To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets

Xin Liu
Dept. of Computer Science
University of California, Irvine
xinl@uci.edu

Xiaowei Yang
Dept. of Computer Science
University of California, Irvine
xwy@uci.edu

Yanbin Lu
Dept. of Computer Science
University of California, Irvine
yanbinl@uci.edu

ABSTRACT

This paper presents the design and implementation of a filter-based DoS defense system (StopIt) and a comparison study on the effectiveness of filters and capabilities. Central to the StopIt design is a novel *closed-control*, *open-service* architecture: any receiver can use StopIt to block the undesired traffic it receives, yet the design is robust to various strategic attacks from millions of bots, including filter exhaustion attacks and bandwidth flooding attacks that aim to disrupt the timely installation of filters. Our evaluation shows that StopIt can block the attack traffic from a few millions of attackers within tens of minutes with bounded router memory. We compare StopIt with existing filter-based and capability-based DoS defense systems under simulated DoS attacks of various types and scales. Our results show that StopIt outperforms existing filter-based systems, and can prevent legitimate communications from being disrupted by various DoS flooding attacks. It also outperforms capability-based systems in most attack scenarios, but a capability-based system is more effective in a type of attack that the attack traffic does not reach a victim, but congests a link shared by the victim. These results suggest that both filters and capabilities are highly effective DoS defense mechanisms, but neither is more effective than the other in all types of DoS attacks.

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, Filter, Capability

1. INTRODUCTION

Large-scale denial of service (DoS) attacks remain a serious threat to the reliability of the Internet. Despite much improved software security, botnets are still getting bigger. In March 2007, the number of bot-infected machines tracked by a single group was estimated to reach 1.2 million [17]. In June 2007, a presentation from Support Intelligence Inc. reported 48 million infected IP addresses observed in a six month period [35]. In September 2007, the estimated size of the Storm botnet alone reached 50 million [16, 30]. It is a distressing fact that the dark side possesses this vast amount of computing power: if each bot sends one full-sized packet per second (1500 bytes), the aggregated attack traffic from a 10-million-node botnet would exceed 120 Gbps, sufficient to take down anyone on the Internet. The recent attacks on Estonia [23] are perhaps only the tip of the iceberg on what attackers are capable of.

Many solutions have been proposed to battle the DoS problem [2, 4, 5, 8, 13, 20, 21, 25, 28, 31–34, 36, 37]. Yet there lacks a consensus on how to build a DoS-resistant network architecture. Among various proposals, two schools of thought are particularly intriguing: the capability-based approach [4,25,36,37] and the filter-based approach [5, 20]. Both advertise to enable a receiver to control the traffic it receives, but differ significantly in methodology. The capability approach proposes to let a receiver explicitly authorize the traffic it desires to receive, while the filter approach proposes to let a receiver install dynamic network filters that block the traffic it does not desire to receive. Advocates of filters have argued that "capabilities are neither sufficient nor necessary to combat DoS" [6], while proponents of capabilities "strongly disagree" [25].

As a first step towards reaching a consensus, we aim to understand the roles of filters and capabilities: which one is a more effective DoS defense mechanism? Ideally, to answer this question, one can systematically compare filter-based designs and capability-based ones. Unfortunately, this simple approach is not viable because capability-based systems [25, 36, 37] have been improved much in the past few years, yet there lacks a comprehensive filter-based architecture to compare with. The most complete work on filters, AITF [5], has a few limitations that prohibit a fair comparison between filters and capabilities. For instance, AITF verifies the legitimacy of a filter request using a three-way handshake. If the flooded link is outside a victim's AS, the three-way handshake may not complete because the handshake packets traverse the same flooded link as the attack traffic, and filters may not be installed. Another filter-based system, Pushback [20], does not completely block attack traffic. Instead, it aims to rate limit the attack traffic to its fair share of bandwidth.

