Hindawi
Security and Communication Networks
Volume 2022, Article ID 2566681, 20 pages
https://doi.org/10.1155/2022/2566681

Research Article

A Persistent Route Diversification Mechanism for Defending against Stealthy Crossfire Attack

Boyang Zhou,[1] **Chunming Wu**,[2] **Qiang Yang**,[3] **Xiang Chen**,[2] **and Dong Zhang**[4]

[1]*Intelligent Networks Institute, Zhejiang Lab, Hangzhou 311100, China*
[2]*College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*
[3]*College of Electrical Engineering, Zhejiang University, Hangzhou 310027, China*
[4]*College of Computer and Data Science, Fuzhou University, Fuzhou 350108, China*

Correspondence should be addressed to Boyang Zhou; zhouby@zhejianglab.com

Received 28 September 2022; Revised 5 December 2022; Accepted 15 December 2022; Published 31 December 2022

Academic Editor: Wenjuan Li

Computer networks are facing the challenge of stealthy crossfire attacks that flood through persistent routes (PRs) towards their decoys at a low rate for disrupting end-to-end connectivity of the target. At first, the PRs can be stealthily probed at the initial stage of the attack. Later, some undefended and vulnerable PRs can be speculated at the renaissance stage of the attack, which yet remains unconcerned. To achieve an effective defense against the two-stage attacks, this paper investigates a new persistent route diversification defense (PRDD) mechanism to mitigate each identified PR under the attacks. The PRDD effectively stops the flooding on the PRs to mitigate their congestion. Meanwhile, it makes the adversary unable to probe or speculate the PRs under their corresponding attack stages. Thus, it disables every flooding choice for the adversary, avoiding the attacks. The PRDD is designed with scalable algorithmic complexity in computation and overhead. The PRDD is extensively assessed using NS-3 and Mininet, and the results show the following. (a) It is more effective in mitigating more attacked PRs compared with the existing solutions. (b) The defense performance of the PRDD remains highly scalable in computation, while maintaining an acceptable overhead.

1. Introduction

Recently, link flooding attacks (LFAs) have become serious threats to the end-to-end data connections of computer networks. LFAs have caused large repercussions in massive system outages and heavy subsequent losses via high throughput traffic, e.g., Cloudflare attacked at 400 Gbps in 2014 [1], Dyn at 620 Gbps in 2016 [2], and Amazon in 2019 [3]. Currently, a new kind of LFAs named as the stealthy crossfire attack [4], hard to be defended, has been investigated. The attack can significantly disrupt end-to-end data connections to a target, e.g., cutting off 53% of the US Internet connections [4]. This necessitates an effective corresponding defense solution.

In the stealthy crossfire attack, the adversary commands bots to first find the persistent routes (PRs) and then flood over the PRs to a set of decoys non-adjacent to the target [4]. Here, the PR is the link non-adjacent to the target and passes through a number of forwarding paths originating from some bots to the target. Furthermore, the PRs can be found in two ways as follows: (a) probing the PRs out of the links connecting to the target via sending the Internet control message protocol (ICMP) packets [4] and (b) speculating the vulnerable PRs out of the previously obtained PRs in terms of verifying the flooding destructiveness to the target over each obtained PR. In addition, we assume that the adversary is initially unknown about locations where the PRs to be attacked next are in the network. Hence, at first, the adversary usually prefers to probe PRs at the *initial* stage of the crossfire attack and later speculate on some vulnerable PRs at the *renaissance* stage of the attack. However, at the initial attack stage, the existing defense approaches can leave PRs undefended and vulnerable due to the following reasons: (a) hardness of probing [5, 6] and flooding [7, 8] detection for

the stealthier ICMP or flooding traffic sent from the bots at a low rate, (b) non-changed routes and decoys' addresses that can be speculated for the network topology obfuscation [5, 6, 9, 10] and ICMP detection [11, 12] approaches, and (c) the impossibility in diverging overloaded traffic on the critical PRs for the route mutation [13] and traffic throttling [7, 8, 14] approaches. In addition, these approaches do not protect the vulnerable PRs from being speculated and hence have not been concerned with such a renaissance attack stage.

Currently, the software-defined networking (SDN) paradigm [15] has shown its flexibility in defending LFAs [14, 16, 17]. SDN is practical [18–22] and compatible with IP ones [23], with security assurance [24]. SDN can flexibly program the forwarding rules in the flow granularity via the OpenFlow (OF) protocol [25] and efficiently measure the host-level packet receiving rate [26].

To effectively defend against the two-stage attacks, we investigate a new SDN-based persistent route diversification defense (PRDD) mechanism using the defense idea as follows. First, the PRDD identifies all PRs under the two-stage attacks disruptive to the target via measuring the link congestion level. Then, PRDD stops the current flooding on the identified PRs to mitigate their congestion. Meanwhile, it makes the adversary unable to find the identified PRs via PR probing in the initial attack stage or PR speculation in the renaissance one. To achieve that, PRDD diversifies each potential decoy and target for all identified PRs into multiple proxies of the decoy and target. Each proxy is configured with a randomized network address, where the address makes flooding traffic discarded. In addition, with the proxies, the paths over the PRs are obfuscated for bots, which can avoid the bots to probe the PR. Meanwhile, it diverges or suppresses the overloaded traffic forwarded to the identified victim decoys by their related PRs under the renaissance one. As a result, later flooding of the PRs for the verification above is not disruptive to the target, which invalidates the PR speculation. Thus, the adversary cannot find the identified PRs and has no choice for further conducting a disruptive flooding. Hence, the two-stage attack is defended.

We evaluate the defense effectiveness and performance of PRDD using the NS-3 simulator [27] and Mininet emulator [28]. The numerical results demonstrate that PRDD can mitigate the PRs in non-congestion state for the two-stage attack. Meanwhile, the execution time and space as well as overhead of the PRDD are scalable to large networks and all meet with their theoretical estimations. Besides, PRDD is more effective in mitigating more congested PRs and reacting to more bots than the existing defense approaches.

The contributions of the paper are listed as follows:

(i) We present the PRDD defense processes against the two-stage attacks by making the adversary have no choice in launching a disruptive attack on the target

(ii) To implement the defense processes, we present the design of PRDD to efficiently generate and enforce the defense policy with a scalable complexity

(iii) A comprehensive evaluation on the PRDD is given on its defense effectiveness and performance against the two-stage attacks, with comparative studies

This paper is a substantial extension of the letter in [29] recommended by the 1st National Conference on Advanced Computing & Defense after the oral presentation.

The rest of the paper is structured as follows. Section 2 details the threat model and related work. Section 3 presents the defense concept. Section 4 gives the defense mechanism design. Section 5 presents the evaluation results. Section 6 gives conclusive remarks and future work.

2. Preliminaries

2.1. Threat Model. Figure 1 depicts the concrete steps of the stealthy crossfire attack [4] in the initial and renaissance stages with blue and yellow labels, respectively. They are detailed as follows.

2.1.1. Initial Attack Stage. This stage includes the following.

(i) *Probing Forwarding Paths*. Each bot (in the cloud) probes the paths (in the dashed arrows) to the target (in the red box). To realize that, the bot can use the ready-to-use traceroute program that sends the ICMP packets with incremental time-to-live (TTL) values [4]. Furthermore, the bot can send the ICMP packets in a stealthier manner with different TTL values in randomized intervals, e.g., by mimicking benign application behaviors using [30].

(ii) *Selecting PRs*. In the above probed PRs, the adversary first computes all of their links and the number of times that each link is included in the paths (termed as the link density). Then, it selects the links non-adjacent to the target and with a link density as high as possible as the PRs, e.g., such a link is between switches 1 and 4.

(iii) *Finding Decoys for the PRs*. For each PR, the adversary scans the subnetwork address space of the egress points of the PR, e.g., using brute force. Then, from scanned hosts, the adversary selects the corresponding (one or multiple) available decoys for each egress point. The selection is in accordance with a strategy to make sure that the decoy is connected to one of the points and non-adjacent to the target (in the blue box).

(iv) *Flooding*. For an effective attack, the adversary requires to ensure the flooding traffic of bots passing through each PR, where the destination address of the packet is that of a decoy. This means that the adversary acquires the total attack resource involving at least a decoy of each PR. In practice, the bots flood towards the decoys at low rates with valid randomized network addresses in a rolling manner (in the red arrows) [31]. Consequently, the adversary drains the bandwidth of the PRs, which disrupts their related end-to-end connections from the end hosts (end host 1 in the green box) to the target. Meanwhile, there is no flooding traffic in the target area.

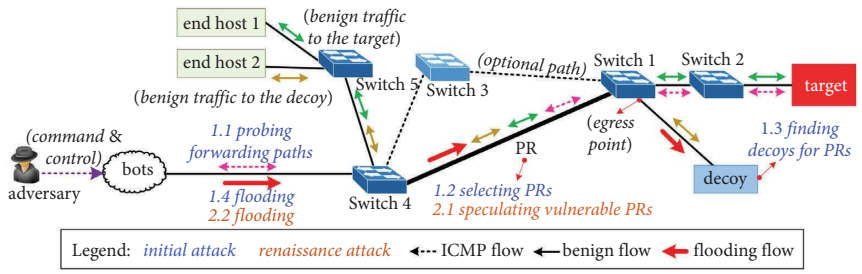


Figure 1: The two-stage stealthy crossfire attack model.

2.1.2. Renaissance Attack Stage. After the initial attack stage, the renaissance attack stage includes the following.

(i) *Speculating on Vulnerable PRs.* The adversary first obtains candidate decoys from the previous round or rescanned ones using step (iii) of the initial attack stage. Then, he commands some bots to flood towards the addresses of the candidate decoys. Later, he commands some rest bots to verify whether the connectivity from the rest bots to the target is disrupted or not, e.g., via accessing a network service of the target. If the verification is positive, the adversary confirms that the PR is vulnerable and its corresponding decoy can be exploited. For example, such a vulnerable PR is between switches 1 and 4.

(ii) *Flooding.* The botnet floods towards the verified valid decoys for attacking the speculated vulnerable PRs as in step (iv) of the initial attack stage. Such flooding can also disrupt the connections to the target. Because each speculated PR connects to the target, the renaissance attack stage can also cause certain disruption to the target. For example, if each PR under the initial attack stage is vulnerable, the renaissance one can definitely make the disruption in the same degree as the initial one. Hence, such an attack also requires an effective defense.

2.2. Related Work on Defense Approaches

2.2.1. Defense against the Initial Attack Stage. The existing defense approaches are built on the moving target defense (MTD) [32, 33] and traffic throttling as follows.

MTD. There are three categories of methods as follows.

(i) *Changing Host Locations.* Venkatesan et al. presented a proactive proxy-based moving target architecture (PROTAG) in [34] that proactively replaces the target with application-level proxies reachable to the target. Then, it shuffles hosts for access to the new proxy to isolate bots for mitigating the initial attack stage. However, the PRs related to some proxies can remain unchanged and probable that are vulnerable for the adversary, so that the adversary can still make effective flooding over the PRs via their connected proxies.

(ii) *Route Mutations.* Aydeger et al. proposed a defense mechanism [16] that detects the traceroute patterns and computes the link-disjoint path excluding the congested links. Later, Aydeger et al. proposed a strategic defense technique that identifies the illegal ICMP probing and performs the corresponding random route mutation [35]. However, the detection does not concern with the stealthier ICMP probing. Meanwhile, Xu et al. proposed an algorithm to mutate the routes related to a link or a host under the DDoS attack [36]. The mutation is taken by diverging the routes to optional ones with a context-aware Q-learning algorithm to adaptively adjust the mutation period and learning rate. Later, the route mutation was exploited in defending the DDoS attack in the vehicle ad hoc network (VANET). For example, Zhang et al. proposed a hierarchical routing scheme-based route mutation by choosing a neighbored grid in the communication area in VANET [37]. The mutation is realized by the multi-agent reinforcement learning based on the observation of nodes in the Markov process. Meanwhile, Zhang et al. proposed an intelligent defense scheme against DDoS in the software-defined Internet of vehicles in VANET [38]. The scheme periodically mutates the network communication range and access capacity of the road side units based on deep reinforcement learning. However, those solutions have insufficiency in defense against the crossfire attack as follows. (a) Defenses in VANET [37, 38] are designed based on ad hoc routes that are suitable for general networks. (b) The solutions [36–38] have not mentioned how to identify the links or hosts under attack. (c) For all of them [36–38], finding the link-disjoint path or the route mutation is impossible for the critical PRs in general networks. Later, Gupta et al. developed a defense mechanism to change interdomain paths and performed address hopping for setting up a honey pot [13]. However, the new paths may include congested PRs.

