NetFence: Preventing Internet Denial of Service from Inside Out

Xin Liu
Dept. of Computer Science
Duke University
xinl@cs.duke.edu

Xiaowei Yang
Dept. of Computer Science
Duke University
xwy@cs.duke.edu

Yong Xia
Networking Systems Group
NEC Labs China
xia_yong@nec.cn

ABSTRACT

Denial of Service (DoS) attacks frequently happen on the Internet, paralyzing Internet services and causing millions of dollars of financial loss. This work presents NetFence, a scalable DoS-resistant network architecture. NetFence uses a novel mechanism, secure congestion policing feedback, to enable robust congestion policing inside the network. Bottleneck routers update the feedback in packet headers to signal congestion, and access routers use it to police senders' traffic. Targeted DoS victims can use the secure congestion policing feedback as capability tokens to suppress unwanted traffic. When compromised senders and receivers organize into pairs to congest a network link, NetFence provably guarantees a legitimate sender its fair share of network resources without keeping per-host state at the congested link. We use a Linux implementation, ns-2 simulations, and theoretical analysis to show that NetFence is an effective and scalable DoS solution: it reduces the amount of state maintained by a congested router from per-host to at most per-(Autonomous System).

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms

Design, Security

Keywords

Internet, Denial-of-Service, Capability, Congestion Policing

1. INTRODUCTION

Large-scale Denial of Service (DoS) attacks remain as a potent threat to the Internet. A survey from Arbor Networks shows that DoS attacks continue to grow in both scale and sophistication [4]. The largest observed attack reached 49Gbps in 2009, a 104% growth over the past two years. The survey also ranks DoS attacks as the largest anticipated threat in the next 12 months. This result is not surprising, as tens of gigabits flooding traffic could easily overwhelm most links, routers, or sites on the Internet.

The destructive nature of DoS attacks has brought forth a fundamental research challenge: how can we design an open network architecture that is resistant to large-scale DoS attacks? There have been several proposals addressing this challenge [5,27,35,48,47,3]. These proposals enable DoS victims to suppress attack traffic using network capabilities or filters, but when malicious sender-receiver pairs collude to flood a link, the best defense mechanism these systems can offer is per-host queuing at the flooded link to separate legitimate traffic from attack traffic. This solution faces a scalability challenge, as a flooded router may forward packets for millions of (malicious and legitimate) end systems.

This paper presents the design and evaluation of NetFence, a scalable DoS-resistant network architecture. NetFence provably guarantees each sender its fair share of bandwidth without keeping per-host state at bottleneck routers even when malicious senders and receivers collude into pairs to flood the network. It also enables DoS victims to suppress unwanted traffic as in a capability-based system [48, 35]. A key departure of NetFence from previous work is that it places the network at the first line of DoS defense rather than relies on end systems (be it senders or receivers) to suppress attack traffic.

The NetFence design places a robust traffic policing control loop inside the network (§ 3 and § 4). Packets carry unforgeable congestion policing feedback stamped by routers that suffer excessive congestion (caused either by DoS attacks or other reasons, which NetFence does not distinguish). Access routers at the trust boundaries between the network and end systems examine the feedback and police the senders' traffic. A malicious sender cannot gain more than its fair share of bandwidth even if it colludes with a compromised receiver, because it cannot spoof valid congestion policing feedback. Innocent DoS victims can use the unforgeable congestion policing feedback as capability tokens to suppress the bulk of unwanted traffic, by not returning the feedback to malicious senders. To be fail-safe in case access routers are compromised, NetFence uses Autonomous System (AS)-level queues (or rate-limiters) to separate traffic from different source ASes, limiting DoS damage to the ASes that harbor the compromised routers.

We have implemented NetFence in Linux and evaluated its overhead and performance using theoretical analysis (§ 3.4), testbed experiments, and large-scale simulations (§ 6). Our analysis shows that regardless of attackers' strategies, NetFence provides a legitimate sender its fair share of bottleneck bandwidth. The simulation results correlate well with this analysis, and also show that NetFence performs similarly to state-of-the-art capability- or filter-plus-fair-queuing DoS defense systems [27, 48]. Our Linux pro-

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.

totype benchmarking results show that NetFence's per-packet processing overhead is low.

These results suggest that NetFence is an effective and scalable DoS solution. NetFence's bottleneck routers have $O(1)$ per-packet computational overhead, and maintain at most per-AS state (more scalable design alternatives exist as discussed in § 4.5), while previous work requires these bottleneck routers to keep per-host state to protect legitimate traffic. One concern for the NetFence design is that access routers need to keep per-(sender, bottleneck link) state (§ 3), but we show in § 5.1 today's access routers can meet such scalability requirements.

The key contributions of this paper include a new DoS defense primitive: secure congestion policing feedback, and based on it, the construction of a robust, network-based, closed-loop congestion policing architecture that scalably and effectively limits the damage of DoS flooding attacks. With a closed-loop design, NetFence can flexibly place different functionalities at different locations: lightweight attack detection and congestion signaling at bottleneck links, and congestion policing that requires per-(sender, bottleneck link) state at access routers. This design makes it scale much better than previous open-loop approaches that employ per-host queuing at bottleneck routers [27, 48].

2. ASSUMPTIONS AND GOALS

Before we present the design of NetFence, we first describe its threat model, assumptions, and design goals.

2.1 Threat Model and Assumptions

Flood-based network attacks: NetFence focuses on mitigating network-layer flooding attacks where attackers send excessive traffic to exhaust network resources such as link capacity or router processing power. It does not aim to mitigate DoS attacks that exploit application vulnerabilities to exhaust end system resources.

Strong adversary: We assume that attackers can compromise both end systems and routers. Compromised end systems involved in an attack can grow into millions; they may launch brute-force or strategic flooding attacks. For instance, they may disguise attack traffic as legitimate traffic, launch on-off attacks, or collude into sender-receiver pairs to send flooding traffic. Attack traffic may or may not be distinguishable from legitimate traffic.

We make two assumptions to assist NetFence's design.

Trust: We assume that routers managed by the network are much less likely to be compromised than end systems. We thus place policing functions on routers rather than end systems. As a tradeoff for scalability, we treat each AS as a trust and fate sharing unit. When compromised routers exist, we aim to localize the damage to the ASes that harbor compromised routers rather than protect all the legitimate hosts within such ASes.

Line-speed lightweight cryptography: We assume that symmetric key cryptography can be supported at line-speed. Some current hardware can support AES operations at 40Gbps [20], and the latest Intel Westmere processors have native support for AES [21].

2.2 Goals

NetFence aims to meet several design goals. It is these goals that distinguish NetFence from previous work.

i) Guaranteed network resource fair share: When DoS victims can identify attack traffic, we aim to enable them to suppress the attack traffic near the origins. This prevents attack traffic from wasting network resources. When DoS victims fail to identify attack traffic, or attackers collude into sender-receiver pairs to flood the network, we resort to a weaker goal to guarantee a legitimate sender its fair share of network resources. That is, for any link of capacity C shared by N (legitimate and malicious) senders, each sender with sufficient demand should be guaranteed at least $O(\frac{C}{N})$ bandwidth share from that link. This mitigates the effect of large-scale DoS attacks from denial of service to predictable delay of service.

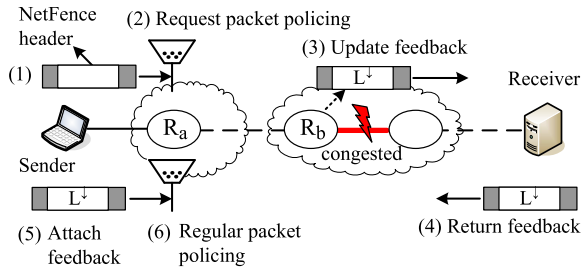

Figure 1: The NetFence architecture. Packets carry unspoofable congestion policing feedback stamped by bottleneck routers (R_b in this figure). Access routers (R_a) use the feedback to police senders' traffic, preventing malicious senders from gaining unfair shares of bottleneck capacity. DoS victims can use the congestion policing feedback as capability tokens to suppress unwanted traffic.

ii) Open network: NetFence aims to keep the network open to new applications, and thus places the attack traffic identification function at the receivers to avoid false positives introduced by in-network traffic classification. This goal is also shared by previous work [3, 48, 5].

iii) Scalable and lightweight: NetFence may face millions of attackers that attempt to congest a single link. To be effective at such a scale, it does not assume that a router always has sufficient resources to warrant per-flow or per-host state management. It aims to keep little or no state in the core network and avoid heavyweight operations such as per-flow/host fair queuing in the core network. To facilitate high-speed router implementation, NetFence aims to incur low communication, computation, and memory overhead.

iv) Robust: NetFence should be robust against both simple, brute-force flooding attacks and sophisticated ones that attempt to bypass or abuse NetFence itself.

v) Incrementally adoptable: We aim to make NetFence incrementally deployable on today's Internet. Specifically, we aim to provide early adopters immediate deployment benefits: they can form an "overlay" network of deployed regions and benefit collectively from the deployment. We aim not to require hop-by-hop deployment from a congested link to compromised end systems to be effective, unlike [30].

vi) Network self-reliant defense: We aim for a self-reliant solution that depends on only routers in the network, not other infrastructures such as trusted host hardware [2] or DNS extensions [35]. Our hypothesis is that extra dependencies increase security risk and may create deployment deadlocks. That is, without the deployment or upgrade of other infrastructures, the design is not effective. Hence, there is little incentive to deploy it, and vice versa.