To address this issue, we first design and implement a secure and effective filter-based DoS defense architecture StopIt. StopIt employs a novel *closed-control* and *open-service* architecture to combat various strategic attacks at the defense system itself, and to enable any receiver to block the undesired traffic it receives. Unlike previous work [5], StopIt is resistant to strategic filter exhaustion attacks (§ 4) and bandwidth flooding attacks that aim to prevent the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-175-0/08/08 ...$5.00.

timely installation of filters. We implement the StopIt design on Linux using Click [14] and evaluate it on Deterlab [9]. Our experiments suggest that StopIt enables a receiver to block the undesired traffic from a few millions of attackers in tens of minutes; routers with 256K hardware filters and less than 200MB DRAM can block the attack traffic from misbehaving hosts without inflicting damage to legitimate traffic.

The StopIt design demonstrates the feasibility of a filter-based approach and enables a systematic comparison between filters and capabilities. We compare StopIt with two well-known capability-based systems (TVA [37] and Portcullis [25]) together with previous filter-based designs (AITF [5] and Pushback [20]) using ns-2 simulations. We simulate how different systems perform under various DoS attacks. The simulation results show that StopIt outperforms AITF and Pushback in all types of attacks in terms of protecting legitimate communications from being disrupted. This is because it is designed to be resistant to strategic attacks, and filters can still be installed under those attacks, while other systems either fail to install filters or do not entirely block attack traffic. However, StopIt does not always outperform a capability-based system. In the case that the attack traffic does not reach a victim, but congests a link shared by the victim, for instance, if the attack traffic reaches a non-upgraded receiver, or the TTLs of the attack traffic expire before it reaches the victim, filters are not installed and a capability-based system outperforms StopIt. This is because capabilities robustly enable a destination to control the bulk of a link's bandwidth even if the attack traffic does not reach it.

These results suggest that both filters and capabilities are viable choices to build a DoS-resistant network architecture, although neither is more effective than the other in all types of attacks. A DoS-resistant network architecture is likely to incorporate multiple mechanisms. We suspect that the combination of StopIt and capabilities would be the most effective solution, but it may be too expensive in terms of deployment cost. On the other hand, the combination of source address authentication, per-AS bandwidth fairness, capabilities, and moderate bandwidth provision would be the most cost-effective solution due to the robustness of capabilities and the relative simplicity of a capability-based design. It is our future work to validate these hypotheses.

The rest of the paper is organized as follows. § 2 defines the design space of StopIt. We provide the design, implementation and evaluation of the StopIt architecture in § 3 – § 7. We compare the StopIt architecture with other DoS defense systems in § 8. § 9 discusses related work, and we conclude in § 10.

2. DESIGN SPACE

Before we dive into the design details of the StopIt architecture, we describe the threat model the design aims to combat, the assumptions we make, and the design goals.

2.1 Threat Model

The key threat we are concerned with is the network resource exhaustion attacks, in which compromised machines send packet floods to exhaust shared network resources such as link bandwidth and routers' memory or CPU.

We assume both routers and hosts can be compromised, but user-administered hosts are more likely to be compromised than operator-administered routers and servers. Our design places more trust in routers and servers managed by the network than end systems. We also assume that an Autonomous System (AS) is a fate sharing and trust unit. If one component in an AS (e.g., a router) is compromised, we consider the AS as compromised. A compromised host can inject arbitrary traffic into the network. A compromised AS can not only inject traffic, but also eavesdrop, modify, or discard the traffic that it forwards. A compromised AS that is on the forwarding path from a source to a destination is referred to as an on-path attacker or otherwise an off-path attacker.

While we cannot foresee all types of DoS flooding attacks, we focus on two general ones:

Destination Flooding Attacks: Attackers send traffic floods to a destination in order to disrupt the destination's communications.

Link Flooding Attacks: This type of attack aims to congest a link and disrupt the legitimate communications that share the link. The destinations of the attack traffic will not attempt to stop the attack traffic. This could happen in many scenarios such as: 1) the attack traffic is diffused among a large set of destinations, each receiving only a small amount that is not worth blocking; 2) the attack traffic's TTLs expire before it reaches the destinations; 3) no hosts are residing at the destination addresses; 4) the destinations have not deployed a DoS defense system; 5) or the destinations are compromised machines that coordinate the attacks.

