Efficient Defense Against DNS Floods Using Machine Learning in SDN-Based Networks

You-Chiun Wang and Chen-I Wei
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan
Email: ycwang@cse.nsysu.edu.tw; da30231@gmail.com

***Abstract*—Distributed denial-of-service (DDoS) attacks are serious network threats. *Domain name system (DNS)* floods belong to DDoS attacks, where victim DNS servers are overwhelmed by lots of malicious DNS queries, making web services unavailable to hosts. The paper combines *machine learning (ML)* and *software-defined networking (SDN)* to cope with DNS floods. We analyze the performance of detecting malicious DNS queries using different ML techniques. Based on the analysis, we propose an *efficient DT-based defense against DNS floods (ED3F)* scheme. The SDN controller checks if the frequency of DNS queries is high and employs a trained ML model to identify malicious DNS queries. Moreover, ED3F adjusts its blocking time depending on the status of a host sending malicious DNS queries. Using simulation, we show that the ED3F scheme can quickly and accurately block malicious DNS queries, thereby defending against DNS floods efficiently.**

***Keywords*—DDoS, DNS flood, machine learning, SDN.**

I. Introduction

Distributed denial-of-service (DDoS) attacks are common cyberattacks targeting network, transport, or application layers. Victim servers are deluged with many requests in an attempt to consume resources, which prevent legitimate requests from being served. Such attacks are usually launched using a botnet composed of compromised hosts and devices. Since a DDoS attack's packets originate from numerous sources, it cannot be simply resisted using IP filtering (e.g., via a firewall).

Domain name system (DNS) floods are a form of DDoS attack. The attacker aims to obstruct the resolution of resource records by a DNS server responsible for a zone. DNS servers offer the Internet's roadmap, which helps hosts look up the IP addresses of destinations (e.g., web servers) to access contents. In a DNS flood, lots of DNS queries are sent to the DNS server to drain computing resources, impeding its ability to process legitimate requests. This forces web services unavailable to hosts, as they cannot obtain the IP addresses of web servers.

Machine learning (ML) studies how computers learn from data and accomplish tasks (e.g., classification and prediction) using what they have learned. ML has been applied to detect various cyberattacks, like eavesdropping [1], keylogging [2], and false data injection [3]. They analyze attack datasets to find attack signatures. Common ML techniques include K-means, *K-nearest neighbors (KNN)*, *naive Bayes (NB)*, *support vector machine (SVM)*, and *decision tree (DT)* [4].

In *software-defined networking (SDN)*, a controller is used to supervise switches to make network management easy [5]. It communicates with switches using the OpenFlow protocol. Each switch has flow tables to store flow entries sent by the controller. It checks if a packet meets conditions given in a flow entry. If so, the switch obeys the entry's instructions to handle the packet. The controller can also query switches about the quantity and types of packets transferred via them.

This paper combines ML and SDN to resist DNS floods. We analyze the performance of recognizing malicious DNS queries using ML techniques. With the analysis, we design an *efficient DT-based defense against DNS floods (ED3F)* scheme. The controller checks if the frequency of DNS queries is high (i.e., a symptom of DNS floods). If so, the controller uses a trained DT model to identify malicious DNS queries and asks switches to drop them. Since botnet members may dynamically change, we cannot block DNS queries of a host for a long time. Hence, ED3F adjusts its blocking time based on the status of a host sending malicious DNS queries. Simulation results reveal that our ED3F scheme can quickly and accurately block malicious DNS queries, thereby resisting DNS floods efficiently.

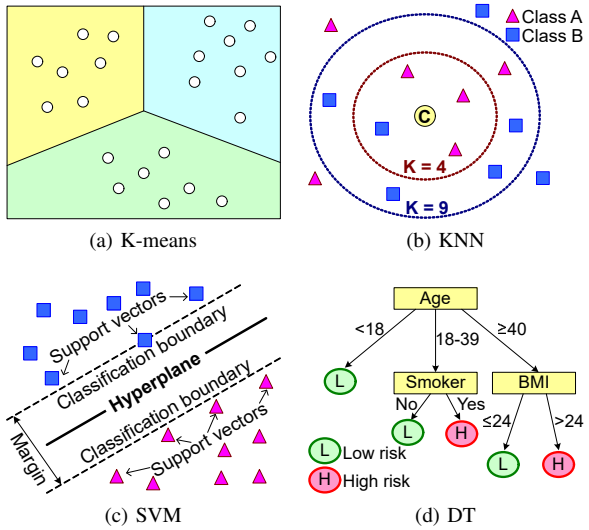
II. Overview of Common ML Techniques

K-means divides data points into K groups. Beginning with K random centroids, it assigns each data point to a group such that the data point is the closest to the group's centroid (i.e., mean). Then, the centroid of each group is recalculated. The above operation is repeated until all centroids have stabilized. Fig. 1(a) gives an example, where $K = 3$.

