SPIFFY: Inducing Cost-Detectability Tradeoffs for Persistent Link-Flooding Attacks

Min Suk Kang
Carnegie Mellon University
minsukkang@cmu.edu

Virgil D. Gligor
Carnegie Mellon University
gligor@cmu.edu

Vyas Sekar
Carnegie Mellon University
vsekar@andrew.cmu.edu

Abstract*—We have recently witnessed the real life demonstration of link-flooding attacks—DDoS attacks that target the core of the Internet that can cause significant damage while remaining undetected. Because these attacks use traffic patterns that are indistinguishable from legitimate TCP-like flows, they can be persistent and cause long-term traffic disruption. Existing DDoS defenses that rely on detecting flow deviations from normal TCP traffic patterns cannot work in this case. Given the low cost of launching such attacks and their indistinguishability, we argue that any countermeasure must fundamentally tackle the root cause of the problem: either force attackers to increase their costs, or barring that, force attack traffic to become distinguishable from legitimate traffic. Our key insight is that to tackle this root cause it is sufficient to perform a *rate change* test, where we temporarily increase the effective bandwidth of the bottlenecked core link and observe the response. Attacks by cost-sensitive adversaries who try to fully utilize the bots' upstream bandwidth will be detected since they will be unable to demonstrably increase throughput after bandwidth expansion. Alternatively, adversaries are forced to increase costs by having to mimic legitimate clients' traffic patterns to avoid detection. We design a software-defined network (SDN) based system called SPIFFY that addresses key practical challenges in turning this high-level idea into a concrete defense mechanism, and provide a practical solution to force a tradeoff between cost vs. detectability for link-flooding attacks. We develop fast traffic-engineering algorithms to achieve effective bandwidth expansion and suggest scalable monitoring algorithms for tracking the change in traffic-source behaviors. We demonstrate the effectiveness of SPIFFY using a real SDN testbed and large-scale packet-level and flow-level simulations.

I. INTRODUCTION

Over the last few years, *link-flooding* DDoS attacks have been proposed that can cause substantial damage to the core of the Internet [51] [31]. Unlike traditional DDoS attacks that exhaust the resources (access bandwidth or computation) of the end targets, these DDoS attacks target the connectivity infrastructure of the targets. These attacks utilize distributed botnets to create a large number of low-rate attack flows that traverse a set of chosen network links. Such flows can cause severe congestion at the targeted links and ultimately significantly degrade the connectivity of target hosts or servers. These attacks have quickly moved from the realm of academic curiosity [31], [51] to real-world incidents [28], [17].

NDSS '16, 21-24 February 2016, San Diego, CA, USA
Copyright 2016 Internet Society, ISBN 1-891562-41-X
http://dx.doi.org/10.14722/ndss.2016.23147

Link-flooding attacks have two specific characteristics that make them exceedingly effective at scale while rendering traditional defense mechanisms irrelevant. First, they attack targets *indirectly*. As the locus of attack is different from the targeted end servers, they cannot be easily detected by intrusion-detection systems and firewalls at the end servers. Second, these attacks use protocol-conforming traffic flows that are *indistinguishable* from legitimate flows, thereby causing high collateral damage when flows are dropped to relieve congestion. Consequently, traditional defenses that rely on detecting anomalous flows with specific attack signatures (e.g., SYN floods) or sources that appear in "elephant flows" [34], [58] will simply be ineffective against these attacks, and may in fact adversely impact legitimate connections.

Moreover, attackers have a fundamental *cost-asymmetry* advantage with respect to defenders. On the one hand, the cost of flooding a 10 Gbps network link can be as low as US $80 and averages US $920, assuming 1 Mbps upload bandwidth per bot [18]. On the other hand, the cost of the backbone link bandwidth is *orders of magnitude* higher. For example, 10 Gbps bandwidth in the Internet transit costs about US $6,300 as of 2015 [3]. This is approximately 7 – 80 times more expensive relative to the equivalent attack bandwidth. Unfortunately, removing the attack-defense cost asymmetry is very difficult to achieve since these costs are determined by two fundamentally independent markets, namely, pay-per-install bot markets [18] and Internet transit markets [53]. Taken together, indistinguishability and cost asymmetry enable attackers to launch pernicious attacks on critical infrastructures and services with impunity.

In this paper, we argue that *any solution* that purports to defend against indistinguishable link-flooding attacks must achieve at least one of the following goals: (1) remove or reduce the cost asymmetry; (2) ensure adversary detection; or (3) create an untenable tradeoff between the cost and detectability. By definition, any ISP that can either substantially increase the attack cost relative to the defense cost or induce detectability will deter attacks by *rational* adversaries; i.e., cost-sensitive adversaries who wish to remain undetected.[1]

We show, perhaps surprisingly, that is indeed possible to force the adversary into an untenable tradeoff that either increases the attack cost or forces detectability. The high-level intuition behind our approach is as follows; viz., Figure 1. Suppose we know the locus of the attack L (i.e., a specific ISP link) and we have some capability to *logically* increase

[1]In contrast, countermeasures for cost-insensitive, irrational adversaries are known to be harder and more expensive to orchestrate and deploy; e.g., CoDef [38]. Thus, SPIFFY's efficient deterrence is a very desirable first-line defense for cost-sensitive, rational adversaries, which are believed to be the majority of DDoS adversaries; viz., Section VIII-A.

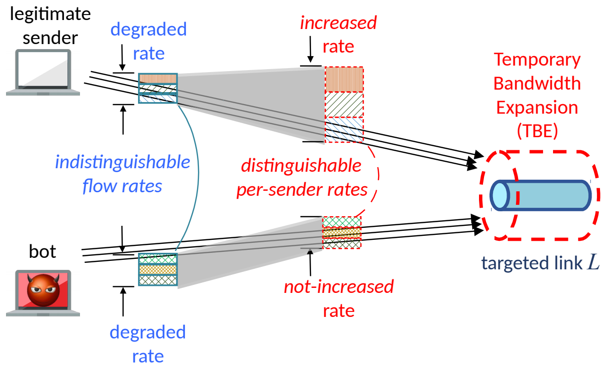
legitimate sender
degraded rate
increased rate
Temporary Bandwidth Expansion (TBE)
indistinguishable flow rates
distinguishable per-sender rates
bot
not-increased rate
targeted link L
degraded rate

Fig. 1: Intuition for distinguishing legitimate senders from bots via temporary bandwidth expansion.

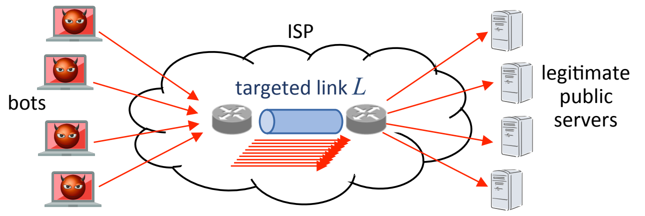

Fig. 2: An example of link-flooding attacks. Legitimate looking connections between the bots and the legitimate public servers cross the link L in the ISP [31].

the bandwidth of L by some factor M temporarily. After the increase, we observe the response of the traffic source IPs that were traversing L. Now, legitimate sources running TCP-like flows will naturally see a corresponding increase in their throughputs as the bandwidth of their bottleneck link has increased. Attack sources, however, will not observe this increase as a rational cost-sensitive attacker would have chosen to *fully utilize* the available bandwidth of the upstream links of the sources in the first stage; i.e., before the temporary bandwidth expansion. Thus, the bottleneck bandwidth increase will induce no increase in the effective throughput of the attack sources. Alternatively, to avoid detection, the attacker could choose to keep each bot's attack traffic rate much lower than the available bandwidth of its upstream link. Note, however, that this will increase the number of required bots and thus *increase attack cost* proportionally. In essence, adversaries are forced to either allow their attack sources to be detected (via rate-change measurements) or accept an increase in attack cost. Note that the key requirement is to monitor the *change in throughput* for traffic sources after the bottleneck bandwidth increase; measuring the raw throughput itself alone will *not* help detection as the attack flows are indistinguishable from normal flows.

However, there are three practical challenges that need to be addressed before this high-level intuition can turn into a practical defense mechanism:

(1) **Implementing bandwidth expansion:** First, we need some mechanism for increasing the logical bandwidth of L with a sufficiently large expansion factor. Note that a larger expansion factor will: (a) make it easier to distinguish bots vs. legitimate sources (e.g., to create a clear separation accounting for measurement noise) and (b) equivalently increase the effective attack cost. However, it is infeasible and uneconomical for ISPs to have spare dark fibers for each link, and thus we need deployable mechanisms to virtually increase the bandwidth, if only temporarily.

(2) **Fast workflow:** Second, we need the defense workflow to be fast and responsive to be effective against real attacks. If the temporary bandwidth expansion and detection takes several hours, then the damage is already done.

(3) **Robust rate-change detection:** Third, we need per-sender rate change measurements at scale, which may in turn require high processing requirements on monitoring routers as well as high control overhead for reporting these measurements. Furthermore, this detection must be robust to TCP effects, especially given that many legitimate flows on the Internet are short flows.

We address these practical challenges and present the design and implementation of *SPIFFY*.[2] To address (1), SPIFFY presents a new traffic engineering [25] technique based on software-defined networking (SDN) whereby one can virtually increase the bandwidth by routing around the bottleneck. To address (2), we develop fast greedy algorithms to solve a traffic optimization problem, which would otherwise take several hours even with state-of-art solvers [2]. Finally, to address (3), we suggest simple sketch-based change detection algorithms that can measure rate changes with low overhead [33], [58]. We develop a proof-of-concept prototype using `POX` [7] and use a combination of real testbed evaluation and large-scale simulations to validate the effectiveness of SPIFFY against link flooding attacks. SPIFFY relies on SDN's centralized control and traffic visibility to develop the first-known defense against such link-flooding attacks.

Contributions: In summary, this paper makes the following contributions:

- A practical solution to force link-flooding adversaries into an untenable tradeoff between cost and detectability, which provides an effective first-line defense;
- A bandwidth expansion mechanism for SDN via traffic engineering based on a fast heuristic for solving the underlying optimization problem;
- An SDN-based implementation of SPIFFY and an extensive evaluation demonstrating its robustness with realistic TCP effects.

II. Background and Threat Model

In this section, we review the types of link-flooding attacks we address in this paper and then formally characterize the attacker goals and constraints.

Background: The link-flooding attacks we consider in this paper target network links in the *core* of the Internet (e.g., backbone links in large ISPs or inter-ISP links) and create a large number of attack flows crossing the targeted links to flood and virtually disconnect them; viz., Figure 2. This is in sharp contrast to traditional DDoS attacks that aim to choke the resources of the end target; e.g., computation, memory, or

[2]SPIFFY stands for handling 'Scalable Persistent Indistinguishable link-Flooding attacks with reduced cost asymmetrY.'

access link bandwidth. Recent research (e.g., Coremelt [51] and Crossfire [31]) and real-life attacks against core routers of upstream networks (e.g., ProtonMail attack [28], Spamhaus attack [17]) are the examples of such attacks.

