest path between source and destination switches can be built. Then, the controller sets forwarding rules for switches on the path by referring to the link table. However, when h_i wants to send a request to a public server, the controller needs to issue a special forwarding rule to the source switch. Let us take Fig. 1 as an example, where host h_1 wants to send a request to a public server whose IP address is $a_s^{\texttt{IP}}$. Moreover, the shortest path is $s_3 \rightarrow s_2 \rightarrow s_1$. For switch s_3 (i.e., the source switch), the forwarding rule is as follows:

[Match fields] in_port=1, eth_type=0x0800, ip_proto=0x11, eth_src=$a_1^{\texttt{MAC}}$, ipv4_src=$a_1^{\texttt{IP}}$, ipv4_dst=$a_s^{\texttt{IP}}$, udp_dst=p_y
[Action] Forward 4

In the match fields, terms "ipv4_src=$a_1^{\texttt{IP}}$" and "ipv4_dst=$a_s^{\texttt{IP}}$" mean that the source IP address is $a_1^{\texttt{IP}}$ (i.e., h_1's IP address) and the destination IP address is $a_s^{\texttt{IP}}$ (i.e., the public server's IP address). Compared to regular forwarding rules, we additionally indicate the source host's MAC address (i.e., eth_src=$a_1^{\texttt{MAC}}$) and the destination UDP port for the request (i.e., udp_dst=p_y). These two terms are used to identify IBM attacks, as discussed later in Section 5.2. Then, the instruction in the action field is to forward the packet to s_3's port 4 (to relay the packet to switch s_2). On the other hand, a regular forwarding rule is set in switch s_2 (i.e., a non-source switch) as follows:

[Match fields] in_port=1, eth_type=0x0800, ip_proto=0x11, ipv4_src=$a_1^{\texttt{IP}}$, ipv4_dst=$a_s^{\texttt{IP}}$
[Action] Forward 2

Forwarding rules are used to route packets in the SDN-based network. Their priorities are not specified because they will not clash with other flow rules used for attack defense.

5.1.3 Applying NAPT

When some hosts in $\hat{\mathcal{H}}$ send requests to public servers, the requests are relayed to a gateway. The gateway translates the source IP addresses and port numbers in the requests. Then, it allows only the response packets (returned by public servers) whose destination IP addresses are translated addresses to enter the SDN-based network. This way, even if EBMs forge IP addresses (using the IP address of a targeted host) and send many requests to public servers, causing public servers to send numerous response packets to the targeted host, these packets will be discarded by the gateway. The reason is that EBMs have no idea about the translated IP addresses and port numbers. They can only use the targeted host's original IP address to launch a DrDoS attack. Due to the filtering rule mentioned in Section 5.1.1, malicious response packets will be blocked by the gateway.

To perform the above translation, we adopt the NAPT technique discussed in Section 2.3. In particular, the controller uses a *port pool* that contains unused protocol ports to allocate an *alias port* for each flow and maintains an *NAPT-mapping table* to record the allocation. Suppose that a host $h_j \in \hat{\mathcal{H}}$ sends a request whose source IP address is $a_j^{\texttt{IP}}$ and whose source port is p_o to a public server whose IP address is $a_s^{\texttt{IP}}$. The controller picks a port p_r from the port pool and adds an entry $\langle (a_j^{\texttt{IP}}, p_o), p_r \rangle$ to the NAPT-mapping table. Then, it

issues the following flow rule to the gateway that handles h_j's request (received via its port ξ_k) to conduct translation:

[Match fields] in_port=ξ_k, eth_type=0x0800, ip_proto=0x11, ipv4_src=$a_j^{\mathtt{IP}}$, ipv4_dst=$a_s^{\mathtt{IP}}$, udp_dst=p_y
[Priority] $\alpha_{\mathtt{L}}$
[Action] Set-Field ipv4_src=$a_c^{\mathtt{IP}}$, udp_src=p_r; Forward ξ_e

The conditions in the match fields include UDP packets from the gateway's port ξ_k (i.e., in_port=ξ_k, eth_type=0x0800, ip_proto=0x11), and these packets are h_j's requests that will be forwarded to the public server (i.e., ipv4_src=$a_j^{\mathtt{IP}}$, ipv4_dst=$a_s^{\mathtt{IP}}$, udp_dst=p_y). In the action field, the instruction "Set-Field ipv4_src=$a_c^{\mathtt{IP}}$, udp_src=p_r" allows the gateway to modify two fields of the packet header in h_j's requests: 1) change the source IP address from $a_j^{\mathtt{IP}}$ (i.e., h_j's original IP address) to an alias IP address $a_c^{\mathtt{IP}}$, and 2) change the UDP source port from p_o (i.e., h_j's original port) to an alias port p_r. After modifying packet headers, the gateway can send these requests to the public server (via its port ξ_e that connects to the external network).

To ensure that h_j can receive responses from the public server, the gateway has to replace the alias IP address and port back to h_j's IP address and port in the response packets. This is known as *reverse translation*. To do so, the controller needs to issue another flow rule to the gateway:

[Match fields] in_port=ξ_e, eth_type=0x0800, ip_proto=0x11, ipv4_src=$a_s^{\mathtt{IP}}$, ipv4_dst=$a_c^{\mathtt{IP}}$, udp_src=p_y, udp_dst=p_r
[Priority] $\alpha_{\mathtt{H}}$
[Action] Set-Field ipv4_dst=$a_j^{\mathtt{IP}}$, udp_dst=p_o; Forward ξ_k

In the match fields, the term "in_port=ξ_e" asks the gateway to handle packets from the external network. The terms "ipv4_src=$a_s^{\mathtt{IP}}$", "ipv4_dst=$a_c^{\mathtt{IP}}$", "udp_src=p_y", and "udp_dst=p_r" mean that the packet is a response sent from the public server (whose IP address is $a_s^{\mathtt{IP}}$ and UDP port is p_y) to the SDN-based network (using alias IP address $a_c^{\mathtt{IP}}$ and alias port p_r). Then, the instructions in the action field let the gateway perform reverse translation (i.e., replacing $a_c^{\mathtt{IP}}$ by $a_j^{\mathtt{IP}}$ in the destination IP address and replacing p_r with p_o in the UDP destination port of the packet header) and forward the modified packet to h_j (via the gateway's port ξ_k). Here, we set the priority of this flow rule to $\alpha_{\mathtt{H}}$, where $\alpha_{\mathtt{H}} \in \mathbb{Z}^+$ and $\alpha_{\mathtt{H}} > \alpha_{\mathtt{L}}$. In other words, the flow rule is given precedence over the default filtering rule. Hence, legitimate response packets will not be dropped by the gateway.

