On the Interplay of Link-Flooding Attacks and Traffic Engineering

Dimitrios Gkounis[1], Vasileios Kotronis[2], Christos Liaskos[3], Xenofontas Dimitropoulos[3*]
[1]NEC Laboratories Europe, Germany [2]ETH Zurich, Switzerland [3]FORTH, Greece
emails: dimitrios.gkounis@neclab.eu, vkotroni@tik.ee.ethz.ch, {cliaskos, fontas}@ics.forth.gr

ABSTRACT

Link-flooding attacks have the potential to disconnect even entire countries from the Internet. Moreover, newly proposed *indirect* link-flooding attacks, such as "Crossfire", are extremely hard to expose and, subsequently, mitigate effectively. Traffic Engineering (TE) is the network's natural way of mitigating link overload events, balancing the load and restoring connectivity. This work poses the question: *Do we need a new kind of TE to expose an attack as well?* The key idea is that a carefully crafted, attack-aware TE could force the attacker to follow improbable traffic patterns, revealing his target and his identity over time. We show that both existing and novel TE modules can efficiently expose the attack, and study the benefits of each approach. We implement defense prototypes using simulation mechanisms and evaluate them extensively on multiple real topologies.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General-security and protection; C.2.3 [**Computer Communication Networks**]: Network Operations-network management

Keywords

DDoS defense; link-flooding attack; traffic engineering.

1. INTRODUCTION

Some of the most powerful DDoS (Distributed Denial of Service) attacks ever have been observed during 2013 and 2014, reaching traffic rates greater than 300 *Gbps* [1]. Moreover, new types of indirect DDoS link-flooding attacks have been recently proposed, which are extremely difficult to mitigate [11, 21]. In particular, the *Crossfire* attack [11] (illustrated in Fig. 1) seeks to cut-off a given network area (target) from the Internet, while sending *no* attack traffic directly to the target as follows: *(i)* the attacker detects the link-map around the target by executing traceroutes towards many points within the network, *(ii)* locates critical links that connect the intended target to the Internet, *(iii)* deduces non-targeted network areas (decoys) that are also served via the critical links, and *(iv)* consumes the bandwidth of the critical links with multiple low-bandwidth flows (e.g., normal HTTP messages) originating from attacker-controlled bots and destined towards the decoys. Thus, the target loses Internet connectivity, without noticing the attacker's traffic.

*This work was funded by the European Research Council, grant EU338402.

Traffic Engineering (TE) is the expected reaction of a network on link-overload events. Regardless of their cause, TE is triggered to perform re-routing and load-balance the network traffic. Thus, there is a natural interplay between the Crossfire attack and TE. Following a TE round, the attack will pinpoint and flood new links and the cycle repeats, potentially using different decoys and bots [11]. Despite this core-placement of TE in the attack cycle, its effects on the attack exposure potential remain unknown. Nonetheless, even existing TE schemes could potentially expose the attack, without alterations. For instance, an administrator could exploit the repeated TE, and monitor network areas that are persistently affected by link-flooding events. Such areas can be marked as probable targets, gradually implying that an attack is in progress. Additionally, traffic sources that persistently react to re-routing can be monitored for the purpose of affecting a specific target. If certain sources are recorded several times in links that are DoS'ed, effectively behaving as a bot swarm, they can be marked as suspicious.

The present work studies the effect of TE on the attack exposure via analysis and experimentation. The focus is on the destination-based routing case, which remains widely adopted [20]. Our contribution is two-fold, showing that: *(i)* Crossfire attacks can be exposed in a manner agnostic to the underlying, attack-unaware TE scheme, and *(ii)* certain attack-aware TE schemes can contain the effects of the attack within a small part of the network, whereas attack-unaware TE may cause network-wide routing changes. Nonetheless, attack containment can increase the attack exposure time, while optimizing this trade-off is an interesting open problem.

2. SYSTEM MODEL

The study assumes a destination-based routed network and the Crossfire attack model [11]. The network can have multiple gateways to the Internet, while each network node represents an area that may contain one or more physical machines. The network contains a node that represents the *target* area of a bot swarm. Bots are entities with unique identifiers (e.g., machines with different IPs) that operate from beyond or within the studied network. The attack model defines a continuous cycle of interactions between the *attacker* (bot swarm) and the *defender* (network administrator) [11].

Attacker Model: The attacker seeks to cut-off the paths connecting the gateways to the target. On these paths there exist public servers, the *Decoy Servers*, and the respective *Critical Links*, which lead to both the target and the decoy servers. The attacker first constructs a map of the network links (the *link-map*) around the target, e.g., by executing

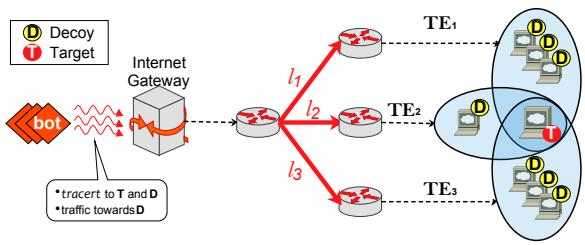

Figure 1: Attacker-Defender interaction. The attacker floods a link l_1. The defender then re-routes traffic (TE_2). The attacker updates the selected decoy servers, flooding link l_2. The defender replies with TE_3 and the attacker floods link l_3, and so on.

traceroutes towards multiple points in the network [5]. Then, he floods critical links by sending traffic only to decoy servers. Thus, all paths connecting the target to the gateway(s) are blocked due to congestion at some links, while the target sees no apparent cause, such as a high volume of direct traffic.