In the general case, link-flooding attacks may flood multiple link targets, as exemplified in Crossfire [31]. For simplicity of presentation, we focus on the link-flooding attacks against a single infrastructure link throughout this paper. However, our system is also robust to multiple link-flooding attacks; viz., Section VIII-C for a detailed discussion.

Threat model: We consider a *rational* adversary who wants to inflict as much damage as possible on legitimate flows of the target network link using as few resources as possible, and while remaining indistinguishable. Formally, our link-flooding adversary pursues three goals:

- **Attack-Strength Maximization** ($G_{strength}$): Suppose the network a mechanism to guarantee a per-flow rate under "normal" network operation when there are no attacks; e.g., through a combination of link capacity provisioning and traffic engineering [12]. Let this *guaranteed rate* be denoted by r_g. The adversary aims to reduce the per-TCP-flow fair-share rate of flows traversing the target link to the *degraded rate*, denoted by r_d. The degraded rate will be much smaller than the guaranteed rate (i.e., $r_d \ll r_g$); otherwise (e.g., $r_d \sim r_g$) the attack would fail to degrade a legitimate flow much beyond the guaranteed rate r_g.

  The degraded rate r_d is an adversary-chosen parameter that measures the *attack strength*. A smaller r_d indicates a stronger attack since legitimate flow rates would be degraded more.
- **Attack Persistence** ($G_{persistence}$): To circumvent detection and hence be persistent, a link-flooding attack needs to mimic legitimate traffic patterns. That is, the attack flows are indistinguishable from legitimate ones via traffic analysis of headers/payloads and/or intrusion detection at the target link. For instance, this can be achieved by using legitimate looking web sessions to decoy servers [31]. To this end, the adversary uses TCP-based flooding attacks. Because TCP traffic constitutes the majority of the Internet backbone traffic, as it represents about 90 – 98% of the byte volume of the backbone links, these attacks are more difficult to detect and filter than UDP-based flooding attacks [59].
- **Attack-Cost Minimization** (G_{cost}): A *rational* adversary will seek to minimize the cost of the attack. In this paper, we assume that the cost of the attack is proportional to the *number of bots* necessary for the attack; thus, the number of bots is a good proxy for the attack cost. This assumption is based on the observation that, in general, bots are sold in bulk (e.g., several thousands) in the pay-per-install markets [18].

We assume that the network follows a *per-flow fair-share* allocation of link bandwidth to all flows served. This is already widely observed in today's Internet since TCP flows adjust their rates in response to congestion, and thus approximate per-flow max-min fair rates [8]. If senders do not conform to the TCP flow control (i.e., they send flows faster than the fair-share rates) they can be detected by other mechanisms [34].

Based on this threat model defined, we develop and evaluate SPIFFY in the following sections.

III. SPIFFY Intuition and Security Analysis

In this section, we provide the intuition behind SPIFFY and the security analysis showing why it forces adversaries to either increase costs (G_{cost}) or forgo indistinguishability ($G_{persistence}$) while achieving rate degradation for legitimate flows ($G_{strength}$).

A. High-level Idea

The key reason why link-flooding attacks are so successful and dangerous is that they are affordable and indistinguishable. Thus, our overall goal is to force attackers to compromise on either $G_{persistence}$ or G_{cost} for a given $G_{strength}$; i.e., the attackers either become detectable or pay an increased cost.

The intuition behind our approach is as follows. During a link-flooding attack, a legitimate sender would only be able to send traffic at a much lower per-host rate compared to the desired application-layer data rates. This is because the attack with the rate-reduction goal ($G_{strength}$) decreases legitimate flow rates significantly. However, an attacker who is trying to optimize cost (G_{cost}) would have all its bots send at their highest per-host send rate (i.e., saturate its upstream bandwidth), by creating additional attack flows whenever its upstream bandwidth allows it. Due to these fundamental goal differences, a legitimate sender and an adversary's bot would react very differently when the congestion is *relieved*; viz., Figure 1. A legitimate sender would very likely increase its send rate to meet its rate demand (e.g., buffered traffic from application layer) due to TCP rate control while a bot would have no available bandwidth left for further rate increase.

We can implement the controlled congestion relief by what we call the *temporary bandwidth expansion* (TBE). That is, we temporarily increase the virtual bandwidth of the target link by some factor M to allow senders suffering from congestion to increase their send rates. TBE enables us to measure the rate increases of senders and ultimately distinguish bots from legitimate senders.

In order to prevent bots from being detected, a link-flooding adversary must give up fully utilizing the upstream bandwidth of bots and *mimic* the legitimate senders' rate increase when congestion is relieved, as we will see below. However, this rate-increase mimicry will lower the bandwidth utilization of each bot and in turn cause the link-flooding adversary to significantly increase the number of attack bots. As a result, the link-flooding adversary faces an *untenable choice*: (1) she could maintain the low-attack cost while allowing her bots to be detected, or (2) she could make bots indistinguishable while increasing the attack cost significantly.

B. Security Analysis

We begin by formulating the optimal attack strategy for a scenario *without* the SPIFFY defense and then argue why SPIFFY creates a fundamental cost-detectability tradeoff for link-flooding adversaries. For simplicity of presentation and without loss of generality, the following analysis assumes a *homogeneous* bot deployment where each bot has the same upstream bandwidth u. Let B be the bandwidth of the target link which is under the link-flooding attack.

Optimal adversary strategy without SPIFFY ($AS_{\neg spiffy}$)**:** An adversary can optimally satisfy the attack goals $G_{strength}$, G_{cost}, $G_{persistence}$ by using B/u bots, where each bot creates u/r_d attack flows and saturates its upstream bandwidth u.

Proof: To congest the target and reduce the per-flow TCP fair-share rate $G_{strength}$, the attack first has to flood the target link. Thus, the minimum number of bots required for the attack G_{cost} is $n_b = B/u$. Also, due to TCP per-flow fairness, the fair-share rate provided by the target is $r_{FS} = \frac{B}{N_b}$, where N_b represents the total number of attack flows. This assumes no legitimate flows in the target link: the attack strategy designed without considering legitimate flows guarantees meeting the attack goals even when legitimate flows exist. Also, due to the rate-reduction goal the fair-share rate is reduced to the degraded rate $G_{strength}$, $r_{FS} = r_d$. Since N_b attack flows are created by n_b bots, on average each bot creates N_b/n_b attack flows, which is $N_b/n_b = (B/r_d)/(B/u) = u/r_d$.[3] ■

An adversary with attack strategy $AS_{\neg spiffy}$ has already saturated the upstream bandwidth of each bot. As a result, bots cannot increase their sending rate by a factor of M and cannot avoid being detected by SPIFFY. We argue that for the adversary to evade the test, it must satisfy a property we call *rate-increase mimicry*.

- **Rate-increase Mimicry (RM):** Bots are capable of instantly increasing their send rate by a factor of M when congestion is relieved at the bottlenecked link. This implies that bots must use only u/M of their upstream bandwidth when congesting the target link.

The **RM** property enables the adversary to simultaneously satisfy $G_{strength}$ and $G_{persistence}$ while compromising G_{cost}. If all bots are capable of rate increase with a factor of M, they pass the SPIFFY test and thus bots remain undetected. This leaves the adversary with the following new attack strategy under SPIFFY.

Optimal attack strategy with SPIFFY (AS_{spiffy}): The attack strategy must satisfy the two conditions to achieve the two attack goals $G_{strength}$ and $G_{persistence}$ under SPIFFY.

(1) the attack utilizes $M \cdot (B/u)$ bots and
(2) each bot creates $(u/r_d)/M$ attack flows by utilizing only $1/M$ of its upstream bandwidth u.

Proof: The proof is similar to that of the optimal attack strategy with SPIFFY ($AS_{\neg spiffy}$). However, due to the **RM** property, when attacking the target link, each bot uses only $1/M$ of its bandwidth limit u. Therefore, the attack requires $B/(u/M) = M \cdot (B/u)$ bots, where each bot creates $u/r_g = u/(M \cdot r_d)$ attack flows. ■

Thus, a link-flooding adversary now faces the following mutually-exclusive options forcing a fundamental tradeoff between cost and detectability:

1) Adversary follows $AS_{\neg spiffy}$ and requires (B/u) bots, potentially allowing detection of his/her bots by SPIFFY.

[3]For simplicity, we ignore small errors generated when converting real values to integers.

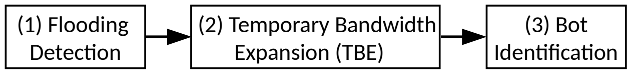

Fig. 3: Workflow of SPIFFY

2) Adversary follows AS_{spiffy} and requires $M \cdot (B/u)$ bots, circumventing the bot detection.

We argue that an adversary has no other options than those listed above. To see why, let us consider two attack strategies that differ from these: (1) *Per-flow rate increase strategy*: In this strategy, bots saturate their bandwidth to attain the cost-minimization goal G_{cost}. They quickly detect the bandwidth expansion and instantly allocate increased bandwidth to a set of selected flows by pausing (or terminating) other attack flows, making the selected flows look legitimate. However, since SPIFFY measures *per-sender* (**not** per-flow) rate changes, such bots would be detected due to their unchanged per-sender rates. (2) *Bot replacement strategy:* This strategy also saturates the bots to achieve G_{cost}. The adversary replaces his/her bots in operation with new bots whenever the current bots are detected by SPIFFY. Although this strategy can be efficient for a short period of time, the cost of maintaining the attack persistence grows linearly with attack duration increases since bots need to be replaced repeatedly.

IV. SPIFFY System Overview

In this section, we describe an end-to-end view of SPIFFY and highlight key practical challenges that we need to address to realize it. We envision SPIFFY being run by an ISP where the target link L is located, since the end customer who is the eventual target of the attack cannot detect or respond to link-flooding attacks. We believe that ISPs have a natural economic incentive to protect their immediate customers (e.g., as a value-added service [1]) and offer such capabilities on demand to create new revenue streams.