In KNN, every object is assigned to the class that is most prevalent among its K closest neighbors. Fig. 1(b) shows an example, where objects of classes A and B are represented by triangles and squares. When $K = 4$, object C is categorized to class A, since its four neighbors contain three triangles and one square. If $K = 9$, object C is assigned to class B, as there are four triangles and five squares in its neighborhood.

NB is a probability model, considering that the features of a target class are conditionally independent. Let a problem instance be represented as a vector $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ encoding n features (i.e., independent variables). NB gives a probability for each possible class X_k: $P(X_k \mid v_1, v_2, \cdots, v_n) \propto P(X_k) \prod_{i=1}^{n} P(v_i \mid X_k)$, where $\propto$ denotes proportionality.

SVM uses a hyperplane for classification, where margin is the distance between two classification boundaries, as Fig. 1(c) shows. Data points from each class nearest to its boundary are support vectors. Training SVM is to find the maximum margin and pick support vectors (i.e., the most useful data points, as they are most likely to be classified incorrectly). Thus, SVM can expedite classification by using only support vectors.



(a) K-means (b) KNN

(c) SVM (d) DT

Fig. 1. Examples of ML techniques.

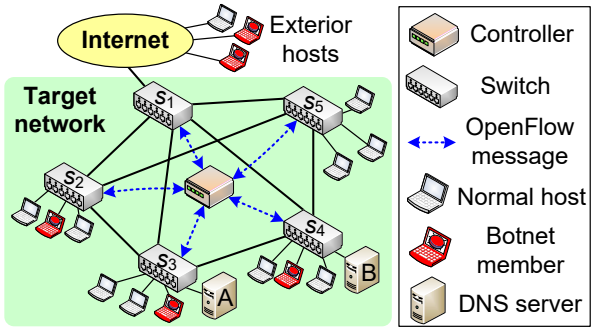


Fig. 2. A schematic diagram of DNS floods in an SDN-based network.

DT has a flowchart-like tree structure. Each internal node gives one test on an attribute, each branch expresses the test's outcome, and each leaf shows a decision made after evaluation. The path from the root to each leaf indicates a classification rule. Fig. 1(d) presents an example of using DT to assess the risk of heart disease. The first attribute is age. If the age is below 18, the risk is low. When the age falls between 18 and 39, we judge the risk based on whether the tester smokes. If the age is 40 or above, we judge the risk using BMI.

III. Related Work

Various SDN-based solutions to DDoS attacks are designed. The study [6] infers that a UDP flood occurs if a switch has far more received packets than sent packets. In [7], when a new host sends UDP packets to a server, the server transmits a keepalive packet for the host to answer. If some hosts send many SYN packets without finishing handshakes, the work [8] treats them as attackers of SYN floods. In [9], the controller finds anomalous packets via an intrusion prevention system. The work [10] checks if there are DDoS flows based on flow sizes, IP variability, and duration. The study [11] uses principal component analysis to find DDoS attacks. However, in DNS floods, botnet members behave like normal hosts, except that they send more DNS queries. Furthermore, DNS requires only a two-way handshake, that is, a host sends a DNS query and the server returns a DNS reply. The two properties make the above methods unable to effectively solve DNS floods.

Many studies measure the entropy of IP addresses in packets to find DDoS attacks. The Shannon entropy is adopted in [12]–[14], as computed by $E(Y) = -\sum_{i=1}^{n} p_i \log_2 p_i$. In random variable $Y = \{y_1, y_2, \cdots, y_n\}$, y_i $(i = 1 \ldots n)$ denotes the event that packets with a certain IP address are sent to a server, and p_i gives its probability. The work [15] uses the φ-entropy, as defined by $E_\varphi(Y) = -\sum_{i=1}^{n} p_i \sinh(\varphi \log_2 p_i)/\sinh(\varphi)$. Here, $\sinh(\cdot)$ is a hyperbolic sine function. In [16], the entropy of flows in a time window is compared with a threshold to detect attacks. However, legitimate DNS queries may also be discarded by the entropy-based methods, causing false alarms.