3. ARCHITECTURE

In this section, we present an overview of the NetFence architecture, and defer design details to § 4.

3.1 System Components

NetFence has three types of packets: *request* packets, *regular* packets, and *legacy* packets. The first two, identified by a special

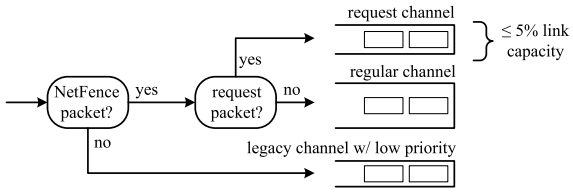

Figure 2: Each NetFence router keeps three channels.

protocol number in the IP header, have a shim NetFence header between their IP and upper-layer protocol headers. The NetFence header carries unforgeable congestion policing feedback generated by the network (§ 3.2 and § 4.4). A NetFence-ready sender sends request and regular packets, while a non-NetFence sender sends only legacy packets.

Each NetFence router, depicted in Figure 2, keeps three channels, one for each of the three packet types discussed above. To motivate end systems to upgrade, the NetFence design gives legacy channel lower forwarding priority than the other two. To prevent request flooding attacks from denying legitimate requests, NetFence has a priority-based backoff mechanism for the request channel (§ 4.2). The request channel is also limited to consume no more than a small fraction (5%) of the output link capacity, as in [48,35].

NetFence places its feedback and policing functions at bottleneck and access routers that are either inside the network or at the trust boundaries between the network and end systems. It does not place any trusted function at end systems. As shown in Figure 1, a NetFence sender starts an end-to-end communication by sending request packets to its NetFence-ready receiver (Step 1). The access router inserts the nop feedback in the NetFence header of the packet (Step 2, § 4.1). Along the path, a bottleneck router might modify the feedback, in a way similar to TCP ECN [37] (Step 3). After the receiver returns the feedback to the sender (Step 4), the sender can send valid regular packets that contain the feedback (Step 5). In Step 4, two-way protocols like TCP can piggyback the returned feedback in their data packets, while one-way transport protocols such as UDP must send extra, low-rate feedback packets from a receiver to a sender.

A NetFence router periodically examines each output link to decide if an attack is happening at the link. It uses a combination of link load and packet loss rate as an attack indicator (§ 4.3.1). If an attack is detected, NetFence starts a monitoring cycle, which lasts until i) no more attack is detected during the cycle, and ii) the cycle has existed for an extended period (typically a few hours) after the most recent attack is detected. During a monitoring cycle, the mon congestion policing feedback (containing the link ID l, an $action$ field, etc.) is stamped into the NetFence header of all the passing request/regular packets (§ 4.3.2). The sender's regular packets must include this mon feedback to be considered valid, and they will be policed by the access router (Step 6, § 4.3.3).

An access router maintains one rate limiter for every sender-bottleneck pair to limit a sender's regular traffic traversing a bottleneck link. The router uses an Additive Increase and Multiplicative Decrease (AIMD) algorithm to control the rate limit: it keeps the rate limit constant within one pre-defined control interval (a few seconds); across control intervals, it either increases the rate limit additively or decreases it multiplicatively, depending on the particular mon feedback it receives (§ 4.3.4). We use AIMD to control the rate limit because it has long been shown to converge onto efficiency and fairness [11]. Other design choices exist; they have different cost-performance tradeoffs, and are discussed in [28].

When no attack is detected, a downstream router will not modify the nop feedback stamped by an access router. When the sender obtains the nop feedback and presents it back to its access router in a packet, the packet will not be rate-limited. That is, when no attack happens, NetFence stays in idle state. The overhead during such idle periods is low, because 1) the NetFence header is short (20 bytes) (§ 6.1); 2) the bottleneck attack detection mechanism only involves a packet counter and a queue sampler; and 3) an access router only needs to stamp and validate (not rate limit) the NetFence header for each packet. Only when an attack is detected at a bottleneck link, does NetFence activate its policing functions, which add additional processing overhead at bottleneck and access routers. We show the overhead benchmarking results in § 6.2.

3.2 Unforgeable Congestion Policing Feedback

Congestion policing feedback must be made unforgeable so that malicious nodes cannot evade NetFence's traffic policing functions. NetFence achieves this goal using efficient symmetric key cryptography. An access router inserts a periodically changing secret in a packet's NetFence header. A bottleneck router uses this secret to protect its congestion policing feedback, and then erases the secret. The access router, knowing the secret, can validate the returned feedback. We describe the details of this design in § 4.4, and discuss how to limit the effect of compromised access routers in § 4.5.

3.3 Congestion Feedback as Capability

If a DoS victim can identify and desires to bar attack traffic, NetFence's unspoofable congestion policing feedback also serves as a capability token: a receiver can return no feedback to a malicious sender. Because the malicious sender cannot forge valid feedback, it cannot send valid regular packets. It can at most flood request packets to a destination, but an access router will use a priority-based policing scheme to strictly limit a sender's request traffic rate (§ 4.2). Alternatively, it can simply flood to congest its local area network, but this attack is easy to detect and the damage is confined to the local area network.

3.4 Fair Share Guarantee

With the above-described closed-loop network architecture, we are able to prove that NetFence achieves per-sender fairness for single bottleneck scenarios.

Theorem*: Given G legitimate and B malicious senders sharing a bottleneck link of capacity C, regardless of the attack strategies, any legitimate sender g with sufficient demand eventually obtains a capacity fair share $\frac{\nu_g \rho\, C}{G+B}$, where $0 < \nu_g \leq 1$ is a parameter determined by how efficient the sender g's transport protocol (*e.g.*, TCP) utilizes the rate limit allocated to it, and ρ is a parameter close to 1, determined by NetFence's implementation-dependent AIMD and attack detection parameters.*

Due to lack of space, we briefly describe why this theorem holds, but leave a detailed proof in the technical report [28].

Proof sketch: In NetFence, an access router keeps one rate limiter for each sender-bottleneck pair when a monitoring cycle is triggered during attack times. Based on the unspoofable congestion feedback from the bottleneck, the access router dynamically adjusts the rate limits using a robust AIMD algorithm (§ 4.3.4). Since AIMD has been shown to converge onto efficiency and fairness [11], all the rate limits will eventually converge to the fair share of the bottleneck capacity. Thus, any sender, whether legitimate or malicious, can send at most as fast as its fair share rate.

4. DESIGN DETAILS

In this section, we show the design details of NetFence. For clarity, we first present the design assuming unforgeable congestion

policing feedback and non-compromised routers. We then describe how to make congestion policing feedback unforgeable and how to handle compromised routers. Key notations used to describe the design are summarized in Figure 3.

4.1 Congestion Policing Feedback

NetFence uses three types of congestion policing feedback:

- nop, indicating no policing action is needed;
- $L^{\downarrow}$, indicating the link L is overloaded, and an access router should reduce the traffic traversing L;
- $L^{\uparrow}$, indicating the link L is underloaded, and an access router can allow more traffic traversing L.

We refer to $L^{\uparrow}$ and $L^{\downarrow}$ as the mon feedback. Each congestion policing feedback includes a timestamp to indicate its freshness.

4.2 Protecting the Request Channel

Attackers may simply flood request packets to congest downstream links. NetFence mitigates this attack with two mechanisms. First, it limits the request channel on any link to a small fraction (5%) of the link's capacity, as in [48, 35]. This prevents request packets from starving regular packets. Second, it combines packet prioritization and priority-based rate limiting to ensure that a legitimate sender can always successfully transmit a request packet if it waits long enough to send the packet with high priority. This mechanism ensures that a legitimate user can obtain the valid congestion policing feedback needed for sending regular packets.

In NetFence, a sender can assign different priority levels to its request packets. Routers forward a level-k packet with higher priority than lower-level packets, but the sender is limited to send level-k packets at half of the rate of level-(k-1) packets. An access router installs per-sender token-based rate limiters to impose this rate limit. It removes 2^{k-1} tokens from a request packet rate limiter when admitting a level-k packet. Level-0 packets are not rate-limited, but they have the lowest priority.

This request channel policing algorithm guarantees that a legitimate sender can eventually send a request packet to a receiver regardless of the number of attackers [35]. It holds because the arrival rate of request packets decreases exponentially as their priority level increases. Thus, the arrival rate of high priority request packets will eventually be smaller than the request channel capacity.

NetFence does not use computational puzzles as in [35]. This is because computational resources may be scarce [13], especially in busy servers and handheld devices. In addition, NetFence's design has the flexibility that an access router can configure different token refill rates for different hosts on its subnet. Legitimate servers could be given a higher rate to send more high priority request packets without purchasing additional CPU power.

When an access router forwards a request packet to the next hop, it stamps the nop feedback into the packet, ensuring that a sender can obtain valid feedback if the receiver desires to receive from it.

4.3 Protecting the Regular Channel

Malicious senders may flood regular packets when they can obtain valid congestion policing feedback from their colluding receivers. We describe how to mitigate this attack.

4.3.1 A Monitoring Cycle

When a router suspects that its outgoing link L is under attack, it starts a monitoring cycle for L. That is, it marks L as in the mon state and starts updating the congestion policing feedback in packets that traverse L (§ 4.3.2). Once a sender's access router receives such feedback, it will start rate limiting the sender's regular packets that will traverse the link L (§ 4.3.3).