To understand the key challenges in this deployment model, let us consider the three logical stages in the SPIFFY workflow as seen in Figure 3:

1. **Flooding detection.** SPIFFY detects the existence of a link-flooding attack against a link (e.g., via SNMP-based link-utilization measurements [21]) and estimates the degraded rate r_d for the attack by measuring the fair-share flow rate of the target link.
2. **Temporary bandwidth expansion (TBE).** For *all* senders that use the target link, SPIFFY provides a temporarily expanded bandwidth $M\times$(current per-sender rate), where bandwidth expansion factor $M \gg 1$. For the time being, let us imagine an *ideal* TBE that increases the target link's physical bandwidth B to $M \times B$. Section V explains how TBE can be implemented in real networks. Note that the bandwidth expansion is *temporary* (e.g., $<$ 5 seconds) and the bandwidth of the link returns to B after TBE. The bandwidth expansion factor M is set to the ratio of the guaranteed rate r_g to the degraded rate r_d, namely, $M = r_g/r_d$, to let the legitimate senders increase rates from r_d to r_g in response to TBE. This value of M enables SPIFFY to identify senders with the per-sender rate change close to M as legitimate ones.

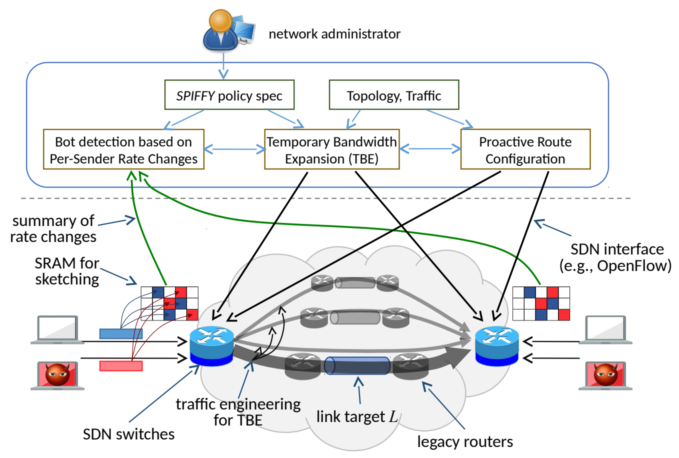
network administrator
SPIFFY policy spec
Topology, Traffic
Bot detection based on Per-Sender Rate Changes
Temporary Bandwidth Expansion (TBE)
Proactive Route Configuration
summary of rate changes
SRAM for sketching
SDN interface (e.g., OpenFlow)
traffic engineering for TBE
link target L
SDN switches
legacy routers

Fig. 4: Overview of the SPIFFY using an SDN in the Internet core.

3. **Bot identification.** SPIFFY measures the *per-sender* rate changes of all the senders that use the target link. It starts measuring the per-sender rate before TBE and stops measuring after TBE. The frequency of measurements should be high enough to capture the rate increase and calculate the ratio of the increase; e.g., every 1 second. Before TBE, flows from a legitimate sender will have the flow rate r_d (viz., $G_{strength}$), but during TBE the majority of the legitimate flows increases their rates at least up to r_g, and thus the total per-sender rate increases by a factor close to or higher than M $(= r_g/r_d)$. In contrast, a bot would *not* increase its send rate even if the bandwidth allocated to it is expanded due to its saturated upload bandwidth; viz., Attack Strategy $AS_{\neg spiffy}$.

Challenges: Our focus in this paper is on steps (2) and (3) of this workflow. We assume that existing monitoring mechanisms are used for (1); e.g., [21]. Our two key challenges arise for steps (2) and (3).

First, the challenge in designing TBE is to provide the senders significantly expanded bandwidth. Ideally, we want to physically increase the target link bandwidth, but this may not be viable unless the target network has spare dark fiber. Instead, our goal is to find an immediate solution that does not rely on spare optical fibers. Moreover, the operation of TBE has to be real-time to quickly react to the flooding attacks; e.g., in a few seconds.

Second, bot identification is challenging because it requires real-time per-sender rate measurements for *all* senders at the target link. In practice, it is difficult to keep track of these rate changes because the number of senders might easily go up to tens or hundreds of thousands. Finally, the rate-change estimation must be robust to real-world considerations; e.g., TCP effects in reacting to changes in the RTT or the impact on short legitimate flows.

Key ideas: We address these two challenges as follows. SPIFFY can leverage recent advances in software-defined networking (SDN) to implement the above workflow. An SDN's central controller provides new capabilities for network management [24], [29], [43], [44]. While we do not claim that SDN is *necessary* for countering link-flooding attacks, we consider it to be a natural enabler for realizing the SPIFFY workflow. The overall system is illustrated in Figure 4.

- *Practical TBE:* To enable practical TBE, we develop a traffic engineering application that dynamically changes traffic routing to meet desired goals [25]. At a high level, we increase the effective bandwidth of the link-flooding target link by routing flows around the bottleneck. We also provide practical techniques to work around the constraints of real networks where the bandwidth expansion factor (M) might be low. Finally, we develop fast heuristics to solve the traffic engineering optimization.
- *Robust bot detection:* First, to detect bots, we provide a scalable monitoring mechanism that relies on simple "sketching" algorithms running in the edge switches [58]. This algorithm guarantees the accurate per-flow rate change measurement with only small size of SRAM and few hash computations. Second, to obtain the robust bot-detection results, we develop strategies that yield very low false-positive rate. We investigate several cases where legitimate senders might be misidentified as bots (e.g., legitimate senders that do not react to TBE or TCP effects in response to changed RTT measurements) and propose solutions to remove such undesirable events.

V. Scalable and Practical TBE

In this section, we focus on a practical implementation of TBE. As discussed earlier, there are two key challenges here. First, given that networks do not have spare fibers lying around, we need a network-layer solution for TBE. Second, we need this step to be fast because it ultimately impacts our ability to rapidly test and detect bots.

Our *network-layer* TBE approach dynamically reroutes the flows traversing the target link through other under-utilized links in the network. It computes the new routes, which provide large bandwidth expansion to all senders using the target link simultaneously, *as if* the target link bandwidth is physically expanded. The new routes are calculated at a central controller and installed in SDN-supported switches at the edge. Note that for ease of explanation we refer to the physical bandwidth expansion as the *ideal* TBE.

The goal of the network-layer TBE is to emulate the ideal TBE with large bandwidth expansion factor. The ideal bandwidth expansion factor we wish to achieve is $M_{ideal} = r_g/r_d$, as described earlier. Then the question that arises is how the network-layer TBE can achieve this high M_{ideal}. To answer the question, we first look at how much bandwidth expansion can be achieved by the network-layer TBE for a given network. Then we evaluate whether the bandwidth expansion factor is large enough for M_{ideal}.

We formulate the routing problem of finding the maximum bandwidth expansion factor, denoted as $M_{network}$, for a given a network configuration. Let us assume that we are given a network graph $G = (V, E)$, where V represents the set of routers and E represents the set of links between the routers. We denote by $b(x, y)$, where $(x, y) \in E$, the bandwidth that is not used at the time of TBE; i.e., residual bandwidth. We define a *flooding traffic matrix* T where each ingress/egress pair (s, t) denotes the total traffic rate $T(s, t)$ between s and t that contribute to the flooding at the target link. Note that we assume that the residual bandwidth $b(x, y)$ and the flooding traffic matrix T are unchanged during TBE operation. We associate a variable $f^{(s,t)}_{(x,y)}$, with each pair (s, t) and each link $(x, y) \in E$, that denotes the fraction of the traffic flow from s to t over the link (x, y). The problem of finding the maximum bandwidth expansion factor for network-layer TBE is defined by following linear program:

$$\max \quad m \tag{1}$$

$$\text{(LP)} \quad \text{s.t.} \sum_{y:(x,y)\in E} f^{(s,t)}_{(x,y)} = \begin{cases} -m \cdot T(s,t), & \text{if } y = s \\ m \cdot T(s,t), & \text{if } y = t \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$\sum_{(s,t)} f^{(s,t)}_{(x,y)} + f^{(s,t)}_{(y,x)} \le b(x,y),\ (x,y) \in E \tag{3}$$

We denote the objective value of this linear program by $M_{network}$. The objective (1) in LP is to maximize the bandwidth expansion factor via rerouting. Conditions (2) represent the flow conservation constraints that ensure the m-times expanded traffic $T(s,t)$ is routed from the attack relevant ingress/egress router pairs (s, t). Constraints (3) define the load on each link and ensure it is smaller than its residual link bandwidth.[4]

Here, we face two challenges in solving the LP and applying it for the ideal TBE emulation.

- *Scalability:* Although the LP can be solved in polynomial time, the size of the problem becomes impractically large when the number of routers R is large (e.g., $R > 100$). The time to solve the problem grows rapidly with the network size and becomes unrealistic for real-time operations; e.g., few thousand seconds in a network of size $R = 196$.
- *Small $M_{network}$ compared to M_{ideal}:* In practice, the value of $M_{network}$ is small compared to M_{ideal}. As we will see in detail in Section VII, $M_{network}$ is typically in the range of 2 – 3 and it is likely that $M_{ideal} = r_g/r_d$ is 5 – 10 times larger than $M_{network}$.

We solve the TBE scalability problem by greedy algorithm (Section V-A) and the small $M_{network}$ problem by randomized sequential TBE (Section V-B).

A. Greedy algorithm for TBE Scaling

As a solution to the scalability problem, we propose a simple greedy routing algorithm that runs much faster than solving LP with off-the-shelf solvers. One can also use other efficient ways of solving such problems in general; e.g., [13], [47]. The greedy algorithm presented here is one possible way for calculating an approximate solution.

The greedy algorithm takes as inputs the set of ingress/ egress pairs and the number of their flows that cross the target link, the desired bandwidth expansion factor m, and the network graph $G = (V, E)$ and the residual link bandwidth $b(x, y)$ for each link $(x, y) \in E$. Then it outputs a feasible routing solution $R(s, t)$ for all ingress/egress pairs (s, t). The pseudocode of this algorithm is given in the following Algorithm 1.

[4]Since we use the undirected graph G, for any link $(x, y) \in E$ we set $b(x, y) = b(y, x)$ but we activate only one them for the constraints (3).

Algorithm 1 Greedy algorithm for TBE

```
1: Inputs: Set of ingress/egress pairs $(s,t)$ crossing target link,
2:         Number of flows on each ingress/egress pair $n(s,t)$,
3:         Desired bandwidth expansion factor $m$,
4:         Network topology graph $G = (V,E)$, and
5:         Residual link bandwidth $b(i,j)$ for $(i,j) \in E$.
6: while ($\exists(s,t)$ that has not yet selected) do
7:     Select ingress/egress pair $(s,t)$ at random w/o replacement.
8:     Calculate the available network graph $G' = (V, E \setminus E')$,
       where $E'$={links w/ available bandwidth $\ge m \cdot n(s,t) \cdot r_d$}.
9:     Calculate new route $R(s,t)$ in $G'$.
10:    if $R(s,t) \ne NULL$ then
11:        Move all flows in $(s,t)$ to $R(s,t)$.
12: Output: New routes $R(s,t)$ for all ingress/egress pairs $(s,t)$.
```

We use a binary search procedure over the value of m in Algorithm 1 and obtain the estimate of the maximum bandwidth expansion factor $\widehat{M}_{network}$ and the corresponding routing solution. We show in Section VII that the difference between $M_{network}$ and $\widehat{M}_{network}$ is negligible in practice.