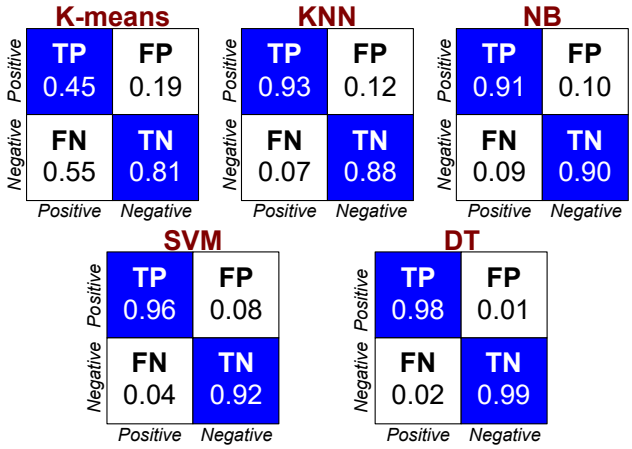
In [17], if DNS floods occur, DNS queries are passed from the attacked server to other servers. A program is set on users' local servers to allow the IP addresses of paid websites to be maintained in caches. Thus, the websites can be accessed even if DNS servers are down. However, malicious DNS queries are not blocked and continue to attack servers. Datta et al. [18] propose a two-level method called *DNSguard*. In the first level, they consider IoT devices connected to a gateway and authorize MUD-compliant devices to send DNS queries, where MUD stands for manufacturer usage description. Then, the second level mitigates DNS floods by monitoring the rate of DNS traffic. However, MUD is not a widely used protocol.

Evidently, the issue of how to resist DNS floods efficiently has not been intensively investigated yet. Compared to previous studies, our work combines ML and SDN to quickly and accurately identify malicious DNS queries, thereby effectively defending DNS servers against such attacks.

IV. System Model

We consider a *target network* using SDN for management, as Fig. 2 shows. It contains DNS servers, hosts, and switches. DNS servers provide DNS services available to the public, so exterior hosts can also request their services. Switches form the network backbone and support OpenFlow. The target network is directed by a controller that issues commands to switches by sending them flow entries. Moreover, switches report state-related information (e.g., the number of packets handled and their types) to the controller. These operations are done using OpenFlow messages exchanged between the controller and switches. In this way, the controller can grasp the overall status and manage flows in the target network.

An attacker will launch DNS floods on some DNS servers through a botnet. Botnet members may be inside or outside the target network. They send many DNS queries to DNS servers to use up their resources, preventing them from serving hosts. Since botnet members may be unknown and can change, our objective is to quickly and accurately identify DNS floods and stop them. Moreover, we shall distinguish between malicious and legitimate DNS queries, so normal hosts can still use DNS services when the attack is being blocked.



(a) confusion matrices

Technique	Accuracy	Precision	Recall	F1-score
K-means	0.6300	0.7031	0.4500	0.5488
KNN	0.9050	0.8857	0.9300	0.9073
NB	0.9050	0.9010	0.9100	0.9055
SVM	0.9400	0.9231	0.9600	0.9412
DT	0.9850	0.9899	0.9800	0.9849

(b) evaluation using four metrics

Fig. 3. Detection performance of the five ML techniques.

V. The Proposed ED3F Scheme

In ED3F, we select an ML technique and train its model to detect DNS floods. Then, the controller monitors the frequency of DNS queries sent to each DNS server. Once the frequency exceeds a threshold δ, the DNS server is overburdened, and there is a good possibility that DNS floods are taking place. Hence, the controller identifies malicious DNS queries using the trained model and issues flow rules to switches to block the queries. The blocking time will be adaptively adjusted based on the behavior of a suspicious host sending DNS queries.

A. Selection of ML Techniques

We measure the performance of K-means, KNN, NB, SVM, and DT, on detecting DNS floods. A powerful packet manipulation tool called *Scapy* [19] is used to generate legitimate and attack traffics. We refer to the *MazeBolt open dataset* [20] to model DNS floods. The controller collects 50 seconds of traffic information during the attacking phase from each switch and employs 65% of the packets as a training set to discipline each ML model. The remaining packets are used as a test set.