5.1.4 Remark on the EBM Attack Resisting Module

For EBMs, we cannot stop them from sending malicious request packets to public servers using forged source IP addresses (i.e., the targeted host's IP address). However, since response packets from public servers to the targeted host must pass through a gateway, we can let the gateway weed out attack response packets. By adopting the policy of *security by default*[2], the gateway blocks all response packets to any host in $\hat{\mathcal{H}}$. When a host $h_j \in \hat{\mathcal{H}}$ ever sent requests to a public server, legitimate response packets sent from that public server to h_j will be allowed to enter the SDN-based network and be forwarded to h_j. To do so, we implement the NAPT functions in the gateway using an NAPT-mapping table and setting appropriate flow rules. Using NAPT has two benefits. First, compared to other similar techniques like NAT, NAPT can effectively reduce the usage of alias IP addresses required for address translation [28]. Second, the targeted host and EBMs will not be aware of IP translation by NAPT. Hence, the targeted host can request services from public servers using normal procedures without making any changes. On the other hand, EBMs cannot attack the targeted host, as they do not know how to translate the alias IP address to the targeted host's IP address.

Like existing DrDoS countermeasures discussed in Section 3, our objective is to avoid malicious service responses (caused by attacks) entering the SDN-based network and consuming the targeted host's resources. Therefore, we let the gateway drop malicious responses. However, these malicious responses might occupy bandwidth of the gateway's external link. This problem can be mitigated by using some techniques such as setting delimitations for link capacity or traffic cleaning. However, these techniques have to be applied to the ISP router at the other endpoint of the gateway's external link. The ISP router is not a part of the SDN-based network and thus cannot be instructed by the controller. Since the above issue is beyond the scope of this paper, we leave the details in [29].

5.2 IBM Attack Resisting Module

Since the sources of IBM attacks are some hosts in $\hat{\mathcal{H}}$, the IBM attack resisting module aims to prevent their malicious requests from being sent to public servers. Specifically, this module is composed of three mechanisms:

- *Period adjustment:* To strike a good balance between the cost and performance on identifying attacks, the controller adaptively adjusts the period length for attack detection.
- *Two-phase detection:* In each period, the controller judges whether IBM attacks occur by checking if there are anomalies in the number of requests and their source IP addresses.
- *Blocking requests:* On detecting attacks, the controller identifies malicious requests sent from IBMs and asks switches to drop them to block attacks.

Next, we detail these three mechanisms and discuss the IBM attack resisting module.

5.2.1 Period Adjustment

The controller detects the occurrence of IBM attacks, period by period. The length $T_{\mathtt{poll}}$ of a period determines not only the burden on the controller but also the performance of resisting attacks. In particular, if $T_{\mathtt{poll}}$ is set too short, the controller needs to frequently query switches about packet information, which imposes a heavy load and wastes network bandwidth. On the contrary, if $T_{\mathtt{poll}}$ is set too long, many malicious requests may not be blocked in time, causing significant damage to targeted hosts. Hence, borrowing the notion of binary exponential backoff [30], we adaptively adjust the period length as follows:

$$T_{\mathtt{poll}} = T_{\mathtt{base}} \times 2^{\lfloor \min\{\tau_{\mathtt{u}}, \tau_{\mathtt{max}}\}/\tau \rfloor}, \tag{6}$$

where $T_{\mathtt{base}}$ is the basic period length (in seconds), $\tau_{\mathtt{u}}$ is the number of consecutive periods in which no IBM attack was

2. Security by default means that the default configuration settings are the most secure. In an operating system, this typically implies that no network ports are open after installation. Only when a service is required will the corresponding port be allowed to open [27].

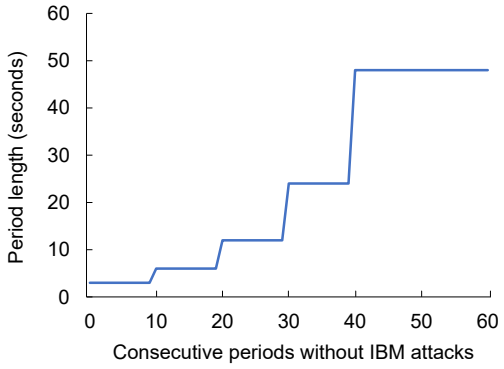


Fig. 4: Growth curve of period length $T_{\texttt{poll}}$.

detected, and τ is a constant used to control the exponential growth. Here, we have $\tau \in \mathbb{Z}^+$ and $\tau \geq 2$. To avoid excessive growth of the exponent, making $T_{\texttt{poll}}$ too long, we use an upper bound $\tau_{\texttt{max}}$ in Eq. (6), where $\tau_{\texttt{max}} \in \mathbb{Z}^+$ and $\tau_{\texttt{max}} > \tau$.

The adaptive adjustment of the period length for attack detection can help alleviate the controller's load while maintaining effective defense against IBM attacks. On one hand, if no IBM attacks occur since the last attack, due to the design of Eq. (6), the period length will show a stepwise exponential growth (until reaching the maximum value $T_{\texttt{base}} \times 2^{\lfloor \tau_{\texttt{max}}/\tau \rfloor}$). Fig. 4 shows the growth curve of period length $T_{\texttt{poll}}$ as the number of consecutive periods without IBM attacks (i.e., $\tau_{\texttt{u}}$) increases, where $T_{\texttt{base}} = 3$s, $\tau = 10$, and $\tau_{\texttt{max}} = 40$. The maximum period length is $3 \times 2^{\lfloor 40/10 \rfloor} = 48$s. In this case, the time interval for the controller to collect packet information from switches for attack detection will increase gradually and significantly. Hence, as long as no IBM attacks occur for a long time, the burden on the controller can be substantially reduced. On the other hand, once the controller has detected IBM attacks, $\tau_{\texttt{u}}$ will become zero, making $T_{\texttt{poll}}$ reset to $T_{\texttt{base}}$ (i.e., the basic period length). This way, the controller can perform the next attack detection in a short period of time to effectively combat the attack.

5.2.2 Two-Phase Detection

In each period, the controller queries each switch s_i about service requests sent from its attaching hosts and store the information in a *request statistics table*. The format of each entry is $\langle (a_j^{\texttt{IP}}, a_j^{\texttt{MAC}}, s_i, \xi_k, p_x), N_j, L_j \rangle$, which means that a host whose IP address is $a_j^{\texttt{IP}}$ and whose MAC address is $a_j^{\texttt{MAC}}$ has sent N_j requests via s_i's port ξ_k, and p_x is their UDP ports[3]. In addition, L_j is a label used to distinguish between legitimate and malicious requests. Specifically, L_j is set to zero (i.e., legitimate) for each entry in the beginning, and it may be changed to one (i.e., malicious) by the mechanism in Section 5.2.3. Table 6 presents an example. Here, some hosts (i.e., IBMs) may forge source IP addresses in their requests. Thus, there could be multiple entries that record hosts with the same IP address but different MAC addresses (e.g., entries e_1 and e_4).