B. Randomized Sequential TBE

To solve the problem of small value of $M_{network}$, we use a randomized sequential TBE approach. That is, we test only a subset $R_{TBE} = M_{network}/M_{ideal}$ of senders at each TBE round so that a subset of senders can have feasible routes that provide M_{ideal}-times bandwidth expansion. We then repeat this process until most of the senders are tested. If the obtained $M_{network}$ from LP is larger than or equal to M_{ideal}, no more than a single TBE is required.

Random sender selection. We randomly select a fraction R_{TBE} of senders in each ingress/egress pair and reroute them using the solution of LP. The obtained routing solution provides the selected senders M_{ideal}-times expanded bandwidth. Since we randomly sample the senders at every TBE, an adversary *cannot* anticipate when a particular bot will be tested. The number of TBE rounds that needs to be performed to test the majority (e.g., 90%) of the senders depends on the fraction R_{TBE}.

VI. Rate-change measurement tests

In this section, we focus on the rate-change measurement test. As previously mentioned, there are two key challenges in designing the test. First, stateful per-sender rate monitoring could be expensive and induce high control overhead at the SDN controller. Second, the robustness can be undermined by real world TCP effects; e.g., prevalence of short-lived TCP flows or reaction to RTT changes.

A. Sketch-based Per-Sender Rate Change Detection

As mentioned, SPIFFY requires rate change detection for all senders that cross the target link. This raises concerns about the computation and memory complexity of such "stateful" operations. Another concern is that when the real-time per-sender rate measurements are reported back to the SDN controller the control channel could be easily congested due to the large volume of control messages.

SPIFFY can address these challenges by utilizing sketch-based measurements [30]. Sketch is a memory-efficient data structure that stores summaries of streaming data. In particular, a simplified variant of sketch-based rate change detection [33] can be used for efficiently and quickly detecting per-sender rate changes. With the sketch-based rate change detection, the edge switches report *only* the measurement summary to the SDN controller, such as list of bot IPs, and significantly minimize the control channel overhead.

Sketch-based rate-change measurement: We use the original sketch-based change detection mechanism by Krishnamurthy *et al.* [33] for measuring per-sender rate changes. In fact, SPIFFY needs a simpler version of the original sketch-based change detection since it measures with the granularity of a sender (i.e., source IP), which is coarser than the granularity of a flow. The three basic components are the sketch module, the forecasting module, and the change-detection module [33]:

1) The sketch module creates a sketch; i.e., a $H \times K$ table of SRAM memory. When a packet arrives at an edge switch, the source IP is fed into the H independent hash functions. Based on the $\mathtt{mod}\ K$ of the H hash outputs, H registers in the H rows are updated by the packet size u. By updating all packets in a time interval t, we obtain a sketch $S(t)$ at the end of the interval t.
2) The forecasting module uses the observed sketches in the past intervals $S(t')$ $(t' < t)$ to compute the forecast sketch $S_f(t)$.
3) The change-detection module constructs the forecast error sketch $S_e(t) = S(t) - S_f(t)$. For each sender's IP$_{src}$, this module calculates the forecast error.

Estimated measurement complexity: We analyze the estimated memory size and the sketch computations. The required memory size is determined by the number of independent hash functions H and the size of sketch bins K for each hash function. For a real Internet trace dataset with more than 60 million flows, $H = 5$ and $K = 32K$ produce very accurate rate-change measurement; e.g., 95% accuracy for top 1000 flows with the maximum rate changes [33]. Our rate-change measurement will also be accurate with these parameters, since each edge switch will not need to measure more than 60 million senders in most cases. When we assume 3 bytes for each register in the sketch memory, each edge switch requires 480 KB SRAM memory space.

Sketch-based measurement requires H hash operations for individual incoming packets. Also, each SRAM access requires few tens of nano seconds. However, since the hash operations and SRAM access can be implemented in parallel in hardware, these per-packet operations can be very efficiently implemented and thus do not affect the data plane throughput [58].

At every rate-change detection interval, each edge switch calculates the forecast sketches $S_f(t)$ and the forecast error sketches $S_e(t)$. These and the final rate-change calculation requires a computational overhead of about 1.91 seconds, when for example $H = 5$, $K = 64K$, and 10 million flows are monitored [33]. Our per-sender rate-change measurement would require much shorter (e.g., $\ll 1$ sec) time for the computation at each edge switch since today's commodity CPUs are at least 3-4 times faster than the one used (900 MHz CPU clock speed) more than a decade ago [33].

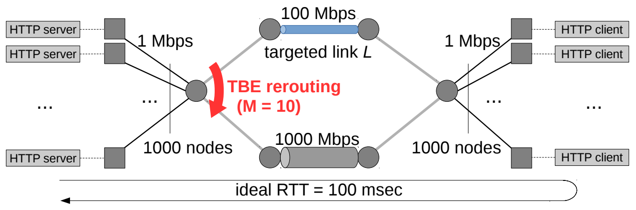

Fig. 5: Simulation setup.

Bot-detection summary reports: Instead of reporting the rate-changes of all senders, each edge switch can report only the subset of senders that are determined as bots (or legitimate senders). Thus, the aggregate bandwidth for control channel can be limited to a few Mbps or less; e.g., only 4 MB data transfer is required even when 1 million bot IPs are reported. Notice that the reports are made only when the TBE is performed.

B. *Bot Detection Robustness to TCP Effects*

The robustness of SPIFFY's bot detection relies on the prompt and fast rate increase of legitimate senders when TBE is performed. The rate increase is mainly determined by TCP operations at the senders since they control the maximum flow rates at a given time. However, achieving robust bot detection can be challenging due to the two following TCP effects: (1) short-lived flows (e.g., few packets in a flow) in the Internet terminate before TCP increases their rates; (2) when TBE's route changes cause sudden increase of RTT values, TCP might decrease the send rates by decreasing congestion windows and/or causing spurious timeouts.

To achieve robust bot detection, our primary focus is to maintain low false-positive rate because false-positive events cause collateral damage to legitimate senders. In contrast, false-negative rate (i.e., the rate in which bots are misidentified as legitimate senders) is not a particularly useful metric since SPIFFY allows false-negative events to happen for the cost-detectability tradeoffs. For example, if an adversary is determined to remain undetected, she can make the false-negative rate to be practically one at a highly increased attack cost.

Robustness to short TCP flows: Unlike long-lived flows, short-lived flows might *not* increase their rates in response to TBE because they may not last long enough (e.g., few seconds) when the bottlenecked bandwidth is expanded. Therefore, when the majority of flows are short-lived (as is the case of today's Internet traffic), per-sender rate of *legitimate senders* could be almost unchanged when TBE is performed, causing false-positive events.

Here, we first observe that the prevalence of short-lived flows does *not* affect the rate changes of senders that create realistic traffic with the mixture of short- and long-lived flows. Moreover, we show that SPIFFY can maintain false-positive rate as low as 1% or less by exempting senders with per-sender rates lower than *minimum per-sender rate* from the bot detection process regardless of their rate-change ratios.

To test our claims, we perform simulations with a synthetic web-traffic generator. We use the *ns2* simulator with PackMime-HTTP web-traffic generator to construct diverse

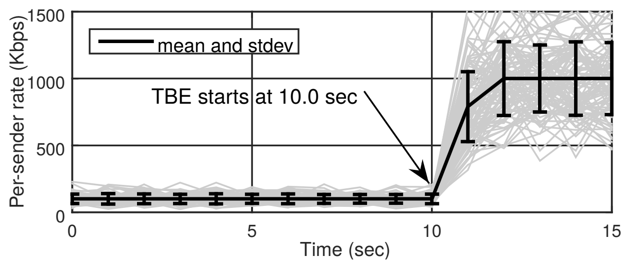

Fig. 6: An example per-sender rate change measurements of randomly selected 100 legitimate senders with mean and standard deviation when the bandwidth expansion factor $M = 10$.

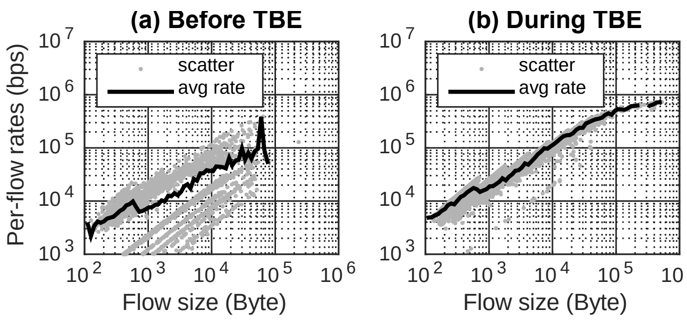

Fig. 7: Simulated per-flow rates of flows in realistic HTTP web traffic (a) before and (b) during TBE with bandwidth expansion factor $M = 10$.

network environments and simulate accurate TCP operations with realistic HTTP application traffic demand [4], [19]. Approximately 70% of the synthetic web-traffic flows have size smaller than a single IP packet's maximum size (1,500 Bytes) while a small number of large flows exist. We determine the queue size based on a rule-of-thumb practice; i.e., $QueueSize = \overline{RTT} \times C$, where $\overline{RTT}$ is the average round-trip time of the flows crossing the link and C is the data rate of the link [11]. For TBE, we assume that the bandwidth of the target link is expanded by a factor of $M = 10$. As shown in Figure 5, we simulate 1000 pairs of clients/servers exchanging HTTP traffic through a target network link. We set the ideal (i.e., when no traffic on the path) round-trip time of 100 msec and the application-layer data rate of 1000 Kbps for the purpose of clear illustration.

Figure 6 shows a rate measurements of 100 randomly selected senders. Before TBE starts at $t = 10$ seconds, all senders achieve approximately 100 Kbps with small standard devidations; however, after TBE starts, most senders achieve 10 times higher rates within 2 seconds. This result shows that the rate change detection is robust for the legitimate senders with realistic flows, in particular with large portion of short-lived flows.

The reason for the negligible effect of short-lived TCP flows on the effectiveness of rate-change detection is that a few long-lived flows from senders increase their rate significantly once TBE is performed and thus induce the overall per-sender rate change. Figure 7 shows the simulated flow rates versus flow sizes. Notice that before TBE only short-flows (i.e., small flow size) are observed. They achieve low rates and long-lived flows are not even able to complete their TCP connections.