Defender Model: The goal of the defender is to keep the network running without any severe performance degradation (e.g., flooded links) and to expose probable targets and bots. Therefore, he: *(i)* monitors traffic load on his network and detects links that are flooded, *(ii)* balances the incoming traffic by re-routing load destined to different destinations (including the target, decoys, etc.), and *(iii)* executes actions to expose probable attackers and their target.

Attacker - Defender Interaction: The attacker monitors the network routes and reacts to routing changes performed by the defender, as exemplary shown in Fig. 1. Bots will then change their decoy server selection in case the re-routing has diverted their load from the critical link(s), repeating the cycle. Thus, the attacker updates the link-map, recalculates the critical links and floods them again with several bot-originated flows. In the example of Fig. 1, both the links and the decoys vary per cycle. The attacker may also vary the bots that participate in each attack cycle, in an effort to make their presence less persistent. In this case, the defender can reply by simply distributing traffic to the target across multiple routes, making future flooding attempts harder.

3. ANALYSIS

The defense objectives are assumed to be the following: *(i)* expose probable targets and, hence, an attack, and *(ii)* expose traffic sources which are consistently involved in it.

The exposure of probable attack targets is considered as the primary goal, and it is based on the fact that areas affected by malevolent link-floods contain the attack target. Thus, as shown in Fig. 1, if the intersection of the affected areas is not empty, then there must be an attack towards a target in this area intersection.

The exposure of bots relies on the assumption that malevolent traffic sources (bots) tend to: *(i)* change their destinations, or *(ii)* open new connections, persistently affecting a probable target (Fig. 1). On the contrary, benign causes of link-floods (e.g., flash-crowds) do not re-adjust to routing changes in such a persistent manner [24].

Nonetheless, studies have proven that bot detection approaches based on the bot reuse assumption does not guarantee detection [11, 15]. For instance, if the botnet size is vast, the attacker needs not use each bot frequently. In such cases, the defender can raise his bot detection ratio by deploying multiple, generic heuristics in parallel, such as reflector nets, white holes, bot-hunters, and rate limiters [19]. The attacker should then avoid all deployed traps on a regular basis and for every bot, which can be extremely challenging. Thus, the proposed scheme also considers the smooth collaboration with other third-party defense mechanisms in order to increase the chances of bot detection.

In light of these remarks, we operate as follows. Section 3.1 presents the preliminaries. The defense workflow is modeled and analyzed in Section 3.2, assuming independence from the underlying TE module. In Section 3.3, we study the case of attack-aware TE modules. Collaboration with third-party defense mechanisms is discussed in Section 3.4.

3.1 Notation and Assumptions

Notation. The network is denoted as $W = \{\mathcal{N}, \mathcal{L}\}$ comprising a set of nodes $\mathcal{N}$ and links $\mathcal{L}$. A single node is denoted as $n \in \mathcal{N}$ and a directed network link as $l \in \mathcal{L}$. $s(l)$ and $d(l)$ represent the source and destination nodes of l. The network has a set G of gateway nodes $g \in G$ and a persistently targeted node $target \in \mathcal{N}$. The path connecting a node n to a destination m is expressed as $\overrightarrow{p_{(n,m)}} = \{l_1, l_2, l_3, \ldots, l_k\}$, where $\{\ldots\}$ are the ordered links comprising the path, with $s(l_1) = n$, $d(l_k) = m$. The *free* bandwidth of link l is $\mathcal{B}(l)$.

Link-flooding attacks deplete the bandwidth of certain network links, using bots to send traffic over the gateways to the network. The defender deduces if a link l is flooded, i.e., $flooded(l) \rightarrow {}^{true}/_{false}$, based on existing approaches (e.g., [24]). We define $\mathbf{Src}\,\{l\}$ as the set of IP addresses, i.e., malevolent or legitimate data sources that contribute to the traffic flowing within link l. Likewise, $\mathbf{Dst}\,\{l\}$ denotes the nodes served *via* link l in the examined down-stream direction, i.e., $\mathbf{Dst}\,\{l\} = \left\{n : l \in \overrightarrow{p_{(g,n)}}\right\}$.