Fig. 3(a) shows the confusion matrices of ML techniques. In a confusion matrix, each row gives an instance (i.e., positive or negative) of the predicted class, and each column displays an instance of the actual class. It has four parts. *True positive (TP)* gives the proportion of DNS queries where an ML model says that they are malicious, and the queries indeed are. *False positive (FP)* indicates the proportion of DNS queries where an ML model states that they are malicious, yet the queries are legitimate. *False negative (FN)* shows the proportion of DNS queries where an ML model claims that they are legitimate, but the queries are malicious. *True negative (TN)* presents the proportion of DNS queries where an ML model predicts that they are legitimate, and these queries are legitimate.

As can be seen, K-means has the lowest TP and TN values (i.e., correct cases) and the highest FP and FN values (i.e., error cases). KNN, NB, SVM, and DT perform much better than K-means on TP and FN. Moreover, DT has the highest TP and TN values and also the lowest FP and FN values. This means that DT can identify malicious DNS queries more accurately while reducing more false alarms.

Through TP, FP, FN, and TN, we can quantify the detection performance of each ML technique using four metrics:

$$\textbf{accuracy} = (\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN}), \quad (1)$$

$$\textbf{precision} = \text{TP}/(\text{TP}+\text{FP}), \quad (2)$$

$$\textbf{recall} = \text{TP}/(\text{TP}+\text{FN}), \quad (3)$$

$$\textbf{F1-score} = 2(\text{precision} \times \text{recall})/(\text{precision}+\text{recall}). \quad (4)$$

Fig. 3(b) compares K-means, KNN, NB, SVM, and DT via the four metrics. Evidently, DT performs the best in all metrics. Based on the analysis, we thus choose DT as the ML technique used to detect DNS floods in our ED3F scheme.

B. Model Training and Usage

To produce an attack dataset for training the DT model, we inject attack traffics and let the controller collect packet data about DNS queries. In the attack dataset, eight features are extracted: *datapath_id*, *flow_id*, *ip_src*, *ip_dst*, *tp_src*, *packet_count*, *flow_duration_sec*, and *label*. Both datapath_id and flow_id uniquely identify switches and flows. We know from ip_src and ip_dst the hosts that sent DNS queries and the victim DNS servers, and tp_src indicates what services are employed (tp_src is 53 for DNS). In DNS floods, botnet members transmit many DNS queries over a longer period of time. Thus, packet_count and flow_duration_sec could be manifest features. Moreover, DT is a supervised ML technique, so the label is used to distinguish between malicious and legitimate DNS queries for learning. Then, a DT model is trained based on the correlation between the label and the other seven features of every datum in the attack dataset. The trained DT model is stored in a PKL (Python pickle) file.

In actual testing applications, the controller gathers packet data about DNS queries from switches, converts data into the CSV format, and removes unnecessary fields. Then, it loads the trained DT model from the PKL file and uses the model to classify DNS queries into malicious and legitimate groups. The label is only used to train the DT model. This feature does not appear in practical packet data. Then, if the malicious group is not empty, the controller adopts the mechanism in Section V-C to drop subsequent DNS queries sent from suspicious hosts.

C. Attack Blocking

Once the controller discovers that some hosts send malicious DNS queries (using the trained DT model), it then commands responsible switches to drop subsequent DNS queries sent by these suspicious hosts (as they are possibly botnet members). More concretely, if a suspicious host, say, h_i is located in the target network, the responsible switch is the one to which h_i attaches. Otherwise, h_i is an exterior

host, and the responsible switch will be a gateway switch (e.g., switch s_1 in Fig. 2).

Let a_i be the IP address of h_i and a_v be the IP address of the victim DNS server that h_i attacks. The controller installs a flow entry in the responsible switch as follows:

[Match fields] eth_type=0x0800, ipv4_src=a_i, ipv4_dst=a_v, udp_dst=53

[Action] drop **[Timeout]** hard_timeout=$T(\tau)$

In the match fields, the term "eth_type = 0x0800" indicates IPv4 packets. If IPv6 is used, we can set eth_type to 0x86DD. Fields ipv4_src and ipv4_dst designate the packet's source and destination IP addresses, which are set to the IP addresses of the suspicious host (i.e., a_i) and the victim server (i.e., a_v). When using IPv6, fields ipv4_src and ipv4_dst are replaced with ipv6_src and ipv6_dst. The term "udp_dst = 53" means that the packet is a DNS query. Then, if a packet meets all conditions in the match fields, the switch drops that packet, as indicated in the action field.