Then, the controller judges whether IBM attacks occur using the two-phase detection mechanism. In phase 1, the controller checks if the number of service requests is unusually large, which can be considered a symptom of attacks. If so, the controller performs judgment in phase 2, which estimates the entropy of the source IP addresses of request packets. If the entropy is small, meaning that many request packets come from just a few sources, the controller infers that an IBM attack is taking place. In this case, the mechanism in Section 5.2.3 is used to block malicious requests.

3. Recall that in Section 5.1.2, s_i has a forwarding rule to send the host's request, which indicates the host's IP and MAC addresses (i.e., $a_j^{\texttt{IP}}$ and $a_j^{\texttt{MAC}}$), s_i's port ξ_k, and UDP port p_x. To obtain N_j, s_i can count how many requests are sent using this forwarding rule.

TABLE 6: Example of the request statistics table.

entry	(IP, MAC, switch, port, UDP port)	packet count	label
e_1	(10.0.0.1, 00:00:00:00:00:01, 3, 1, 53)	25	0
e_2	(10.0.0.2, 00:00:00:00:00:02, 3, 2, 123)	271	0
e_3	(10.0.0.3, 00:00:00:00:00:03, 4, 1, 123)	42	0
e_4	(10.0.0.1, 00:00:00:00:00:03, 4, 1, 53)	323	0

Phase 1: The criterion to identify anomalies in the number of service requests is whether the average number of requests sent per second exceeds a threshold $\delta_{\texttt{REQ}}^t$. Since there is variability in the network, this threshold should be dynamically adjusted based on the network status. Hence, we adopt the *exponentially weighted moving average* model, which is widely used to handle serial data and forecast the value of the current time series according to the observed value [31]. Given the number of request packets sent per second in the previous period (denoted by Γ_{t-1}), threshold $\delta_{\texttt{REQ}}^t$ is calculated as follows:

$$\delta_{\texttt{REQ}}^t = (1 - \beta_{\texttt{REQ}}) \times \delta_{\texttt{REQ}}^{t-1} + \beta_{\texttt{REQ}} \times \Gamma_{t-1}, \tag{7}$$

where $\beta_{\texttt{REQ}} \in [0, 1]$ is a coefficient revealing the degree of weighting. Using a dynamic threshold via Eq. (7), the controller can recognize anomalies in the number of requests sent by hosts once IBMs launch an attack.

Phase 2: In a DrDoS attack, botnet members send many service requests to public servers using a targeted host's IP address as source IP addresses. Hence, concentrated source addresses in request packets will be a distinctive feature. To assess the degree of concentration, the controller computes the Shannon entropy of source IP addresses in request packets as follows:

$$E(\hat{\mathcal{A}}) = -\sum\nolimits_{a_j^{\texttt{IP}} \in \hat{\mathcal{A}}} P(a_j^{\texttt{IP}}) \times \log_2 P(a_j^{\texttt{IP}}), \tag{8}$$

where $\hat{\mathcal{A}}$ is the set of source IP addresses of all request packets sent in a period. Moreover, $P(a_j^{\texttt{IP}})$ is the probability that an IP address $a_j^{\texttt{IP}}$ appears, which is defined by the number of request packets whose source IP addresses are $a_j^{\texttt{IP}}$ divided by the number of all request packets. The value of $E(\hat{\mathcal{A}})$ is between 0 and $\log_2 |\hat{\mathcal{A}}|$, and a smaller $E(\hat{\mathcal{A}})$ value (i.e., lower entropy) implies that source IP addresses are more concentrated. In phase 2, the criterion to identify anomalies in source IP addresses is whether entropy $E(\hat{\mathcal{A}})$ is below a dynamic threshold $\delta_{\texttt{ETP}}^t$ (at period t). Similarly, let $E_{t-1}(\hat{\mathcal{A}})$ be the value of $E(\hat{\mathcal{A}})$ estimated by Eq. (8) at period $t-1$. Threshold $\delta_{\texttt{ETP}}^t$ is calculated as follows:

$$\delta_{\texttt{ETP}}^t = (1 - \beta_{\texttt{ETP}}) \times \delta_{\texttt{ETP}}^{t-1} + \beta_{\texttt{ETP}} \times E_{t-1}(\hat{\mathcal{A}}), \tag{9}$$

where the value of coefficient $\beta_{\texttt{ETP}}$ is between zero and one.

5.2.3 Blocking Requests

After discovering IBM attacks using the two-phase detection mechanism, the controller identifies malicious requests and issues flow rules to switches to block them. To do so, we use the K-means method to divide entries in the request statistics table into two groups based on their packet counts,

Algorithm 1: Grouping entries in the request statistics table using K-means

Data: entries $\{e_1, e_2, \cdots, e_n\}$ in the request statistics table
Result: two groups (normal and abnormal)
Randomly choose two entries as the initial members of two groups;
repeat
  Assign each entry e_j to a group whose average packet count is the closest to that of e_j;
  Update the average packet count for each group;
until *the members in each group no longer change, or the number of executed iterations exceeds a predefined upper bound*;

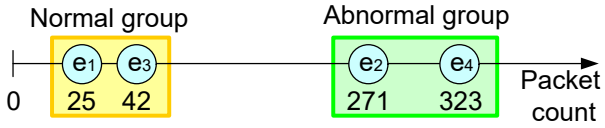


Fig. 5: Example of grouping entries in the request statistics table.

where Algorithm 1 gives the pseudocode. The *normal group* contains entries that have relatively fewer packet counts, while the *abnormal group* includes entries whose packet counts are unusually large (i.e., a sign of DrDoS attacks). However, those requests indicated in entries of the abnormal group may not necessarily be malicious. Some legitimate hosts might send many requests to public servers for themselves. Hence, we further check if the relationship between IP and MAC addresses of such an entry (i.e., $a_j^{\texttt{IP}}$ and $a_j^{\texttt{MAC}}$) is consistent with that recorded in the host table. If not, the requests indicated in the entry are malicious, so the controller changes its label L_j to one.

Fig. 5 presents an example. The normal group has entries e_1 and e_3, and the abnormal group contains entries e_2 and e_4. Hence, we further check the relationship between IP and MAC addresses in entries e_2 and e_4. Suppose that there are entries $\langle h_1, (10.0.0.1, 00{:}00{:}00{:}00{:}00{:}01)\rangle$ and $\langle h_2, (10.0.0.2, 00{:}00{:}00{:}00{:}00{:}02)\rangle$ in the host table. Since the relationship indicated in entry e_4 is inconsistent with that recorded in the host table, the controller thus sets $L_4 = 1$ (i.e., e_4's label).