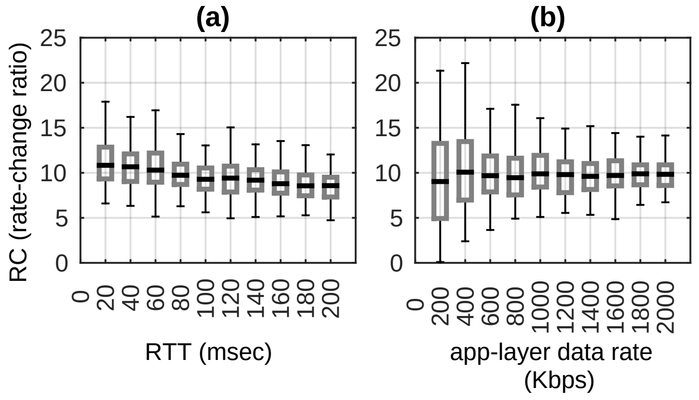

Fig. 8: Whisker plots representing the rate-change ratios for varying RTT/application-layer data rates when the bandwidth expansion factor $M = 10$.

This is because short-lived flows spend most of their life in the TCP slow start and thus they can rapidly capture a greater proportion of resources than long-lived flows in TCP congestion avoidance, often driving the long-lived flows into timeouts. After TBE starts, long-lived flows achieve much higher rates whereas short-lived flows achieve only slightly higher rates than before. This is because long-lived flows now have enough time to increase the congestion windows.

Next, we evaluate the false-positive rate of the SPIFFY's bot detection with realistic traffic and propose a mechanism to maintain low false-positive rate. To simulate various types of realistic legitimate senders in different locations with different traffic rates, we vary the end-to-end propagation delays (in msec) and the application-layer data rate (in Kbps) per sender.

Figure 8 shows the measured rate-change ratio (RC) when the ideal round-trip time (RTT) (i.e., RTT measured when no traffic on the path) or the application-layer data rate (i.e., average HTTP data rate) vary. Figure 8a shows that the vast majority of measured rate-change ratios are close to the bandwidth expansion factor $M = 10$ and largely independent of the ideal RTT of the flows. This suggests that bot detection can achieve low false-positive ratio when it uses a rate-change ratio threshold RC_{th} close to M to identify senders with $RC < RC_{th}$ as bots. However, as shown in Figure 8b, the rate-change ratio RC is heavily affected by the application-layer data rate. While senders with high application-layer data rates show rate-change ratios very close to the bandwidth expansion factor $M = 10$, senders with low rates result in rate-change ratios that are spread over a large range. This would potentially induce non-negligible false-positive ratios when bots are identified by thresholding the rate-change ratios.

From this observation, we set the *minimum per-sender rate* ($rate_{min}$) and *exempt* the senders with per-sender rate lower than $rate_{min}$ from bot detection. In other words, senders with per-sender rate lower than $rate_{min}$ are not tested by the target network regardless of their rate change ratios. By exempting these low-rate senders from the bot detection, we can also protect the legitimate, inherently low-rate senders from being misidentified as bots; e.g., legitimate users with slow legacy cellular connections or casual web surfing users with light activity are protected by this exemption.

Figure 9 shows the false-positive rate for varying rate-

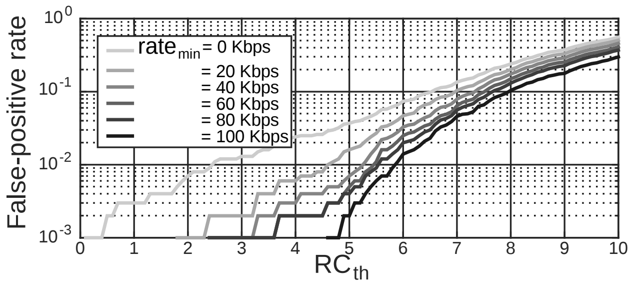

Fig. 9: False-positive rate for varying rate-change ratio thresholds (RC_{th}) and minimum per-sender rates ($rate_{min}$).

change ratios RC_{th} and for several values minimum per-sender rate $rate_{min}$. We first observe that the larger threshold ratio RC_{th}, the higher false-positive rate is expected because small rate fluctuations can cause false positive when the threshold ratio RC_{th} is high. We also notice that as we exclude more low-rate senders (i.e., set higher minimum per-sender rate $rate_{min}$), we can reduce the false-positive rate. As shown in Figure 9, with proper parameters we can easily maintain very low false-positive rate; e.g., 1% or less. Note that the exemption of low-rate senders could contribute to some false-negative errors; i.e., indicating bots as legitimate. However, the influence of the non-detected bots is limited since they do not send at the rate higher than the minimum per-sender rate $rate_{min}$, which is the chosen small rate value.

Note also that adversaries *cannot* exploit the exemption of low-rate senders. An adversary might configure her bots to send at a rate lower than the minimum per-sender rate $rate_{min}$ to avoid detection, but this only increases the attack cost significantly because more bots are needed to create the same amount of attack traffic to congest the target link.

Robustness to sudden RTT increase: Our TBE mechanism reroutes traffic around the target link. Rerouting may find a new route longer than the initial one. According to our experiments (Section VII-B), TBE increases the route length (i.e., number of routers in a route) on average by up to 24%. This raises the question of whether this suddenly increased RTT adversely impacts the false-positive rates of SPIFFY. We list two possible cases where sudden RTT increase might cause false-positive events: (1) Some delay-based TCP variants (e.g., Compound TCP [52] and TCP Vegas [16]) use RTT measurements at receivers to adjust TCP congestion windows. These TCP variants consider RTT increase as the sign of congestion and reduces their sending rates; (2) TCP senders might experience spurious timeouts and drop sending rates significantly. A spurious timeout occurs when RTT suddenly increases and exceeds the retransmission timer that had been determined a priori [39].

Here, we claim that such rate decrease due to RTT increase is *not* likely to happen becuase RTT will actually be reduced significantly when TBE is performed. The rationale behind this is that TBE removes high queueing delay at the (almost) full buffer of the target link. The RTT reduction due to this congestion relief is in general much larger than the RTT increase due to TBE rerouting, ultimately causing RTT reduction.

To support our claim, we measure RTT changes when TBE is performed in a simulation. We set the ideal (i.e., when no congestion on a path) RTT to 100 msec and assume 25% increase of the RTT when rerouting takes place. We assume the rule-of-thumb queue size (i.e., RTT times link capacity [11]) at the target link. As shown in Figure 10a, as soon as TBE is executed at time 5.0 sec, the measured RTT is significantly reduced to the near ideal RTT value. The new measured RTT is 25% higher than the ideal RTT due to TBE's rerouting, but it is still significantly smaller than the RTT measurements before TBE.

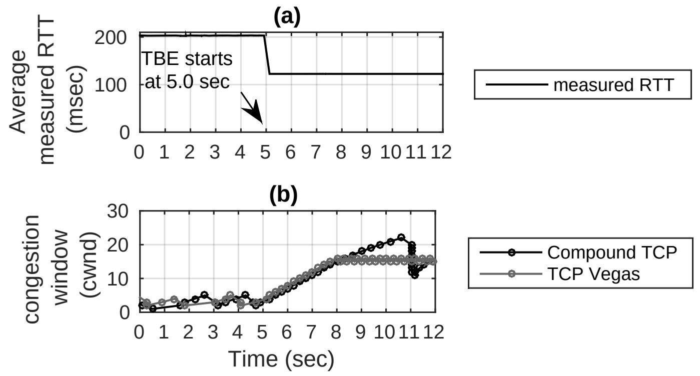

Fig. 10: RTT and congestion window changes when TBE is performed.

We also test how the two delay-based TCP variants, Compound TCP and TCP Vegas, adjust their congestion window in response to TBE. Figure 10b shows that both TCP variants increase their congestion window promptly when TBE is performed and reach the converged points less than 3 seconds.

VII. Evaluation

In this section, we evaluate SPIFFY in an SDN testbed to show its effectiveness (§VII-A). Then we evaluate it using flow-level simulations to show its feasibility in large ISP networks (§VII-B).

A. Testbed Experiments

Our evaluations are executed on a server-grade Dell R720 machine with 20-core 2.8 GHz Xeon CPUs and 128 GB of memory, which runs the KVM hypervisor on CentOS 6.5 (Linux kernel v2.6.32). We use Open vSwitch (OVS v2.3), virtual switches [6]. OVS v2.3 supports the OpenFlow v1.3 [5] specification. We use OpenFlow-enabled switches only at the edges of our test network and traditional switches inside the network. Note that we will interchangeably use switches and routers in this paper. We implement SPIFFY as a `POX` application [7] on the centralized network controller. Notice that in these SDN testbed experiments, we test only long-lived TCP flows generated by `iperf3`. The effects of short-lived flows are studied in packet-level simulations, as discussed in Section VI-B.

1) Effectiveness of Bot Detection: We evaluate how effective SPIFFY is in identifying bots when they are mixed with legitimate senders. We implement the bots based on the Attack Strategy $AS_{\neg spiffy}$. Bot upstream is saturated by attack flows, each of which have the degraded rate, r_d. Note that the adversary in this evaluation does not apply the rate-increase

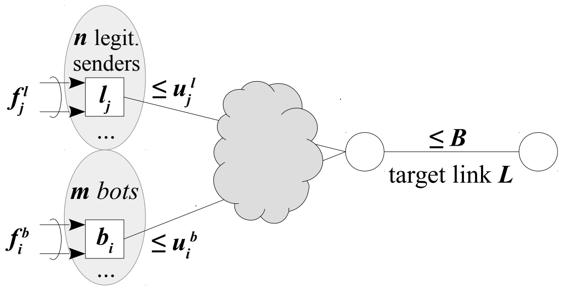

Fig. 11: Parameters for SPIFFY experiments.

mimicry (**RM**) and thus her bots have no available bandwidth to demonstrate the rate increase.

In our simplified ISP network with two edge switches (one ingress and one egress) and 10 parallel links that connect the two edge switches (one of them is the target link of the attack), we reroute traffic crossing the target link to other parallel links and provide ten-times expanded bandwidth (i.e., $M = 10$) to the two senders. For this, the SPIFFY application installs rules and MPLS labels (which are prevalently used in large ISPs [22] and can be implemented by SDN switches [48]) at the edge switches.

Figure 11 shows the general parameters for the SPIFFY experiments. A bot b_i $(1 \leq i \leq m)$ has upstream bandwidth u_i^b and a legitimate sender l_j, $(1 \leq j \leq n)$ has upstream bandwidth u_j^l. We set the number of bots m and their upstream bandwidths u_i^b in such a way that the fair-share per-flow rate at the target link L equals r_d (i.e., $\frac{B}{\sum_{i=1}^{m} f_i^b + \sum_{j=1}^{n} f_j^l} = r_d$) to achieve the attack goal $G_{strength}$. Notice that all bots generate $f_i^b = u_i^b / r_d$ flows of rate r_d to saturate their upstream bandwidth.