The hard_timeout field decides the duration (in seconds) of the flow entry. After $T(\tau)$ time, the flow entry is automatically removed [21]. Hence, the amount of time to block h_i's DNS queries is $T(\tau)$. We set $T(\tau) = T_{\texttt{base}} \times \tau$, where $\tau \in \mathbb{Z}^+$. $T_{\texttt{base}}$ is the basic time length (in seconds), and τ is initially set to 1. Let f_i be the frequency with which h_i previously sent DNS queries. Suppose that $N_{\texttt{avg}}$ is the average number of DNS queries sent by normal hosts during $T(\tau)$ time. Then, we adjust the τ value to compute the new blocking time for h_i based on four cases:

Case 1: $N_i > \alpha(f_i \times T(\tau))$, where N_i is the number of DNS queries sent by h_i during the blocking time, and $0.75 < \alpha \leq 1$. This case indicates that h_i continues to send many malicious DNS queries to the victim DNS server. Therefore, we set τ to $\min\{\tau + 1, \tau_{\texttt{max}}\}$ to punish h_i by extending its blocking time, where $\tau_{\texttt{max}}$ is the maximum value of τ.

Case 2: $\beta(f_i \times T(\tau)) < N_i \leq \alpha(f_i \times T(\tau))$, where $0 < \beta < \alpha$. Here, the τ value will not change.

Case 3: $(1+\gamma)N_{\texttt{avg}} < N_i \leq \beta(f_i \times T(\tau))$, where γ is a small positive value (e.g., $\gamma = 0.1$). This case means that h_i significantly reduces its frequency of sending DNS queries. However, h_i still sends more DNS queries than normal hosts. Thus, we set τ to $\max\{\tau - 1, 1\}$ to shorten its blocking time.

Case 4: $N_i \leq (1+\gamma)N_{\texttt{avg}}$. In this case, h_i behaves just like normal hosts. Hence, we stop blocking h_i's DNS queries.

In cases 1–3, the controller reinstalls a flow entry with the new blocking time in the switch. Let us consider an example, where $T_{\texttt{base}} = 10\,\text{s}$, $\tau = 3$, $\tau_{\texttt{max}} = 6$, $f_i = 1\,\text{query/s}$, $N_{\texttt{avg}} = 10$, $\alpha = 0.8$, $\beta = 0.5$, and $\gamma = 0.1$. Then, the current blocking time is $T(\tau) = 10\,\text{s} \times 3 = 30\,\text{s}$. If h_i sends 28 DNS queries in the 30 s blocking time, since $N_i = 28 > \alpha(f_i \times T(\tau)) = 0.8(1 \times 30)$, case 1 is applied. Hence, the new blocking time is $T(\tau) = 10\,\text{s} \times \min\{3+1, 6\} = 40\,\text{s}$. On the other hand, if h_i sends 18 DNS queries, case 2 will be applied, so the new blocking time is kept to 30 s. If h_i sends 13 DNS queries, case 3 is applied. Therefore, the new blocking time is $T(\tau) = 10\,\text{s} \times \max\{3-1, 1\} = 20\,\text{s}$. Only if h_i sends no more than 11 DNS queries (i.e., $(1+\gamma)N_{\texttt{avg}} = (1+0.1) \times 10 = 11$), its subsequent DNS queries will not be blocked. Notice that in this example, the maximum blocking time is $T_{\texttt{base}} \times \tau_{\texttt{max}} = 10\,\text{s} \times 6 = 60\,\text{s}$, and the minimum blocking time is $T_{\texttt{base}} \times 1 = 10\,\text{s}$.

The adaptive blocking mechanism has two benefits. First, if a host keeps sending many DNS queries, $T(\tau)$ gradually rises. Hence, the overhead to reinstall flow entries can be cut off. Second, when the number of DNS queries reduces, $T(\tau)$ decreases. This helps ED3F recognize the termination of an attack more quickly.

TABLE I
DURATIONS OF TRAFFIC GENERATION STAGES (UNIT: SECOND).