For each entry $e_j = \langle (a_j^{\texttt{IP}}, a_j^{\texttt{MAC}}, s_i, \xi_k, p_x), N_j, L_j\rangle$ such that $L_j = 1$, the controller issues the flow rule to switch s_i:

[Match fields] in_port=ξ_k, eth_type=0x0800, ip_proto=0x11, ipv4_src=$a_j^{\texttt{IP}}$, eth_src =$a_j^{\texttt{MAC}}$, udp_dst=p_x
[Priority] $\alpha_{\texttt{L}}$ **[Action]** Drop

In this way, s_i will drop subsequent malicious requests (using UDP port p_x) sent by the host that claims its IP address to be $a_j^{\texttt{IP}}$ and MAC address to be $a_j^{\texttt{MAC}}$.

5.2.4 Remark on the IBM Attack Resisting Module

For IBMs, their malicious requests will also be converted by NAPT. Hence, attack response packets returned from public servers can pass through a gateway and be sent to targeted hosts. Consequently, IBM attacks are trickier than EBM attacks, and their malicious responses cannot be blocked by gateways. In this case, the controller shall differentiate between malicious and legitimate requests sent by hosts in $\hat{\mathcal{H}}$. To avoid burdening the controller with a heavy load, instead of making the controller examine requests in each fixed-length period, we adaptively adjust the period length (i.e., $T_{\texttt{poll}}$) using Eq. (6). In the case of no attacks, the controller can gradually extend the period of time for collecting packet information and examining requests to reduce its burden. On the other hand, when detecting an attack, the controller is capable of performing the next attack detection in a short period of time for a fast reaction to attacks. To discover IBM attacks, we design a two-phase detection mechanism by checking for anomalies in the number of requests and their IP entropy. When many requests are sent and most of them have the same source IP address, we infer that an IBM attack has taken place. Thus, the controller identifies malicious requests and instructs switches to drop them, thereby stopping attacks.

To identify malicious requests, the controller checks if the IP-MAC relationship in each entry of the abnormal group in the request statistics table is consistent with that in the host table. If not, the entry indicates malicious requests. Here, we employ MAC addresses for three reasons. First, the IBM attack resisting module determines whether requests sent from internal hosts are malicious. In an SDN-based network, obtaining the MAC address of each internal host is easy, as the host must attach to a switch (with a functioning layer 2). Second, the IP-MAC mapping information recorded in the host table is acquired via ARP. In this case, it is difficult for IBMs to spoof switches with fake MAC addresses (or otherwise, these IBMs cannot receive their packets). Third, ARP is a basic protocol that every host inherently performs. Getting the IP-MAC relationship for internal hosts using ARP thus incurs no extra overhead.

5.3 Superiority and Limitations of EC-NTD

Let us discuss the superiority and limitations of our EC-NTD scheme. From a theoretical perspective, the superiority of EC-NTD lies in its ability to effectively resist DrDoS attacks across different network services. Specifically, to launch a DrDoS attack, a network service that has a dedicated well-known port must be used. EC-NTD employs UDP ports instead of IP addresses to recognize service responses replied from public servers. Therefore, when attackers use new network services to launch DrDoS attacks, EC-NTD is capable of handing these attacks by simply specifying their ports in flow rules. From a practical perspective, EC-NTD's superiority is that it takes account of diverse attack sources (i.e., EBMs and IBMs) and adopts different defense strategies. In this way, EC-NTD can provide more comprehensive protection against DrDoS attacks.

Regarding limitations, like all SDN-based solutions, our EC-NTD scheme relies on OpenFlow-compatible switches for operation. If replacing all switches in a network with OpenFlow-compatible switches is difficult (for example, due to budget considerations), we suggest that gateways should be replaced by OpenFlow-compatible switches first. This way, both mechanisms of filtering responses and applying NAPT in Section 5.1 can be implemented (on gateways) to safeguard against EBM attacks. However, as the IBM attack resisting module in Section 5.2 cannot be fully implemented, EC-NTD's ability to deal with IBM attacks will be weakened.

TABLE 7: Tools used to build the simulation.

item	tool (software program)	version
operating system	Ubuntu	16.04
network simulator	Mininet	2.3.0
SDN controller	Ryu	3.27
switch	Open vSwitch	2.8.1
southbound protocol	OpenFlow	1.3
attack generation	Tcprewrite and Tcpreplay	3.4.4

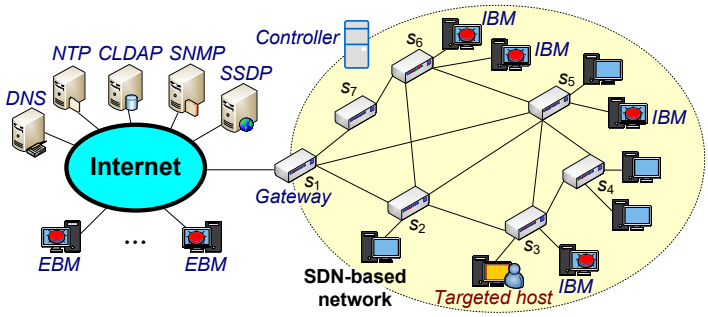


Fig. 6: Network topology used in the simulation.

6 Performance Evaluation

We develop a simulation to evaluate system performance. Section 6.1 introduces tools used to build the simulation and also the network topology. Section 6.2 explains how to generate DrDoS attacks and gives methods for comparison with our EC-NTD scheme. In Section 6.3, we measure the number of response packets that a targeted host receives when different DrDoS attacks occur. Section 6.4 compares the attack detection performance of each method. Then, Section 6.5 gives a comparison between different grouping methods used to divide entries in the request statistics table (as discussed in Section 5.2.3).

6.1 Simulation Setups

Table 7 lists tools (i.e., software programs) used to develop the simulation, which is installed on a virtual machine executing Ubuntu with four processors and 32 GB of RAM. To construct the network environment, we employ Mininet, a powerful network emulation framework, to perform virtualization of network nodes (e.g., switches and hosts) via a Linux kernel [32]. To carry out the OpenFlow protocol in Mininet, the controller and switches are implemented by the Ryu SDN framework [33] and the Open vSwitch module [34], respectively. Moreover, we adopt Tcprewrite and Tcpreplay tools [35] to help generate DrDoS attacks.

Fig. 6 gives the network topology. We consider an SDN-based network with seven OpenFlow-compatible switches, which is managed by a controller. Switch s_1 acts as a gateway. We pick one host linked to switch s_3 as the targeted host. Five public servers outside the SDN-based network offer DNS, NTP, CLDAP, SNMP, and SSDP services. Some botnet members will launch DrDoS attacks on the targeted host. They are located outside and inside the SDN-based network (i.e., EBMs and IBMs, respectively), as shown in Fig. 6.