(iii) *Topology Obfuscation.* Obfuscation [39, 40] provides fake outward paths to the adversary for preventing leaks of the PRs in the probing. To realize that,

Meier et al. proposed NetHide [9] to modify ICMP probing packets with similar link delays. Ding et al. proposed Linkbait to reroute the tracing flow to bait links [6]. Hou et al. proposed to fake the topology of the detected bots using k-nearest neighbor analysis [5]. Kim et al. proposed EqualNet with the best effect to create a virtual topology with its nodes equalizing the tracing flow [10]. However, the approaches [9, 10] have not mentioned the attacked link detection which is a prerequisite in performing obfuscation. In addition, the ICMP probing flow detection [5, 6] can be inaccurate for the randomized ones. They do not change the decoy addresses, which cannot stop the flooding.

Traffic Throttling. The approach performs defense reactions in the following major ways:

(i) *Detecting Flooding Traffic*. Kang et al. proposed the SPIFFY approach [7] that detects the bot in terms of virtually increasing the PR bandwidth, which, however, is hard to identify the flooding at a consistently low rate. Later, Zheng et al. proposed an SDN-based defense (RADAR) approach to detect the attack via correlation analysis on suspicious flows and link utilization changes [8]. However, the bots can adjust their flooding rates frequently enough to undermine the sampling accuracy of flow statistics according to the Nyquist–Shannon theorem. Thus, the detection can be ineffective for such bots. They all meet the performance scalability bottlenecks for the anomaly sampling, and there is no divergence for the critical PRs.

(ii) *Detecting ICMP Traffic*. Sakuma et al. proposed to analyze the concentration degree of the traceroute packets on the links near the target [11]. However, it is hard to distinguish the sparse long-term probing behaviors. Later, Wang et al. proposed a link flooding attack defender (LFADefender) [12]. It first detects the anomaly traceroute packets via the k-nearest neighboring estimation and then blocks the detected bots or diverges the PRs. However, they are hard to cope with the randomized PR probing and critical PR.

(iii) *Traffic Diverging*. Belabed et al. proposed an SDN-based defense mechanism for the attack. The mechanism detects congested links according to the throughout increase, and then diverges the paths of the decoys and target under attack to new paths [14]. However, the mechanism does not consider the hardness in identifying the normal link loss events caused by the benign traffic sent at high throughput. Ravi et al. proposed a service-based hybrid SDN to find the location of congested links in the crossfire attack via computing the entropy of the data flow throughput features [17]. However, the later mitigation does not identify the bots with low-rate flooding flow, which cannot stop the flooding of the attack.

(iv) *Anomaly Traffic Detection Based on Programmable Switches*. These studies focus on leveraging switch abstraction in providing a flexible defense. Zhang et al. proposed Poseidon to provide simple and modular primitives to switches for defending Tbps level of attack traffic [41]. Later, Liu et al. proposed Jaqen to detect and mitigate volumetric DDoS attacks entirely in switches at high line rate by handling TCP, UDP, and ICMP attacks [42]. Meanwhile, Xing et al. proposed Ripple to provide a new abstraction to detect flow throughput through the bloom filter entirely on the switch [43]. Their programmable APIs provided can be used to support our proposed mechanism for achieving high performance.

Hence, the above approaches are insufficient in defending against the initial attack stage with the stealthier PR probing. Consequently, they can leave some undefended and vulnerable PRs for the adversary, which can be confirmed by the evaluation results in Section 5.6. The ICMP used by the adversary is necessary in discovering the maximum transmission unit for IPv6 [44], which cannot be blocked.

2.2.2. Defenses against the Renaissance Attack Stage. The existing defense solutions cannot deal with the speculation on the vulnerable PRs after defense at the initial attack stage. First, the MTD approach may leave non-changed routes involving the PRs and cannot diverge the critical PRs [13, 16, 34, 35]. In addition, the topology obfuscations [5, 6, 9, 10] do not change the underlying routes and host addresses, which makes the decoys to be reflooded. Second, the traffic throttling approaches are hard to detect randomized testing flooding at low rate [7, 8, 11, 12]. Such deficiencies make the vulnerable PRs capable to be speculated by the adversary, where the PR is reachable and destructive to the target. The PRs can be obtained from a long-term probing before congestion, which is hard to be prevented. Hence, they are not concerned with the defense against the renaissance.

3. Defense Concept

To cope with the two-stage crossfire attack, the proposed PRDD takes two different defense processes for all identified PRs under attack according to their different stages, respectively, as follows.

3.1. Defense Process in Initial Attack Stage. At the initial attack stage, the PRDD creates multiple *proxies* corresponding to each destination host over the PR at the ingress switch of the PR under attack in a scalable manner. The proxy works at the network layer in a light-weighted manner. In detail, the proxy delegates all traffic sent to or from the destination host corresponding to the proxy, where the real address of the destination host is dynamically replaced with a randomized preset address. The proxy can be

realized via the OpenFlow (OF) [25] based rules, which has been proven viable in [45]. The realization is detailed in Section 2. Figure 2 shows the defense process that includes the following steps.

(i) *Identifying the PRs under the Initial Attack Stage*. For example, such a PR is between switches 1 and 4.

(ii) *Identifying the Threatened Hosts (THs)*. The step extracts all destination hosts in the downstream of the PR under attack which we term as THs, e.g., THs 1 and 2. The THs must contain not only the target and decoys but also the benign hosts in the downstream of the PR. This is because the adversary can potentially exploit any TH except the target as a decoy for attack. Meanwhile, the defender is unaware of the target of the adversary.

(iii) *Installing Proxies*. For each TH above (e.g., TH 1 or 2), the step installs multiple proxies for the TH (e.g., see the dashed squares). Each proxy is installed at the location of the ingress switch (e.g., switch 4) of a PR reachable to the TH, where the PR has been identified under the initial stage in step (i). The method to find the location is detailed in Section 2. Furthermore, each proxy at a switch (e.g., switch 4) delegates all data packets sent between a TH corresponding to the proxy and any *non-threatened host* (NTH) via a random network address as follows. (a) The switch directly replies the ICMP request with the address of the proxy (e.g., see the blue dashed arrows). In addition, the switch discards further ICMP replies corresponding to the request, which prevents the PR from being leaked to the bots (e.g., the red cross on the pink dashed arrow). The switch decides the ICMP replies to discard by determining whether each reply is with the source address of an identified TH or not. Additionally, the switch forwards the ICMP request sent to a TH (e.g., see the pink dashed arrows) to the next hop (e.g., switch 1). Meanwhile, the switch allows to forward the ICMP replies not relating to the request, where each reply is not with its source address of an identified TH. Thus, all NTHs can only probe the paths excluding the PR. (b) The TH corresponding to a proxy actually responds any non-ICMP data packet sent to/from the address of the proxy (see the solid thin arrows).

(iv) *Restoring NTH Connections*. The step restores the end-to-end connections to the above THs. It is accomplished via making their NTHs reconnected to the addresses of the proxies corresponding to the THs installed in step (iii).

Discussion. Consequently, the switch will discard the flooding packets halfway before reaching the PRs (e.g., see the cross on the red arrow between the NTHs M and switch 4). This is because step (iii) changes the addresses of the THs to the ones of their proxies. Meanwhile, in the further initial attack stages on the same PRs, bots cannot probe any path including the PRs because the NTHs include the bots. Thus, the adversary has no choice for disruptive flooding. Hence, the PRDD defends the initial attack stage. Note that there can be vulnerable PRs left, e.g., the PR between switches 1 and 4, that can be speculated in carrying out the later renaissance attack stages. For large-scale networks, we need to concern with performance scalability and overhead issues in generating proxies for defense design.

3.2. Defense Processes in Renaissance Attack Stage. At the renaissance attack stage, the PRDD steers the high rate traffic reaching the THs of the PRs under attack to mitigate the PRs into non-congestion. Meanwhile, it updates the addresses of the proxies corresponding to the THs of their related switches. Figures 3 and 4 depict different defense processes for the renaissance attack stage in the two corresponding cases on whether there is an optional path detouring the PR or not, respectively. The two processes have similar steps described as follows.

(i) *Identifying the PRs under the Renaissance Attack Stage*. For example, such a PR is between switches 1 and 4 as shown in Figures 3 and 4.

(ii) *Identifying the Victim Decoys*. The step first extracts the THs related to the PRs similarly as in the initial attack stage above, e.g., THs 1 and 2 in both Figures 3 and 4. Then, the step identifies the *victim decoys* that are some THs above with a higher packet receiving rate. This is due to the volume accumulation effect of the flooding.

(iii) *Traffic Steering for Victim Decoys*. The steering aims at hindering the vulnerability speculation on the PRs in the stage. The solution is to make the attack unable to bring the connectivity disruption to the target via the victim decoys. To realize that, all data flows sent from any NTH to each victim decoy via a related PR under attack are steered into non-congestion states. Such steered flows include the flooding ones. The different kinds of steering are performed on the condition whether there is an optional path detouring the PR or not, respectively, as follows:

(a) *Traffic Diverging*. In Figure 3, such an optional path passes switches 4, 3, and 1 (in dashed lines). It diverges the flows from any NTH to each victim decoy over the PR to the optional path (see red and yellow arrows).

(b) *Traffic Suppressing*. In Figure 4, it suppresses the forwarding rates of the flows to the victim decoys (see the red and yellow arrows in the dashed circle).

For such two conditions, the step will mitigate the PRs into the non-congestion state if the PRs are under attack by the same bots at any time. This yields that each PR above is impeccable for flooding towards any proxy of the victim decoy connecting to the PR. Hence, the adversary will fail in speculating

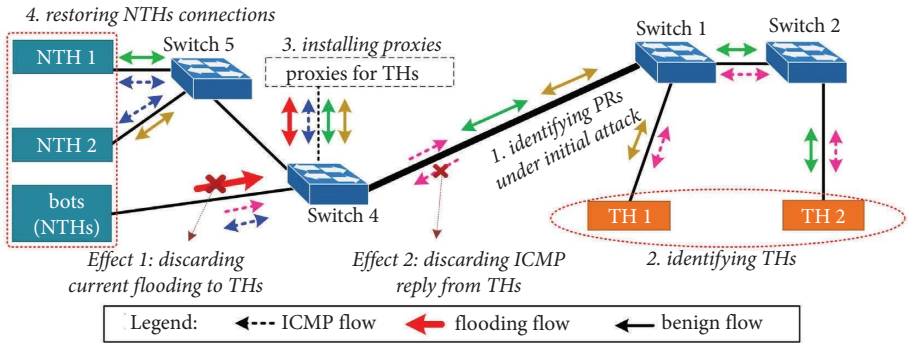


Figure 2: Defense process using the proxy creation.

on the same decoys for the above PRs in carrying out their further renaissance stage attacks.

(iv) *Updating Proxy Addresses*. The step rerandomizes all addresses of the proxies previously installed on the switches of their related forwarding paths to new ones, e.g., see the green arrows in both Figures 3 and 4. Later, the switches discard the flooding packets in the stage, e.g., between the bots and switch 4 (see the red arrows in both Figures 3 and 4). Such updating forces the ingress switches on the PRs under attack to discard the flooding packets, which mitigates the PRs into non-congestion states.

(v) *Restoring NTH Connections*. The step is accomplished in the same way as in Section 3.1.

Discussion. Eventually, step (iv) makes the adversary to speculate the PRs under attack to find the vulnerable ones for the later renaissance stage attacks on them. Meanwhile, step (iii) makes no vulnerable PRs being found and then makes no choice in launching a disruptive flooding, all for the adversary. Thus, the renaissance attack stage is defended.

Therefore, the above two defense processes can effectively cope with the initial and renaissance attack stages.

4. Defense Mechanism

To realize the aforementioned defense processes, the proposed PRDD is designed with its concrete algorithms as well as their computation complexity and overhead analysis.

4.1. Overview Design. The PRDD is designed as a control service of the SDN controller [46]. The PRDD consists of link congestion level analysis (LCLA), defense policy generation (DPG), and defense policy enforcement (DPE) components. In defense, Figure 5 depicts the collaboration of the components in the following phases. (i) The LCLA computes the congestion levels of all links by distinguishing the flooded ones from legitimate ones. (ii) The DPG first identifies all PRs under the two-stage attack based on the congestion levels. Then, it generates the defense policy involving countermeasures corresponding to the stages. (iii) The DPE enforces the policy on its related switches and then restores the end-to-end communications of the impacted NTHs via their proxy addresses. Meanwhile, we assume that there is no duplicated network address, and the switch automatically decreases the TTL value of the outgoing packet by 1 in its forwarding. The designs of the LCLA, DPG, and DPE are presented in the following sections, respectively. The common notations used in the paper are listed in Table 1.