Name	Value	Meaning
l_1	1 ms	level-1 request packet rate limit
I_{lim}	2 s	Rate limiter ctrl interval length
w	4 s	Feedback expiration time
Δ	12 kbps	Rate limiter additive incr
δ	0.1	Rate limiter multiplicative decr
p_{th}	2%	Packet loss rate threshold
Q_{lim}	0.2s $\times$ link bw	Max queue length
$minthresh$	$0.5\,Q_{lim}$	RED algorithm parameter
$maxthresh$	$0.75\,Q_{lim}$	RED algorithm parameter
w_q	0.1	EWMA weight for avg queue length

Figure 3: Key parameters and their values in our implementation.

It is difficult to detect if L is under an attack because the attack traffic may be indistinguishable from legitimate traffic. In NetFence, L's router infers an attack based on L's utilization and the loss rate of regular packets. If L is well-provisioned and its normal utilization is low (a common case in practice), it can be considered as under an attack when its average utilization becomes high (*e.g.*, 95%); if L always operates at or near full capacity, its router can infer an attack when the regular packets' average loss rate p exceeds a threshold p_{th}. A link's average utilization and p can be calculated using the standard Exponentially Weighted Moving Average (EWMA) algorithm [18]. The threshold p_{th} is a local policy decision of L's router, but it should be sufficiently small so that loss-sensitive protocols such as TCP can function well when no attack is detected. Attackers may launch a mild attack and evade the detection by keeping p below p_{th}, but the damage is also limited.

When the attack detection is based on the packet loss rate p, a flash crowd may also be considered as an attack. We do not distinguish these two because it is too difficult to do so. As shown by our simulation results (§ 6), starting a monitoring cycle for a link does not have much negative impact on a legitimate sender.

It is undesirable to infinitely keep a monitoring cycle due to the added overhead. Thus, a NetFence router terminates a link L's monitoring cycle when L is no longer under attack (*e.g.*, $p < p_{th}$) for a sufficiently long period of time T_b. The router will mark L as in the nop state and stop updating the congestion policing feedback in packets traversing L. Similarly, an access router will terminate a rate limiter (src, L) if it has not received any packet with the $L^{\downarrow}$ feedback and the rate limiter has not discarded any packet for T_a seconds.

Routers should set T_a and T_b to be significantly longer (*e.g.*, a few hours) than the time it takes to detect an attack (T_d). This is because attackers may flood the network again after T_a (or T_b) seconds. By increasing the ratio of the monitored period $min(T_a, T_b)$ to the unprotected period T_d, we reduce the network disruption time. Network disruption during an attack detection period cannot be eliminated unless compromised senders are patched up, but we do not assume routers have this ability.

4.3.2 Updating Congestion Policing Feedback

When a link L is in the mon state, its router R_b uses the following ordered rules to update the congestion policing feedback in any request/regular packet traversing L:

1. If the packet carries nop, stamp $L^{\downarrow}$;
2. Otherwise, if the packet carries $L'^{\downarrow}$ stamped by an upstream link L', do nothing;
3. Otherwise, if L is overloaded, stamp $L^{\downarrow}$.

The router R_b never stamps the $L^{\uparrow}$ feedback. As we will see in § 4.3.3, only an access router stamps $L^{\uparrow}$ when forwarding a packet. If the $L^{\uparrow}$ feedback reaches the receiver of the packet, it indicates that the link L is not overloaded, because otherwise the router R_b would replace the $L^{\uparrow}$ feedback with the $L^{\downarrow}$ feedback.

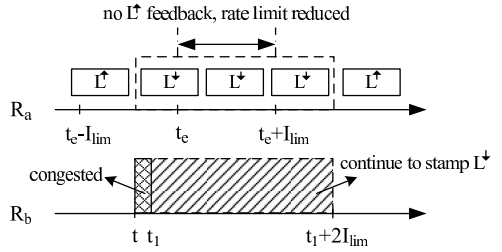

Figure 4: Once a router R_b encounters congestion between time $[t, t_1]$, it will continuously stamp the $L^{\downarrow}$ feedback until $t_1 + 2I_{lim}$.

A packet may cross multiple links in the mon state. The access router must ensure that the sender's rate does not exceed its legitimate share at any of these links. The second rule above allows NetFence to achieve this goal, gradually. This is because the first link L_1 on a packet's forwarding path that is both overloaded and in the mon state can always stamp the $L_1^{\downarrow}$ feedback, and downstream links will not overwrite it. When the $L_1^{\downarrow}$ feedback is presented to an access router, the router will reduce the sender's rate limit for the link L_1 until L_1 is not overloaded and does not stamp $L_1^{\downarrow}$. This would enable the next link (L_2) on the path that is both in the mon state and overloaded to stamp $L_2^{\downarrow}$ into the packets. Gradually, a sender's rate will be limited such that it does not exceed its fair share on any of the on-path links in the mon state.

4.3.3 Regular Packet Policing at Access Routers

A sender src's access router polices the sender's regular packets based on the congestion policing feedback in its packets. If a packet carries the nop feedback, indicating no downstream links require congestion policing, the packet will not be rate-limited. Otherwise, if it carries $L^{\uparrow}$ or $L^{\downarrow}$, it must pass the rate limiter (src, L).

We implement a rate limiter as a queue whose de-queuing rate is the rate limit, similar to a leaky bucket [44]. We use the queue to absorb traffic bursts so that bursty protocols such as TCP can function well with the rate limiter. We do not use a token bucket because it allows a sender to send short bursts at a speed exceeding its rate limit. Strategic attackers may synchronize their bursts to temporarily congest a link, leading to successful on-off attacks.

When an access router forwards a regular packet to the next hop, it resets the congestion policing feedback. If the old feedback is nop, the access router refreshes the timestamp of the feedback. If the old feedback is $L^{\downarrow}$ or $L^{\uparrow}$, the access router resets it to $L^{\uparrow}$. This design reduces the computational overhead at the link L's router, as it does not update a packet's feedback if L is not overloaded.

For simplicity, NetFence uses at most one rate limiter to police a regular packet. One potential problem is that a flow may switch between multiple rate limiters when its bottleneck link changes. We discuss this issue in § 4.3.5.

4.3.4 Robust Rate Limit Adjustment

The $L^{\uparrow}$ and $L^{\downarrow}$ feedback enables an access router to adjust a rate limiter (src, L)'s rate limit r_{lim} with an AIMD algorithm. A strawman design would decrease r_{lim} multiplicatively if the link L is overloaded and stamps the $L^{\downarrow}$ feedback, or increase it additively otherwise. However, a malicious sender can manipulate this design by hiding the $L^{\downarrow}$ feedback to prevent its rate limit from decreasing.

To address this problem, we periodically adjust a rate limit, use $L^{\uparrow}$ as a robust signal to increase the rate limit, and ensure that a sender cannot obtain valid $L^{\uparrow}$ feedback for a full control interval if its traffic congests the link L. Let I_{lim} denote the control interval length for rate adjustment on an access router. Suppose a downstream bottleneck router R_b has a link L in the mon state. R_b monitors L's congestion status using a load-based [46] or a loss-based algorithm such as Random Early Detection (RED) [18]. If R_b detects congestion between time t and t_1, it will stamp the $L^{\downarrow}$ feedback into all packets traversing L from time t until two control intervals after t_1: $t_1 + 2I_{lim}$, even if it has considered the link not congested after t_1. This hysteresis ensures that if a sender congests a link L during one control interval, it will only receive the $L^{\downarrow}$ feedback in the following control interval, as shown in Figure 4.

For each rate limiter (src, L), the access router R_a keeps two state variables: t_s and $hasIncr$, to track the feedback it has received. The variable t_s records the start time of the rate limiter's current control interval, and $hasIncr$ records whether the rate limiter has seen the $L^{\uparrow}$ feedback with a timestamp newer than t_s. At the end of each control interval, R_a adjusts the rate limiter (src, L)'s rate limit r_{lim} as follows:

1. If $hasIncr$ is true, R_a compares the throughput of the rate limiter with $\frac{1}{2}r_{lim}$. If the former is larger, r_{lim} will be increased by Δ; otherwise, r_{lim} will remain unchanged. Checking the rate limiter's current throughput prevents a malicious sender from inflating its rate limit by sending slowly for a long period of time.
2. Otherwise, R_a will decrease r_{lim} to $(1 - \delta)r_{lim}$.

We discuss how to set the parameters Δ, δ, etc. in § 4.6.

We now explain why this AIMD algorithm is robust, *i.e.*, a malicious sender cannot gain unfair bandwidth share by hiding the $L^{\downarrow}$ feedback: if a sender has sent a packet when a link L suffers congestion, the sender's rate limit for L will be decreased. Suppose L's router R_b detects congestion and starts stamping the $L^{\downarrow}$ feedback at time t, and let t_e denote the finishing time of an access router's control interval that includes the time t, as shown in Figure 4. R_b will stamp the $L^{\downarrow}$ feedback between $[t, t_1 + 2I_{lim}]$. Since $t_e \in [t, t + I_{lim}]$, a sender will only receive the $L^{\downarrow}$ feedback for packets sent during the control interval $[t_e, t_e + I_{lim}]$, because $t_e \geq t$ and $t_e + I_{lim} < t_1 + 2I_{lim}$. It can either present the $L^{\downarrow}$ feedback newer than t_e to its access router, or present one older than t_e, or not send a packet. All these actions will cause its rate limit to decrease according to the second rule above.