6.2 Attack Settings

To generate DrDoS attacks, we employ the CIC-DDoS2019 dataset, a DDoS evaluation dataset released by the Canadian Institute for Cybersecurity [36]. This dataset includes PCAP (packet capture) files that describe multiple DrDoS attacks using different services. To inject attack packets in PCAP files into our simulated network in Fig. 6, we use the Tcprewrite tool to modify their source and destination IP addresses (e.g., the IP addresses of the targeted host and public servers) and recalculate checksums in packet headers. Then, we adopt the Tcpreplay tool to control when to inject attack packets.

TABLE 8: Settings for different DrDoS attacks.

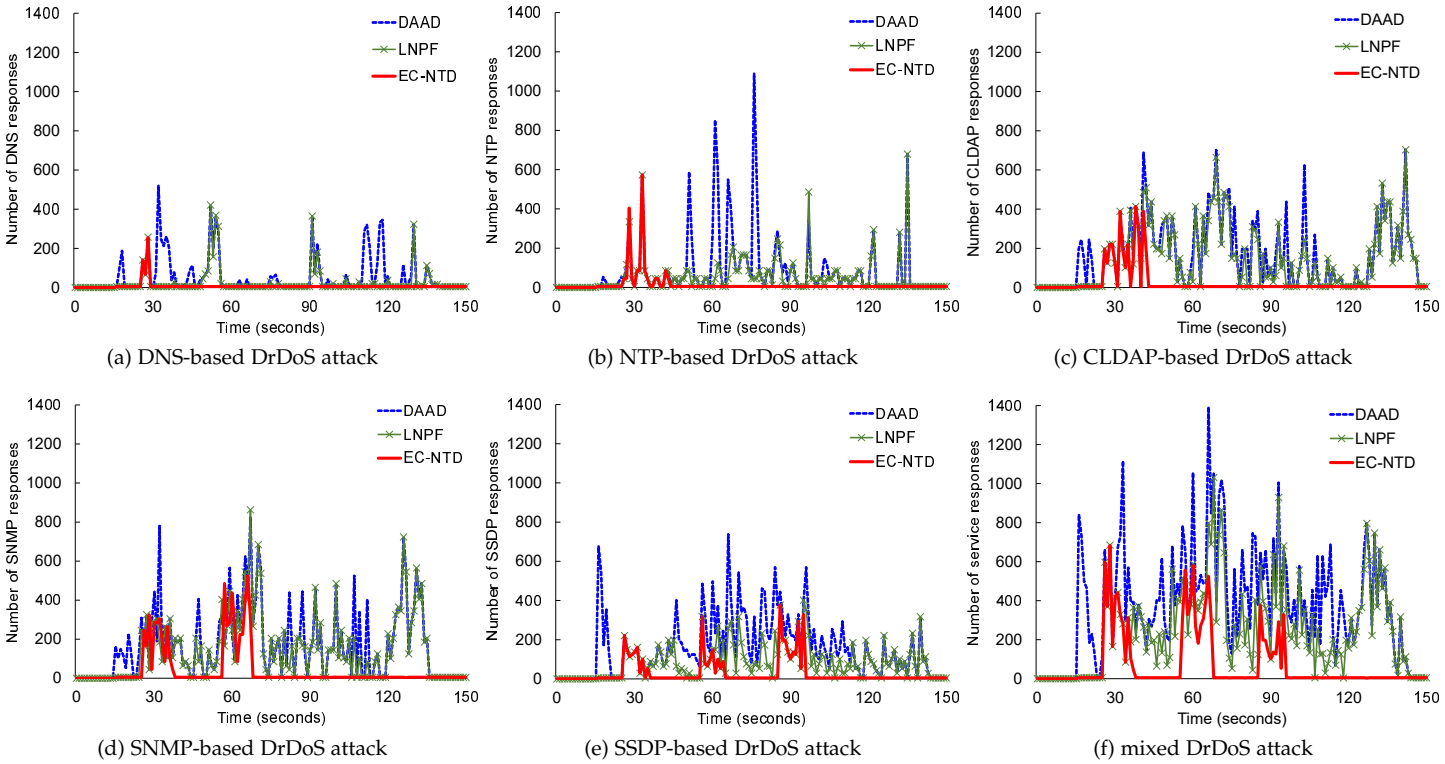
attack	stage 1	stage 2	stage 3	EBM:IBM	requests
DNS	5s–24s	25s–129s	130s–140s	7:3	11,390
NTP	5s–24s	25s–119s	120s–135s	7:3	17,112
CLDAP	5s–24s	25s–124s	125s–145s	2:8	27,264
SNMP	5s–24s	25s–119s	120s–135s	2:8	26,496
SSDP	5s–24s	25s–124s	125s–143s	5:5	28,416
mixed	5s–24s	25s–124s	125s–143s	4:6	67,652

As discussed in Section 2.1, five network services with large bandwidth amplification factors are chosen for launching DrDoS attacks, including DNS, NTP, CLDAP, SNMP, and SSDP. In addition to these services, we produce *mixed* DrDoS attacks, which contain DrDoS attacks using DNS, SNMP, and SSDP. To measure the effect of attacks coming from different sources, we divide the duration of each DrDoS attack into three stages. In stage 1, only EBMs generate attack packets. Then, both EBMs and IBMs send malicious service requests to public servers during stage 2. In stage 3, only IBMs make fake requests. Table 8 shows the settings for different DrDoS attacks, including the duration of each stage, the ratio of requests generated by EBMs and IBMs, and the total number of requests sent by botnet members. In addition, starting from the 15th second, the targeted host sends normal requests to public servers at the speed of five packets per second.

Two SDN-based DrDoS countermeasures mentioned in Section 3.2 are chosen for comparison. In the DAAD method [21], when a switch receives service requests from internal hosts, the controller sets a flow rule for upcoming responses. DAAD is designed for DNS-based DrDoS attacks, so we modify flow rules to let it be adapted to DrDoS attacks using other services. Then, the *lightweight NAT-based packet filtering (LNPF)* method [25] applies the NAT protocol to gateways to drop malicious responses, which replaces the targeted host's IP address with an alias IP address in legitimate requests. As discussed in Section 3.2, these two methods do not need extra components (e.g., detection agent, honeypot, or symmetric routing). Hence, we compare our EC-NTD scheme with both DAAD and LNPF.

6.3 Comparison of Received Responses

We measure the number of service responses received by the targeted host per second under different DrDoS attacks. Fig. 7(a) gives the experimental result with a DNS-based DrDoS attack, where the ratio of attack packets generated by EBMs and IBMs is set to 7:3. Since LNPF asks the gateway to drop response packets by default, it performs better than DAAD. However, both LNPF and DAAD cannot efficiently defend against attacks coming from IBMs, so the targeted host receives many DNS responses in stages 2 and 3 (i.e., the 25th–140th seconds). Thanks to the two-phase detection mechanism, our EC-NTD scheme can quickly recognize malicious requests sent by IBMs. Hence, after the 30th second, the targeted host receives merely legitimate responses (i.e., corresponding to the requests issued by the targeted host). In other words, all malicious requests originated inside the SDN-based network can be correctly discarded by EC-NTD.