4.2. Link Congestion Level Analysis. The LCLA determines the congestion levels of all links in the network in two phases as follows:

(i) *Link Throughput Sampling*. The LCLA samples two statistics for each link in the network at its egress and ingress switches, respectively, in a period of τ_s, as follows: (a) λ^t_{RX} is the number of receiving packets of the link at t obtained via an OFPMP_PORT or OFPMP_PORT_STATS message [25] and (b) λ^t_{TX} is the number of the packets of the data flows forwarded to the same physical ingress port of the link at t obtained in a group via an OFPMP_GROUP_STATS message [25]. Furthermore, the sampling efficiency is ensured as follows. The rules for the data flows forwarded to the same egress port point to the same group entry [25]. The entry defines the forwarding behavior and provides the statistics with four fields as follows: (a) the identifier distinguishes different groups; (b) the group type is initialized to "all"; (c) the counter indicates the number of receiving packets forwarded to the same port; and (d) the action set defines the selection of an output forwarding port via the OFPAT_OUTPUT instruction. Hence, it collects those statistics in the coarse-grained unit of the physical port of switches, which requires only a roundtrip of OF control messages per unit, i.e., in $O(1)$. Thus, the number of control messages has a low complexity of $O(|L|)$.

(ii) *Congestion Level Analysis*. With the sampled statistics, Algorithm 1 analyzes the congestion level of each link (l_i) in the following steps. First, in line 2, for a time range of $[t - \tilde{\tau}, t]$, it computes a series of the numbers of the in-flight packets forwarded at an ingress switch of the l_i as a sequence of λ_a. Similarly, in line 3, it obtains a series of the numbers of the packets received

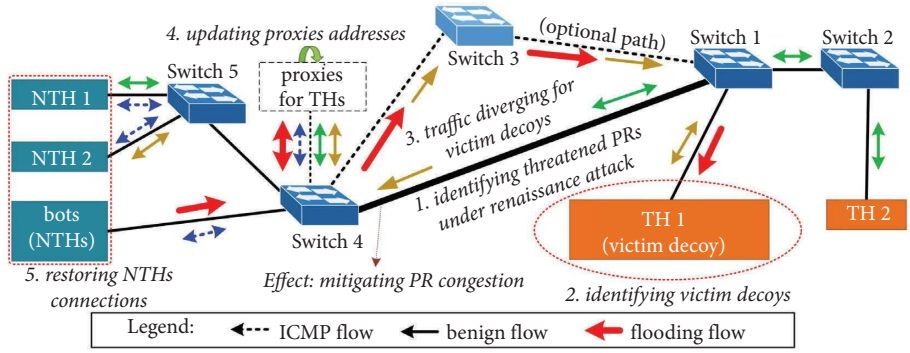


Figure 3: Defense process using traffic diverging.

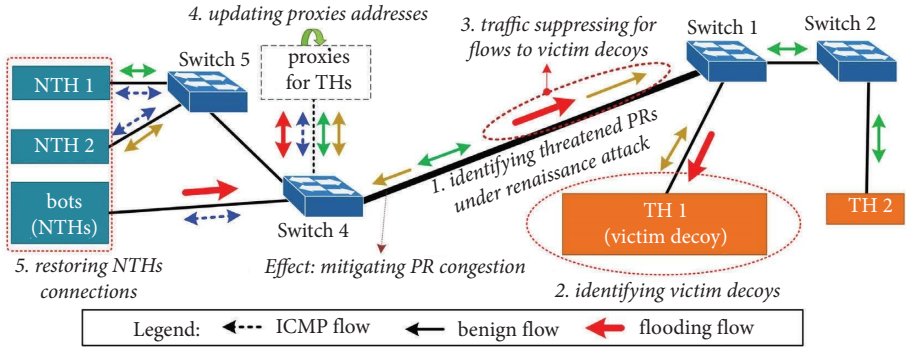


Figure 4: Defense process using traffic suppressing.

at an egress switch of l_i as λ_b. Second, it computes H_1 and H_2 as the two series of the correlation coefficients between λ_a and λ_b and the time range in line 4 and 5, respectively. Third, in line 6, it computes the correlation between H_2 and H_2 as $\tilde{\kappa}$, where $\tilde{\tau}$ is the smoothing period. Fourth, it computes the level for the two different conditions as follows.

The first condition is $\tilde{\kappa} > C_1$, i.e., λ_a is changed in accordance with λ_b in a higher degree. This implies an affluent link bandwidth. Hence, the level is 0 in line 7. In addition, the second one is when the adversary has exhausted the link bandwidth in attack. To cope with that, in line 8, it identifies a starting index within λ_a as b of which its value is suddenly changed beyond a C_2 constant proportion of $\lambda_a[1]$. Next, in line 9, it computes a right triangle with the following features. (a) Its bottom edge starts from $(b, \lambda_a[b])$ on the left and ends at $(|\lambda_a|, \lambda_a[b])$. (b) Its hypotenuse is with the minimum slope ratio of $s_{\min}$. This makes the triangle covering at least a constant proportion of C_3 of the area between the bottom edge and the line of λ_a. In line 9, it obtains $s_{\min}$ via searching among the candidate ones of which each is computed via (1). In line 10, it computes the congestion level as κ as the product of the two following parts: (a) the area proportion corresponding to $s_{\min}$, i.e., $A(b, s, \lambda_a)$, and (b) the angle percentage between the corner cut of the slope and a right one. Finally, it returns κ in line 11.

Algorithm 1 can effectively identify the flooding traffic from the legitimate one in the current Internet. It is because the values of λ_b in the second condition fluctuate around a constant. Meanwhile, λ_a for the legitimate traffic is more decurved than that for the flooding one closing to a straight line. This is due to the end-to-end congestion controls adopted in a great majority of data transports, e.g., the NewReno, LinuxReno, CUBIC, and Illinois of TCP [47], as well as QUIC [48]. Furthermore, the controls will suddenly decrease their window size upon the packet loss. In comparison, the flooding packets are without the congestion control that will continuously overload the link bandwidth and eventually lead to a constant rate. Hence, a link under heavier flooding is incorporated with higher area proportion and hypotenuse slope than a lighter one. Line 10 of Algorithm 1 can effectively obtain the congestion level.

The parameters can be turned by the operator for performance optimization according to the following principles. (a) The smaller value of τ_s yields less response time for detecting link congestion changes. (b) C_1, C_2, and C_3 are all in $(0, 1)$. In this paper, they are configured as 0.5, 0.05, and 0.99, respectively, for the tolerance of the throughput statistics fluctuations. (c) $\tilde{\tau}$ is determined for achieving a proper resolution in computing the correlations above. Furthermore, the LCLA returns κ to DPG for triggering the execution of GenPolicy to generate the defense policy.

$$A(b, s, \lambda_a) = \frac{\sum_{b+1 \le i \le |\lambda_a|} \min\left((i-b) \cdot s, \lambda_a[i]\right)}{\sum_{b+1 \le i \le |\lambda_a|} \left(\lambda_a[i] - \lambda_a[b]\right)}. \quad (1)$$

4.3. Defense Policy Generation. This section presents the defense policy generation algorithms with computation complexity and overhead analysis.

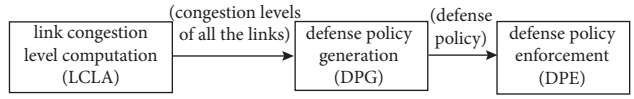


Figure 5: Component collaboration of PRDD in defense.

Table 1: Mathematical notations.

Notation	Description
G	An undirected graph of data plane, $G = (N, L)$
N	A set of all nodes in the data plane, $N = Y \cup W$
Y	A set of switches, $Y = \{y_1, y_2, \ldots, y_{\|Y\|}\}$
W	A set of end hosts, $W = \{w_1, w_2, \ldots, w_{\|W\|}\}$
L	A set of links connecting switches, $L = \{l_1, l_2, \ldots, l_{\|L\|}\}$
$p(w_r, w_s)$	The forwarding path starting from w_r to w_s that is a sequence of nodes, i.e., $(w_r, y_{j_1}, y_{j_2}, \ldots, y_{j_m}, w_s)$
$P_{\mathrm{PR}}^{\mathrm{th}}$	PRs under attack, $P_{\mathrm{PR}}^{\mathrm{th}} \subset L$
W^{th}	A set of THs, $W^{\mathrm{th}} \subset W$
τ_s	The period for link statistics sampling
$\widetilde{\tau}$	Link throughput smoothing period
ς	The minimum congestion level threshold value
η	The minimum attack time threshold on a PR identified as in the renaissance attack stage
$\nu(w_i)$	The proxy of w_i
Υ	A defense policy, denoted as $\Upsilon = (V, R, S)$, where V, R, and S are proxy creation, diverging, and suppressing countermeasures, respectively
$\kappa(l_i, t)$	The congestion level of l_i at the time tick of t
t	The time tick
ρ	The minimum proportional threshold for identifying a threatened host as a victim decoy

(1) **function** LCLA($\lambda_{\mathrm{RX}}^t, \lambda_{\mathrm{TX}}^t$)
(2) $\lambda_a = [(\lambda_{TX}^t - \lambda_{RX}^t)/\mathrm{delay}(l_i) | t - \widetilde{\tau} \le t\prime \le t]$
(3) $\lambda_b = [\lambda_{RX}^{t'} - \lambda_{RX}^{t'-1} | t - \widetilde{\tau} \le t' \le t]$
(4) $H_1 = [\mathrm{Corrcoef}_{1,2}([t' - \widetilde{\tau}, t'], \lambda_a) | t - \widetilde{\tau} \le t' \le t]$
(5) $H_2 = [\mathrm{Corrcoef}_{1,2}([t' - \widetilde{\tau}, t'], \lambda_b) | t - \widetilde{\tau} \le t' \le t]$
(6) $\widetilde{\kappa} = \mathrm{Corrcoef}_{1,2}(\mathbf{H}_1, \mathbf{H}_2)$
(7) **if** $\widetilde{\kappa} > \mathrm{C}_1$ **then return** 0
(8) $b = \mathrm{argmax}_b b$, subject to: $\bigwedge_{1 \le i \le b} (|\lambda_a[i] - \lambda_a[1]| \le C_2 \cdot \lambda_a[1]) = \mathrm{True}$
(9) $s_{\min} = \mathrm{argmin}_s A(b, s, \lambda_{\mathrm{a}})$, subject to $A \ge C_3$
(10) $\kappa = 2 \cdot A(b, s_{\min}, \lambda_{\mathrm{a}}) \cdot (\arctan(s_{\min})/\pi)$
(11) **return** κ
(12) **end function**

Algorithm 1: Link congestion level analysis.

4.3.1. Defense Policy Generation Algorithm. Algorithm 2 presents the defense policy generation algorithm with the following inputs: (a) the network topology of G, (b) a set of all forwarding paths within the current data plane of P_{all}, and (c) a map from each link to its congestion level at the current time of t, i.e., $\Omega(t) = \{l_i \longrightarrow \kappa^t | \forall l_i \in L\}$, that the LCLA has computed in Section 4.2. Its execution steps are as follows. First, in line 2 of Algorithm 2, based on $\Omega(t)$, it identifies the attack intention using AttackIntention function. Figure 6 illustrates the function workflow for identifying the components involved in such attack intention as follows:

(1) The PRs under attacks, i.e., $P_{\mathrm{PR}}^{\mathrm{th}}$, are extracted from all links using the following criteria: (a) their congestion levels no less than ς and (b) their link density no less than 2. Then, the PRs are partitioned into two subsets of Θ and Λ in the light of the number of times under attack less than η or not, respectively.

(2) The THs under the initial attack stage, i.e., W_{Θ}^{th}, are computed as the combination of each destination host of the paths passing via a link in Θ.

(3) The victim decoys under the renaissance attack stage, denoted as $W_{\Lambda}^{\mathrm{th}}$, are computed for each host in the network using the following criteria. (a) The forwarding path of the host passes through a PR in Λ. (b) The proportion between the data packet receiving rate of the host and the bandwidth of its corresponding PR is no less than ρ. Furthermore, it efficiently measures the above rate via the sketch hashing count approach provided by the OF controller itself [26]. This is because a benign host will

suddenly decrease its receiving rate by its congestion control in dealing with the continuous packet losses led by the flooding [47, 48].

(4) The NTHs, i.e., W^{no}, are computed as all destination hosts of the paths disjoint from $P^{\text{th}}_{\text{PR}}$.