Routing Changes. We next define $\mathcal{R}_{TE}$ as the set that contains all routing configurations that can be deployed in W. The instance of routing rules presently adopted within the network is denoted as $r_{TE} \in \mathcal{R}_{TE}$. Migrating from r_{TE} to another configuration r'_{TE} requires the addition/removal of rules at the routing tables T_n, $n \in \mathcal{N}$, of the network nodes that route/switch traffic. In order to denote the rule modifications needed for this migration, we define the associative operator $\ominus$ on two routing instances, r_{TE} and r'_{TE}, which counts the required routing table changes as [22]:

$$r_{TE} \ominus r^*_{TE} = \sum_{\forall n \in \mathcal{N}} \left\| T_n^{r_{TE}} \cup T_n^{r'_{TE}} - T_n^{r_{TE}} \cap T_n^{r'_{TE}} \right\| \quad (1)$$

where $\cap$, $\cup$, $-$ are the set intersection, union and difference operators, and $\|*\|$ is the cardinality of a set $*$. The migration from r_{TE} to r'_{TE} is done with existing approaches [10].

Attack. The attacker first discovers the paths $\overrightarrow{p_{(g,target)}}$, $g \in G$, for the current routing tree, r_{TE}. A practical procedure based on distributed traceroutes is detailed in [5]. Then, for each path, he selects and attacks the link with the smallest amount of free bandwidth, seeking to limit the number of required bots [11]. In the *traceroutes* approach, link loads can be inferred via the corresponding delays. Increased link delay indicates high queuing delay and, therefore, high load as well. In addition, the selected link should be on the path to some minimum number of decoy nodes, D_{min} [11]. More decoys mean that the attacker can mask his traffic more effectively as legitimate, since each destination will receive less bot traffic. Thus, the critical links are selected as:

$l = argmin_{(l)} \left\{ \mathcal{B}(l) : l \in \overrightarrow{p_{(g,target)}},\ \|\mathbf{Dst}\{l\}\| \geq D_{min} \right\}$
Finally, he selects the bots with the lowest participation in past attack steps and launches the new attack.

3.2 Defense Analysis and Workflow

Let $t = 0, 1, \ldots$ denote the time moments when the network administrator notices link-flooding events. Let the flooded links at time t be l_i^t, $i = 0, 1, \ldots$. The set of nodes whose connectivity is affected by these flooded links is:

$$\mathbf{Dst}^t = \bigcup\nolimits_{\forall i} \mathbf{Dst}\left\{l_i^t\right\} \tag{2}$$

The TE module is naturally called to load-balance the traffic, relieving the congested links and restoring connectivity. In terms of attack target exposure, it would be desirable to apply a new routing tree, r_{TE}^{t+1}, that decouples each node $n \in \mathbf{Dst}^t$ from the rest of the set $\mathbf{Dst}^t$. In Fig. 1, for example, the attack at cycle 1 affects 3 decoys and the intended target. Thus, the defender sees 4 probable targets. If the subsequent TE assigned dedicated, link-disjoint paths to each of the four nodes, the attacker would not affect the 3 original decoys again. Thus, the real target would stand out. In general, the r_{TE}^{t+1} that connects each node $n \in \mathbf{Dst}^t$ to the network gateways via link-disjoint paths is:

$$r_{TE}^{t+1} : \bigcap\nolimits_{\forall n \in \mathbf{N}^t} \overrightarrow{p_{(G,n)}^{t+1}} = \emptyset \tag{3}$$

Based on condition (3), we observe:

Remark 1. *Topological path-diversity favors the exposure of Crossfire attack targets.*

Condition (3) and Remark 1 are generally aligned to the load-balancing objective of TE. Relieving flooded links means that certain nodes need to be served via decoupled paths, while path-diversity is known to favor the efficiency of TE [8, 13]. Nonetheless, in terms of attack exposure, r_{TE}^{t+1} should also decouple $\mathbf{Dst}^t$ from all past sets $\mathbf{Dst}^\tau$, $\tau = 0 \ldots t-1$:

$$r_{TE}^{t+1} : \left\| \bigcap\nolimits_{\tau=0}^{t+1} \mathbf{Dst}^\tau \right\| = 1 \tag{4}$$

Condition (4) also incorporates the concern that no past r_{TE} should be repeated at $t+1$, in order to avoid loops, which benefit the attacker. Thus, the paths available to the TE process will reduce with t, eventually hindering its load-balancing potential.

Due to this limitation, we initially study the case where TE and defense are decoupled. This constitutes the common, real-world case, where TE is executed just for load-balancing, without Crossfire-derived path restrictions or memory [8, 13]. Then, assuming sufficient path-diversity for efficient load balancing, we study the probability of a network node being coupled to the attack target *by chance*.

Lemma 1. *The probability of a non-targeted node being coupled to the target at any attack step is bounded in* $[0, 1/2]$.

Proof. See Appendix. □

Lemma 1 states that, despite the lack of interaction between the TE and defense modules, the target will still stand-out from the remaining nodes, with at least two times more appearances in attack-affected areas than the next suspect, *in the worst case*. The probability reaches zero when the attack takes place near the network leaves (see Appendix). The upper bound corresponds to attacks closer to the gateways. In both cases, however, the *cumulative* appearances of nodes within $\mathbf{Dst}^t$ favor the exposure of the target. Similarly, cumulative appearances of IPs in flooded links favor in principle the exposure of bots.