2.2 Assumptions

We make a few assumptions about other design modules and the underlying network conditions on which this work may depend. These assumptions allow us to limit the scope of this work, but nonetheless we make the StopIt design fail-safe: even if some assumptions do not hold, the damage is limited locally to where they fail, not globally.

- **Securable Intra-AS Communications:** We assume that communications within an AS can be made secure if the AS desires to do so. In particular, an AS may prevent source address spoofing within its network using any anti-spoofing method such as ingress filtering [11] or link-layer security protocols [1]. It can ensure the integrity of communications between the routers or servers under its administration.
- **Attack Traffic Classification:** We assume that it is possible for an end system to detect and classify DoS flooding traffic, although it is not guaranteed. We believe this is not an over-optimistic assumption, because there is evidence that intrusion detection works to some extent, and reverse Turing tests such as CAPTCHA can distinguish bots from human users. Even under a more pessimistic assumption that attack traffic may become absolutely indistinguishable, a defense system that can stop distinguishable attack traffic still raises the bar to launch successful attacks. Therefore, we think it is worthwhile to explore this design point.
- **The Battle Ground**: Defense and attack is an arms race. We cannot predict accurately the growth of botnets, or the growth of network resources and technology advancement. Therefore, we choose to hypothesize the power on each side based on the current best estimate. We assume that a DoS attack may involve multi-million compromised machines; attackers may compromise a significant fraction of ASes, but not the majority of the Internet. We also assume that routers and hosts have bounded bandwidth, memory, and computation power. If those assumptions do not hold in the future, it requires future work to adapt the design to a different battle field.
- **Upgradable Components:** We assume that both router and host software can be upgraded, but we do not assume special (e.g., tamper-proof) hardware upgrade. Some may consider this assumption is unrealistic, but we prefer to work out an architectural design with this underlying assumption, because if deployed, an architectural solution has the advantage of protecting everyone.

- **Feasible:** The design should be efficient such that it can be implemented on high-speed routers within the reach of the present or foreseeable future technology. For this reason, we constrain the StopIt design not to involve public key cryptography operations at packet forwarding time for their high processing cost. We also avoid per-flow state in the network.
- **Dependable Routing:** We assume that the BGP routing system can be made to correctly forward packets towards their destinations. While presently prefix hijacking attacks do occur, we consider it a separate problem from DoS attacks.

2.3 Goals

Under the above assumptions, we aim to achieve the following design goals:

- **Effective (§ 3):** If a receiver can detect attack traffic, the StopIt architecture should enable it to stop the traffic without inflicting damage on other legitimate hosts using network filters.
- **Resistant to Strategic Attacks (§ 4)**: A main challenge in building a DoS defense system is to secure the defense system itself. Unavoidably, attackers will aim to defeat or abuse the defense system. We refer to such attacks as strategic attacks. We aim to make StopIt resistant to strategic attacks as well as Destination and Link Flooding Attacks. In § 4, we describe those attacks and mechanisms to combat them. Although we cannot claim or prove that we have considered all possible attacks, we have included as many as we can think of, and to the best of our knowledge, our list is much more complete than that considered by previous filter-based designs [5, 20].
- **Fail Safe (§ 5):** If filters fail to install, either due to the failure of attack detection or because some design assumptions do not hold, StopIt should limit the impact of the failure, and provide gracefully degraded service to legitimate communications. In addition, it should not make legitimate hosts worse off than without it, either in its failure or normal operation mode.
- **Incremental and Incentive-compatible Deployment (§ 6):** The StopIt design must enable incremental deployment, and provide immediate benefits to early adopters.

3. STOPIT OVERVIEW

For clarity, we first describe the high-level StopIt architecture. This basic design is vulnerable to strategic attacks. In the next section, we describe those attacks and how to secure StopIt under those attacks. Figure 1 lists the notations used to describe the design. We will also define them when we first refer to them.