Second, in lines 3 ~ and 4, Algorithm 2 calls Algorithms 3 and 4 to generate the countermeasures corresponding to the initial and renaissances stages of the attack, denoted as V and (R, S), respectively. Then, in line 5, Algorithm 2 returns the defense policy, denoted as Υ, including the above countermeasures.

Furthermore, the operator can configure the above parameters as follows: (a) the smaller values of η (no less than 2) and ς (more than 0) for increasing sensitivity and (b) ρ in $(0, 1)$ for a trade-off between accuracy (to 1) and sensitivity (to 0) in the victim decoy detection.

4.3.2. Countermeasures for Initial Attack Stage. According to the attack intention of the above Θ and W^{th}_{Θ} in the initial attack stage, Algorithm 3 generates the countermeasures for the stage. Algorithm 3 computes the proxy creation locations for each TH for minimizing switch resource usage in the following steps. In lines 2 ~ and 3, it declares a queue for conducting a BFS in the later steps. In addition, it creates two arrays with the size of $|N|$ as follows: (a) the torch records the link identifiers (IDs) for generating the proxy locations; and (b) the visited records whether a node has been iterated in a BFS for a threatened host. In lines 4 ~ 23, it iterates each proxy to identify a set of corresponding OF switches required to create the proxies for w^{th}_i in terms of $V_{w^{\text{th}}_i}$ (line 22). In detail, line 5 provides a set of links of all paths forwarding to each host in W^{th}_{Θ}, that is, L. In line 7, all elements in the torch and visited are initialized as -1 and False, respectively. In lines 8 ~ 21, it conducts a breadth-first search (BFS) for w^{th}_i. Specifically, in lines 10 ~ 20, it propagates the link IDs within all links in Θ along the trajectory from a parent node to its child node, working in torch pass. Therefore, it propagates only one of the link IDs closest to the leaf nodes among the links belonging to the same path to the leaf node. As a result, it minimizes the number of created proxies due to the maximum separation of the bots to their corresponding THs. Meanwhile, it creates these proxies at the ingress points of the PRs under attack for their THs as V that combines all $V_{w^{\text{th}}_i}$ for each w^{th}_i in W^{th}_{Θ} (see lines 22 and 24), where each item of V, i.e., $(\nu(w_i), y_j)$, indicates that the proxy of w_i is at y_j.

Proxy Implementation. Each proxy is implemented as per the following rules of a corresponding switch: (i) the rules discard any received ICMP reply sent from the TH and (ii) the rules redirect to the controller for any ICMP request sent from any NTH to the proxy. Moreover, the rules forward the ICMP reply sent from the controller to its corresponding host. (iii) The rules perform the bidirectional network address translation for the data packet sent between the TH and any NTH as follows. If any NTH sends a packet, the switch replaces its destination field with the proxy address with the real address of its corresponding TH. Otherwise, the switch replaces the source field with the TH address with the proxy address. The DPG realizes such replacement through the OFPAT_SET_FIELD instruction [25]. In addition, if the packet matches with the first or second rule, the switch will not execute the third rule. To accomplish that, the controller inserts these rules in front of the existing rules of the paths between the THs and NTHs of the switch. Thus, such implementation above is with only $O(1)$ additional process delay which can be neglected.

4.3.3. Countermeasures for Renaissance Attack Stage. According to the attack intention of above Λ, W^{th}_{Λ}, and W^{no} in the renaissance attack stage, Algorithm 4 presents the traffic steering countermeasures for the stage. In detail, Algorithm 4 takes the following steps. In line 2, it returns an empty set if there is no link in Λ. In lines 4 ~ 11, it computes R and S for all $w_i \in W^{\text{th}}_{\Lambda}$ as follows. Initially, in lines 5 ~ and 6, it computes the optional paths in $G(N, (L \setminus \Lambda))$ originating from w_i with the bandwidth and delay constrains (Ξ) as P', which is detailed in [49]. Then, it computes a set of THs required to be suppressed as W^{th}_{S} according to line 7. Each host is unreachable via any path in the P' that starts from any NTH in W^{no} to w_i. Thus, it generates S_{w_i} for each $w_j \in W^{\text{th}}_{\text{S}}$ that is reachable to w_i along with a path in P_{all} via a PR in Λ in line 8, where the Ingress (l, p) function obtains the ingress switch of l in p. Meanwhile, it generates R_{w_i} as a set of paths starting from each $w_j \in (W^{\text{no}}/W^{\text{th}}_{\text{S}})$ to w_i, and each path is different from its old one in P_{all}, in line 9. Then, it adds S_{w_i} and R_{w_i} to R and S, respectively, in line 10. Finally, it returns (R, S) in line 12, where each item of R, i.e., (p', p, ν), indicates that the above flows on p' need to be diverged to p. In implementation, it generates the new forwarding rules of p for each switch on p and then replaces the existing rules of p'. In addition, it updates the proxies corresponding to (R, S) to ν. Each item of S, i.e., (y_m, p, ν), indicates that the above flows on p need to be suppressed at y_m. In realization, it generates the meter rules for y_m that are inserted in front of all existing rules of p via the OFPIT_METER message [25]. Meanwhile, it updates the proxy relating to ν.

4.3.4. Execution Complexity of Algorithms. The complexity analysis of the GenPolicy algorithm (see Algorithm 2) in both time and space is as follows. For G, its $|L|$ is equal to $O(\log|N| \cdot |N|)$, $O(|N|)$, and $O(|N|^2)$ for the expected, best, and worst cases, respectively; suppose that $|W| = O(|N|)$. For the time complexity, in Algorithm 2, the execution time of lines 2 is $O(|W^{\text{th}}| \cdot \log|N|)$. Then, in line 3 of Algorithm 2, the execution time of Algorithm 3 is $O(|W^{\text{th}}| \cdot |N|)$ which considered the use of BFS for each $w^{\text{th}}_i \in W^{\text{th}}_{\Theta}$. In line 4 of Algorithm 2, the execution time of Algorithm 4 is $\zeta = O(|W^{\text{th}}| \cdot (|N| \cdot \log|N| + |L| + |W| + |W| \cdot \log|N|))$ with the probability $g = \Pr\{|\Lambda| \geq 1\}$ (see line 2 of Algorithm 4), where g is a coefficient that positively correlates with the attack intensity degree, that is, the number of PRs that have been defended by the proxy countermeasure and later, but, successfully speculated and flooded by a renaissance attack

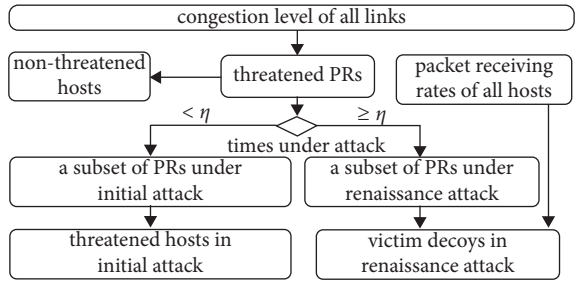


Figure 6: Workflow for identifying attack intention.

(1) **function** GENPOLICY($G, P_{\text{all}}, \Omega(t)$)
(2) $(\Theta, \Lambda, W_{\Theta}^{\text{th}}, W_{\Lambda}^{\text{th}}, W^{\text{no}}) = \text{AttackIntention}(\Omega(t))$
(3) $V = \text{Genproxies}(G, P_{\text{all}}, \Theta, W_{\Theta}^{\text{th}})$
(4) $(R, S) = \text{GenTrafficSteers}(G, P_{\text{all}}, \Lambda, W_{\Lambda}^{\text{th}}, W^{\text{no}})$
(5) **return** $\Upsilon = (V, R, S)$
(6) **end function**

Algorithm 2: Defense policy generation algorithm.

(1) **function** GENPROXIES $(G, P_{\text{all}}, \Theta, W_{\Theta}^{\text{th}})$
(2) Q = Queue, torch = $[-1]_{|N|}$
(3) visited = $[\text{False}]_{|N|}$
(4) **for** $w_i^{\text{th}} \in W_{\Theta}^{\text{th}}$ **do**
(5) $\tilde{L} = \{l_j | l_j \in p_k \wedge p_k[|p_k|] \in W_{\Theta}^{\text{th}}, \quad \forall p_k \in P_{\text{all}}\}$
(6) Q.Add(w_i^{th})
(7) torch$[1 \cdots] = -1$, visited$[1 \cdots]$ = False
(8) **while** $|Q| > 0$ **do**
(9) $n_u = Q$.Poll
(10) **for** $l_j \in \text{Edges}(n_u)$ **do**
(11) $n_v = \text{PeerNode}(n_u, l_j)$
(12) **if** $l_j \notin \tilde{L} \vee$ visited$[v]$ **then** continue
(13) **if** $l_j \in \Theta$ **then**
(14) torch$[v] = j$
(15) **else**
(16) torch$[v]$ = torch$[u]$
(17) **end if**
(18) torch$[u] = -1$
(19) Q.Add(n_v), visited$[v]$ = True
(20) **end for**
(21) **end while**
(22) $V_{w_i^{\text{th}}} = \{(v(w_i^{\text{th}}), y_j) | \text{torch}[j] \neq -1, \quad \forall y_j \in Y\}$
(23) **end for**
(24) **return** $V = \cup_{w_i^{\text{th}} \in W_{\Theta}^{\text{th}}} V_{w_j^{\text{th}}}$
(25) **end function**

Algorithm 3: Proxy creation generation algorithm.

stage. Thus, the overall execution time of Algorithm 2 is $O(|W^{\text{th}}| \cdot \log|N| + |W^{\text{th}}| \cdot |N| + g \cdot \zeta)$ that can be further simplified in different cases as follows: (a) in both the expected and best cases, $O(|W^{\text{th}}| \cdot |N| \cdot (1 + g) + |W^{\text{th}}| \cdot \log|N| \cdot (1 + g \cdot |N|))$; and (b) in the worst case, $O(|W^{\text{th}}| \cdot |N| \cdot (1 + g) + |W^{\text{th}}| \cdot \log|N| \cdot (1 + g \cdot |N|) + g \cdot |W^{\text{th}}| \cdot |N|^2)$. Thus, when only proxy countermeasure is generated, the time is $O(|W^{\text{th}}| \cdot (|N| + \log(|N|)))$ for all cases. Otherwise, the time will be increased depending on the g value. Hence, the algorithm is scalable in time.

For the space complexity, the different parts of the above algorithms consume their corresponding memory sizes, during execution, as follows: (a) Lines 2 ~ 10 of Algorithm 2 is in $O(|W^{th}| \cdot \log(|N|)|)$; (b) Algorithm 3 is in $O(|N|)$ (see lines 2 ~ 3); and (c) The line 7 of Algorithm is in $O(|N|)$ and

(1) **function** GENTRAFFICSTEERS $(G, P_{\text{all}}, \Lambda, W^{\text{th}}_{\Lambda}, W^{\text{no}})$
(2) **if** $\Lambda = \varnothing$ **then return** $\varnothing$
(3) $R = \varnothing, S = \varnothing$
(4) **for all** $w_i \in W^{\text{th}}_{\Lambda}$ **do**
(5) $G' = (Y \cup W, L \setminus \Lambda)$
(6) $P' = \text{ShortestPaths}(G', w_i, \Xi)$
(7) $W^{\text{th}}_{S} = \left\{ w_j | p(w_j, w_i) \notin P', \quad \forall w_j \in W^{\text{no}}_{\Lambda} \right\}$
(8) $S_{w_i} = \left\{ (y, p, \nu^{new}_{ran\,d}(w_i)) | y = \text{Ingress}(l, p) \wedge . p(w_j, w_i) \in P_{all} \wedge (l \in \Lambda, \exists l \in p), \quad \forall w_j \in W^{th}_{S} \right\}$
(9) $R_{w_i} = \left\{ (p', p, \nu^{\text{new}}_{\text{rand}}(w_i)) | p' \neq p \wedge p'(w_j, w_i) \in P' \wedge p(w_j, w_i) \in P_{\text{all}}, \quad \forall w_j \in (W^{\text{no}} \setminus W^{\text{th}}_{S}) \right\}$
(10) $S = S \cup S_{w_i}$, $R = R \cup R_{w_i}$
(11) **end for**
(12) **return** (R, S)
(13) **end function**

Algorithm 4: Traffic steering generation algorithm.

the lines 8∼10 to store both R and S is in $\vartheta = O(|W^{\text{th}}| \cdot (|W| + \log|N| \cdot |W|))$. Thus, for all cases, the total memory usage is $O(|W^{th} \cdot \log|N| + |N| + g \cdot \vartheta)$ that is simplified into $O(|W^{\text{th}}| \cdot \log|N| \cdot (1 + g \cdot |N|) + |N|)$, which indicates a scalable complexity.