A legitimate sender should always present $L^{\uparrow}$ feedback to its access router as long as the feedback has not expired, even if it has received newer $L^{\downarrow}$ feedback. This design makes a legitimate sender mimic an aggressive sender's strategy and ensures fairness among all senders.

4.3.5 Handling Multiple Bottlenecks

When a flow traverses multiple links in the mon state, the flow's access router will instantiate multiple per-(sender, bottleneck link) rate limiters for the sender. The present NetFence design sends a regular packet to only one rate limiter for simplicity, but it may overly limit a sender's sending rate in some cases. This is because when a sender's packets carry the congestion policing feedback from one of the bottleneck links, all other rate limiters stay idle. The sender's access router will reduce their rate limits, if they are idle for longer than a full control interval, as described above (§ 4.3.4). Consequently, the idle rate limiters' rate limits may become smaller than a sender's fair share rates at those bottleneck links. When a sender's bottleneck link changes, it may obtain less than fair share bandwidth at the new bottleneck initially, until its rate limit for the new bottleneck converges. Additionally, if a sender's bottleneck link changes frequently due to congestion dynamics, its packets may switch between different rate limiters. If those rate limiters' rate limits differ greatly, it may be difficult for a transport protocol such as TCP to fully utilize its available bandwidth.

We have considered various solutions to address this problem. One simple solution is to allow a packet to carry all feedback from all the bottleneck links on its path. An access router can then pass the packet through all the on-path rate limiters, each receiving its own feedback and policing the packet independently. This solution requires a longer and variable-length NetFence header. Another one is for an access router to infer the on-path bottleneck links of a packet based on history information and send the packet through all the inferred rate limiters.

We do not include these solutions in the core design for simplicity. We describe the details of these solutions in [28], and use simulations to evaluate how NetFence performs against those alternative designs when there are multiple bottlenecks. The results suggest that NetFence's performance is acceptable. Thus, we consider it a worthy tradeoff to keep the design simple.

4.4 Securing Congestion Policing Feedback

Congestion policing feedback must be unforgeable. Malicious end systems should not be able to forge or tamper the feedback, and malicious routers should not be able to modify or remove the feedback stamped by other routers. The NetFence design uses efficient symmetric key cryptography to achieve these goals.

Feedback format: A congestion policing feedback consists of five key fields as shown in Figure 5: $mode$, $link$, $action$, ts, and MAC. When the $mode$ field is nop, it represents the nop feedback. When the $mode$ field is mon, the $link$ field indicates the identifier (an IP address) of the corresponding link L, and the $action$ field indicates the detailed feedback: if $action$ is $incr$ ($decr$), it is the $L^{\uparrow}$ ($L^{\downarrow}$) feedback. The ts field records a timestamp, and the MAC field holds a MAC signature that attests the feedback's integrity.

In addition to the five key fields, a mon feedback also includes a field $token_{nop}$. We explain the use of this field later in this section.

Stamping nop feedback: When an access router stamps the nop feedback, it sets $mode$ to nop, $link$ to a null identifier $link_{null}$, $action$ to $incr$, ts to its local time, and uses a time-varying secret key K_a known only to itself to compute the MAC:

$$token_{nop} = MAC_{K_a}(src, dst, ts, link_{null}, nop) \qquad (1)$$

The MAC computation covers both the source and destination addresses to prevent an attacker from re-using valid nop feedback on a different connection.

Stamping $L^{\uparrow}$ feedback: When an access router stamps the $L^{\uparrow}$ feedback, the $mode$ field is already mon, and the $link$ field already contains the link identifier L. The router sets $action$ to $incr$ and ts to its local time, and computes the MAC field using the secret key K_a:

$$token_{L^{\uparrow}} = MAC_{K_a}(src, dst, ts, L, mon, incr) \qquad (2)$$

The router also inserts a $token_{nop}$ as computed in Eq (1) into the $token_{nop}$ field.

Stamping $L^{\downarrow}$ feedback: When a link L's router R_b stamps the $L^{\downarrow}$ feedback, it sets $mode$ to mon, $link$ to L, $action$ to $decr$, and computes a new MAC value using a secret key K_{ai} shared between its AS and the sender's AS:

$$token_{L^{\downarrow}} = MAC_{K_{ai}}(src, dst, ts, L, mon, decr, token_{nop}) \qquad (3)$$

The shared secret key K_{ai} is established by piggybacking a distributed Diffie-Hellman key exchange in BGP as in [26]. The router R_b includes $token_{nop}$ stamped by the sender's access router in its MAC computation, and erases it afterwards to prevent malicious downstream routers from overwriting its feedback.

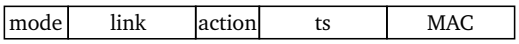

Figure 5: The key congestion policing feedback fields.

If R_b is an AS internal router that does not speak BGP, it may not know K_{ai}. In this case, R_b can leave the MAC and $token_{nop}$ fields unmodified and let an egress border router of the AS update their values when the packet exits the AS. This design reduces the management overhead to distribute K_{ai} to an internal router R_b.

Validating feedback: When a source access router receives a regular packet, it first validates the packet's congestion policing feedback. If the feedback is invalid, the packet will be treated as a request packet and subject to per-sender request packet policing.

A feedback is considered invalid if its ts field is more than w seconds older than the access router's local time t_{now}: $|t_{now} - ts| > w$, or if the MAC field has an invalid signature. The MAC field is validated using Eq (1) and Eq (2) for the nop and $L^{\uparrow}$ feedback, respectively. To validate $L^{\downarrow}$ feedback, the access router first re-computes the $token_{nop}$ using Eq (1), and then re-computes the MAC using Eq (3). The second step requires the access router to identify the link L's AS in order to determine the shared secret key K_{ai}. We can use an IP-to-AS mapping tool [33] for this purpose, as the feedback includes the link L's IP address.

4.5 Localizing Damage of Compromised Routers

The NetFence design places enforcement functions that include feedback validation and traffic policing at the edge of the network to be scalable. However, if an access router is compromised, attackers in its subnet or itself may misbehave to congest the network. NetFence addresses this problem by localizing the damage to the compromised AS. If an AS has a compromised router, we consider the AS as compromised, and do not aim to provide guaranteed network access for that AS's legitimate traffic.

A NetFence router can take several approaches to localize the damage of compromised ASes, if its congestion persists after it has started a monitoring cycle, a signal of malfunctioning access routers. One approach is to separate each source AS's traffic into different queues. This requires per-AS queuing. We think the overhead is affordable because the present Internet has only about 35K ASes [7]. We may replace per-AS queuing with per-AS rate limiting and set the rate limits by periodically computing each AS's max-min fair share bandwidth on the congested link as in [30]. Another more scalable approach is to use a heavy-hitter detection algorithm such as RED-PD [31] to detect and throttle high-rate source ASes. A heavy-hitter detection algorithm is suitable in this case because legitimate source ASes will continuously reduce their senders' traffic as long as they receive the $L^{\downarrow}$ feedback. The detected high-rate ASes are likely to be the compromised ASes that do not slow down their senders.

All these approaches require a router to correctly identify a packet's source AS, which can be achieved using an IP-to-AS mapping tool if the packet's source IP address is not spoofed. NetFence uses Passport [26] to prevent source address spoofing. A Passport header is inserted between IP and the NetFence header. Passport piggybacks a distributed Diffie-Hellman key exchange in the inter-domain routing system to enable each pair of ASes to share a secret key. A source AS uses a key it shares with an AS on the path to a destination to compute a secure MAC and inserts it into a packet's Passport header. Each AS on the path can verify that a packet is originated from the source AS by validating the corresponding MAC. NetFence also uses Passport to establish the shared secret keys between ASes to secure the congestion policing feedback (§ 4.4).

4.6 Parameter Settings

Figure 3 summarizes the main parameters in the NetFence design and their values used in our implementation. The level-1 request packets (l_1) are rate limited at one per 1 ms. A request packet size is estimated as 92 bytes that includes a 40-byte TCP/IP header, a 28-byte NetFence header (Figure 6) and a 24-byte Passport header [26]. We set the control interval I_{lim} to 2 seconds, one order of magnitude larger than a typical RTT ($<$ 200ms) on the Internet. This allows an end-to-end congestion control mechanism such as TCP to reach a sender's rate limit during one control interval. We do not further increase I_{lim} because a large control interval would slow down the rate limit convergence.

The rate limit AI parameter Δ can be neither too small nor too large: a small Δ would lead to slow convergence to fairness; a large Δ may result in significant overshoot. We set Δ to 12Kbps because it works well for our targeted fair share rate range: 50Kbps $\sim$ 400Kbps. A legitimate sender may abort a connection if its sending rate is much lower than 50Kbps, and 400Kbps should provide reasonable performance for a legitimate sender during DoS flooding attacks. The rate limit MD parameter δ is set to 0.1, a value much smaller than TCP's MD parameter 0.5. This is because a router may stamp the $L^{\downarrow}$ feedback for two control intervals longer than the congestion period (§ 4.3.4).