(a) DNS-based DrDoS attack (b) NTP-based DrDoS attack (c) CLDAP-based DrDoS attack

(d) SNMP-based DrDoS attack (e) SSDP-based DrDoS attack (f) mixed DrDoS attack

Fig. 7: Comparison of the number of service responses received by the targeted host per second.

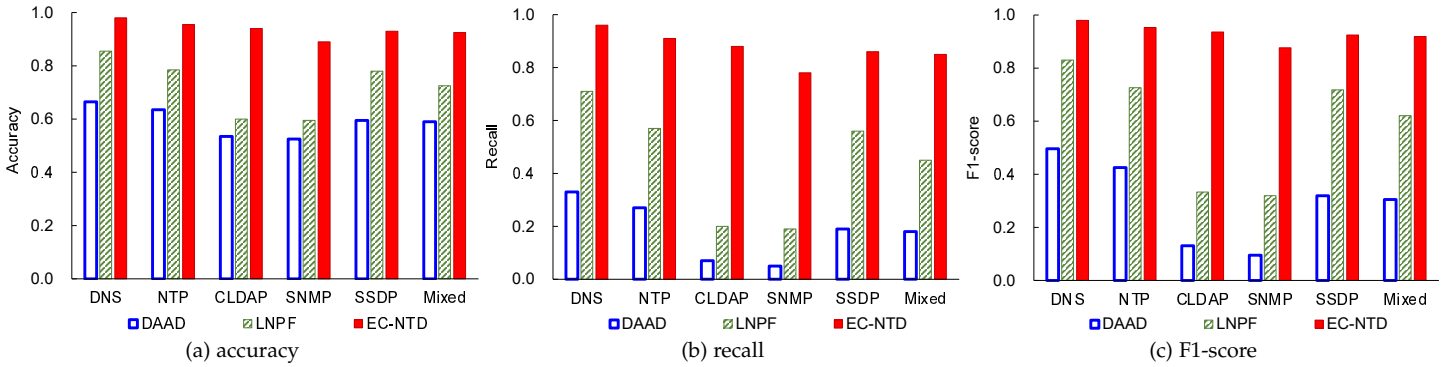
Fig. 7(b) shows the experimental result with an NTP-based DrDoS attack. The ratio of attack packets from EBMs and IBMs is the same as that in the DNS-based DrDoS attack (i.e., 7:3), but more NTP requests are generated. Thus, the targeted host receives more NTP responses per second in Fig. 7(b) as compared to DNS responses in Fig. 7(a). Moreover, there are six particularly high peaks (more than 500 NTP responses) at the 33rd, 51st, 61st, 66th, 76th, and 135th seconds. That is because botnet members tend to send numerous NTP requests suddenly in this attack. As can be seen, these six peaks all occur in DAAD, which reveals that DAAD cannot well resist such bursty transmissions of NTP requests. On the contrary, after the 43rd second, there is no peak in our EC-NTD scheme. This means that all attack packets have been blocked by EC-NTD.

With a CLDAP-based DrDoS attack, Fig. 7(c) presents the number of CLDAP responses received by the targeted host. In this attack, we change the ratio of malicious requests generated by EBMs and IBMs to 2:8. As many attack packets originate from the SDN-based network (i.e., IBMs), the targeted host is flooded with CLDAP responses in DAAD and LNPF, especially after the 25th second (i.e., the starting time of stage 2). By applying NAPT to the gateway to drop malicious responses caused by EBMs and using two-phase detection to identify forged requests from IBMs, EC-NTD can efficiently block the CLDAP-based DrDoS attack after the 42nd second.

For the SNMP-based DrDoS attack, the ratio of SNMP requests from EBMs and IBMs is kept to 2:8. However, IBMs are divided into two batches for attack in stage 2, where each batch contains 50% of IBMs. The first batch of IBMs launches the attack on the 25th second. Then, the second batch of IBMs participate in the attack after the 55th second. As can be seen in Fig. 7(d), the performance of DAAD and LNPF does not change significantly compared to the result in Fig. 7(c). They cannot effectively block malicious SNMP responses caused by botnet members. On the other hand, the targeted host receives some malicious SNMP responses during the 26th–36th and 57th–67th seconds when using our EC-NTD scheme. The reason is that the controller checks IBM attacks period by period in EC-NTD. When a new batch of IBMs partake in the attack, the controller requires some time to conduct detection. Despite this, EC-NTD can efficiently block most malicious SNMP responses compared to DAAD and LNPF.

Then, Fig. 7(e) gives the result when an SSDP-based DrDoS attack takes place. EBMs and IBMs generate a similar number of malicious SSDP requests. Moreover, we divide IBMs into three groups of equal size. These groups of IBMs participate in the attack in sequence at the 25th, 55th, and 85th seconds (in stage 2). As mentioned earlier, EC-NTD needs some time to detect attacks from IBMs. Hence, the targeted host receives some malicious SSDP responses during the 26th–35th, 56th–64th, and 86th–95th seconds. As for DAAD and LNPF, they cannot prevent IBMs from sending malicious SSDP requests to the public server, so the targeted host continues to be attacked. The above result demonstrates that our EC-NTD scheme outperforms both DAAD and LNPF in terms of resisting the SSDP-based DrDoS attack.

When evaluating countermeasures against DrDoS attacks, the capacity to cope with malicious requests is also an important metric. To measure the capacity and defense effectiveness of DAAD, LNPF, and EC-NTD, we generate one mixed DrDoS attack, which is essentially a combination of DNS-based, SNMP-based, and SSDP-based DrDoS attacks. According to Table 8, the ratio of malicious requests sent by EBMs and IBMs is set to 4:6, and the number of malicious requests in the mixed DrDoS attack is much more than other DrDoS attacks. Fig. 7(f) shows the number of service responses received by the targeted host per second. In each method, the quantity



(a) accuracy (b) recall (c) F1-score

Fig. 8: Comparison of the detection performance of each method.

of received responses substantially increases, as compared to Fig. 7(a)–(e). Such a phenomenon is particularly obvious in the DAAD and LNPF methods, which implies that the number of malicious requests is far beyond their capacity. This reflects the small capacity of DAAD and LNPF and their low effectiveness of defenses when faced with numerous malicious requests. In contrast, for the EC-NTD scheme, the targeted host receives relatively more malicious responses during three short periods (i.e., the 26th–36th, 56th–67th, and 86th–95th seconds), but the number of received responses is significantly lower than that in DAAD and LNPF. Furthermore, except for the three periods, EC-NTD is capable of dropping almost all attack packets. The above result demonstrates that compared to DAAD and LNPF, the large capacity of our EC-NTD scheme can help safeguard against the mixed DrDoS attack that generates many malicious requests more efficiently.