4.3.5. Overhead Analysis in Defense Policy.

(i) *The Number of Generated Proxies.* In lines 8 ∼ 21, Algorithm 3 visits the PR using BFS on the shortest path tree (SPT) from the source of W^{th}_{Θ}. Meanwhile, it checks the reachability of each PR from W^{th}_{Θ} (see line 12). Furthermore, we present the tree as a graph of $G_{\text{tree}} = (N, L^{\text{c}})$. In addition, $|L^{\text{c}}| = |N| - 1$, and each $l'_i \in L^{\text{c}}$ is also in L with the probability of $(1/\log(|N|))$. Correspondingly, let us consider a subgraph of $G'(W^{\text{th}} \cup \epsilon(P^{\text{th}}_{\text{PR}}), L_{\text{PR}})$ of $G(N, L)$, where ϵ is a function to extract the egress node from a set of given links. In addition, each $l^{PR}_i \in L_{PR}$ is a hypothetical link if there is a subpath in $(G_{\text{tree}} \setminus P_{\text{PR}})$ directly connecting two different $\epsilon(p^{\text{PR}}_i \in P^{\text{th}}_{\text{PR}})$ with each other. Furthermore, G' is the combination of all traces of the torch passing steps following the number of links of the SPT starting from each w^{th} in lines 8 ∼ 21. Thus, the SPT in the G' equals to $|\epsilon(P^{\text{th}}_{\text{PR}})| - 2$ which is in $O(|W^{\text{th}}|)$. Hence, for the w^{th}_i in line 22, the number of generated proxies, i.e. the $|V_{w^{\text{th}}_i}|$ is in$O(|W^{\text{th}}|/\log|W^{\text{th}}|)$. It is because the probability that the algorithm selects each $w^{\text{th}}_i \in W^{\text{th}}$ as a torching passing node is equivalent to the one that the selection of a SPT link is in G'. Totally, the number of generated proxies is $O(|W^{\text{th}}|^2/\log(|W^{\text{th}}|))$. Hence, such complexity is irrelevant to the topology size, rather the number of THs, which indicates the suitableness for large-scale networks.

(ii) *The Number of Control Messages.* The number of messages for enforcing proxy creation, diverging, and suppressing countermeasures is $O(|V|)$, $O(|R| \cdot (\log|N| + 1))$, and $O(|S|)$, respectively. Furthermore, we simplify them as $O(|W^{\text{th}}|^2/\log(|W^{\text{th}}|))$, $O((1 - h) \cdot |\Lambda| \cdot (\log|N| + 1))$, and $O(h \cdot |\Lambda|)$, respectively, where h is the average probability of $P' = \varnothing$ (see line 6 of Algorithm 4), and $\Lambda = O(|W^{\text{th}}|)$. Hence, the total number of control messages is $O(|W^{\text{th}}|^2/\log(|W^{\text{th}}|) + (1 - h) \cdot \log|N| \cdot |W^{\text{th}}| + h \cdot |W^{\text{th}}|)$. Furthermore, the term relates to the attacking scale in the number of THs, that is, $O(|W^{\text{th}}|^2/\log(|W^{\text{th}}|))$ when $h = 1$. Meanwhile, it relates to both the attacking scale $(O(|W^{\text{th}}|^2/\log(|W^{\text{th}}|)))$ and also the network size in a logarithm of $\log|N|$ when $0 < h < 1$. Overall, this indicates a scalable message complexity.

4.4. Defense Policy Enforcement

4.4.1. Defense Policy Reconfiguration. First, the DPE replaces all old rules currently contained in all switches with the new ones corresponding to the rules in the policy. The replacement compares any old rule and any new one with each other according to their real source and destination addresses. Second, upon a new data flow arrival at a switch, the DPE first finds the rules in the policy with their fields matching to those of the flow. Then, the DPE reconfigures the found rules into the switches relating to the policy in a differential and on-demand manner.

4.4.2. Proxy Address Updates. According to the policy, the NTHs obtain the new proxy addresses in two following ways. (a) The active way: the DPE immediately sends a DNS update message (see IETF RFC 2136 [50]) carried with the addresses to the NTHs. Each NTH runs with an application daemon to receive the message like antivirus programs and actively delivers its addresses to its related applications. (b) The passive way: the DNS service implemented in the controller works as a transparent proxy between all hosts within the current OF data plane and the DNS server via intercepting DNS packets. If one of the hosts is installed with a proxy, the service dynamically translates the domain name of the host to its corresponding proxy address. Otherwise, the service translates the name into the real address of the

host. Thus, the NTH can request new addresses by looking up the DNS service. Finally, in either of the two ways, each NTH reconnects to its corresponding THs with its notified or requested new address.

Therefore, the above designs of PRDD realize the two defense processes for both initial and renaissance attack stages with scalable performance and acceptable overhead.

5. Performance Evaluation and Results

We evaluate the defense effectiveness, performance, and advantages of the proposed PRDD in mitigating the stealthy crossfire attack in a computer network as follows.

5.1. Experimental Settings and Parameters. We randomly synthesized the network topology with 10 ~ 2000 nodes with randomized degrees using the Erdős–Rényi model [51] for generality. Here, the nodes include all switches and all hosts. Table 2 lists its related parameters as follows: (a) the total number of switch links is twice the number of OF switches, ensuring a certain proportion of links with optional paths; (b) each switch connects to a host that can abstract the multiple hosts situation; and (c) the link transmission delay is randomly distributed in [1, 200] ms, and the link bandwidth and MTU are 10 MBps and 1500 B. Each benign host is configured with the standard NewReno TCP/IP stack [52] using the parameters in Table 2. The host sends a file with infinite length to mimic high-throughput applications.

The PRDD was comprehensively tested in the following two ways. (i) The NS-3-based discrete event simulations [27]: based on NS-3, the test environment is built with the OF switches that have been implemented by an existing work named OFSwitch13 [53]. OFSwitch13 supports the OpenFlow v1.3 protocol and allows NS-3 to dynamically interact with an external OF controller during simulations. In the environment, Figure 7 depicts the concrete experiment workflow. At first, the NS-3 parses the topology to generate its corresponding OF switches, hosts, and bots. Then, the OF controller exchanges the OF messages with the switches via the OF stub module. The stub is implemented as a transparent network layer proxy for all switches based on a tap bridge created by OFSwitch13. The advantage of the simulations is the easy tracking of the detailed testing results. (ii) The Mininet emulations [28]: the PRDD is deployed on the Floodlight OF controller v1.2 [46], due to its simplicity and support of OF v1.3 [25]. The controller interacts with switches in an out-of-band manner.

Furthermore, the effectiveness of LCLA (see Section 4.2) was tested via simulations under two different scenarios as follows. The first scenario is the current high performance transmission protocol including (a) NewReno, LinuxReno, CUBIC, and Illinois under the same configuration in Table 2 [52] and (b) the default QUIC [54]. Each benign host sends traffic in on-off at the same rate as that of the bot. The second one is the realistic Internet traffic that is synthesized with 90.04% TCP flows and 9.96% non-TCP ones closing to a CAIDA real Internet trace found in [55]. In detail, the hosts send the packets of each flow according to the Poisson Pareto burst process model using the tool mentioned in [56]. In addition, we choose NewReno for the TCP for generality. Furthermore, the model is configured with the Hurst and mean parameters of 0.7 and 0.2 s, respectively, as recommended in [56]. The model determines the traffic volume by the two parameters of the mean rate of each data flow (MRDP) as well as the mean arriving rate of the data flows (termed as *flow intensity*). To this end, the LCLA is tested for the same link in twofold. (i) The sensitivity for identifying the different intensities of flooding: the testing flows are mixed with both the Internet flows with a constant MRDP at 100 kbps and a different flow intensity ranging from 100 ~ to 500, as well as flooding sent from 0 ~ to 500 bots. The Internet traffic is generated with different MRDP and flow intensities ranging from 100 ~ to 500 kbps and 100 ~ to 500, respectively. (ii) The accuracy in identifying if there is flooding traffic in a heavily congested link: the corresponding heavy flooding is sent from 200 bots. Thus, the volume of all traffic is enough to drain the PR bandwidth.

Meanwhile, the parameters of PRDD are listed in Table 2 as follows: (a) for the two scenarios of LCLA above, the two kinds of ς are set to 0.2 and 0.8 ensuring the detection sensitivity and accuracy, i.e., ς_1 and ς_2, respectively; (b) τ_s of the LCLA is 100 ms small enough in the link statistics sampling for throughput changes; (c) $\tilde{\tau}$ of LCLA is 1 s big enough for smoothing the captured throughput changes; (d) η of DPG is 2 for the steering countermeasure; and (e) ρ of DPG is 0.9 to ensure the victim decoy detection accuracy.

5.2. Stealthy Crossfire Attack Simulations. We choose a synthesized topology above with the sizes of $|Y| = 2000$ and $|W| = 2000$ for evaluating the defense effectiveness and advantages of the PRDD. 30% of the hosts are bots, just sufficient for exhausting link bandwidth. Thus, the adversary carries out the attack in the following two stages:

(i) *A Stealthier Initial Attack Stage*. The adversary takes the following attacking steps. First, each bot sends irregular ICMP requests to the target with a randomized interval time in a uniform distribution within [1, 10] s to probe the paths to the target. Afterwards, all bots return their probed paths to the adversary. Second, he selects the top 40 PRs to the target in the link density, where the number is sufficient in causing a large-scale disruption to the network. Third, he discovers the decoys for the PRs. Fourth, he assigns the bots for flooding with their decoys on the PRs.

(ii) *An Extreme Renaissance Attack Stage*. The adversary launches the attack in an extreme case: all PRs defended in the initial attack stage and their available decoys have been successfully speculated in 0 seconds. It is used to fully test the defense effectiveness of the PRDD.

Figure 8 presents the different number of bots versus the number of their assigned PRs. The average, minimum, and maximum number of bots is 581.625, 525, and 600, respectively. These bots flood a PR in a rolling manner with the

Table 2: Experiment setup parameters.

Simulation topology and PRDD	
Parameter	Value
Bandwidth (l_i)	10 Mbps
Delay (l_i)	[1, 200] ms
η for DPG	2
ρ for DPG	0.9
$\widetilde{\tau}$ for LCLA	1 s
MTU (l_i)	1500 B
$\lvert L\rvert$	$2 \cdot \lvert Y\rvert$
ς_1 for LCLA	0.2
ς_2 for LCLA	0.8
τ_s for LCLA	100 ms
TCP stack parameter	Value
Maximum receiving buffer size (bytes)	1 MB
Maximum sending buffer size (bytes)	1 MB
Writing buffer size for emulation	1 MB
Number of packets to wait before sending a TCP ACK	2
Timeout value for TCP delayed ACKs (seconds)	0.2
Nagle algorithm	Disable

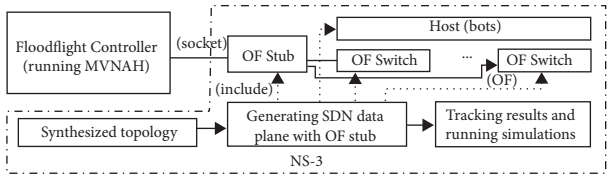


Figure 7: Experiment workflow of PRDD in NS-3.

dummy UDP packets at the low rate of 1.719 kB/s in on-off for saturating the PR bandwidth [4, 31]. Meanwhile, the normal hosts send the ICMP packets with the randomized interval time in a uniform distribution within $[0, 20]$ s to mimic the ICMP sending behavior of benign applications.

5.3. Evaluation Metrics of PRDD Solution. Thus, the evaluation of the PRDD in defending against the two-stage attack (see Section 5.2) is in the following aspects.

(i) *Defense Effectiveness*. We checked the following metrics in attacking: (a) the congestion level detected by LCLA; (b) the TCP receiving rate sharpness with the decrease of the packet losses; and (c) the CWnd size indicating the attack mitigation effect. The results are detailed in Section 5.4.

(ii) *Defense Performance*. We examined the execution time and the number of control messages of the algorithm, as well as the number of generated proxies. In addition, we compared the tested values with their theoretical estimations in Sections 4 and 5 and gave the results in Section 5.5.