Algorithm 1 Defense Workflow against Crossfire.

```
1: On init do //Applies at initialization as well.
2:   suspect_IPs ← ∅;   //Hash[IP] → penalty.
3:   suspect_targets ← ∅;   //Hash[node] → penalty.
4:   r_TE ←any from R_TE;   //Initial TE routing rules.
5: On event flooded(l) do
6:   async_call: Penalize(Src{l}, Dst{l})   //Non-blocking call.
7:   r_TE ← r'_TE ∈ R_TE;   //Execute new TE routing.
8: Procedure Penalize(Src, Dst) //Reinforcement learning.
9:   suspect_IPs ← penalize_IPs(suspect_IPs, Src);
10:  suspect_targets ← penalize_nodes(suspect_targets, Dst);
11:  if penalization_conclusive(suspect_IPs, suspect_targets)
12:    Deduce the attack target from suspect_targets;
13:    Deduce bots using suspect_IPs;
14:    suspect_IPs ← ∅;
15:    suspect_targets ← ∅;
16:  end if
```

Based on these remarks, we define a TE-agnostic defense workflow, formulated as Algorithm 1. The workflow is event-based, defining an initialization event and a TE event. The initialization takes place once, at the defense module setup phase. Any routing instance $r_{TE} \in \mathcal{R}_{TE}$ can be active in the network at this stage. The TE event is triggered when $flooded(l)$ yields true for any network link(s), which is the common case. The TE event then calls a reinforcement learning module to update the probable bots and targets, and executes the underlying TE scheme. The reinforcement learning module is called in a non-blocking fashion, therefore causing no delay or other overhead to the TE process.

A generic template for the reinforcement learning module is outlined in lines 8−16. It requires two standard hash tables (i.e., $O(1)$ average complexity for insert, search and delete operations) for storing the needed state. The two hash tables assign "penalties" to traffic sources and to network nodes. Penalizing an IP (*penalize_IPs*) has the generic meaning of gradually collecting evidence of its malevolence, taking into account its persistence (i.e., after re-routing). On the other hand, penalizing a node (*penalize_node*) corresponds to the generic process of gradually gathering evidence of an attack being launched against it (e.g., flooding its connecting links). If a custom criterion (*penalization_conclusive*) yields that the penalization process is conclusive (e.g., penalties surpass a threshold set by the network administrator), the network administrator can enforce an attack mitigation policy of his choice. For instance, blacklisting can be applied to the deduced bots, while extra paths can be deployed between the network gateways and the attack target. Examples of instantiating the mentioned abstract functions and mitigation policies are given in Section 4.

We should note that the modeled workflow does not require any additional input parameters, apart from the ones already gathered by most network operators. Particularly, network routes, and subsequently $\mathbf{Dst}\{l\}$, are the *needed* output of any TE scheme, which is critical to the operation of any network. In addition, the monitoring of flows over each route, and subsequently $\mathbf{Src}\{l\}$, is *required* by standard defense mechanisms for commonplace attacks [9]. A trivial example is the mitigation of link-flooding due to direct attacks with

elephant flows. Moreover, the asynchronous call mode of the learning module ensures that the defense module runs in parallel to the TE objectives, without altering or obstructing its operation. Finally, the complexity of the discussed defense workflow is determined by the chosen reinforcement learning module.

3.3 Defense with Attack-aware TE

While an attack-unaware TE process allows for the exposure of the attack, potentially more can be achieved by incorporating an attack-aware TE module to the modeled workflow. Particularly, a classic, load-balancing TE will remain oblivious of the attack and overlook the continuous routing changes required to migrate from r_{TE}^{t} to r_{TE}^{t+1}. In cases of repeated and frequent migrations, such as in DDoS attacks, these changes can be expensive in terms of routing table space and disruptions caused to applications [6]. Most importantly, the attacker is inherently allowed to affect the routing of potentially the whole network. Therefore, the posed question is whether an attack-aware TE module can limit the routing changes and contain the disruptions, while still exposing the attack.

First, we employ the routing disruption metric $\mathcal{M}(t)$ [22]:

$$\mathcal{M}(t) = \sum_{\tau=1}^{t} r_{TE}^{\tau+1} \ominus r_{TE}^{\tau} \tag{5}$$

Then, the novel ReMOTE (Routing MOdification minimizing TE) limits $\mathcal{M}$ by *(i)* reusing big parts of r_{TE}^{t}, effectively retaining some memory of past routing paths, and *(ii)* affecting potential target areas only. ReMOTE is essentially a binary search for the target over several attack cycles, while ensuring that all flooded links are relieved, i.e., their loads remain below a given threshold. ReMOTE comprises two steps, executed at each cycle:

❶. Bisect the routing tree defined by node set $\mathbf{Dst}^t$ and r_{TE}^{t}, e.g. by using the process of Reed et al. [18] which has $O(\|\mathbf{Dst}^t\|)$ complexity. Thus, $\mathbf{Dst}^{t+1}$ will comprise half of the $\mathbf{Dst}^t$ set.