We set the attack detection threshold p_{th} to 2%, since at this packet loss rate, a TCP flow with 200ms RTT and 1500B packets can obtain about 500Kbps throughput [34]. We set a link's maximum queue length Q_{lim} to 200ms $\times$ the link's capability. We use a loss-based algorithm RED to detect a link's congestion status. It is our future work to implement a load-based algorithm (*e.g.*, [46]).

5. ANALYSIS

In this section, we analyze the scalability and security of NetFence, and discuss the incremental deployment issues.

5.1 Scalability

As a closed-loop design, NetFence can place different functions at different locations to provide per-sender fairness. It places per-sender traffic policing at access routers, and lightweight congestion detection, feedback stamping, and AS-level policing at bottleneck routers. In contrast, per-host fair queuing, an open-loop solution used in previous work [48, 27], does not have this flexibility. Every bottleneck router must keep per-host queues to provide per-sender (or per-receiver) fairness. There are only 35K ASes on today's Internet [7], while the number of compromised hosts involved in a DoS attack could reach millions [17]. Thus, compared to per-host fair queuing, NetFence can significantly reduce the amount of state kept by a bottleneck router.

However, NetFence access routers need to perform per-(sender, bottleneck link) rate limiting. Our calculation suggests that with today's hardware technology, they can afford to do so and will not become a new scaling bottleneck. While we do not have accurate data to estimate the number of bottlenecks a sender traverses during attack times, we think 100 links per legitimate sender is a reasonable upper bound. An access router can aggregate a sender's rate limiters by bottleneck links' prefixes if a sender needs more than 100 rate limiters. If an access router serves 10K end hosts, it will have at most one million rate limiters in total. Each rate limiter needs about 24 bytes of memory for state variables (1 bit for $hasIncr$, 8 bytes for two timestamps, 4 bytes for the rate limit, and 12 bytes for a queue object) and another 1500 bytes to queue at least one packet. The total amount of memory requirement is less than 2GB, and we can use fast DRAM for this purpose as access routers' line speeds are typically slower than those of core routers.

The processing overhead of an access router is also acceptable. The per-packet processing time on our benchmarking PC is less than $1.3\mu s$ during attack times (§ 6.2). This translates into a throughput of 770K packets per second, or more than 9 Gbps, assuming 1500-byte packet size and CPU is the throughput bottleneck. Implementing the cryptographic operations in hardware can further improve an access router's throughput.

5.2 Security

Next we summarize how NetFence withstands various attacks.

5.2.1 Malicious End Systems

Forgery or Tampering: Malicious end systems may attempt to forge valid congestion policing feedback. But NetFence protects congestion policing feedback with MAC signatures. As long as the underlying MAC is secure, malicious end systems cannot spoof valid feedback. A malicious sender may selectively present $L^{\uparrow}$ or hide $L^{\downarrow}$ to its access router, but NetFence's robust AIMD algorithm (§ 4.3.4) prevents it from gaining a higher rate limit.

Evading attack detection: Malicious end systems may attempt to prevent a congested router from starting a monitoring cycle. This attack is ineffective when a well-provisioned router uses high link utilization to detect attacks. When an under-provisioned router uses the packet loss rate to detect attacks, NetFence limits the damage of this attack with a low loss detection threshold p_{th} (§ 4.3.1).

On-off attacks: Attackers may attempt to launch on-off attacks. In a macroscopic on-off attack, attackers may flood the network again after a congested router terminates a monitoring cycle. NetFence uses a prolonged monitor cycle (§ 4.3.1) to mitigate this attack. In a microscopic on-off attack, attackers may send traffic bursts with a short on-off cycle, attempting to congest the network with synchronized bursts, yet maintaining average sending rates lower than their rate limits. Our theoretical bound in § 3 and simulation results in § 6.3.2 both show that the shape of attack traffic cannot reduce a legitimate user's guaranteed bandwidth share, because a sender cannot send faster than its rate limit at any time (§ 4.3.3), and NetFence's robust rate limit adjustment algorithm (§ 4.3.4) prevents a sender from suddenly increasing its actual sending rate.

5.2.2 Malicious On-path Routers

A malicious router downstream to a congested link may attempt to remove or modify the $L^{\downarrow}$ feedback stamped by a congested router in order to hide upstream congestion. But such attempts will make the feedback invalid, because the router does not know the original $token_{nop}$ value needed to compute a valid MAC (§ 4.4).

A malicious on-path router may discard packets to completely disrupt end-to-end communications, duplicate packets, or increase packet sizes to congest downstream links. It may also change the request packet priority field in a NetFence header to congest the request channel on downstream links. Preventing such attacks requires Byzantine tolerant routing [36], which is not NetFence's design goal. Instead, we aim to make these attacks detectable. Passport [26], the source authentication system NetFence uses, partially protects the integrity of a packet and enables duplicate detection. It includes a packet's length and the first 8 bytes of a packet's transport payload (which includes the TCP/UDP checksum) in its MAC computation. We can further extend Passport's MAC computation to include NetFence's request packet priority field to protect it.

5.3 Incremental Deployment

NetFence can be incrementally deployed by end systems and routers. Since the NetFence header is a shim layer between IP

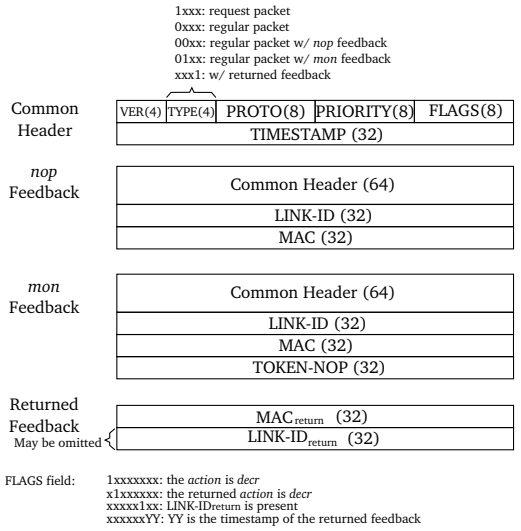

Figure 6: The NetFence header format.

and upper layer protocols, legacy applications need not be modified. Legacy routers can ignore the NetFence header and forward packets using the IP header. Routers at congested links and access routers need to be upgraded, but well-provisioned routers that can withstand tens of Gbps attack traffic may not need to upgrade. The deployment can take a bump-in-the-wire approach, by placing inline boxes that implement NetFence's enforcement functions in front of the routers that require upgrading. Middleboxes such as firewalls need to be configured to permit NetFence traffic.

NetFence provides deployment incentives to both end systems and ASes, because legacy traffic is treated by deployed ASes with lower priority (Figure 2). Deployed ASes can form a trusted overlay network and protect each other's legitimate traffic within their networks. Their traffic is not protected at undeployed networks, encouraging them to direct traffic to other deployed ASes using BGP.

6. IMPLEMENTATION AND EVALUATION

We have implemented NetFence prototypes in Linux and in the ns-2 simulator. Next we evaluate the NetFence header and packet processing overhead with our Linux implementation, and use ns-2 simulations to show how effective NetFence mitigates DoS attacks.

6.1 NetFence Header

Figure 6 shows the format of a NetFence header in our Linux implementation. A full NetFence header from a sender to a receiver includes a forward header and a return header. The forward header includes the congestion policing feedback on the forward path from the sender to the receiver, and the return header includes the reverse path information from the receiver to the sender. Most fields are self-explained. A NetFence header is implemented as a shim layer between IP and an upper-layer protocol, and the PROTO field describes the upper-layer protocol (*e.g.*, TCP or UDP). The unit of a timestamp is one second.

The return header may be omitted to reduce overhead if the sender has previously returned the latest feedback to the receiver. Even when the return header is present, it does not always include all the fields. If the returned feedback is nop, the LINK-ID$_{return}$ field will be omitted because it is zero, and one bit in the FLAGS field indicates this omission.

A NetFence header only includes the last two bits of the returned timestamp to save header space. In the subsequent packets from the sender to the receiver, the sender's access router will reconstruct the full timestamp from its local time and the returned two bits, assuming that the timestamp is less than four seconds older than its current time. With this implementation, a NetFence header is 20 bytes in the common case when the feedback is nop for both the forward and return paths. In the worst case that the feedback is mon for both paths, the header is 28 bytes long.

Packet Type	Router Type	Processing Overhead (ns/pkt)	
		NetFence	TVA+
request	bottleneck	w/o attack: 0 w/ attack: 492	389
	access	546	
regular	bottleneck	w/o attack: 0 w/ attack: 554	791
	access	w/o attack: 781 w/ attack: 1267	

Figure 7: NetFence implementation micro-benchmarking results.

6.2 Micro-benchmarking

We have implemented NetFence in Linux using XORP [19] and Click [24]. We modified XORP's BGP module to establish the pairwise symmetric keys shared between ASes. We added the data packet processing logic into Click and ran Click routers in the kernel space for packet forwarding. XORP communicates with Click via the `/click` file system. We added a module between the IP and transport layers on end-hosts to handle NetFence headers. This design keeps upper-layer TCP/UDP protocols and legacy applications unmodified. We use AES-128 as a secure MAC function due to its fast speed and available hardware support [20, 21].