(iii) *Defense Advantages*. We compared the PRDD with the existing MTD and traffic throttling approaches as follows. First, the MTD approaches hide or steer PRs, and hence the two metrics are chosen, respectively, where the higher metric is better, as follows: (a) the *disappeared PR ratio* is the division of the number of disappeared PRs after the defense to the number of congested PRs before the defense and (b) the *steered PR ratio* is the division of the number of diverged or suppressed PRs after the defense to the number of congested PRs before the defense. The comparative solutions are configured as follows. PROTAG [34] is on 1~50% proxies out of all switches according to BIND-SPLIT [34], where each proxy is randomly connected to a switch. Meanwhile, Aydeger and EqualNet accord with [10, 16], respectively. Second, the throttling approaches detect and react to the bots. Hence, we chose the metric *positive ratio*, that is, the percentage of the reacted bots in mitigating a crossfire flooding, where the higher ratio is better. The comparative approaches are configured as follows. (a) In LFADefender [12], k is set to 5 for computing the local outlier factor (LOF) in malicious flow identification [12], which is big enough to capture the density of traceroute packets. (b) In RADAR [8] and SPIFFY [7], their sampling periods are 10 s for the collection of data plane statistics. The periods are small enough to capture the dynamic changes in link throughput under attack. The results are given in Section 5.6.

5.4. Effectiveness for Defending Attack. We evaluated the defense effectiveness of the PRDD for the two-stage attack in two aspects as follows:

(i) *Congestion Level*. The LCLA is evaluated as follows. For the transmission protocol scenario, Figure 9(a) shows that the congestion level of the link attacked by 60 bots raises to 0.211 higher than ς_1. Meanwhile, the level is 0 for the benign clients in TCP and QUIC, due to their congestion control. For the realistic Internet traffic scenario, in the sensitivity, Figure 9(b) shows that the congestion levels are

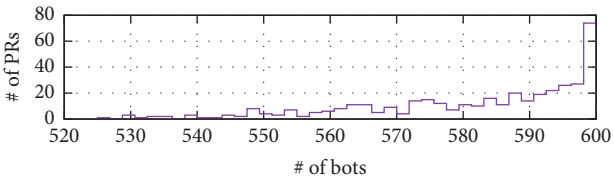


Figure 8: The number of bots assigned to PRs in $|N| = 4000$.

increased with an increase of the bots for 100 ~ 500 flow intensity. In the accuracy, with the flooding, Figure 9(c) depicts that all congestion levels are beyond ς_2 and gradually decrease with an increase of the flow intensity due to the decreased flooding proportions within all link traffic. Without the flooding, Figure 9(d) shows that all congestion levels are below ς_2 and decrease with an increase of the flow intensity due to more decreased CWnd for the benign. Such results confirm the effectiveness.

(ii) *Defense Policy*. The TCP performance between hosts and the target is tested after the defense policy enforcement as follows. At the initial attack stage, the effectiveness of the proxy creation countermeasures is tested. Figure 9(e) shows the TCP CWnd size changes at the target, when a l_i with $e(l_i) = 0$ is flooded in 5 ~ 10 s by 50 to 300 bots. Meanwhile, Figure 9(f) shows the TCP packet receiving rate changes at the target, where the TCP CWnd size and receiving rate are changed according to the congestion state of l_i. In 0 ~ 5 s, they are linearly increased and stable, respectively, without the flooding. In 5 ~ 15 s, they are significantly decreased near to zero for 50 bots due to the heavy link load rather than the attack congestion. Meanwhile, on the contrary, they are zero for 60 ~ 300 bots. The two results above confirm that $\kappa(l_i) < \varsigma$ and $\kappa(l_i) \geq \varsigma$ (see Figure 9(a)), respectively. Meanwhile, the countermeasure is enforced at 15 s for the bots no less than 60. In this case, the host receives the DNS update of the proxy of the target and reconnects to the target via the proxy. Moreover, the TCP CWnd and receiving rate are restored in normal. For 50 bots, DPE is not involved since l_i is still available on the host. Thus, with the countermeasure, the bots cannot relaunch the attack due to the invisibility of the PRs that are then in the non-congestion state. The results show the defense effectiveness at the initial attack stage.

For the renaissance one, the effectiveness of the traffic steering countermeasures is tested. Figure 9(g) shows CDF of the decrease percentage in receiving rate and corresponding CWnd size of TCP when traffic diverging is enforced or not. The results shown in Figure 9(g) are obtained when 300 bots attack on the PRs that are with the most link density. In addition, the results are calculated as the mean values in the corresponding metrics above, where each value is sampled in either the two periods as follows: (a) when TCP is stabilized in 5s, or (b) since that TCP is started in 30s. Overall, the TCP packet receiving rate is only decreased by 3.32% when its CWnd size is stabilized because the average path delay is increased by 14.12%. In addition, Figure 9(h) shows the changes in TCP receiving rate and CWnd when a critical PR is under attack from 300 bots, where traffic suppressing is enforced in 10 ~ 20 s. Furthermore, the attack suppresses the PR bandwidth to 2 Mbps. After 20 s, the bots continue flooding on the PR at a suppressed speed. Meanwhile, the TCP socket of the host is reconnected to the target for sending packets at 80% of its previous receiving rate (i.e., 20% sent by bots). Thus, the PRs between the host and the target are restored into non-congestion states.

Moreover, we evaluate the effectiveness of the defense policy via the Mininet emulation for the topology with 50 switches. Figures 9(i)–9(k) show the bandwidth and CWnd statistics measured by iperf for the proxy creation, diverging, and suppressing countermeasures, respectively. For the initial stage, with the proxies, in Figure 9(i), the flooding on a randomly selected PR since 15 s is defended at 24.3 s due to no probable PR. Later, the renaissance ones are defended as follows. (a) In Figure 9(j), the PRDD diverges the flooding on a non-critical PR under attack to an optional bypassed path. (b) In Figure 9(k), the PRDD suppresses the flooding on a critical PR at 29 s under attack at 20 s, due to suppressing. Hence, the PRDD mitigates the PRs into non-congestion, and the PRs' destructiveness cannot be evaluated. Hence, the PRDD is effective in defending such two-stage attacks.

5.5. Performance in Defense Policy Generation. Furthermore, we evaluated the execution time and overhead of the GenPolicy algorithm in the following aspects:

(i) *Execution Time*. Figure 10(a) shows the execution time of Algorithm 2 under the topology with 20 to 4000 nodes, when $g = 0$ and $g = 0.3$, respectively. The two curves are all close to their estimations in $0.0015 \cdot |N|^2 - 0.1782 \cdot |N| \cdot \log|N| + 233.7825$ and $O(0.0028 \cdot \log|N| \cdot |N|^2 - 0.0184 \cdot |N|^2 + 440.4651)$, respectively. Hence, the execution time is scalable. Consider when 30% PRs are under the renaissance. Figure 10(b) shows the average number of PRs under attack in different topology sizes for different countermeasures. On average, 75.78% of the PRs are suppressed, which demonstrates the suppressing necessity.

(ii) *Proxy Overhead*. Figure 10(c) shows the boxplot of the number of generated proxies under 10 to 400 congested PRs in the synthesized topology with $|N| = 4000$. The mean number of proxies fit the curve of $3.016 \cdot |W^{\text{th}}| \cdot \log|W^{\text{th}}| - 15.651$. The numbers are close to their estimation in Section 5. In addition, Figure 10(d) shows the total number of the proxies under different number of congested PRs. Those proxies fit the curve of $2.834 \cdot |W^{\text{th}}|^2 \cdot \log|W^{\text{th}}| - 1411.8$ which is also close to its estimation in Section 5. When the number of congested

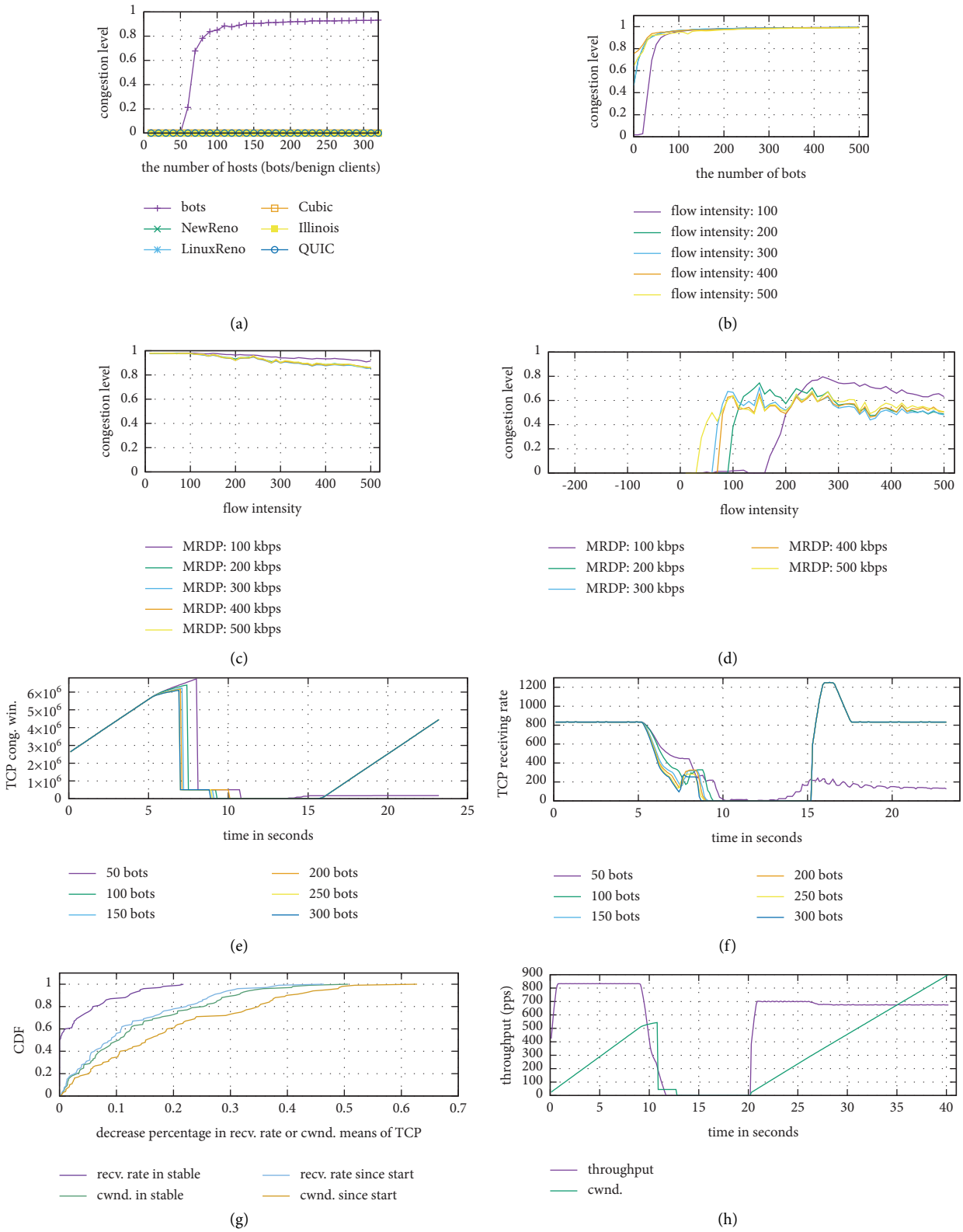


Figure 9: Continued.