❷. Rehome each of the ensuing routing trees to any gateway $g \in G$, using link-disjoint paths that can support the added traffic load. This is achieved with a path finding algorithm (e.g., Dijkstra or A*) for $\overrightarrow{p_{g,d(l)}}$, sequentially for each *flooded* link l and a gateway $g \in G$.

An example of ReMOTE is given in Fig. 2, where the attacker aims to disconnect node 7. He thus sends load to other decoy nodes and tries to cut the link $l_{g,5}$, denoted in bold. The nodes in Fig. 2 are annotated with their traffic demand ratio at this point. Once the link flooding is detected, ReMOTE applies step 1, bisecting the original routing tree comprising nodes 1-9. The tree bisection process seeks to define two routing subtrees that are approximately equal both in terms of total traffic demand and number of nodes. In the context of Fig. 2, the bisection process returns the subtrees comprising nodes 1-5 and 6-9, each yielding 4 nodes and 0.5 total traffic demand.

Subsequently, step 2 of ReMOTE is executed, and the sub-tree with nodes 6-9 is routed to the gateway via a new path $g \rightarrow 6$ (dashed line, Fig. 2) which is link-disjoint to the previous, $g \rightarrow 5 \rightarrow 6$. The new path is selected so as to handle the added traffic demand of nodes 6-9 (i.e., 0.5) without causing new flooding events. At the next attack cycle, ReMOTE will focus on nodes 7-8, etc., eventually pinpointing the target.

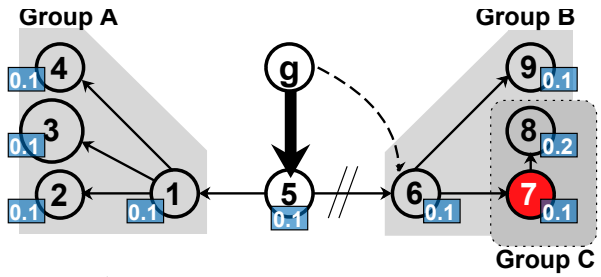

Figure 2: An example of ReMOTE in practice. The link $g, 5$ is flooded by an attacker, and ReMOTE replies by isolating nodes 1-5 from 6-9 while also balancing their annotated load. Subsequent attacks lead to the isolation of nodes (7, 8) from (6, 9), closing in on the target (7).

Finally, we collectively handle all exceptional cases, occurring, e.g., when the maximum link utilization remains high after TE, or when step 1 of ReMOTE returns empty sets. In any exceptional case, ReMOTE hands over operation to a pure load-balancing TE [13], for the current cycle only.

In the described manner, the attack is in principle contained within a progressively-shrinking node set. Likewise, the defender's actions will disrupt smaller parts of the network. As a binary search approach, ReMOTE is completed at time $\tau \approx log_2 \|\mathbf{Dst}^1\|$. However, due to step (1), certain nodes will be persistently coupled with the target. In particular, at time τ, one node (i.e., $2^1 - 1$) will have been coupled to the target $\tau - 1$ times, $2^2 - 1$ nodes $\tau - 2$ times, and so on. Thus, in order to finally differentiate between the target and suspect nodes, additional iterations are needed as follows. The defender alters the specific routing of the most probable target(s) (as of time τ), deploying node-disjoint paths per iteration. Thus, persistent couplings are broken and the target becomes more discernible.

Notice that, counter-intuitively, an attack-unaware TE will generally yield a more *confident* target detection. According to Lemma 1, at an attack cycle τ, the attack-unaware TE will have counted τ participations of the real target in the affected node sets, whereas any other node will have been observed just $\frac{\tau}{2}$ times. ReMOTE, on the other hand, will have counted $\tau - 1$ participations (out of τ) for one non-targeted node, yielding a smaller detection confidence. Thus, a possible trade-off between the two types of TE is revealed. Particularly, an attack-unaware TE may discern the attack target better than ReMOTE for the same number of iterations (i.e., faster detection). On the other hand, an attack-unaware TE can alter the routing of the complete network, as opposed to the logarithmically shrinking area-of-effect of ReMOTE. This trade-off is evaluated in Section 4.

3.4 Collaboration with other defense schemes

The effective mitigation of Crossfire requires collaboration among existing defenses, especially when countering large botnets, as mentioned in Section 3. Within an Autonomous System (AS), for example, existing solutions can force the attacker to use more bots per attack iteration, facilitating their exposure [11]. Particularly, defenses based on flow rate monitoring can force the attacker to use less traffic per bot and, hence, more bots to achieve the same impact [3]. Packet inspectors and other heuristics can also peel-off the botnet by detecting and blocking malevolent traffic sources independently. Thus, the bots at the attacker's disposal decrease, forcing him to increase the reuse rate of the remaining ones [24]. For instance, IP traceback and TTL inspectors can

detect the origins of spoofed packets [4]. Bot-Hunters can detect bots based on the similarity of their traffic patterns [19]. Phantom Nets can mislead an attacker into producing a false topology of the network, while White-Holes can disguise real network nodes as honeypots, tricking the attacker into an inefficient use of his bots [19]. Such solutions can run independently, forming a defense stack, while not being obstructed by (or obstruct) the TE module.