In our experiments, we set all senders (both bots and legitimate senders) to send 50 long-lived TCP flows to make them indistinguishable to any per-host rate filtering mechanisms. Accordingly, all bots are set to have upstream bandwidth $u_i^b = f_i^b \times r_d = 0.5$ Mbps when $r_d = 10$ Kbps. We set all legitimate senders' bandwidth to accommodate all 50 legitimate flows with guaranteed rate r_g. That is, $u_j^l = f_j^l \times r_g = 5$ Mbps when $r_g = 100$ Kbps. Note that the upstream bandwidth parameters for bots and legitimate senders are selected for illustrative purpose only. SPIFFY is effective for any practical upstream link bandwidth. RTTs are set to be 200 msec to experiment the practically worst-case rate-change responsive time for TBE operation.

Figure 12 shows the per-sender rate changes of the two senders measured every second by the edge switches. The rate is measured from $t = 0$ to $t = 20$ seconds, when the TBE operation is performed at $t = 10$ second. Notice that before TBE (i.e., at $t < 10$), the two senders' rates are almost identical. However, once TBE is performed, within less than 5 seconds (i.e., at $t < 15$), the two senders show very different rate changes; the legitimate sender's rate increases by almost 10 times whereas the bot's per-sender rate remains the same. At the legitimate sender TCP adapts to the expanded bandwidth in less than 5 seconds. Note that after TBE ends (i.e., at $t = 15$), SPIFFY immediately starts the bot identification. The target network notices the difference in the rate changes, identifies the bot, and filters its source IP at the corresponding ingress

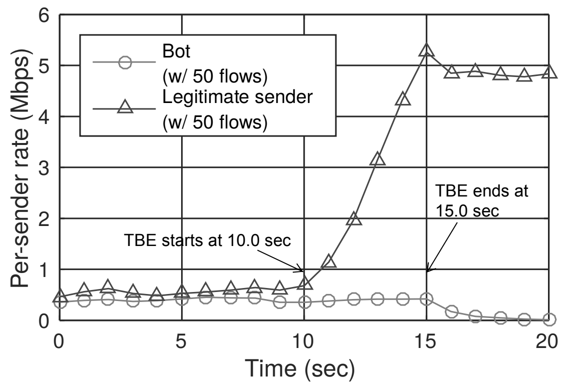

Fig. 12: Effectiveness of TBE for bot identification.

switch of the network. As a result, after $t = 15$ the bot's rate tapers off quickly while the legitimate sender achieves the guaranteed rate $r_g = 100$ Kbps $= 5$ Mbps / 50 flows.

2) Effectiveness of Increasing Attack Cost: Unlike the previous experiment, in this evaluation an adversary decides to follow the rate-increase mimicry (**RM**) and increases her attack cost. To demonstrate how the number of bots required to achieve $G_{strength}$ differ for defense strategies, we implement a simple adversary program that manages the bots and adapts to the defense changes at the target network. This program *increases* the number of bots in the attack; i.e., if the attack is unsuccessful (i.e., the average per-flow rate, r_{avg}, at the target network is larger than r_d), it adds more bots at the rate of one additional bot per second.

We evaluate the effectiveness and the cost of the attack against the three different defense strategies: (a) *no defense*: a strategy that only provides per-flow fairness, which is automatically achieved by TCP's congestion control mechanism; (b) *ordinary traffic engineering (TE)*: a strategy that *provisions* additional bandwidth by rerouting traffic crossing the target link (both malicious and benign) persistently as long as the flooding continues; and (c) *SPIFFY*: a strategy that performs TBE and rate-increase measurement *on demand* to test the bots. Note that ordinary TE provisions the additional bandwidth persistently without attempting to detect the bots, while the TBE operation is temporary and only for testing bots.

We utilize 130 bots and each of them have 1 Mbps of upload bandwidth limit. The per-flow rate demand for legitimate senders is $r_g = 100$ Kbps while bots have the rate demand of $r_d = 10$ Kbps for no defense and ordinary TE. However, since the attack against SPIFFY has the demand-rate mimicry goal (**RM**), its bots have the rate demand of $r_g = 100$ Kbps. The target link bandwidth is set to be 8 Mbps. The number of senders and the bottleneck bandwidth are limited by our experiment setup. Through additional packet-level simulations (Section VI-B) and flow-level simulations (Section VII-B), we show that the results from these limited-bandwidth experiments scale to large configurations.

Figure 13 shows the results of the evaluation over the three defense strategies. In the two plots, the x-axis represents the wall-clock time of the experiment. The adaptive adversary program starts from time $t = 0$, increasing its number of bots by 1 every second, if the adversary goal $G_{strength}$ is not satisfied. Figure 13a shows the average per-flow rate changes

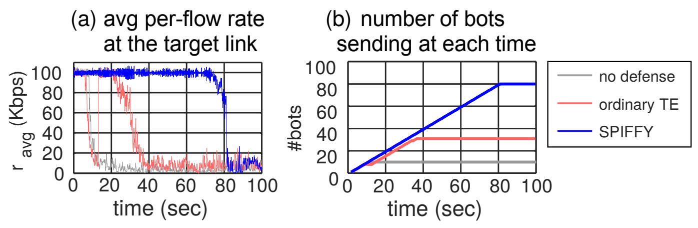

Fig. 13: Measured average per-flow rates (r_{avg}) and the number of used bots ($\#bots$) for the three defense strategies.

	Cogent	Tata	UUNET	NTT	Deutsche Telekom
#routers	196	144	48	46	38
#links	245	194	84	63	55

TABLE I: Five ISP networks used for large-scale simulations and their number of routers and links.

over time. Figure 13b illustrates the number of bots used in the attack, which represents the attack cost. The number of required bots varies widely for different defense strategies. For no defense, only 10 bots are needed to achieve the goal. For ordinary TE, initially the attack needs only 8 bots to achieve its goal. However, as soon as the target network is flooded at around $t = 12$ seconds, ordinary TE expands its defense bandwidth by a factor of three and the average per-flow rate of the target recovers the initial $r_g = 100$ Kbps. As a result, the adversary needs to further increase the number of bots up to 31. Notice that both the attack and defense costs increase roughly three times, which suggests that there is no reduction in cost asymmetry. For SPIFFY, we observe that the adversary requires 80 bots in total to achieve the rate-reduction goal $G_{strength}$ while the defense does not use additional bandwidth.[5] This shows that the rate-increase mimicry (**RM**) costs the adversary use roughly $M = 10$ times more bots to achieve the attack goal $G_{strength}$.

B. Large-Scale Flow-Level Simulations

In this section, we evaluate the feasibility of SPIFFY in large-scale flow-level simulations with up to about 200 routers. In particular, we focus on the implementation of the TBE to show that its proposed design (Section V-B) can be implemented in practical ISP networks. For scalable evaluation (e.g., millions of flooding flows and hundreds of routers), we developed a simulator that models TCP flows (defined by srcIP, dstIP, srcPort, dstPort, and protocol) as fluid flows [14]; i.e., each flow at each time epoch has its flow rate and occupies the same amount of bandwidth at all the network links it travels. We model the behavior of TCP flows by implementing the ideal fair-share rate property (i.e., allocating equal bandwidth to all competing flows) at every attack-targeted link in the network. We examine the TBE algorithm using the flow simulator with millions of flows. Our simulator models the five large ISP topologies from the Topology Zoo database [32] as shown in Table I. We use the uniform link-bandwidth model, where all links have the same bandwidth, and non-uniform model, where links in the center of the ISP topology have higher bandwidth. The simulation proceeds in discrete time ticks. At each tick, the simulation updates the rates of all flows in the network by visiting each network link and updating the rates of all flows on the link.

[5]The TBE operations use additional bandwidth at the target only *temporarily*. Thus, the increase in defense cost on average is negligible.

(in seconds)

	Cogent	Tata	UUNET	NTT	Deutsche Telekom
LP solution $M_{network}$	2,039.06	435.79	0.79	0.27	0.27
Greedy algorithm solution $\widehat{M}_{network}$	14.71	9.07	0.65	0.35	0.26

TABLE II: Execution times for LP solution $M_{network}$ and greedy algorithm solution $\widehat{M}_{network}$.

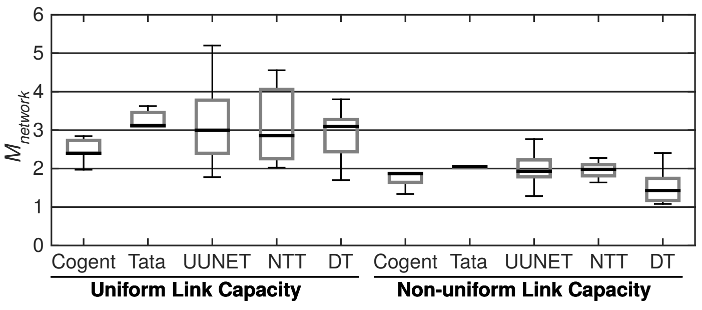

Fig. 14: $M_{network}$ values for the five ISPs in two link-bandwidth models.

Real-time operation of TBE in large networks. The TBE operation needs to calculate the new route sets in real-time; e.g., within few seconds. We evaluate the execution time to calculate the new routes using the greedy routing algorithm (i.e., Algorithm 1) and show how time efficient it is, compared to solving the optimal LP. When solving the greedy algorithm solution $\widehat{M}_{network}$, we apply a binary search; viz., Section V-A. We utilize the multi-core architecture of our SDN controller for the binary search and evaluate 12 values of m concurrently at each iteration. Table II shows the execution time for the LP solution $M_{network}$ and the greedy algorithm solution $\widehat{M}_{network}$. LP is solved with the CPLEX solver in a server-grade machine with 20 cores. As explained, LP requires an impractical amount of time for networks with large number of routers R. In contrast, the greedy algorithm with binary search requires only few seconds in general to calculate $\widehat{M}_{network}$. Even in the largest network we evaluate (i.e., Cogent), it takes only 14.7 seconds, which is less than 1 percent of the time taken by the LP solution, which is 2,039 seconds.