Symbol	Meaning
H_s	Source host
H_d	Destination host
R_s	Access router of the source H_s
R_d	Access router of the destination H_d
S_s	StopIt server at H_s's AS
S_d	StopIt server at H_d's AS
T_b	A filter's block period
T_{max}	The longest block period allowed by an AS
T_f	The duration of a flow cache
N_f, T_i	Filter request limit in a duration to detect malicious sources
N_s	Filter request limit to detect malicious ASes
F_s	The maximum number of filters R_s has
N_a	The number of attacker-triggered filter replacements at R_s
N_u	The maximum number of unacknowledged filters

Figure 1: Notations used to describe the StopIt design. We use the same symbol that refers to an entity to refer to its IP address.

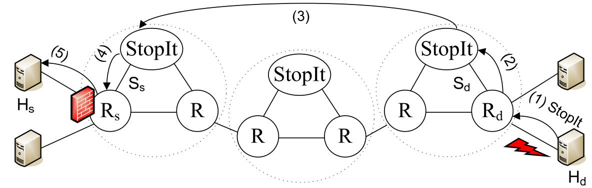

Figure 2: This figure shows the StopIt architecture, and how a destination H_d installs a filter to block the attack flow (H_s, H_d) from a source H_s. The dashed circle represents an AS boundary. Each AS has a StopIt server that sends and receives StoptIt requests, and hosts can only send StopIt requests to their access routers.

3.1 Components

Figure 2 depicts the StopIt architecture. A dashed circle represents an AS boundary. StopIt is designed as an infrastructure service (in analogy to DNS or email service). When a destination H_d detects the attack traffic from a source H_s, it invokes the StopIt service to block the attack flow (H_s, H_d) for a desired period of time T_b. The StopIt design filters packets using the source and destination address fields, as such filters are available at wire speed. We discuss how to prevent source address spoofing in § 4.1.

Each AS has a StopIt server that handles filter requests. Interdomain filter requests can only be sent from one StopIt server to another. The StopIt server acts as an automated abuse contact. It is a logical module, and could be implemented on a router, or run on multiple machines for load balancing and fault tolerance.

A StopIt server needs to know other StopIt servers' addresses to send a filter request. The StopIt design uses BGP to publish StopIt servers' addresses. An AS encapsulates its StopIt server's address or address prefix as an optional and transitive BGP attribute in one of its address prefix announcements. When other ASes receive this announcement via BGP, they learn the StopIt server's address of that AS. A router can be configured with the address of its own AS's StopIt server to send or verify a filter request.

A StopIt server obtains both BGP and IGP feeds from the routing system using passive listening sessions with BGP and IGP routers. It learns the StopIt server's addresses of other ASes from BGP feeds, and the addresses of the routers in its own AS and the prefixes they originate from IGP feeds. It can use this information to locate the access router of an indicted local source in a filter request.

3.2 Interactions

We carefully design the protocol to install a filter to prevent various attacks at the StopIt system. The novelty of this design is that the control channel is closed in the sense that each pair of interacting peers know the identities of each other, yet the system allows any destination to block the attack traffic from any source. Figure 2 illustrates the steps to install a filter:

1. A destination host H_d that wishes to block an attack flow sends a host-router StopIt request to its access router R_d. This request includes the attack flow's source and destination IP addresses: (H_s, H_d) and a block period T_b.
2. The access router R_d verifies this request to confirm that the source H_s is attacking the destination H_d and sends a router-server request to the AS's StopIt server S_d. The verification involves sending end-to-end StopIt requests to H_s (§ 4.6).
3. The StopIt server S_d in the destination H_d's AS forwards an inter-domain StopIt request to the StopIt server S_s in the source H_s's AS to block the flow (H_s, H_d) for T_b seconds.

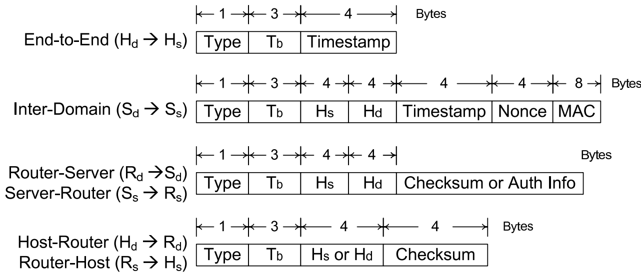

Figure 3: The format of various StopIt requests. The notation $(X \rightarrow Y)$ on the left side of each StopIt request specifies the source and destination IP address of a StopIt request.