Scenario	Stage 1	Stage 2	Stage 3
S1	0th–32nd	33rd–385th	386th–420th
S2	0th–32nd	33rd–385th	386th–420th
S3	N/A	0th–23rd	24th–28th

VI. PERFORMANCE EVALUATION

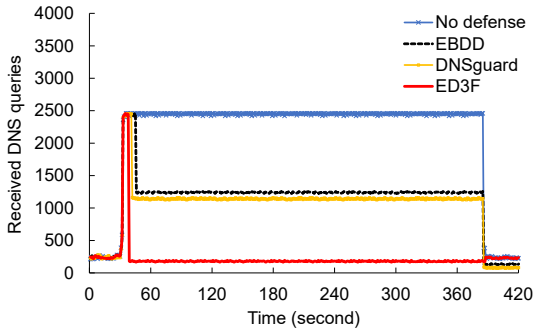
Our simulation runs on a computer with an Intel-i7 3.6 GHz CPU and 32 GB RAM. The operating system is Ubuntu, whose Linux kernel version is 4.4. The controller is implemented via Ryu [22], and Open vSwitch [23] helps create multi-layer virtual switches. To emulate hosts, we employ Docker containers [24], where a container provides an isolated environment to execute programs. DNS servers are implemented using BIND 9 [25], which can resolve domain names to IP addresses and respond to DNS queries. Pipework [26] is used to attach hosts and DNS servers (realized by containers) to switches. There are two ways to generate DNS queries. One is to produce attack and legitimate traffics using Scapy [19], as mentioned in Section V-A. The other is to use Tcprewrite and Tcpreplay [27] to inject traffics into the target network based on PCAP (packet capture) files of attack datasets, as discussed later.

Fig. 2 shows the network topology, where a target network has five switches. Switch s_1 is a gateway linking to exterior networks. Except s_1, each switch connects with three interior hosts. DNS servers A and B connect with switches s_3 and s_4. There is a controller to monitor DNS queries sent to DNS servers and identify DNS floods.

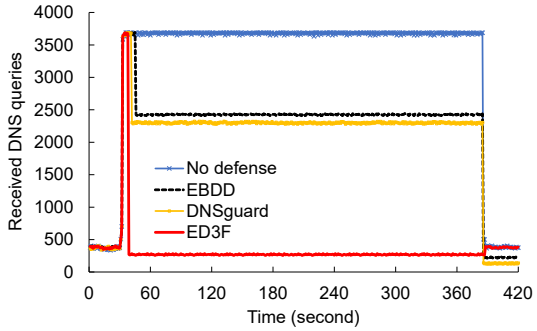
Each normal host asks a DNS server for services. Botnet members also pick a DNS server to attack. For each botnet member, its traffic generation can be divided into three stages. In stage 1, the botnet member pretends to be a normal host and sends a few DNS queries. It launches the attack in stage 2, sending many DNS queries to the victim DNS server. In stage 3, the botnet member stops attacking and returns to the same state as a normal host. Three scenarios for traffic generation and attacks are considered:

- **S1.** Hosts linked to switches s_2 and s_3 send DNS queries to server A. This scenario considers a *private* DNS server handles DNS queries only from the target network.
- **S2.** Exterior hosts and the hosts connected to switches s_4 and s_5 choose server B to send DNS queries. This scenario considers a *public* DNS server.
- **S3.** Only hosts attached to switch s_3 send DNS queries to server A. In this scenario, we use an open attack dataset.

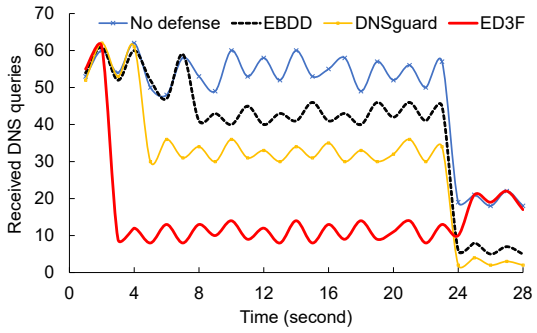
In scenarios S1 and S2, traffics are generated via Scapy. In scenario S3, we employ an open dataset for DNS floods [20]. Tcprewrite is applied to modify IP and MAC addresses (based



(a) scenario S1 (DNS server A)



(b) scenario S2 (DNS server B)



(c) scenario S3 (DNS server A)

Fig. 4. DNS queries received by a victim DNS server in each scenario.

on the topology in Fig. 2) in the dataset's PCAP file. Then, we replay traffics from the modified PCAP file using Tcpreplay. This way, we can inject attack traffics (using an open dataset) into the target network. Table I presents durations of traffic generation stages in each scenario. In scenario S3, there is no stage 1, so attacks occur at the beginning of the simulation.

We choose two methods in Section III for comparison. The *entropy-based DDoS detection (EBDD)* method [14] computes the entropy of IP addresses in packets and then compares the entropy with a threshold to identify attacks. The DNSguard method [18] has two levels. In the first level, only authorized devices (using MUD) can send DNS queries. In the second level, it monitors DNS traffic rate and mitigates DNS floods. DNSguard does not permit exterior hosts to use DNS services (as they are not MUD-compliant), so we only implement its second level for fair comparison.