Additionally, *time-to-mitigation*, as defined by the amount of time taken by a countermeasure to start defense since a DrDoS attack begins, is a metric used to evaluate how fast the countermeasure react to attacks. The significance of minimizing time-to-mitigation in countering DrDoS attacks are twofold. First, we can decrease the number of malicious service responses received and handled by the targeted host to save its resources. Second, the reduction of attack packets helps reduce the bandwidth consumption in the SDN-based network and the burden of packet delivery on switches. Observing from Fig. 7, LNPF and EC-NTD have significantly shorter time-to-mitigation than DAAD in stage 1, where only EBMs launch attacks. This is because both methods drop responses from public servers by default and enable only legitimate responses to pass the gateway. When IBMs join attacks (i.e., stages 2 and 3), DAAD and LNPF cannot prevent their malicious requests from reaching public servers. In other words, their time-to-mitigation in terms of IBM attacks can be viewed as very long. On the contrary, EC-NTD can fast identify IBM attacks and block their malicious requests. In particular, EC-NTD's time-to-mitigation is no more than 15s. This result reveals that the reaction of our EC-NTD scheme to attacks is rapid, which can significantly reduce damages during attacks.

6.4 Comparison of Detection Performance

We then evaluate the detection performance of DAAD, LNPF, and EC-NTD. As mentioned in Section 4.3, four metrics, including accuracy, recall, precision, and F1-score, are used for performance evaluation. Fig. 8(a) gives the accuracy of each method, which is calculated by Eq. (2). As can be seen, all methods have the lowest accuracy in the SNMP-based DrDoS attack because most malicious requests are generated by IBMs, and they participate in the attack at different times. The accuracy of DAAD is between 0.525 and 0.665, while the accuracy of LNPF is between 0.595 and 0.855. On the other hand, the accuracy of our EC-NTD scheme is kept above 0.890. This result reveals that EC-NTD can identify attack packets the most accurately among all methods. Compared to DAAD and LNPF, EC-NTD improves 58.53% and 29.49% of accuracy on blocking attack packets, respectively.

Fig. 8(b) compares the recall of each method, as defined in Eq. (3). Since the ratio of malicious requests from EBMs and IBMs is set to 2:8 in both CLDAP-based and SNMP-based DrDoS attacks (in other words, most malicious requests are sent by IBMs), DAAD and LNPF have the lowest recall in these two attacks. This phenomenon implies that DAAD and LNPF cannot efficiently identify malicious requests generated by IBMs in the SDN-based network. By using different strategies to resist attacks launched by EBMs and IBMs, our EC-NTD scheme can substantially increase the recall. Specifically, EC-NTD improves recall by 380.73% and 95.52% compared to DAAD and LNPF, respectively.

Regarding precision, since some attack packets can be correctly blocked by all methods (i.e., true positives), the condition of $f_{\text{TP}} > 0$ holds for each method. In addition, none of these three methods discard legitimate responses, so there will be no false positives (i.e., $f_{\text{FP}} = 0$). According to Eq. (4), we derive that Precision $= f_{\text{TP}}/(f_{\text{TP}}+f_{\text{FP}}) = f_{\text{TP}}/(f_{\text{TP}}+0) = 1$. Consequently, the precision of each method is always equal to one under different DrDoS attacks. To save space, we omit the result of precision in Fig. 8.

Then, Fig. 8(c) presents the F1-score of each method. Based on Eq. (5), the F1-score is decided by both recall and precision. As the precision of each method is the same (i.e., equal to one), the F1-score will be affected solely by the recall. Hence, the trend of F1-score in Fig. 8(c) is similar to that of recall in Fig. 8(b). Apparently, our EC-NTD scheme results in higher F1-scores than the DAAD and LNPF methods under different DrDoS attacks. In particular, EC-NTD can improve 215.40% and 57.52% of the F1-score compared to DAAD and LNPF, respectively.

6.5 Comparison of Grouping Methods

In the IBM attack resisting module of EC-NTD, we use K-means to divide entries in the request statistics table into normal and abnormal groups (referring to Algorithm 1) for

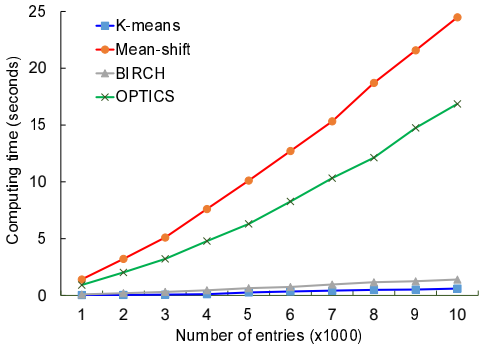


Fig. 9: Comparison of computing time for different clustering methods.

finding malicious requests. In unsupervised machine learning, there are some widely used clustering methods, including K-means [37]. Hence, we choose three methods from them to compare with K-means:

- *Mean-shift* assigns data points to groups iteratively by shifting them towards the highest density of data points in a region.
- *BIRCH (balanced iterative reducing and clustering using hierarchies)* generates a small summary of a large dataset that retains as much information as possible. Then, it clusters this small summary instead of the large dataset.
- *OPTICS (ordering points to identify the clustering structure)* extracts the clustering structure of a dataset by identifying the density-connected points.

We replace Algorithm 1 with mean-shift, BIRCH, and OPTICS in our EC-NTD scheme and conduct the experiments in Section 6.4 again. The detection performance of EC-NTD does not change, which means that using the above clustering methods has the same effect as K-means. Then, we compare the computing time for different clustering methods, as shown in Fig. 9, where the number of entries in the request statistics table is increased from 1000 to 10000. We can observe that K-means has the lowest computing time among all methods. Due to its low computational cost, we select K-means to divide entries in the request statistics table.

7 Conclusion and Future Work

In DrDoS attacks, botnet members send many service requests with forged source IP addresses to public servers, resulting in a flood of service responses to targeted hosts. With SDN, we propose the EC-NTD scheme to resist DrDoS attacks. It applies filtering rules and NAPT to gateways to drop malicious responses caused by EBMs located in external networks. To find malicious requests sent by IBMs inside the SDN-based network, the controller checks for anomalies in requests via a two-phase detection mechanism. The period length for attack detection is also adjusted to balance the controller's load and detection performance. Based on simulation outcomes, we identify three significant findings. First, compared to DAAD and LNPF, EC-NTD can block DrDoS attacks with different services from EBMs and IBMs more effectively. Second, regarding detection performance, EC-NTD performs far better on accuracy, recall, and F1-score than DAAD and LNPF. Third, when IBMs launch a batched attack, EC-NTD takes some time to recognize malicious requests. However, DAAD and LNPF cannot prevent IBMs from sending malicious requests to public servers. The above findings confirm that EC-NTD is capable of better and more comprehensively blocking DrDoS attacks than existing DAAD and LNPF methods.