4. The source StopIt server S_s locates the access router R_s of the attacking source H_s, and sends a server-router request to the access router. A StopIt server ignores inter-domain StopIt requests that block itself to prevent deadlocks.
5. In the last step, the access router R_s verifies the StopIt request, installs a filter, and sends a router-host StopIt request to the attacking source H_s. After receiving this request, a compliant host H_s installs a local filter to stop sending to H_d. If H_s does not stop, it will be punished by R_s (§ 4.5).

Figure 3 shows the format of various StopIt requests. Each StopIt request specifies the attack flow's source and destination IP address (H_s, H_d), and a block duration T_b. If either H_s or H_d is in the IP header, a StopIt request's payload does not duplicate it for efficiency. The design allows the block period T_b to be on the same order as the time it takes to repair a compromised host, e.g., one day. Each AS can have a local limit T_{max} on how long it will block a flow to mitigate issues such as collateral damage caused by dynamic host address allocation. Similar to ICMP, the StopIt protocol uses a raw IP header and has its own protocol number. Each node must verify that a StopIt request comes from the right peer before it honors the request. Otherwise, a malicious host may use StopIt to block other hosts' traffic. We describe how to authenticate each type of StopIt request in § 4.6.

4. SECURE STOPIT

The basic StopIt design is vulnerable to various strategic attacks, which include:

- **Source Address Spoofing Attacks (§ 4.1):** Attackers may spoof source addresses to evade attack detection and filtering.
- **Resource Exhaustion Attacks (§ 4.2 – § 4.5):** Attackers may: 1) flood filter requests to overload routers or StopIt servers' processing power so that legitimate requests cannot be processed; 2) send packet floods to cause filter requests to be discarded so that filters cannot be installed; 3) exhaust routers' filters so that no filters are available to block their DoS flooding traffic.
- **Blocking Legitimate Traffic Attacks (§ 4.6):** Attackers may use the StopIt service to block other legitimate nodes' traffic.

This section describes how we design StopIt to combat those attacks. A comprehensive security analysis of StopIt is shown in [19].

4.1 Passport for Source Authentication

The StopIt design uses a secure source authentication system Passport [18] to prevent source address spoofing. Each packet carries a Passport header to prove the authenticity of its source address. An attacker can not spoof its source address to evade attack detection or filtering. A trustworthy source address also precisely reveals the origin of an attack packet and enables the network to block the attack traffic close to its source. Although StopIt may use any source authentication architecture such as the self-certifying address architecture [3], using Passport has the advantage that the source authentication overhead is equivalent to the capability verification overhead of a capability-based system [37], as both use symmetric key cryptography. This feature facilitates our study on comparing filters with capabilities, as StopIt's packet forwarding overhead is comparable to that of a capability-based system.

For completeness, we briefly summarize how Passport works, and refer interested readers to [18] for more details. Unlike ingress filtering, Passport ensures that no host or AS can spoof the address space of an AS that deploys Passport. Similar to the Internet routing architecture, Passport authenticates source addresses at two levels: intra-domain and inter-domain. At the inter-domain level, a source AS stamps a sequence of Message Authentication Code (MAC) into a packet, each generated with a secret key shared with an AS on the path to the destination. The border router of each AS on the path checks the corresponding MAC to cryptographically verify the source AS of the packet before the packet enters its network. A packet with an invalid MAC will be discarded at the destination AS, and is discarded or demoted at an intermediate AS. Two ASes obtain the pair-wise secret key used for source authentication by piggybacking a standard Diffie-Hellman key exchange in their BGP announcements. At the intra-domain level, Passport assumes that each AS can use any internal mechanism such as ingress filters [11] to prevent source address spoofing.