A. Comparison in Scenario S1 (Private Server)

Fig. 4(a) presents the number of DNS queries received by DNS server A as time goes by. For comparison, we also show the result when no malicious DNS queries are blocked (called *no defense*). In stage 1, botnet members pretend to be normal hosts, so all methods perform the same before the 32nd second. After the 33rd second, the attack starts. Except for no defense, other methods block malicious DNS queries at different times. Specifically, EBDD, DNSguard, and ED3F begin to react (i.e., dropping DNS queries) at 13, 9, and 6 seconds after the attack is launched. Here, EBDD reacts the slowest, as it needs to collect packets for a while to estimate IP entropy. By checking the frequency of DNS queries sent to a DNS server, our ED3F scheme can react faster than DNSguard.

In stage 2 (i.e., the attacking stage), the number of DNS queries received by DNS server A per second keeps above 1,000 when using EBDD and DNSguard. This reflects that many malicious DNS queries cannot be efficiently blocked. Most malicious DNS queries are blocked by ED3F, and the number of DNS queries received by DNS server A per second is below 200. After the 386th second, we enter stage 3, and the attack ends. The number of DNS queries sent to DNS server A in ED3F is slightly higher than that in EBDD and DNSguard. Thanks to the adaptive blocking mechanism, ED3F can be quickly aware of the attack's termination, so it stops blocking DNS queries. By contrast, EBDD and DNSguard still drop some legitimate DNS queries even if there is no attack, causing false alarms.

TABLE II
DEFENSE PERFORMANCE OF EBDD, DNSGUARD, AND ED3F.

(a) scenario S1

Method	Accuracy	Precision	Recall	F1-score
EBDD	0.4100	0.4318	0.5700	0.4914
DNSguard	0.5150	0.5138	0.5600	0.5359
ED3F	0.9850	1.0000	0.9700	0.9848

(b) scenario S2

Method	Accuracy	Precision	Recall	F1-score
EBDD	0.2550	0.2906	0.3400	0.3134
DNSguard	0.3950	0.3793	0.3300	0.3529
ED3F	0.9800	1.0000	0.9600	0.9796

(c) scenario S3

Method	Accuracy	Precision	Recall	F1-score
EBDD	0.5400	0.5541	0.4100	0.4713
DNSguard	0.6050	0.6082	0.5900	0.5990
ED3F	0.9950	1.0000	0.9900	0.9950

Table II(a) shows the defense performance of each method in scenario S1. EBDD is designed for general DDoS attacks, while DNSguard takes account of DNS floods, so DNSguard performs better than EBDD. Our ED3F scheme has the highest values in each metric, which verifies its effectiveness against DNS floods in scenario S1.

B. Comparison in Scenario S2 (Public Server)

Fig. 4(b) displays the number of DNS queries received by DNS server B every second. Overall, the performance trends of all methods in Fig. 4(b) are similar to those in Fig. 4(a). The difference is that DNS server B receives far more DNS queries than DNS server A. The reason is that two and three botnet members send malicious DNS queries to DNS servers A and B. Totally, 865,063 and 1,298,774 DNS queries are generated in stage 2 of scenarios S1 and S2. As can be seen in Fig. 4(b), our ED3F scheme performs better than EBDD and DNSguard in the light of reacting to the attack, blocking malicious DNS queries, and reducing false alarms.

Table II(b) lists the defense performance of all methods in scenario S2. Compared to Table II(a), defense performance degrades as there are more DNS queries produced in scenario S2. For both EBDD and DNSguard, the value of each metric is below 0.4. On the other hand, ED3F keeps the values of all metrics above 0.96. This result shows the superiority of our ED3F scheme over others in scenario S2.

C. Comparison in Scenario S3 (Open Dataset)

Fig. 4(c) presents the number of DNS queries received by DNS server A in scenario S3. As there are fewer DNS queries produced, each method can quickly identify attacks. EBDD, DNSguard, and ED3F start blocking DNS queries at the 7th, 4th, and 2nd seconds. Compared with EBDD and DNSguard, our ED3F scheme can filter out most malicious DNS queries during the attacking stage (i.e., 0th–23rd seconds). After the 24th second, the attack ends, and the number of DNS queries received by DNS server A in ED3F and in no defense is the same. This implies that ED3F can swiftly check if an attack is over to avoid discarding legitimate DNS queries.