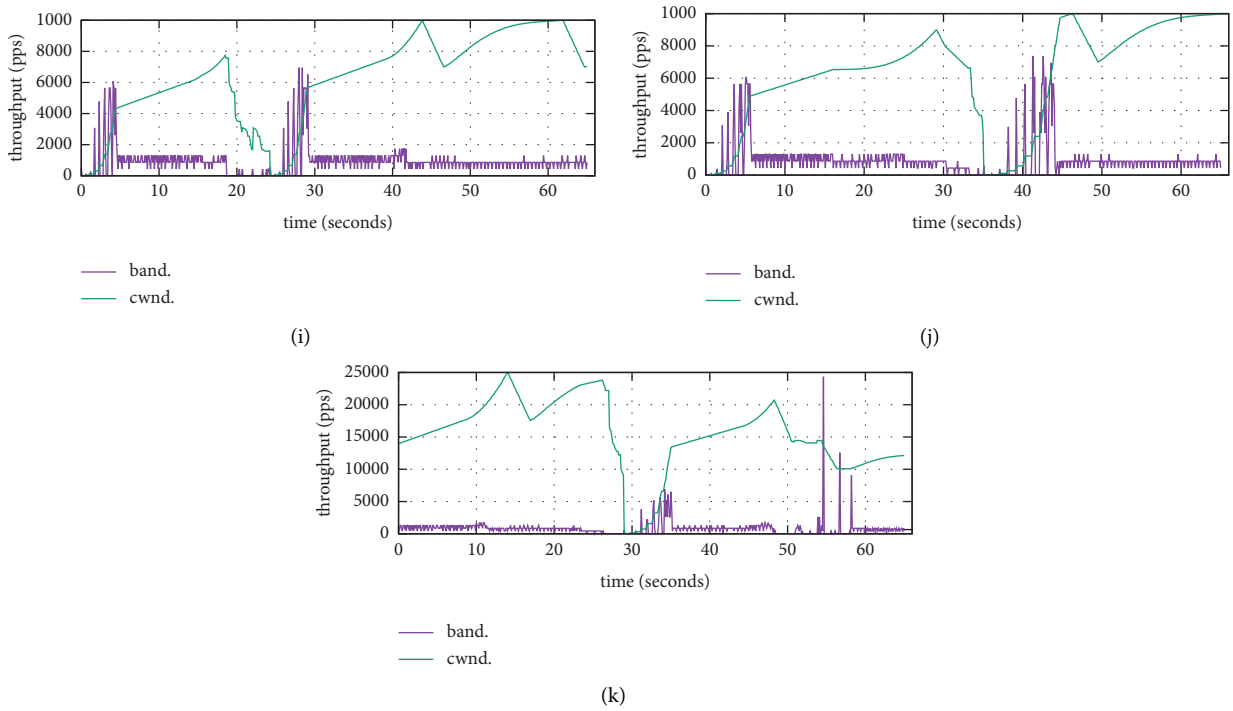


Figure 9: Effectiveness for defending crossfire attack. (a) Congestion level changes for a mixture of flooding and various transmission protocols. (b) Congestion level sensitivity for a mixture of flooding and realistic Internet traffic. (c) Congestion level accuracy for a mixture of flooding and realistic Internet traffic. (d) Congestion level accuracy for the realistic Internet traffic. (e) TCP CWnd size at the target for proxy creation. (f) TCP receiving rate at the target for proxy creation. (g) CDF of decreased TCP receiving rate and CWnd for diverging. (h) TCP receiving rate and CWnd size for suppressing. (i) TCP bandwidth and CWnd for proxies in Mininet. (j) TCP bandwidth and CWnd for diverging in Mininet. (k) TCP bandwidth and CWnd for suppressing in Mininet.

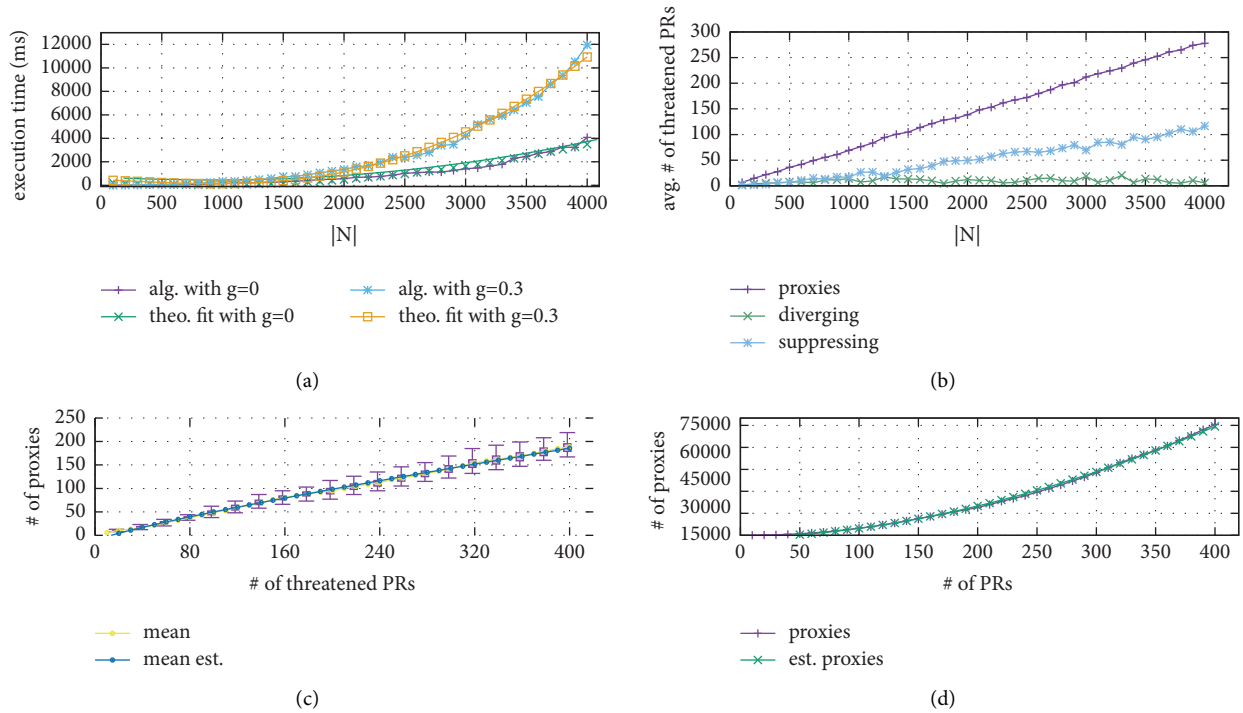
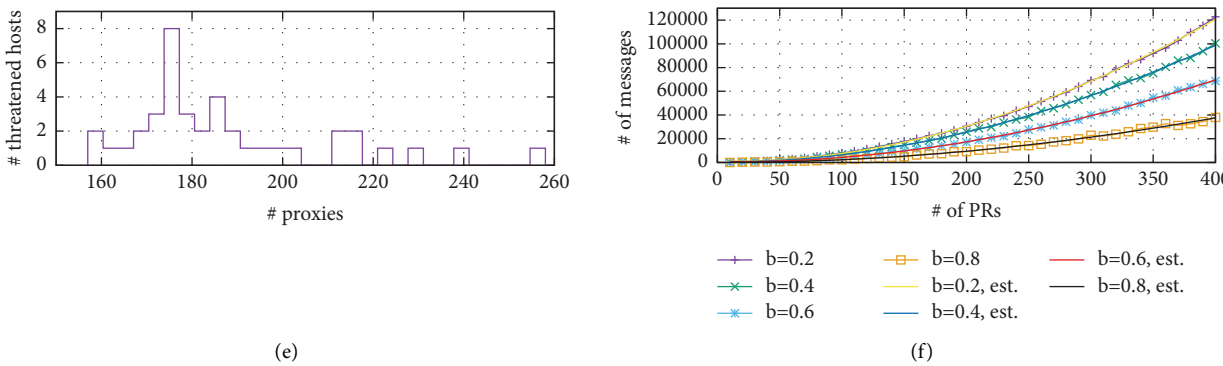


Figure 10: Continued.



(e) (f)

Figure 10: Performance in defense policy generation. (a) Execution time of GenPolicy. (b) The average number of PRs under attack when $g = 0.3$. (c) The number of proxies of each TH when $|N| = 4000$. (d) The number of total proxies when $|N| = 4000$. (e) The number of THs versus the number of their proxies, $|N| = 4000$. (f) The number of control messages in enforcing defense policy.

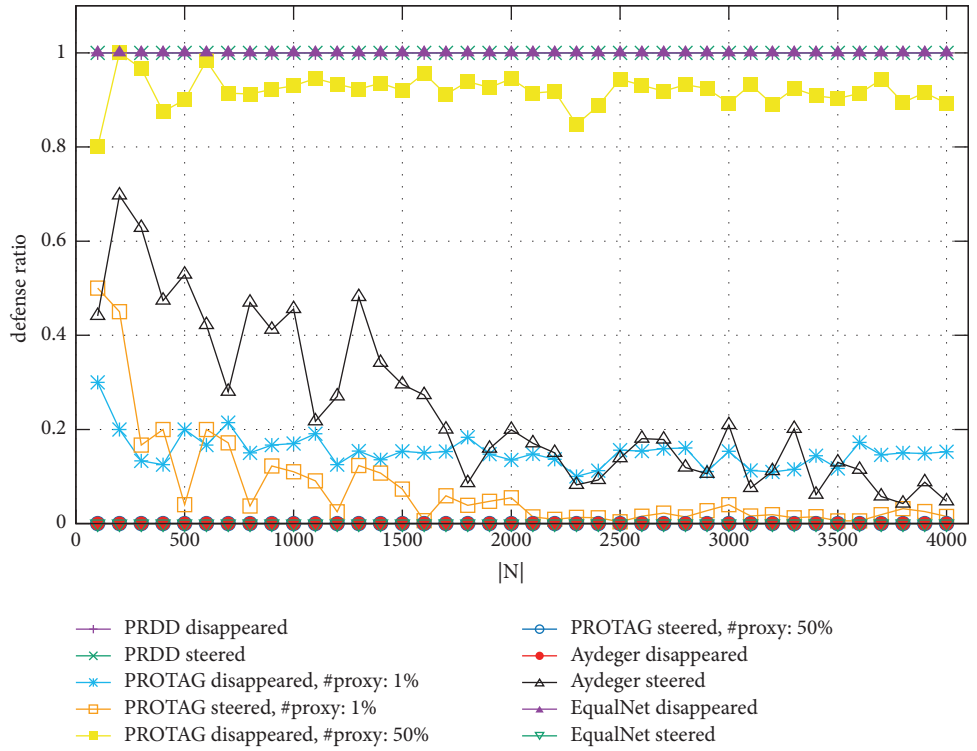


Figure 11: Comparative studies of defense effectiveness in both disappeared PRs and steered PRs ratios.

PRs is 400, Figure 10(e) shows the number of THs under different number of proxies. The mean, minimum, and maximum number of proxies is 188.075, 157, and 258, respectively. The numbers are changed in a shorter range, not exceeding the number of leaf nodes in the SPT from any proxy (see Section 5). Such results show the efficiency in the proxy resource allocation of the DPG and DPE.

(iii) *Message Overhead.* Figure 10(f) shows the number of the generated control messages under 10 to 400 congested PRs during the policy enforcement, where $b = (|\Lambda|/|\tilde{L}|)$ is the ratio of the PRs flooded no less than η. The number of the messages fit their estimation (see Section 4) all in $a_1 \cdot (|W^{\text{th}}|^2/a_2) \cdot \log(|W^{\text{th}}|) + a_3$. For the different b of 0.2, 0.4, 0.6, and 0.8, the corresponding tuples of (a_1, a_2, a_3) equal to (5.238, −51.47, 1370), (4.195, −35.19, 878.7), (3.081, −34.15, 999.3), as well as (1.633, −15.88, 361.6), respectively. These results indicate that the PRDD has a scalable message complexity.

5.6. Comparative Study. To demonstrate the advantage, we compared the PRDD with a number of existing benchmark solutions. Here, the evaluation only considers the initial attack stage, since the renaissance one remains unconcerned by the existing solutions (see Section 2). The results are detailed as follows:

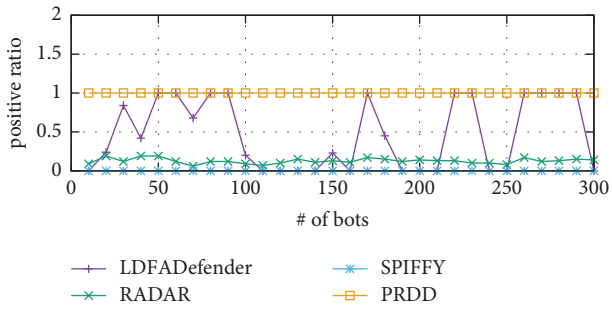


Figure 12: Comparative studies on positive ratios.

(i) *MTD-Based Approach*. Figure 11 shows the defense effect of PROTAG [34], Aydeger [16], and EqualNet [10] in the ratios of the disappeared PR and the steered PR (see Section 5.1) for different network scales. We compared those of PRDD. The results show that the number of disappeared PRs and steered PRs is equal to 100% in PRDD which is higher than that in PROTAG and Aydeger. It indicates a better attack mitigation effect. Furthermore, for PROTAG in the topology with $|N| = 4000$, when the proportion is 1% and 50%, 1.5% and 0% of PRs are diverged, respectively. It is because the target is occasionally moved to the proxies not passed via the PRs. For Aydeger, 95.3% of PRs are unable to be diverged due to their criticality. However, the PRDD can steer all PRs with created proxies to avoid their congestion.

(ii) *Traffic Throttling-Based Approach*. Figure 12 shows the defense effect of LFADefender [12], RADAR [8], and SPIFFY [7] in terms of their positive ratios (see Section 5.3), where the curve of the LFADefender fluctuates for different number of bots. It is due that the LFADefender set with the LOF can be inaccurate for its malicious flow identification. This situdation happens when ICMP packets of normal hosts arrive at switches in the burst by burst according to the on-off model. In addition, the curve of the RADAR has an average positive ratio of 12.63% since the sampling period is too long to capture the throughput changes of bots. Moreover, the positive ratios in the SPIFFY are kept in 0% because that the flooding at the stable rate from bots can escape the detection of the sketch-based measurement in the SPIFFY. However, the positive ratio values in the PRDD are 100% since detecting congested PRs in PRDD is easier than detecting the malicious flows or bots in those three solutions. Meanwhile, the DPE can mitigate each PR.