TE needs to trigger and work closely with inter-AS defenses, in the case where a Crossfire attack has e.g., flooded all links around the gateways. Thus, network-internal TE will be unable to load-balance the traffic, calling for external help. TE is then applied in synergy with the surrounding ASes [14]. The AS-internal defense stack can still run in parallel, removing as many bots as possible in the process.

4. EXPERIMENTAL EVALUATION

The defense workflow is implemented in the AnyLogic simulation platform [23], which offers visual, multi-paradigm programming and debugging, while automating run repetitions to achieve a user-supplied confidence level (set to 95% for the present simulations). We incorporate an attack-unaware TE [13] to Algorithm 1, to validate the analytical findings (Section 3.2). This TE is representative of the load-balancing class of algorithms. It uses path optimization driven by a Genetic Algorithm to achieve a classic min-max link load utilization. We denote this approach as GATE. We then incorporate ReMOTE to Algorithm 1, in order to evaluate the prospects of attack-aware TE (Section 3.3). The attacker model is as detailed in Section 3.1.

Setup. The simulations assume $e = 10,000$ IPs in total, which reside beyond the gateways. Out of these, B_{size} (%) are malevolent. The botnet of size ($B_{size} \cdot e$) uses B_{part} (%) randomly selected bots at each attack cycle, in order to hinder detection. At the same time, a P_{rehome} (%) fraction of the benign $(1 - B_{size}) \cdot e$ hosts randomly picks a new destination within the network, accentuating the appearance of benign link-flooding traffic as well. We use several real topologies from the TopologyZoo [12] (full list in Fig. 3), and set the nodes in each topology with the northern and southern-most geo-coordinates as gateway and target, respectively. The bandwidth of each link is set to $1\,Gbps$. Each host, benign or bot, generates up to $100\,Kbps$, indistinguishably. A link is *flooded* if its load exceeded 90% at the past timestep.

Penalization. We use the algorithm of Misra et al. for the reinforcement learning, which runs on linear complexity over the **Src**, **Dst** data structures and follows an event-counting logic [17]. A node (potential target) receives a +1 penalty each time its connectivity to the gateway is affected by a link-flooding event. Likewise, an IP (potential bot) is penalized by either +1 if it is found sending traffic over a flooded link, or by +2 if it has *also* changed its previous destination to contribute *again* to the link-flooding event. If these criteria are not met, all non-zero penalties of nodes/IPs are reduced by one. The penalization is deemed *conclusive* at timestep τ, when the set of the most penalized nodes has not changed for the last 5 iterations, for stable detection. Then, a host is classified as part of the botnet if it has accumulated more than $\tau + 1$ penalty points. The bot detection is more efficient when B_{part} is near 100%, since there will be at least one penalization per bot per iteration.

Simulation Results. Table 1 shows six evaluation metrics averaged over all topologies and simulation runs for selected parameters. We first observe that the TE choice yields the expected trade-off between detection speed/rate and the number of routing rule modifications needed (Section 3.3). The routing of GATE results in slightly fewer iterations and higher detection rates. Nonetheless, ReMOTE achieves less routing changes per node. Besides, the average bot detection rate is not significantly affected by botnet size variations (scenarios 1, 2 and 3). Scenario 1.a sets $P_{rehome} = 100\%$, accentuating the appearance of random flood events attributed to benign hosts. However, random flood events have no persistent target area in general. In contrast, malicious flooding has a persistent target and can be accurately detected. Scenarios 1.b and 1.c assume an attacker that seeks to avoid detection by decreasing the B_{part} ratio. As expected, the bot detection rate drops considerably [11]. The detection rates of the attack target are promising for both schemes, but GATE has the advantage, as explained in Section 3.3. GATE achieves lower link utilization after TE than ReMOTE, given that it constitutes its sole optimization objective.

Scenario	Parameters (%)			Link util. %	Timesteps	Mod./node	Detect rate %	False pos. %	Tgt. detect %	TE scheme*
	B size	B part	P rehome							
1	**10**	**100**	**0**	**75**	**14**	**13**	**95**	**0**	**95**	R
				65	**12**	**58**	**99**	**0**	**99**	G
1_a	10	100	100	75	14	13	95	0	95	R
				65	12	55	95	0	99	G
1_b	10	50	100	75	14	13	25	0	95	R
				65	12	59	29	0	99	G
1_c	10	10	100	76	14	13	11	0	95	R
				67	12	57	10	0	99	G
2	**50**	**100**	**0**	**72**	**14**	**11**	**98**	**0**	**95**	R
				57	**12**	**56**	**99**	**0**	**99**	G
3	**90**	**100**	**0**	**69**	**14**	**12**	**98**	**0**	**95**	R
				50	**12**	**58**	**99**	**0**	**99**	G

Table 1: Evaluation metrics *averaged over all topologies* using ReMOTE* and GATE* under different parameter settings.