Optimal LP solutions $M_{network}$ and effectiveness of the TBE algorithm. We solve LP in the five ISP networks with two different link-bandwidth models. The uniform link-bandwidth model assumes the same bandwidth of 40 Gbps for all the links. To model more realistic network bandwidth provisioning, we also use the non-uniform model that assigns link bandwidth based on the *betweenness centrality* of each link. The betweenness centrality of a link is the number of shortest-path routes between all pairs of edge routers that include the link [26]. This metric represents how (logically) central the link is in the network topology. We assign 40 Gbps link bandwidth to the 33% of links with the highest centrality, 5 Gbps bandwidth to the 33% of links with the smallest centrality, and 10 Gbps bandwidth to all other links in the middle. For each case, we setup 10 different attacks,

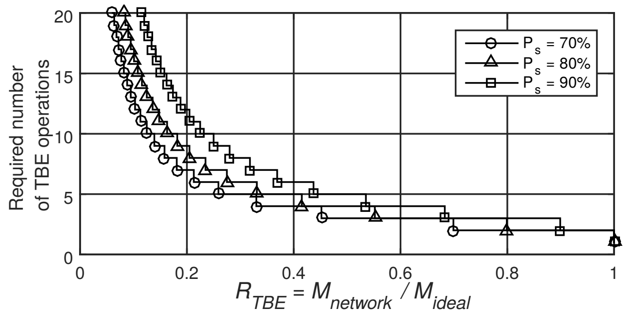

Fig. 15: Required number of TBE operations for varying $R_{TBE} = M_{network}/M_{ideal}$ and P_s.

which target 10 different links for flooding. The targeted links are chosen from the 10 links with the highest betweenness centrality in each ISP topology. We assume that 30% of bandwidth of each link is already used for underlying traffic that is unrelated to the link-flooding attack. Figure 14 shows the distribution of $M_{network}$ in the box plots. We achieve $M_{network}$ close to 3 with the uniform model while $M_{network}$ is close to 2 with the non-uniform model. The non-uniform model has smaller $M_{network}$ since it provides less bandwidth for alternative paths for TBE than the uniform model. As we will see later in this section, a small value of $M_{network} \simeq 2$ can still be effective when used with the sequential TBE. Moreover, we also evaluate the accuracy of the greedy algorithm solution $\widehat{M}_{network}$ compared to the LP solution $M_{network}$. We find that in all five ISP networks the greedy algorithm solution $\widehat{M}_{network}$ is nearly identical to the LP solution $M_{network}$, showing a difference of only few percentages (almost 1–2%). Also, we find that the new routes due to TBE need just 1 to 3 more router hops (or 4 – 24% longer average route length in the target network) compared to the original routes before TBE.

Operation of randomized sequential TBE. We also evaluate how many times the TBE operations need to be performed to test the majority of all senders contributing to the congest on the target link. As explained in Section V, the required number of randomized sequential TBE operations, n, depends on the ratio $R_{TBE} = M_{network}/M_{ideal}$ and the percentage P_s of senders that must be tested at least once. Figure 15 shows the required number of TBE operations for various R_{TBE} and P_s. As expected, the higher R_{TBE}, the lower n. Moreover, the lower P_s, the smaller the number of TBE operations are required. Based on the observation that the five ISPs we evaluate have $M_{network}$ in between 2.21 and 3.19, we conclude that roughly 4 – 10 TBE rounds are required.

VIII. Discussion

A. Handling Cost-Insensitive Irrational Adversaries

Countermeasures for cost-insensitive (e.g., state-sponsored) or irrational adversaries require collaborative defenses, which involve communication and coordination among different ISPs. Collaborative defenses are necessary because cost-insensitive adversaries can flood a large number of (if not all) the links to a target, by definition. These defenses are more complex to deploy and more expensive because they require bilateral deployment agreements and coordination, added infrastructure (e.g., CoDef [38]), and increased run-time detection costs. Hence, collaborative defenses should not be the first-line of defense.

To date, all available evidence indicates that the majority of the link-flooding adversaries are in fact rational; e.g., cost sensitive and stealthy as witnessed by the desire to use high-amplification, low-cost attack traffic afforded by amplification attacks that use hard-to-track sources [45]. Since rational adversaries can always be deterred, SPIFFY can become an effective first-line of defense for an ISP.

B. Legitimate Senders with Application-layer Rate Adaptation

Although the rate of a legitimate sender is mainly determined by its TCP window control (as discussed and evaluated in Section VI-B), application programs might also adapt their data rates and thus affect the send rate of the legitimate senders. Such application-layer rate adaptation can potentially reduce the effectiveness of bot detection. For example, let us assume a legitimate sender that has suffered from severe congestion for few minutes and its application program has adapted (i.e., reduced) its data rate to a low degraded rate. In such a case, if the adaptation of the application-layer data rate is slow (e.g., few minutes), our bot detection mechanism might miss the send rate increase of the sender and false identify the sender as a bot.

In practice, video streaming is one of the most popular examples of application-layer rate adaptation. Today's most video streaming services periodically (e.g., 1 – 10 seconds) adjust the quality of a video stream to provide continuous playback under various range of available network bandwidth. An experimental evaluation study with major video streaming services showed that the rate adaptation algorithm can adapt its bitrate very quickly [9]. In particular, Netflix, the most popular video streaming service, is shown to quickly adapt to the sudden spikes of short-term (e.g., 2, 5, and 10 seconds) bandwidth expansion; viz., Figure 12 in [9]. Based on this experimental evidence, we believe that SPIFFY is effective to most legitimate senders with application-layer rate adaptation.

C. Robustness against Multiple Link-Flooding Attacks

Rational, cost-sensitive adversaries might also target multiple links concurrently to achieve higher damage to end targets.[6] In such cases, individual links interact with the SPIFFY test at each ISP. Thus, all the security analyses in Section III-B are applicable to the multiple link-flooding attacks. That is, the attacks must satisfy the rate-increase mimicry goal **RM** to circumvent the tests launched by each link target and this causes the attacks to increase the number of bots by a factor of M for flooding each target link. If multiple link targets are located in the same network, SPIFFY can simply measure the per-sender rate changes as if single link in the network is being targeted.

Multiplexed link attack: Although simple extensions of single link attacks can be easily handled, when an adversary carefully multiplexes the attack flows across her bots and a small number (e.g., 10) target links, SPIFFY can detect the bots only *probabilistically*. The steps of a multiplexed link

[6]Note that the irrational, cost-insensitive adversaries that flood a large number of (if not all) links to the targets are discussed in Section VIII-A

attack are as follows. Each bot floods multiple (up to M) link targets concurrently while using only $1/M$ of its upload bandwidth (say u) for each link target. When one of the link targets starts testing a bot, the bot allocates all of its upload bandwidth u to the flows that are dedicated to that link by pausing all the other attack flows. As a result, the bot can pass the test by the link, and after the test the bot can continue to flood the multiple link targets again. The bots in this attack can be detected probabilistically when a bot is tested by more than one target link simultaneously since it cannot increase rates for the two tests simultaneously. The detection of the multiplexed link attack can be improved to become *deterministic* when the multiple SPIFFY operations in different ISPs exchange the sender information they are testing (e.g., via standardized channels [41]) and test same bots simultaneously.

D. Multiple senders sharing a single IP address

When multiple senders in a local network are served by a single NAT gateway, they share the same source IP. If some bots are located in the same local network, they might identified as legitimate senders by SPIFFY because their flows are mixed with other legitimate flows under the same source IP.

In such cases, we examine whether a particular IP address is shared or not; e.g., via existing mechanisms [15]. Then, we perform the SPIFFY test with *finer* granularity of *flow aggregates*; e.g., per-source-destination, per-source-protocol. This enables SPIFFY to test different smaller sender groups than the entire sender set sharing the same source IP and thus improve the bot-identification accuracy even when senders share a single source IP.

IX. Related Work

We first summarize link-flooding attacks targeting core network links. We then categorize existing defense approaches, which are insufficient to defend against the link-flooding attacks. Last, we list several other flooding attacks and discuss their relationships with SPIFFY.

Link-flooding attacks. The link-flooding attacks that target the core network links are the main threat model we consider in this paper. The Coremelt attack [51] utilizes bots to send attack traffic to other bots. This Coremelt attack coordinates large numbers of bot pairs in a way that their communication paths share the links in the Internet core. The Crossfire attack [31] coordinates bots to send legitimate-looking low-rate traffic to the attacker-chosen publicly accessible servers (e.g., HTTP servers) in a way that their routes cross the link targets in the core Internet. All attack flows are indistinguishable since they are the connections to the legitimate open services and low-rate protocol-conforming flows.

Profiling-based defense approaches. This type of mechanisms maintain the profiles of legitimate traffic based on their flow rates, source IPs, destination IPs, protocols, etc., and distinguish attack traffic from legitimate one. PSP [20] constructs the profiles of the rate history of origin-destination pairs in a single ISP network. ACC (or Pushback) [40] utilizes the rate/history of flow aggregates at the intermediate routers. Moreover, anomaly detection [37] monitors the unusual changes in the entropy of packet header bits to detect attack traffic. All attack-profiling approaches, however, can be circumvented by an adversary who can freely choose attack sources, destinations, and protocols, and completely conform to network protocols (e.g., TCP congestion control) while successfully flooding a target.

Proof-of-work defense approaches. Proof-of-work mechanisms enable target routers (or servers) to force *both* bots and legitimate senders to submit proofs (e.g., computation resource for solving puzzles [42], [55] or network bandwidth resource [54]) that were performed before allowing access to the target. These systems are fundamentally different from SPIFFY for multiple reasons: (1) proof-of-work systems limit the traffic generation of legitimate senders while SPIFFY limits that of bots only; (2) proof-of-work systems create significant waste of computation/bandwidth at traffic sources, which might be prohibitive in energy/bandwidth-starved devices, whereas SPIFFY does not waste any unnecessary resources; and (3) proof-of-work systems require significant modifications to the current Internet, including senders, routers, end-servers, and protocols between them, while SPIFFY does not require such modifications.

Capacity-provisioning defense approaches. Instead of attempting to distinguish attack traffic or prioritize legitimate traffic, the target network could simply provide more bandwidth either via physical bandwidth addition or traffic engineering for both legitimate and attack traffic [27]. However, capacity provisioning alone cannot be effective because (1) it does not reduce the attack-defense cost asymmetry (i.e., N-times-provision of bandwidth at the target network requires the same factor (N) of increase of attack bandwidth for successful attacks), and (2), if bandwidth is provisioned via traffic engineering, the additional bandwidth available for the provisioning in typical ISP networks is very small (e.g., 2 – 4 times) as shown in our evaluation in Section VII.

Collaboration-based defense approaches. These mechanisms require global collaboration among networks under different ownership. CoDef [38] requires coordination between the attack-target ISP and the ISPs hosting traffic sources to mitigate attack traffic. SENSS [10] assumes the collaboration between the target ISP and the intermediate ISPs in the Internet to control the incoming flooding traffic. Although ISP collaboration in general is not readily available in the current Internet whose relationship between ISPs (e.g., [57]) are competitive rather than collaborative, increasing interest in ISP collaboration for DDoS defense (viz., IETF DDoS Open Threat Signaling (DOTS) Working Group [41]) may make these mechanisms feasible in the near future.

SDN-based DDoS defense approaches. Orthogonal to the above defense approaches, recent proposals utilized SDN-based network architectures to handle DDoS attacks; e.g., Bohatei [23]. However, their focus (in particular Bohatei) is on elastic scaling of defense for *legacy* DDoS attacks and they do not tackle the link-flooding attacks we consider here.