Hence, compared to the solutions mentioned above, the PRDD takes the defense advantages in both mitigating the more congested PRs and reacting to the more flooded bots..

6. Conclusion and Future Work

The proposed PRDD solution effectively defends against the two-stage stealthy crossfire attack targeting the PRs in the network. The PRDD adopts the proxy creation and traffic steering countermeasures for diversifying the identified PRs under its initial and renaissance attack stages, respectively. They can stop the current flooding and make the adversary unable to find the PRs, which defend the two-stage attacks. We comprehensively evaluate the performance of the PRDD in both NS-3 simulation and Mininet emulation. The numerical results confirm that the PRDD can effectively defend the attacks with a scalable computation cost and an acceptable overhead. Meanwhile, the PRDD has a better mitigation effect compared to the existing approaches.

In the future, PRDD can be exploited to defend against other link flooding attacks (LFAs) to consume the bandwidth of certain links or hosts. Concretely, such LFAs can be categorized into the two types. The first one is carried out with the similar ICMP-based PR selection, e.g., Coremelt attack [57]. It is because PRDD can make the PR probing and speculation ineffective. Meanwhile, the second is carried out in the way that bots directly flood towards certain end hosts in UDP with strong attack strategies. For example, in the network time protocol (NTP), an adversary can launch the amplification reflection attack for some victim hosts by exploiting the monlist vulnerability of NTP servers [58]. It is because PRDD can effectively identify PRs under flooding in UDP. Moreover, PRDD can perform various countermeasures to discard flooding traffic over PRs towards the victim hosts like decoys in the crossfire attack. Then, it enforces the steering countermeasure to stop the current flooding traffic on the victim hosts and mitigate the congestion on PRs caused by further flooding attacks. This can recover the end-to-end connectivity related to PRs. Meanwhile, for such exploitation above, the defense performance of PRDD can be optimized with consideration of the adversary's attack strategy in different selections of victim decoys.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (62102375) and the Key Research and Development Program of Zhejiang Province (2020C01021).

References

[1] B. Krebs, "The new normal: 200-400 Gbps DDoS attacks," 2022, https://krebsonsecurity.com/2014/02/the-new-normal-200-400-gbps-ddos-attacks/,%20urldate.

[2] LowEndTalk, "Dyn.com attack analysis," 2022, https://lowendtalk.com/discussion/95204/dyn-com-attack-analysis-21st-october.

[3] D. Reading, "Eight-hour DDoS attack struck AWS customers," 2019, https://www.darkreading.com/cloud/eight-hour-ddos-attack-struck-aws-customers/d/d-id/1336165.

[4] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proceedings of the IEEE Symposium on S & P*, pp. 127–141, Berkeley, CA, USA, May, 2013.

[5] T. Hou, Z. Qu, T. Wang, Z. Lu, and Y. Liu, "ProTO: proactive topology obfuscation against adversarial network topology inference," in *Proceedings of the IEEE INFOCOM*, pp. 1598–1607, Toronto, ON, Canada, July, 2020.

[6] X. Ding, F. Xiao, M. Zhou, and Z. Wang, "Active link obfuscation to thwart link-flooding attacks for Internet of things," in *Proceedings of the IEEE TrustCom*, pp. 217–224, Guangzhou, China, January, 2020.

[7] M. S. Kang, V. D. Gligor, and V. Sekar, "SPIFFY: inducing cost-detectability tradeoffs for persistent link-flooding attacks," in *Proceedings of the NDSS Symposium*, Diego, CA, USA, February, 2016.

[8] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1838–1853, 2018.

[9] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "NetHide: secure and practical network topology obfuscation," in *Proceedings of the USENIX Security Symposium*, pp. 693–709, Berkeley, CA, USA, August, 2018.

[10] J. Kim, E. Marin, M. Conti, and S. Shin, "EqualNet: a secure and practical defense for long-term network topology obfuscation," in *Proceedings of the USENIX NDSS (To Appear)*, San Diego, CA, USA, February, 2022.

[11] K. Sakuma, H. Asahina, and S. Haruta, "Traceroute-based target link flooding attack detection scheme by analyzing hop count to the destination," in *Proceedings of the Asia-Pacific Conference on Communications (APCC)*, pp. 1–6, Perth, Australia, December, 2017.

[12] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao, and F. Yu, "Detecting and mitigating target link-flooding attacks using SDN," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 944–956, 2019.

[13] A. Gupta, L. Vanbever, M. Shahbaz et al., "SDX: a software defined internet exchange," *ACM SIGCOMM - Computer Communication Review*, vol. 44, no. 4, pp. 551–562, 2014.

[14] D. Belabed, M. Bouet, and V. Conan, "Centralized defense using smart routing against link-flooding attacks," in *Proceedings of the Cyber Security in Networking Conference*, pp. 1–8, Paris, France, October, 2018.

[15] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM - Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[16] A. Aydeger, N. Saputro, and K. Akkaya, "Mitigating crossfire attacks using SDN-based moving target defense," in *Proceedings of the IEEE Conference on Local Computer Networks*, pp. 627–630, Dubai, United Arab Emirates, November, 2016.

[17] N. Ravi, S. M. Shalinie, and D. Danyson Jose Theres, "BALANCE: link flooding attack detection and mitigation via hybrid-SDN," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1715–1729, 2020.

[18] S. Tse and G. Choudhury, "Real-time Traffic Management in AT&T's SDN-Enabled Core IP/optical Network," in *Proceedings of the 2018 Optical Fiber Communications Conference and Exposition (OFC)*, San Diego, CA, USA, March, 2018.

[19] C.-Y. Hong, S. Mandal, and M. Al-Fares, "B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proceedings of the ACM SIGCOMM*, pp. 74–87, Florianopolis Brazil, August, 2018.

[20] C.-Y. Hong, S. Kandula, R. Mahajan et al., "Achieving high utilization with software-driven WAN," *ACM SIGCOMM - Computer Communication Review*, vol. 43, no. 4, pp. 15–26, 2013.

[21] M. H. Rehmani, A. Davy, B. Jennings, and C. Assi, "Software defined networks-based smart grid communication: a comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2637–2670, 2019.

[22] R. Alvizu, G. Maier, N. Kukreja et al., "Comprehensive survey on T-SDN: software-defined networking for transport networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2232–2283, 2017.

[23] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: a survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.

[24] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: a survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[25] O. N. Foundation, *OpenFlow Specification v1.3*, Open Networking Foundation Standard, Menlo Park, CA, USA, 2012.

[26] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *Proceedings of the USENIX NSDI*, pp. 29–42, Seattle, WA, USA, April, 2013.

[27] G. F. Riley and T. R. Henderson, "The NS-3 Network Simulator," *Modeling & Tools for Network Simulation*, pp. 15–34, Springer, Berlin, Germany, 2010.

[28] B. Lantz, B. Heller, and N. Mckeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the ACM HotNets*, London, UK, November 2010.

[29] B. Zhou, G. Pan, C. Wu, K. Zhu, and W. Ruan, "Multi-variant network address hopping to defend stealthy crossfire attack," *Science China Information Sciences*, vol. 63, no. 6, Article ID 169301, 2020.

[30] S. Ganesh, A. Sethi, and R. Hardy, "Lightweight scheme for generating stealthy probes," in *Proceedings of the 2009 7th International Workshop on Design of Reliable Communication Networks*, pp. 212–218, Washington, DC, USA, October, 2009.

[31] W. Zhijun, L. Wenjing, L. Liang, and Y. Meng, "Low-rate DoS attacks, detection, defense, and challenges: a survey," *IEEE Access*, vol. 8, pp. 43920–43943, 2020.

[32] R. Zhuang, S. A. Deloach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the ACM Workshop on MTD*, pp. 31–40, Seattle, WA, USA, November 2014.

[33] M. Albanese, S. Jajodia, and S. Venkatesan, "Defending from stealthy botnets using moving target defenses," *IEEE Security & Privacy*, vol. 16, no. 1, pp. 92–97, 2018.

[34] S. Venkatesan, M. Albanese, and K. Amin, "A moving target defense approach to mitigate DDoS attacks against proxy-based architectures," in *Proceedings of the 2016 IEEE Conference on Communications and Network Security*, pp. 198–206, Philadelphia, PA, USA, October, 2016.

[35] A. Aydeger, M. H. Manshaei, M. A. Rahman, and K. Akkaya, "Strategic defense against stealthy link flooding attacks: a signaling game approach," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 751–764, 2021.

[36] C. Xu, T. Zhang, X. Kuang, Z. Zhou, and S. Yu, "Context-aware adaptive route mutation scheme: a reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 528–613 541, 2021.

[37] T. Zhang, C. Xu, B. Zhang, J. Shen, X. Kuang, and L. A. Grieco, "Toward attack-resistant route mutation for VANETs: an online and adaptive multiagent reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 1–14, 2022.

[38] T. Zhang, C. Xu, P. Zou et al., "How to mitigate DDoS intelligently in SD-IoV: a moving target defense approach," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 1097–1106, 2023.

[39] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Proceedings of the 2013 IEEE Military Communications Conference*, pp. 1795–1800, San Diego, CA, USA, November, 2013.

[40] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Cyber deception: virtual networks to defend insider reconnaissance," in *Proceedings of the ACM CCS Workshop on Managing Insider Security Threats*, pp. 57–68, Dallas, TX, USA, November, 2016.

[41] M. Zhang, G. Li, S. Wang et al., "Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches," in *Proceedings of the USENIX NDSS*, San Diego, CA, USA, March, 2020.

[42] Z. Liu, H. Namkung, G. Nikolaidis et al., "A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proceedings of the USENIX Security*, Santa Clara CA USA, August 2019.

[43] J. Xing, W. Wu, and A. Chen, "Ripple: a programmable, decentralized link-flooding defense against adaptive adversaries," in *Proceedings of the USENIX Security Symposium*, San Diego, CA, USA, October, 2021.

[44] H. Robert, *IETF RFC 8200: Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force Standard, Fremont, CA, USA, 2017.

[45] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking," in *Proceedings of the ACM HotSDN*, pp. 127–132, Chicago, IL, USA, August 2012.

[46] F. Controller, "Floodlight controller," 2020, http://projectfloodlight.org/.

[47] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[48] A. Langley, J. Iyengar, and J. Bailey, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the ACM SIGCOMM*, Stowe Vermont, VT, USA, October, 2017.

[49] M. Chen, R. A. Chowdhury, and V. Ramachandran, *Priority Queues and Dijkstra's Algorithm*, Univ. of Texas, Tech. Rep, Texas, TX, USA, 2007.

[50] P. Vixie, *IETF RFC 2136: Dynamic Updates in the Domain Name System (DNS Update)*, Internet Engineering Task Force Standard, Fremont, CA, USA, 1997.

[51] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Physical Review A*, vol. 64, no. 2, Article ID 026118, 2001.

[52] M. Casoni, C. A. Grazia, and M. Klapez, "Implementation and validation of TCP options and congestion control algorithms for NS-3," in *Proceedings of the Workshop on NS-3*, pp. 112–119, Stowe Vermont, VT, USA, June, 2015.

[53] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "OFSwitch13: enhancing NS-3 with OpenFlow 1.3 support," in *Proceedings of the the 2016 Workshop on NS-3*, pp. 33–40, San Diego, CA, USA, March, 2016.

[54] A. De Biasio, F. Chiariotti, and M. Polese, "A QUIC implementation for NS-3," in *Proceedings of the Workshop on NS-3*, pp. 1–8, San Diego, CA, USA, February, 2019.

[55] Caida, "The CAIDA UCSD Anonymized Internet Traces - Equinix-chicago.dira.20160406-130000.UTC.anon," 2016, https://www.caida.org/catalog/datasets/passive_dataset.

[56] D. Ammar, T. Begin, and I. Guerin-Lassous, "A new tool for generating realistic Internet traffic in NS-3," in *Proceedings of the the International ICST Conference on Simulation Tools and Techniques*, pp. 81–83, Cannes France, March, 2011.

[57] A. Studer and A. Perrig, "The Coremelt Attack," in *European Symposium on Research in Computer Security*, pp. 37–52, Springer, Berlin, Germany, 2009.

[58] Z. Li and W. Meng, "Mind the amplification: cracking content delivery networks via DDoS attacks," in *Wireless Algorithms, Systems, and Applications*, pp. 186–197, Springer, Berlin, Germany, 2021.