In the IBM attack resisting module of EC-NTD, the controller finds malicious requests inside the SDN-based network using the relationship between IP and MAC addresses. Some techniques, such as *multi-protocol label switching (MPLS)*, provide additional details about the origin of a flow. As future work, we will consider using MPLS to help identify malicious requests. Furthermore, it deserves further investigation to apply asset-centric threat modeling approaches [38], like DREAD, Trike, OCTAVE, or PASTA, to deal with DrDoS attacks.

References

[1] Securelist, "DDoS reports," https://securelist.com/category/ddos-reports/.
[2] 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.
[3] 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.
[4] 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.
[5] Y.C. Wang and P.Y. Su, "Collaborative defense against hybrid network attacks by SDN controllers and P4 switches," *IEEE Trans. Network Science and Engineering*, vol. 11, no. 2, pp. 1480–1495, 2024.
[6] A. Ahmim, F. Maazouzi, M. Ahmim, S. Namane, and I.B. Dhaou, "Distributed denial of service attack detection for the Internet of Things using hybrid deep learning model," *IEEE Access*, vol. 11, pp. 119862–119875, 2023.
[7] 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.
[8] 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.
[9] Cybersecurity and Infrastructure Security Agency, "UDP-based amplification attacks," https://www.cisa.gov/news-events/alerts/2014/01/17/udp-based-amplification-attacks.
[10] 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.
[11] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *IEEE Comm. Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
[12] P. Srisuresh and K. Egevang, "Traditional IP network address translator (traditional NAT)," Internet Engineering Task Force, RFC 3022, January 2001.
[13] C. Liu, G. Xiong, J. Liu, and G. Gou, "Detect the reflection amplification attack based on UDP protocol," *Proc. Int'l Conf. Comm. and Networking in China*, 2015, pp. 260–265.
[14] S. Yadav, J. Singh, S. Bhatnagar, K. Goyal, and A. Yadav, "The probability Strata: algorithmic approach to DrDoS defense," *Proc. Int'l Conf. Computing for Sustainable Global Development*, 2016, pp. 1865–1870.
[15] H. Fujinoki, "Cloud-base defense against DrDoS attacks," *Proc. IEEE Int'l Conf. Consumer Electronics-Taiwan*, 2018, pp. 1–2.
[16] E. Biagioni, "Preventing UDP flooding amplification attacks with weak authentication," *Proc. Int'l Conf. Computing, Networking and Comm.*, 2019, pp. 78–82.
[17] X.Z. Khooi, L. Csikor, D.M. Divakaran, and M.S. Kang, "DIDA: distributed in-network defense architecture against amplified reflection DDoS attacks," *Proc. IEEE Conf. Network Softwarization*, 2020, pp. 277–281.
[18] K. Ozdincer and H.A. Mantar, "SDN-based detection and mitigation system for DNS amplification attacks," *Proc. Int'l Symp. Multidisciplinary Studies and Innovative Technologies*, 2019, pp. 1–7.
[19] X. Xing, T. Luo, J. Li, and Y. Hu, "A defense mechanism against the DNS amplification attack in SDN," *Proc. IEEE Int'l Conf. Network Infrastructure and Digital Content*, 2016, pp. 28–33.

[20] V. Gupta, A. Kochar, S. Saharan, and R. Kulshrestha, "DNS amplification based DDoS attacks in SDN environment: detection and mitigation," *Proc. IEEE Int'l Conf. Computer and Comm. Systems*, 2019, pp. 473–478.
[21] M. Han, T.N. Canh, S.C. Noh, J.Yi, and M. Park, "DAAD: DNS amplification attack defender in SDN," *Proc. Int'l Conf. Information and Comm. Technology Convergence*, 2019, pp. 372–374.
[22] C.C. Chen, Y.R. Chen, W.C. Lu, S. C. Tsai, and M.C. Yang, "Detecting amplification attacks with software defined networking," *Proc. IEEE Conf. Dependable and Secure Computing*, 2017, pp. 195–201.
[23] Y. Zhauniarovich and P. Dodia, "Sorting the garbage: filtering out DrDoS amplification traffic in ISP networks," *Proc. IEEE Conf. Network Softwarization*, 2019, pp. 142–150.
[24] V. Gupta, S. Saharan, and S. Raje, "SymSDN: a DrDoS attack prevention approach," *Proc. IEEE Wireless Comm. and Networking Conf.*, 2023, pp. 1–6.
[25] T. Lukaseder, K. Stolzle, S. Kleber, B. Erb, and F. Kargl, "An SDN-based approach for defending against reflective DDoS attacks," *Proc. IEEE Conf. Local Computer Networks*, 2018, pp. 299–302.
[26] Y.C. Wang and H. Hu, "An adaptive broadcast and multicast traffic cutting framework to improve Ethernet efficiency by SDN," *J. Information Science and Engineering*, vol. 35, no. 2, pp. 375–392, 2019.
[27] A. Silberschatz, G. Gagne, and P.B. Galvin, *Operating System Concepts*, Hoboken: Wiley, 2019.
[28] M.S. Ferdous, F. Chowdhury, and J.C. Acharjee, "An extended algorithm to enhance the performance of the current NAPT," *Proc. Int'l Conf. Information and Comm. Technology*, 2007, pp. 315–318.
[29] Q. Li, H. Huang, R. Li, J. Lv, Z. Yuan, L. Ma, Y. Han, and Y. Jiang, "A comprehensive survey on DDoS defense systems: new trends and challenges," *Computer Networks*, vol. 233, pp. 1–27, 2023.
[30] X. Sun and L. Dai, "Backoff design for IEEE 802.11 DCF networks: fundamental tradeoff and design criterion," *IEEE/ACM Trans. Networking*, vol. 23, no. 1, pp. 300–316, 2015.
[31] 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.
[32] Mininet, http://mininet.org/.
[33] Ryu, https://ryu-sdn.org/.
[34] Open vSwitch, https://www.openvswitch.org/.
[35] Tcpreplay—PCAP editing and replaying utilities, https://tcpreplay.appneta.com/.
[36] Canadian Institute for Cybersecurity, "DDoS evaluation dataset (CIC-DDoS2019)," https://www.unb.ca/cic/datasets/ddos-2019.html.
[37] A. Gansen, J. Hennicker, C. Sill, J. Dheur, J.S. Hale, and J. Baller, "Melt instability identification using unsupervised machine learning algorithms," *Macromolecular Materials and Engineering*, vol. 308, no. 6, pp. 1–14, 2023.
[38] L.O. Nweke and S.D. Wolthusen, "A review of asset-centric threat modelling approaches," *Int'l J. Advanced Computer Science and Applications*, vol. 11, no. 2, pp. 1–6, 2020.