We benchmark the Linux implementation on Deterlab [14] with a three-node testbed. A source access router A and a destination C are connected via a router B. The B—C link is the bottleneck with a capacity of 5Mbps. Each node has two Intel Xeon 3GHz CPUs and 2GB memory. To benchmark the processing overhead without attacks, we send 100Kbps UDP request packets and 1Mbps UDP regular packets from A to C respectively. To benchmark the overhead in face of DoS attacks, we send 1Mbps UDP request packets and 10Mbps UDP regular packets simultaneously.

The benchmarking results are shown in Figure 7. With NetFence, when there is no attack, a request packet does not need any extra processing on the bottleneck router B, but it introduces an average overhead of 546ns on the access router A because the router must stamp the nop feedback into the packet. A regular packet does not incur any extra processing overhead on the bottleneck router either, but it takes the access router 781ns on average to validate the returned feedback and generate a new one. When the bottleneck link enters the mon state during attack times, the bottleneck router takes 492ns to process a 92B request packet, or at most 554ns to process a 1500B regular packet. The access router takes on average 1267ns to process a regular packet at attack times.

The performance of a capability system TVA+ [27] on the same topology is also shown in Figure 7 for comparison. We can see that the processing overhead introduced by NetFence is on par with that of TVA+. Note that we do not show the result when TVA+ caches capabilities, because such caching requires per-flow state on routers, while NetFence does not have this requirement.

These results show that NetFence's per-packet overhead is low. The CPU-intensive operations are primarily AES computation. Since there exists commercial hardware that can support AES operations at 40Gbps [20], we expect that NetFence's per-packet processing will not become a performance bottleneck. We note that the benchmarking results do not include the Passport overhead, as a Passport header can be updated by inline boxes near an AS's ingress and egress border routers [26].

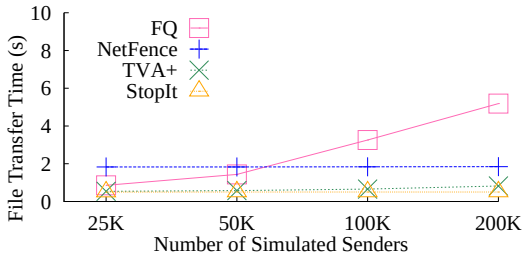

Figure 8: The average transfer time of a 20KB file when the targeted victim can identify and wish to remove the attack traffic. The file transfer completion ratio is 100% in all simulated systems.

6.3 Mitigating DoS Flooding Attacks

Next we evaluate how well NetFence mitigates various DoS flooding attacks using ns-2 simulations. We also compare NetFence with three other representative DoS mitigation schemes:

TVA+ : TVA+ [48, 27] is a network architecture that uses network capabilities and per-host fair queuing to defend against DoS flooding attacks. TVA+ uses hierarchical queuing (first based on the source AS and then based on the source IP address) at congested links to mitigate request packet flooding attacks, and per-receiver fair queuing to mitigate authorized traffic flooding attacks in case (colluding or incompetent) receivers fail to stop attack traffic.

StopIt : StopIt [27] is a filter and fair queuing based DoS defense system. A targeted victim can install network filters to stop unwanted traffic. Similar to TVA+, in case receivers fail to install filters, StopIt uses hierarchical queuing (first based on the source AS and then based on the source IP address) at congested links to separate legitimate traffic from attack traffic.

Fair Queuing (FQ) : Per-sender fair queuing at every link provides a sender its fair share of the link's bandwidth. We use fair queuing to represent a DoS defense mechanism that aims to throttle attack traffic to consume no more than its fair share of bandwidth.

We have implemented TVA+ and StopIt as described in [27, 48]. We use the Deficit Round Robin (DRR) algorithm [39] to implement fair queuing because it has $O(1)$ per packet operation overhead. In our simulations, attackers do not spoof source addresses because NetFence uses Passport [26] to prevent spoofing. Thus, routers could queue attack traffic separately from legitimate traffic.

6.3.1 *Unwanted Traffic Flooding Attacks*

We first simulate the attack scenario where the attackers directly flood a victim, but the victim can classify the attack traffic, and uses the provided DoS defense mechanism: capabilities in TVA+, secure congestion policing feedback in NetFence, and filters in StopIt, to block the unwanted traffic.

We desire to simulate attacks in which thousands to millions of attackers flood a well provisioned link. However, we are currently unable to scale our simulations to support beyond several thousand nodes. To address this limitation, we adopt the evaluation approach in [48]. We fix the number of nodes, but scale down the bottleneck link capacity proportionally to simulate the case where the bottleneck link capacity is fixed, but the number of attackers increases.

We use a dumb-bell topology in which ten source ASes connect to a destination AS via a transit AS. Each source AS has 100 source hosts connected to a single access router. The transit AS has two routers R_{bl} and R_{br}, and the destination AS has one victim destination host. The link between R_{bl} and R_{br} is the bottleneck link, and all other links have sufficient capacity to avoid congestion. We vary the bottleneck link capacity from 400Mbps to 50Mbps to simulate the scenario where 25K $\sim$ 200K senders (both legitimate and malicious) share a 10Gbps link. Each sender's fair share bandwidth varies from 400Kbps $\sim$ 50Kbps, which is NetFence's targeted operating region. The propagation delay of each link is 10ms.

In the simulations, each sender is either a legitimate user or an attacker. To stress-test our design, we let each source AS have only one legitimate user that repeatedly sends a 20KB file to the victim using TCP. We let each attacker send 1Mbps constant-rate UDP traffic to the victim. We measure the effectiveness of a DoS defense system using two metrics: 1) the average time it takes to complete a successful file transfer; and 2) the fraction of successful file transfers among the total number of file transfers initiated. We set the initial TCP SYN retransmission timeout to 1 second, and abort a file transfer if the TCP three-way handshake cannot finish after nine retransmissions, or if the entire file transfer cannot finish in 200 seconds. We terminate a simulation run when the simulated time reaches 4000 seconds.

For each DoS defense system we study, we simulate the most effective DoS flooding attacks malicious nodes can launch. In case of an unwanted traffic flooding attack, the most effective flooding strategy in NetFence and TVA+ is the request packet flooding attack. Under this attack, each NetFence sender needs to choose a proper priority level for its request packets. We make an attacker always select the highest priority level at which the aggregate attack traffic can saturate the request channel. A legitimate sender starts with the lowest priority level and gradually increases the priority level if it cannot obtain valid congestion policing feedback.

Figure 8 shows the simulation results. The average file transfer completion ratio is omitted because all file transfers complete in these simulations. As can be seen, StopIt has the best performance, because the attack traffic is blocked near the attack sources by network filters. TVA+ and NetFence also have a short average file transfer time that only increases slightly as the number of simulated senders increases. This is because in a request packet flooding attack, as long as a legitimate sender has one request packet delivered to the victim, it can send the rest of the file using regular packets that are not affected by the attack traffic. The average file transfer time in NetFence is about one second longer than that in TVA+, because a legitimate sender will initially send a level-0 request packet that cannot pass the bottleneck link due to attackers' request packet floods. After one second retransmission backoff, a sender is able to retransmit a request packet with sufficiently high priority (level-10) to pass the bottleneck link. Attackers cannot further delay legitimate request packets, because they are not numerous enough to congest the request channel at this priority level.

Figure 8 also shows that FQ alone is an ineffective DoS defense mechanism. With FQ, the average file transfer time increases linearly with the number of simulated senders, as each packet must compete with the attack traffic for the bottleneck bandwidth.

These results show that NetFence performs similarly to capability-based and filter-based systems when targeted victims can stop the attack traffic. A legitimate sender may wait longer in NetFence to successfully transmit a request packet than in TVA+ or StopIt. This is because NetFence uses coarse-grained exponential backoff to schedule a request packet's transmission and set its priority, while TVA+ uses fine-grained but less scalable per-sender fair queuing to schedule a request packet's transmission, and StopIt enables a victim to completely block unwanted traffic.

6.3.2 *Colluding Attacks*

Next we present our simulation results for regular traffic flooding attacks where malicious sender-receiver pairs collude to flood the

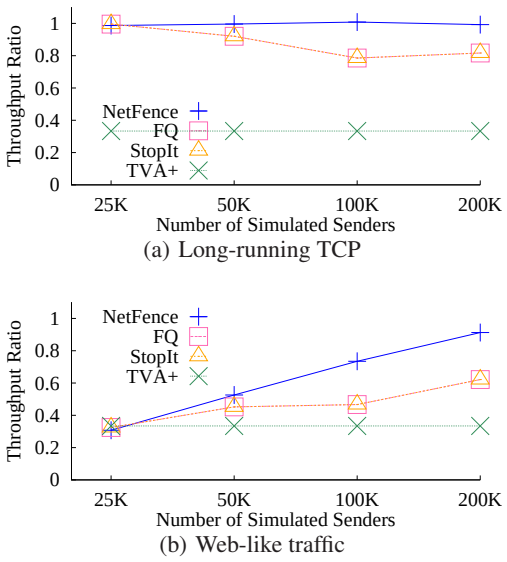

Figure 9: ***Throughput Ratio*** **between legitimate users and attackers when receivers fail to suppress the attack traffic.** ***Fairness Index*** **among legitimate users is close to 1 in all the simulations.**

network. Such attacks may also occur if DoS victims fail to identify the attack traffic.