Other attacks. Although our main focus in this paper is on the link-flooding attacks that target the Internet core, here we list some other DDoS attacks and discuss why SPIFFY is not appropriate for defending against these attacks. First, *opt-ack* attacks [49] send TCP Ack packets with the sequence numbers that the receiver (attacker) has not yet received (i.e., optimistic Acks) to servers to cause servers to send at a high rate without noticing the congestion in the network. Such high-rate flows would be considered as protocol non-conforming at the bottlenecked link and thus could be detected by existing "elephant" detection mechanisms [34]. Second, *TCP amplification attacks* [35] exploit protocol and implementation vulnerabilities in the TCP connection setup and amplify the attack traffic volume. Since this attacks also generate high-rate attack flows, they can be detected in a similar way the opt-ack attacks are detected. Third, *shrew attacks* [36] and its variants (e.g., CXPST [46]) create low-rate flows with bursts of packets that can cause the synchronized TCP timeouts at a targeted link. These attacks are very different from other volumetric DDoS attacks (including our persistent the link-flooding attacks) and therefore could be handled by other existing dedicated countermeasures [56]. Lastly, denial-of-service attacks targeting SDN data-control channels have different goals from the link-flooding attacks we consider here and thus need to be addressed independently [50].

X. Conclusion

Handling Internet link-flooding attacks is an extremely challenging problem because adversaries (1) can generate attacks flows that are indistinguishable from legitimate flows; and (2) incur a much lower cost for attacks than the defense for countermeasures. We propose a system called SPIFFY that forces an adversary to choose between two unpleasant alternatives, namely either allow bot detection or accept an increase in attack cost. SPIFFY removes the key enabler of link-flooding attacks, namely undetectable attacks at low cost, and provides an effective first-line of defense for the common case; i.e., for attacks by the cost-sensitive adversaries who wish to remain undetected. Hence, more complex and expensive collaborative defenses among ISPs are required only for the far fewer, cost-insensitive or irrational adversaries.

References

[1] Denial of Service Protection: Business protection for business as usual. *http://www.business.att.com/enterprise/Service/network-security/threat-vulnerability-management/ddos-protection/*.

[2] IBM ILOG CPLEX Optimization Studio. *http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html*.

[3] Internet Transit Pricing: Historical and Projected. *http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php*.

[4] ns2: The Network Simulator. *http://www.isi.edu/nsnam/ns/*.

[5] Open Flow. *https://www.opennetworking.org*.

[6] Open vSwitch. *http://openvswitch.org*.

[7] POX, Python-based OpenFlow Controller. *http://www.noxrepo.org/pox/about-pox/*.

[8] Afanasyev, A., Tilley, N., Reiher, P., and Kleinrock, L. Host-to-host congestion control for TCP. *IEEE Communications Surveys & Tutorials* (2010).

[9] Akhshabi, S., Begen, A. C., and Dovrolis, C. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. ACM MMSys* (2011).

[10] Alwabel, A., Yu, M., Zhang, Y., and Mirkovic, J. SENSS: observe and control your own traffic in the Internet. In *Proc. ACM SIGCOMM* (2014).

[11] Appenzeller, G., Keslassy, I., and McKeown, N. *Sizing router buffers*. Proc. ACM SIGCOMM, 2004.

[12] Awduche, D., Chiu, A., Elwalid, A., Widjaja, I., and Xiao, X. Overview and principles of Internet traffic engineering. Tech. rep., RFC 3272, may, 2002.

[13] Awerbuch, B., and Khandekar, R. Greedy distributed optimization of multi-commodity flows. In *Proc. ACM PODC* (2007).

[14] Baccelli, F., and Hong, D. Flow level simulation of large IP networks. In *Proc. IEEE INFOCOM* (2003).

[15] Bellovin, S. M. A technique for counting NATted hosts. In *Proc. ACM SIGCOMM Workshop on Internet Measurement* (2002).

[16] Brakmo, L. S., and Peterson, L. L. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* (1995).

[17] Bright, P. Can a DDoS break the Internet? Sure... just not all of it. In *Ars Technica* (April 2, 2013). http://arstechnica.com/security/2013/04/can-a-ddos-break-the-internet-sure-just-not-all-of-it/.

[18] Caballero, J., Grier, C., Kreibich, C., and Paxson, V. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *Proc. USENIX Security* (2011).

[19] Cao, J., Cleveland, W. S., Gao, Y., Jeffay, K., Smith, F. D., and Weigle, M. Stochastic models for generating synthetic HTTP source traffic. In *Proc. IEEE INFOCOM* (2004).

[20] Chou, J. C.-Y., Lin, B., Sen, S., and Spatscheck, O. Proactive Surge Protection: a Defense Mechanism for Bandwidth-based Attacks. *IEEE/ACM Transactions on Networking* (2009).

[21] CISCO. How To Calculate Bandwidth Utilization Using SNMP. http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/8141-calculate-bandwidth-snmp.html.

[22] Donnet, B., Luckie, M., Mérindol, P., and Pansiot, J.-J. Revealing MPLS tunnels obscured from traceroute. *ACM SIGCOMM CCR 42*, 2 (2012), 87–93.

[23] Fayaz, S. K., Tobioka, Y., Sekar, V., and Bailey, M. Bohatei: Flexible and Elastic DDoS Defense. In *Proc. USENIX Security* (2015).

[24] Fayazbakhsh, S. K., Chiang, L., Sekar, V., Yu, M., and Mogul, J. C. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. USENIX NSDI* (2014).

[25] Fortz, B., Rexford, J., and Thorup, M. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine* (2002).

[26] Freeman, L. C. A set of measures of centrality based on betweenness. *Sociometry* (1977).

[27] Gkounis, D., Kotronis, V., and Dimitropoulos, X. Towards Defeating the Crossfire Attack using SDN. In *arXiv:1412.2013* (2014).

[28] Goodin, D. How extorted e-mail provider got back online after crippling DDoS attack. In *Ars Technica* (November 10, 2015). http://arstechnica.com/security/2015/11/how-extorted-e-mail-provider-got-back-online-after-crippling-ddos-attack/.

[29] Gupta, A., Vanbever, L., Shahbaz, M., Donovan, S. P., Schlinker, B., Feamster, N., Rexford, J., Shenker, S., Clark, R., and Katz-Bassett, E. SDX: A software defined Internet exchange. In *Proc. ACM SIGCOMM* (2014).

[30] Indyk, P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. IEEE FOCS* (2000).

[31] Kang, M. S., Lee, S. B., and Gligor, V. D. The Crossfire Attack. In *Proc. IEEE S&P* (2013).

[32] Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., and Roughan, M. The Internet Topology Zoo. *IEEE JSAC* (2011).

[33] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based change detection: methods, evaluation, and applications. In *Proc. ACM IMC* (2003).

[34] Krishnan, R., Durrani, M., and Phaal, P. Real-time SDN Analytics for DDoS mitigation. *Open Networking Summit* (2014).

[35] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Hell of a handshake: Abusing TCP for reflective amplification DDoS attacks. In *Proc. USENIX WOOT* (2014).

[36] Kuzmanovic, A., and Knightly, E. W. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proc. ACM SIGCOMM* (2003).

[37] Lakhina, A., Crovella, M., and Diot, C. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM CCR* (2005).

[38] Lee, S. B., Kang, M. S., and Gligor, V. D. CoDef: collaborative defense against large-scale link-flooding attacks. In *Proc. ACM CoNEXT* (2013).

[39] Ludwig, R., and Katz, R. H. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM Computer Communication Review* (2000).

[40] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM CCR* (2002).

[41] Mortensen, A. DDoS Open Threat Signaling Requirements. *IETF draft-mortensen-threat-signaling-requirements-00* (2015).

[42] Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. Portcullis: protecting connection setup from denial-of-capability attacks. In *Proc. ACM SIGCOMM* (2007).

[43] Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D. A., Kern, R., Kumar, H., Zikos, M., Wu, H., Kim, C., and Karri, N. Ananta: Cloud Scale Load Balancing. In *Proc. ACM SIGCOMM* (2013).

[44] Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., and Yu, M. SIMPLE-fying middlebox policy enforcement using SDN. In *Proc. ACM SIGCOMM* (2013).

[45] Rossow, C. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Proc. NDSS* (2014).

[46] Schuchard, M., Mohaisen, A., Foo Kune, D., Hopper, N., Kim, Y., and Vasserman, E. Y. Losing control of the Internet: using the data plane to attack the control plane. In *Proc. NDSS* (2010).

[47] Sen, S., Shue, D., Ihm, S., and Freedman, M. J. Scalable, optimal flow routing in datacenters via local link balancing. In *Proc. ACM CoNEXT* (2013).

[48] Sharafat, A. R., Das, S., Parulkar, G., and McKeown, N.

[48] Sharafat, A. R., Das, S., Parulkar, G., and McKeown, N. MPLS-TE and MPLS VPNs with OpenFlow. In *ACM SIGCOMM CCR* (2011).

[49] Sherwood, R., Bhattacharjee, B., and Braud, R. Misbehaving TCP receivers can cause Internet-wide congestion collapse. In *Proc. ACM CCS* (2005).

[50] Shin, S., and Gu, G. Attacking software-defined networks: A first feasibility study. In *Proc. ACM HotSDN* (2013).

[51] Studer, A., and Perrig, A. The Coremelt attack. In *Proc. ESORICS* (2009).

[52] Tan, K., Song, J., Zhang, Q., and Sridharan, M. A compound TCP approach for high-speed and long distance networks. In *Proc. IEEE INFOCOM* (2006).

[53] Valancius, V., Lumezanu, C., Feamster, N., Johari, R., and Vazirani, V. V. How many tiers?: pricing in the Internet transit market. In *ACM SIGCOMM CCR* (2011).

[54] Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., and Shenker, S. DDoS defense by offense. In *Proc. ACM SIGCOMM* (2006).

[55] Wang, X., and Reiter, M. K. Defending against denial-of-service attacks with puzzle auctions. In *Proc. IEEE S&P* (2003).

[56] Yang, G., Gerla, M., and Sanadidi, M. Defense against low-rate TCP-targeted denial-of-service attacks. In *Proc. IEEE ISCC* (2004).

[57] Yegulalp, S. Level 3 accuses Comcast, other ISPs of 'deliberately harming' broadband service. In *InfoWorld* (May 6, 2014).

[58] Yu, M., Jose, L., and Miao, R. Software Defined Traffic Measurement with OpenSketch. In *Proc. USENIX NSDI* (2013).

[59] Zhang, M., Dusi, M., John, W., and Chen, C. Analysis of UDP traffic usage on Internet backbone links. In *Proc. IEEE SAINT* (2009).