Table II(c) presents the defense performance of EBDD, DNSguard, and ED3F in scenario S3. Our ED3F scheme performs the best, meaning that it can efficiently resist DNS floods in this scenario (i.e., using the open attack dataset).

VII. Conclusion

DNS floods aim to consume the resources of DNS servers by sending lots of DNS queries, making web services unavailable to hosts. They cannot be well resolved using firewalls and most DDoS countermeasures. With ML and SDN, we propose the ED3F scheme to defend against DNS floods. If the frequency of sending DNS queries is high, the controller uses a trained DT model to differentiate between malicious and legitimate DNS queries. The amount of time to block DNS queries from a suspicious host is adjusted based on its frequency of sending DNS queries. Simulation results display that the ED3F scheme can quickly and accurately restrain malicious DNS queries compared to EBDD and DNSguard. For future work, we will study other performance aspects of ED3F, such as scalability and real-time reaction in a large network.

Acknowledgment

This work was supported by National Science and Technology Council, Taiwan under Grant 113-2221-E-110-056-MY3.

References

[1] 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. 31 595–31 607, 2021.

[2] Y. C. Wang and P. Y. Su, "Collaborative defense against hybrid network attacks by SDN controllers and P4 switches," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 2, pp. 1480–1495, 2024.

[3] 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 Transactions on Network Science and Engineering*, vol. 9, no. 6, pp. 4349–4362, 2022.

[4] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, "A survey on machine learning techniques for cyber security in the last decade," *IEEE Access*, vol. 8, pp. 222 310–222 354, 2020.

[5] Y. C. Wang and H. Hu, "An adaptive broadcast and multicast traffic cutting framework to improve Ethernet efficiency by SDN," *Journal of Information Science and Engineering*, vol. 35, no. 2, pp. 375–392, 2019.

[6] 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.

[7] E. Biagioni, "Preventing UDP flooding amplification attacks with weak authentication," in *International Conference on Computing, Networking and Communications*, 2019, pp. 78–82.

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

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

[10] Y. C. Wang and Y. C. Wang, "Efficient and low-cost defense against distributed denial-of-service attacks in SDN-based networks," *International Journal of Communication Systems*, vol. 33, no. 14, pp. 1–24, 2020.

[11] L. Q. Han and Y. Zhang, "PCA-based DDoS attack detection of SDN environments," in *International Conference on Big Data Analytics for Cyber-Physical-Systems*, 2020, pp. 1413–1419.

[12] 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," in *IEEE International Conference on Consumer Electronics*, 2021, pp. 1–6.

[13] C. S. Whittle and H. Liu, "Effectiveness of entropy-based DDoS prevention for software defined networks," in *IEEE International Symposium on Technologies for Homeland Security*, 2021, pp. 1–7.

[14] G. Fioravanti, M. G. Spina, and F. D. Rango, "Entropy based DDoS detection in software defined networks," in *IEEE Consumer Communications & Networking Conference*, 2023, pp. 636–639.

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

[16] H. Lotfalizadeh and D. S. Kim, "Investigating real-time entropy features of DDoS attack based on categorized partial-flows," in *International Conference on Ubiquitous Information Management and Communication*, 2020, pp. 1–6.

[17] T. Mahjabin, Y. Xiao, T. Li, and C. L. P. Chen, "Load distributed and benign-bot mitigation methods for IoT DNS flood attacks," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 986–1000, 2020.

[18] S. Datta, A. Kotha, K. Manohar, and U. Venkanna, "DNSguard: A Raspberry Pi-based DDoS mitigation on DNS server in IoT networks," *IEEE Networking Letters*, vol. 4, no. 4, pp. 212–216, 2022.

[19] Scapy. [Online]. Available: https://scapy.net/

[20] MazeBolt Knowledge Base, "DNS request flood." [Online]. Available: https://kb.mazebolt.com/knowledgebase/dns-request-flood/

[21] 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 Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.

[22] Ryu. [Online]. Available: https://ryu-sdn.org/

[23] Open vSwitch. [Online]. Available: https://www.openvswitch.org/

[24] Docker Container. [Online]. Available: https://docs.docker.com/

[25] BIND 9. [Online]. Available: https://www.isc.org/bind/

[26] Pipework. [Online]. Available: https://github.com/jpetazzo/pipework

[27] Tcpreplay. [Online]. Available: https://tcpreplay.appneta.com/