Single Bottleneck: We use a similar topology as in the previous experiments (§ 6.3.1) to simulate colluding attacks. In this simulation topology, the router at the right-hand side of the bottleneck link R_{br} connects to one destination AS with a victim host and nine additional ASes, each having a colluding host (colluder). Each source AS has 25% legitimate users and 75% attackers, simulating the case where the attackers are numerous but there are still a reasonable number of legitimate users in each source AS.

Each legitimate user sends TCP traffic to the victim host. We simulate two types of user traffic: 1) long-running TCP, where a legitimate sender sends a single large file; 2) web-like traffic, where a sender sends small files whose size distribution mimics that of web traffic. We draw the file size distribution from a mixture of Pareto and exponential distributions as in [29], and make the interval between two file transfers uniformly distributed between 0.1 and 0.2 seconds. The maximum file size is limited to 150KB to make the experiments finish within a reasonable amount of time.

To simulate colluding attacks, we let each attacker send 1Mbps UDP traffic to a colluder. The attackers in TVA+ and NetFence send regular packets. Colluders in StopIt do not install filters to stop the attack traffic. We simulate each experiment for 4000 seconds.

When compromised nodes organize into pairs to send attack traffic, NetFence aims to guarantee each legitimate sender its fair share of the bottleneck bandwidth without keeping per-sender queues in the core network. We use two metrics to measure a DoS defense system's performance under this type of attack: 1) *Throughput Ratio*, the ratio between the average throughput of a legitimate user and that of an attacker; and 2) *Fairness Index* among legitimate users [11]. Let x_i denote a legitimate sender i's throughput, and the fairness index is defined as $(\sum x_i)^2/(n\sum x_i^2)$. The ideal throughput ratio is 1, indicating that a legitimate user obtains on average the same bottleneck bandwidth as an attacker. The ideal fairness index is also 1, indicating that each legitimate sender has the same average throughput. We only measure the fairness index among legitimate users because *Throughput Ratio* has already quantified how well a legitimate user performs relatively to an attacker.

Figure 9 shows the simulation results. The fairness index for all systems is close to 1 in all the simulations and is thus not shown in the figure. For long-running TCP, NetFence's throughput ratio is also close to 1. This result shows that NetFence provides a legitimate sender its fair share of bandwidth despite the presence of DoS flooding traffic, consistent with the theoretic analysis in § 3.4. For the web-like traffic, NetFence's throughput ratio increases gradually from 0.3 to close to 1 as the number of simulated senders increases. The throughput ratio is low when the number of senders is small, because a legitimate sender cannot fully utilize its fair share bandwidth: each sender has a large fair share of bandwidth, but a legitimate sender's web-like traffic has insufficient demand and there are gaps between consecutive file transfers.

FQ and StopIt perform exactly the same, because in these colluding attacks, they both resort to per-sender fair queuing to protect a legitimate user's traffic. However, unexpectedly, we note that they provide legitimate users less throughput than attackers even when the user traffic is long-running TCP. By analyzing packet traces, we discover that this unfairness is due to the interaction between TCP and the DRR algorithm. A TCP sender's queue does not always have packets due to TCP's burstiness, but a constant-rate UDP sender's queue is always full. When a TCP sender's queue is not empty, it shares the bottleneck bandwidth fairly with other attackers, but when its queue is empty, the attack traffic will use up its bandwidth share, leading to a lower throughput for a TCP sender.

TVA+ has the lowest throughput ratio among all systems in this simulation setting, indicating that a small number of colluders can significantly impact TVA+'s performance. This is because TVA+ uses per-destination fair queuing on the regular packet channel. With N_C colluders, a DoS victim obtains only $\frac{1}{N_C+1}$ fraction of the bottleneck capacity C at attack times, and each of the victim's G legitimate senders obtains $\frac{1}{G(1+N_C)}$ fraction of the capacity C. The attackers, however, obtain an aggregate $\frac{N_C}{(1+N_c)}$ fraction of C. If this bandwidth is shared by B attackers fairly, each will get a $\frac{N_C}{B(1+N_c)}$ fraction of the bottleneck capacity. A sender's bottleneck bandwidth share in other systems (NetFence, StopIt, and FQ) is $\frac{1}{G+B}$, and does not depend on the number of colluders N_C. In our simulations, $N_c = 9$, $G = 25\% \times 1000$, and $B = 75\% \times 1000$. A legitimate TVA+ sender obtains $\frac{1}{2500}$ of the bottleneck bandwidth, while an attacker obtains $\frac{9}{7500}$ of the bottleneck bandwidth, three times higher than a legitimate sender, as shown in Figure 9.

In these simulations, we also measure the bottleneck link utilization. The result shows that the utilization is above 90% for NetFence, and almost 100% for other systems. NetFence does not achieve full link utilization mainly because a router stamps the $L^{\downarrow}$ feedback for two extra control intervals after congestion has abated, as explained in § 4.3.4.

Multiple Bottlenecks: To evaluate NetFence's performance with multiple bottlenecks, we have also simulated colluding attacks on a parking-lot topology with two bottleneck links. The results show that in a multi-bottleneck topology, NetFence provides a reasonable share of bandwidth to a legitimate TCP sender, but this share may be less than a TCP sender's max-min fair share, because a TCP flow may switch back and forth between two rate limiters and cannot adapt quickly enough to fully use its rate limit. This performance can be improved with a more complicated design, as discussed in § 4.3.5. More discussions and simulation results for the multi-bottleneck topology can be found in [28].

Strategic Attacks: Attackers may launch sophisticated attacks (§ 5.2) than brute-force flooding attacks. We simulate microscopic

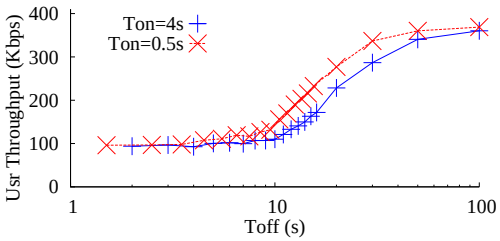

Figure 10: Average user throughput in face of microscopic on-off attacks. The user traffic is long-running TCP. There are 100K senders. Each sender's fair share bottleneck bandwidth is 100Kbps.

on-off attacks and show that with NetFence, the attack traffic's shape does not reduce a legitimate user's throughput.

The simulation topology is the same as in the previous single-bottleneck simulations. All legitimate users send long-running TCP traffic, while attackers send on-off UDP traffic. In the on-period T_{on}, an attacker sends at the rate of 1Mbps; in the off-period T_{off}, it does not send any traffic. All attackers synchronize their on-periods to create the largest traffic bursts. There are 100K simulated senders, each having a fair share bandwidth of at least 100Kbps.

In these simulations, we use two different values for T_{on}: 0.5s and 4s. For each T_{on}, we vary the off-period length T_{off} from 1.5s to 100s. Figure 10 shows the simulation results. As we can see, the average user throughput is at least a user's fair share rate as if attackers were always active (100Kbps), indicating that the attack cannot reduce a legitimate user's fair share of bandwidth. As the attackers' off-period length increases toward 100s, a legitimate user can achieve a throughput close to 400Kbps, indicating that long running TCP users can use most of the bottleneck bandwidth when the attackers' off-period is long.

7. DISCUSSION

Fair Share Bound: When a disproportionally large number (B) of attackers attack a narrow link C (*e.g.*, a million bots attacking a 1Mbps link), the fair share lower bound $O(\frac{C}{G+B})$ achieved by NetFence or per-sender fair queuing (*e.g.*, [27]) is small. However, this lower bound is still valuable, because without it, a small number of attackers can starve legitimate TCP flows on a well-provisioned link (*e.g.*, 10Gbps). Although this guarantee does not prevent large-scale DoS attacks from degrading a user's network service, it mitigates the damage of such attacks with a predictable fair share, without trusting receivers or requiring the network to identify and remove malicious traffic. Other means, like congestion quota discussed below, can be used to further throttle malicious traffic.

Congestion Quota: If we assume legitimate users have limited traffic demand while attackers aim to persistently congest a bottleneck link, we can further weaken a DoS flooding attack by imposing a congestion quota, an idea borrowed from re-ECN [9]. That is, an access router only allows a host to send a limited amount of "congestion traffic" through a bottleneck link within a period of time. Congestion traffic can be defined as the traffic that passes a rate limiter when its rate limit decreases. With a congestion quota, if an attacker keeps flooding a link, its traffic through the link will be throttled after it consumes its congestion quota.

Convergence Speed: It may take a relatively long time (*e.g.*, 100s-200s) for NetFence to converge to fairness. This is because the control interval I_{lim} is on the order of a few seconds (two seconds in our implementation), much longer than a typical RTT on the Internet. This convergence speed is acceptable in the NetFence design, because a rate limiter persists for a much longer period of time (*i.e.*, on the order of hours).

Equal Cost Multiple Path (ECMP): NetFence assumes that a flow's path is relatively stable and the bottleneck links on the path do not change rapidly. One practical concern arises as routers may split traffic among equal-cost multi-paths for load balancing. Fortunately, most ECMP implementations in practice (*e.g.*, [12]) would assign a flow's packets to the same path to avoid packet reordering. Thus, we expect NetFence to work well with ECMP.