On the other hand, ReMOTE requires far fewer routing changes than GATE. Fig. 3 shows boxplots of the number of modifications per node for ReMOTE and GATE. Furthermore, the variance of this number of changes is very low for ReMOTE, given that it contains the attack within rapidly shrinking network areas.

5. RELATED WORK

DDoS attacks have attracted notable research interest, particularly in recent years. Braga et al. [3] presented a low-overhead technique for traffic analysis using Self Organizing Maps (SOMs) to classify flows and enable DDoS attack detection caused by *direct* heavy hitters. Ashraf et al. [2] study several machine learning approaches for use in counter-DDoS heuristics. Hommes et al. investigate DoS attacks from the aspect of *routing table* space depletion [7]. Lim et al. propose a scheme to block botnet-based DDoS attacks that do not exhibit detectable statistical anomalies [16]. They focus on HTTP communications between clients and servers and they employ *CAPTCHA challenges* for HTTP URL redirection. Besides, Xue et al. [24] propose *LinkScope* to detect new classes of link-flooding attacks and locate critical links whenever possible. *LinkScope* employs end-to-end and hop-by-hop network measurement techniques to capture

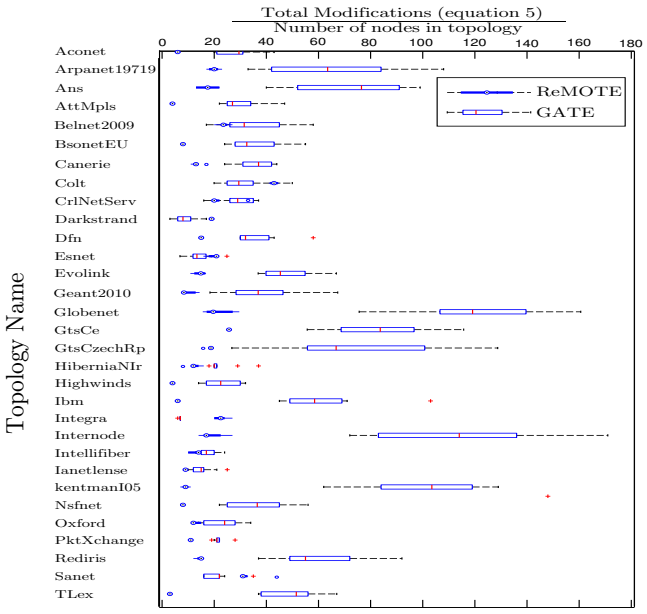

Figure 3: Boxplots of the number of rule modifications per node with ReMOTE and GATE for several real topologies.

abrupt performance degradation, while *packet inspection* is required. The work of Lee et al. (CoDef) [14] is an *inter-AS* approach towards defeating attacks such as Coremelt [21] and Crossfire [11]. CoDef proposes a cooperative method for identifying low-rate attack traffic. Nonetheless, the emphasis is on the communication among distrustful autonomous systems, and not on the role of the TE process. We refer the reader to the survey of Zargar et al. [25] for a comprehensive overview of defenses against link flooding attacks.

The present paper differentiates by studying the role of TE in Crossfire detection and mitigation. We note, however, that the presented workflow can coexist with related solutions as well (cf. Section 3.4). The authors' prior work studied the use of TE against Crossfire in networks that employ *source-based* routing and *static paths* [15]. The present work refers to *destination-based* routing and *variable paths*. In this context, it contributes a practical workflow for Crossfire mitigation, which can be used in conjunction with existing TE, as well as with novel attack-aware TE schemes.

6. CONCLUSION

The present study showed that a class of stealthy link-flooding attacks can be exposed by exploiting existing and novel TE schemes. A generic defense workflow was formulated, which can be used efficiently in conjunction even with common TE algorithms. A trade-off was indicated, showing that there exist TE schemes that contain the attack within isolated network areas, at the cost of slower exposure. Optimizing this trade-off is the objective of future work.

7. REFERENCES

[1] The DDoS That Almost Broke The Internet. `http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet`.

[2] Ashraf, J., and Latif, S. Handling Intrusion and DDoS Attacks in SDNs using Machine Learning. In *IEEE NSEC* (2014).

[3] Braga, R., et al. Lightweight DDoS Flooding Attack Detection using NOX/OpenFlow. In *IEEE LCN* (2010).

[4] Farha, A. Ip spoofing. *The Internet Protocol Jrn. 10*, 4 (2007).

[5] Gkounis, D. Cross-domain DoS Link-flooding Attack Detection and Mitigation Using SDN Principles. *MSc Th., ETH* (2014).

[6] Godfrey, P., et al. Stabilizing route selection in bgp. *IEEE/ACM Trans. on Networking 23*, 1 (2015), 282–299.

[7] Hommes, S., et al. Implications and Detection of DoS Attacks in OpenFlow-based Networks. In *IEEE GLOBECOM* (2014).

[8] Hopps, C. E. Analysis of an Equal-Cost Multi-path Algorithm. *IETF RFC 2992* (2000).