8. RELATED WORK

At the architectural level, NetFence combines the elements of capability-based systems [48,47,35] and re-ECN/re-feedback [8,9]. In contrast to capability tokens, NetFence's congestion policing feedback carries valuable network congestion information. Re-ECN/re-feedback is a congestion policing framework that incentivizes rational senders to honestly report downstream path congestion. Routers will discard the packets from the senders that under-report downstream congestion with high probability before they reach the destinations. In contrast, NetFence is a DoS defense architecture that uses unspoofable congestion policing feedback to scalably and robustly guarantee a sender's fair share of bottleneck bandwidth in face of attacks. Attackers cannot send packets with false congestion feedback reporting no or low levels of congestion to flood a link. Instead, they can at most send packets reporting the actual levels of congestion and will not gain more bandwidth than honest senders. In addition, DoS victims can use the unspoofable feedback as capability tokens to suppress unwanted traffic. ECN-nonce [16] robustly signals congestion from the network to a honest sender even when a receiver attempts to hide congestion, while NetFence enables robust congestion signaling from congested routers to access routers when both senders and receivers are malicious.

NetFence's request packet protection mechanism is inspired by Portcullis [35] that uses computational puzzles to impose delay on senders. Differently, NetFence uses a rate limiting algorithm that does not require proof-of-work (PoW) nor a network-wide puzzle synchronization mechanism. This algorithm is similar in spirit to LazySusan [13] which substitutes resource-based PoW for latency-based PoW. Different from LazySusan, NetFence uses a sender's waiting time to set its request packet's priority level, and guarantees the eventual delivery of a legitimate request packet.

Several DoS defense systems aim to enable a victim to install network filters to stop unwanted traffic [5, 2, 27], or to control who can send to it [6]. Unlike them, NetFence does not use per-host queues at congested routers to separate legitimate traffic from attack traffic in case compromised receivers collude with malicious senders. Pushback [30] sends hop-by-hop pushback messages from a congested router to install per-(incoming interface, destination prefix) rate limiters to reduce DoS flooding traffic. NetFence does not require hop-by-hop deployment, enables a victim to suppress unwanted traffic, and provides per-sender fairness at bottleneck links: attackers cannot diffuse their traffic to many destinations to gain unfair bandwidth shares. AIP [2] uses trusted host hardware to block unwanted attack traffic, while NetFence places policing functions inside the network and does not require trusted host hardware.

Speakup [45] and Kill-Bots [22] address application-layer DoS attacks, while NetFence addresses network-layer DoS attacks. Several systems use overlay networks [1, 23, 15, 40, 38, 42] or middleboxes [10, 32] to mitigate DoS attacks against dedicated destinations. DoS mitigation products on today's market (*e.g.*, [43]) offer in-network anomaly detection and attack traffic removal services near the victims. Kreibich et al. [25] propose to use packet symme-

try to detect and remove attack traffic. This body of work requires fewer changes to routers, but NetFence can remove attack traffic near its origins and protect all destinations on the Internet once deployed. Moreover, it places the attack traffic identification function at the receivers to keep the network open to new applications.

NetFence's approach to scalability is inspired by CSFQ [41] that achieves per-flow fairness without per-flow queues in the core routers. Differently, NetFence enables DoS victims to suppress attack traffic, and provides per-sender rather than per-flow fairness.

9. CONCLUSION

This paper presents the design and evaluation of NetFence, an architecture that places the network at the first line of DoS defense. NetFence uses a key mechanism, secure congestion policing feedback, to enable scalable and robust traffic policing inside the network. Bottleneck routers use the congestion policing feedback to signal congestion to access routers, and access routers use it to robustly police senders' traffic. In case compromised senders and receivers collude in pairs to flood the network, NetFence limits the damage of this attack by providing each sender (malicious or legitimate) its fair share of bottleneck capacity without keeping per-host state at bottleneck routers. In case attackers send DoS floods to innocent victims, NetFence enables the DoS victims to use the secure congestion policing feedback as capability tokens to suppress unwanted traffic. Using a combination of a Linux implementation, simulations, and theoretic analysis, we show that NetFence is an effective DoS solution that reduces the amount of state maintained by a congested router from per-host [48, 27] to per-AS.

Acknowledgment

The authors thank Jeff Chase, David Harrison, Yongqiang Liu, and the anonymous SIGCOMM reviewers for their insightful comments, and David Oran for shepherding this paper. This work is supported in part by NSF awards CNS-0925472 and CNS-0845858.

10. REFERENCES

[1] D. Andersen. Mayday: Distributed Filtering for Internet Services. In *USENIX USITS*, 2003.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *ACM SIGCOMM*, 2008.

[3] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial of Service with Capabilities. In *ACM HotNets-II*, 2003.

[4] Arbor Networks. Worldwide Infrastructure Security Report, Volume V. `http://www.arbornetworks.com/en/research.html`, 2009.

[5] K. Argyraki and D. R. Cheriton. Scalable Network-layer Defense Against Internet Bandwidth-Flooding Attacks. *ACM/IEEE ToN*, 17(4), 2009.

[6] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! In *ACM Hotnets-IV*, 2005.

[7] BGP Routing Table Statistics. `http://bgp.potaroo.net/as6447/`, 2010.

[8] B. Briscoe, A. Jacquet, C. D. Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing Congestion Response in an Internetwork using Re-feedback. In *ACM SIGCOMM*, 2005.

[9] B. Briscoe, A. Jacquet, T. Moncaster, and A. Smith. Re-ECN: A Framework for Adding Congestion Accountability to TCP/IP. `http://tools.ietf.org/id/draft-briscoe-tsvwg-re-ecn-tcp-motivation-01.txt`, 2009.

[10] M. Casado, P. Cao, A. Akella, and N. Provos. Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks. In *IWQoS*, 2006.

[11] D.-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.*, 17(1), 1989.

[12] CSS Routing and Bridging Configuration Guide. `http://www.cisco.com/en/US/docs/app_ntwk_services/data_center_app_services/css11500series/v7.30/configuration/routing/guide/IP.html`, 2010.

[13] J. Crowcroft, T. Deegan, C. Kreibich, R. Mortier, and N. Weaver. Lazy Susan: Dumb Waiting as Proof of Work. Technical Report UCAM-CL-TR-703, University of Cambridge, Computer Laboratory, 2007.

[14] Deterlab. `http://www.deterlab.net/`, 2010.

[15] C. Dixon, A. Krishnamurthy, and T. Anderson. Phalanx: Withstanding Multimillion-node Botnets. In *USENIX/ACM NSDI*, 2008.

[16] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson. Robust Congestion Signaling. In *In IEEE ICNP*, 2001.

[17] F-Secure. Calculating the Size of the Downadup Outbreak. `http://www.f-secure.com/weblog/archives/00001584.html`, 2009.

[18] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM ToN*, 1(4), 1993.

[19] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing Extensible IP Router Software. In *USENIX/ACM NSDI*, 2005.

[20] Helion Technology. AES Cores. `http://www.heliontech.com/aes.htm`, 2010.

[21] Intel AES Instructions Set. `http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/`, 2010.

[22] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving DDoS Attacks that Mimic Flash Crowds. In *USENIX/ACM NSDI*, 2005.

[23] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *ACM SIGCOMM*, 2002.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM TOCS*, 18(3), 2000.

[25] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using Packet Symmetry to Curtail Malicious Traffic. In *ACM Hotnets-IV*, 2005.

[26] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *USENIX/ACM NSDI*, 2008.

[27] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *ACM SIGCOMM*, 2008.

[28] X. Liu, X. Yang, and Y. Xia. NetFence: Preventing Internet Denial of Service from Inside Out. Technical Report 2010-01 (available at `http://www.cs.duke.edu/nds/ddos/netfence-tr.pdf`), Duke University, 2010.

[29] S. Luo and G. A. Marin. Realistic Internet Traffic Simulation through Mixture Modeling and A Case Study. In *Winter Simulation Conference*, 2005.

[30] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM CCR*, 32(3), 2002.

[31] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *IEEE ICNP*, 2001.

[32] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *USENIX/ACM NSDI*, 2007.

[33] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-Level Traceroute Tool. In *ACM SIGCOMM*, 2003.

[34] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM CCR*, 27(3), 1997.

[35] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM*, 2007.

[36] R. Perlman. Network Layer Protocols with Byzantine Robustness. MIT Ph.D. Thesis, 1988.

[37] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, 2001.

[38] E. Shi, I. Stoica, D. Andersen, and A. Perrig. OverDoSe: A Generic DDoS Protection Service Using an Overlay Network. Technical Report CMU-CS-06-114, Carnegie Mellon University, 2006.

[39] M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round Robin. In *ACM SIGCOMM*, 1995.

[40] A. Stavrou and A. Keromytis. Countering DoS Attacks with Stateless Multipath Overlays. In *ACM SIGCOMM CCS*, 2005.

[41] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: a Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks. *IEEE/ACM ToN*, 2003.

[42] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *USENIX Security Symposium*, 2000.

[43] DDoS Mitigation to the Rescue. `https://www.arbornetworks.com/dmdocuments/DDoS%20Mitigation%20to%20the%20Rescue.pdf`, 2010.

[44] J. S. Turner. New Directions in Communications (Or Which Way to the Information Age?). *IEEE Communications Magazine*, 1986.

[45] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS Defense by Offense. In *ACM SIGCOMM*, 2006.

[46] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One More Bit is Enough. *IEEE/ACM ToN*, 16(6), 2008.

[47] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Security Symposium*, 2004.

[48] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting Network Architecture. *IEEE/ACM ToN*, 16(6), 2008.