[9] Hoque, N., et al. Network attacks: Taxonomy, tools and systems. *Jrn. of Netw. and Comp. App. 40* (2014), 307–324.

[10] Jin, X., et al. Dynamic Scheduling of Network Updates. In *ACM SIGCOMM* (2014).

[11] Kang, M. S., et al. The Crossfire Attack. In *SP* (2013).

[12] Knight, S., et al. The Internet Topology Zoo. *IEEE Jrn. on Select. Areas in Comm. 29*, 9 (2011), 1765–1775.

[13] L.Buriol, S., et al. A Hybrid Genetic Algorithm for the Weight Setting Problem in OSPF/IS-IS Routing. *Networks 46*, 1 (2005), 36–56.

[14] Lee, S. B., et al. CoDef: Collaborative Defense Against Large-scale Link-flooding Attacks. In *ACM CoNEXT* (2013).

[15] Liaskos, C., et al. A Novel Framework for Modeling and Mitigating Distributed Link Flooding Attacks. In *INFOCOM (2016)*.

[16] Lim, S., et al. A SDN-oriented DDoS Blocking Scheme for Botnet-based Attacks. In *IEEE ICUFN* (2014).

[17] Misra, J., and Gries, D. Finding repeated elements. *Sc. of Comp. Prog. 2*, 2 (1982), 143–152.

[18] Reed, B., et al. Finding Disjoint Trees in Planar Graphs in Linear Time. *Contemp. Math. 147* (1993), 295–295.

[19] Shin, S., et al. Fresco: Modular composable security services for software-defined networks. In *NDSS* (2013).

[20] Singh, S., et al. A survey on internet multipath routing and provisioning. *IEEE Communications Surveys & Tutorials 17*, 4 (2015), 2157–2175.

[21] Studer, A., et al. The Coremelt Attack. In *ESORICS* (2009).

[22] Thaler, D., and Hopps, C. E. Multipath Issues in Unicast and Multicast Next-Hop Selection. *IETF RFC 2991* (2000).

[23] XJ Technologies. The AnyLogic Simulator, 2015.

[24] Xue, L., et al. Towards Detecting Target Link Flooding Attack. In *USENIX* (2014).

[25] Zargar, S., et al. A Survey of Defense Mechanisms against Distributed Denial of Service Flooding Attacks. *IEEE Comm. Surv. & Tutor. 15*, 4 (2013), 2046–2069.

Appendix

Proof of Lemma 1. Consider a physical topology and all possible routing tree choices $\mathcal{R}_{TE}$. We will assume that for every possible subset $\mathcal{S}$ of the topology nodes, there exists a link l and a routing tree $r^t_{TE} \in \mathcal{R}_{TE}$ such as $\mathbf{Dst}\{l\} = \mathcal{S}$. In other words, the attack can potentially focus on any node subset. In addition, let the attacker be able to flood links that serve k or less nodes. The variable k qualitatively expresses the severity of the attack. Attacks against network leaves affect few nodes (low k), while attacks near, e.g., a gateway affect great parts of the network (higher k).

The number of all possible subsets of $\mathcal{N}$, with size up to k, that contain one given target is $g_k = \sum_{m=1}^{k} \binom{\|\mathcal{N}\|-1}{m-1}$. Any non-targeted node is contained in such an m-sized set with probability $\frac{m-1}{\|\mathcal{N}\|-1}$. Thus, the probability of a non-targeted node being coupled to the target in groups of up to k nodes is $p_k = \frac{f_k}{g_k}$, where $f_k = \sum_{m=1}^{k} \frac{m-1}{\|\mathcal{N}\|-1}\binom{\|\mathcal{N}\|-1}{m-1} = \sum_{m=1}^{k}\binom{\|\mathcal{N}\|-2}{m-2}$.

Finally, $p_1 = 0$ and $p_{\|\mathcal{N}\|} \approx \frac{1}{2}$ due to the binomial theorem, while p_k is strictly rising:

Let $\Delta p = p_{k+1} - p_k$. Then, via finite differences we write:

$$\Delta p = \frac{\Delta f \cdot g_k - f_k \cdot \Delta g}{g_k \cdot (g_k + \Delta g)}, \Delta g = \binom{\|\mathcal{N}\|-1}{k}, \Delta f = \binom{\|\mathcal{N}\|-2}{k-1} \quad (6)$$

Notice that $\frac{\Delta g}{\Delta f} = \frac{\|\mathcal{N}\|-1}{k}$. Thus, $\Delta p \propto \Delta f \left(g_k - f_k \frac{\|\mathcal{N}\|-1}{k}\right)$. In addition, $f_k = \sum_{m=1}^{k} \frac{m-1}{\|\mathcal{N}\|-1}\binom{\|\mathcal{N}\|-1}{m-1}$. Therefore:

$\Delta p \propto \sum_{m=1}^{k} \left(\frac{k-m+1}{k}\right)\binom{\|\mathcal{N}\|-1}{m-1} > 0$, hence $p_{k+1} > p_k$, QED.■