4.2 Closed Control to Mitigate Request Floods

As shown in Figure 2, the StopIt architecture ensures that a router or a StopIt server only receives StopIt requests from either a local node in the same AS, or another StopIt server. This design prevents a router or a StopIt server from wasting its computational resources to process filter request floods from unknown addresses. A router or a StopIt server can be configured with the addresses of its local hosts, routers, or other StopIt servers from which it will accept StopIt requests, and discard other requests without processing them. If discarded requests from unknown addresses are classified as attack traffic, a node can use the StopIt service itself to block them. If requests from legitimate addresses overload a router or a server, the node can use local scheduling algorithms to fairly process those requests, or discard misbehaving peers' requests temporarily.

4.3 Guard StopIt Requests from Packet Floods

If attackers are able to congest both directions of a bottleneck link, inter-domain StopIt requests from a destination AS to a source AS could be lost due to the flooding attack. The StopIt design is able to protect an inter-domain StopIt request in this scenario, because StopIt servers' addresses are known to routers via BGP. Routers can separate StopIt servers' traffic from other hosts' traffic. As we will soon describe in § 5, this can be done either via hierarchical fair queuing [7] or hierarchical rate limiting.

4.4 Confirm Attacks Before Taking Actions

A compromised destination may initiate a futile filter request to block a legitimate source's traffic to itself. Large botnets may use these futile requests to launch filter exhaustion attacks, or to trigger inter-domain request floods. For instance, they may first send filter requests to block a legitimate host that co-locates with a compromised host and exhausts the filters at the hosts' access router. The compromised host can then send attack traffic, but the access router has no filters left to block it.

To prevent this type of attack, the StopIt design verifies that a host H_s is sending undesired traffic to a destination H_d before it installs a filter. Three nodes independently confirm this before they proceed to the next step. Each node represents a separate trust domain: the destination's access router, the source's access router, and the source itself.

4.4.1 *Confirm Attacks By a Destination Router*

As described in § 3, to block a malicious source H_s, a destination H_d sends a StopIt request to its local access router R_d. R_d must confirm that H_s is sending undesired traffic to H_d before it forwards the StopIt request to a local StopIt server.

The StopIt design uses three mechanisms: flow cache, end-to-end StopIt requests, and local filters, for R_d to confirm that H_s is attacking H_d. R_d uses a flow cache to verify that H_s has sent some traffic to H_d recently. A flow cache records the flows a router forwards in the past T_f seconds. An access router keeps an incoming flow cache as well as an outgoing flow cache.

If R_d finds the flow (H_s, H_d) in its incoming flow cache, it further checks whether H_s is misbehaving. To do so, it installs a local filter (H_s, H_d) and sends an end-to-end StopIt request directly to H_s. This request uses H_d as its source IP address to facilitate H_s's verification. This source IP address "spoofing" should be allowed by an AS because in a sense R_d owns its stub network's addresses. In the StopIt design, a compliant host H_s must stop sending to H_d after receiving a legitimate end-to-end StopIt request. If it does not stop, R_d's local filter will catch the traffic from H_s to H_d. This confirms that H_s is sending undesired traffic to H_d, and R_d proceeds to send a router-server StopIt request.

R_d may not have enough local filters to verify all StopIt requests it receives, if the number of attacking flows is large, e.g., 10-million bot machines attack every host on a /24 subnet. If R_d replaces an old filter before the requested blocking period T_b expires, a malicious source may pretend to stop after an end-to-end StopIt request and attack a destination after R_d replaces its filter. R_d may fail to catch this behavior and send end-to-end StopIt requests again to block the source. Consequently, malicious sources may continue to send attack traffic without being blocked.

To address this problem, we design a secure filter replacement protocol that enables an access router to deterministically catch a misbehaving source even if it runs out of filters. The key idea is to use an unforgeable filter replacement message to record verification state. When a router receives a new StopIt request and it runs out of filters, it randomly replaces an existing filter (H_s, H_d) with the new one, and sends the host H_d a filter replacement message. If the host H_d continues to receive the attack traffic from H_s, H_d resubmits the router's filter replacement message. This message proves to the router that it has processed a StopIt request from H_d before, i.e., the router has sent an end-to-end StopIt request to H_s using H_d's address. If the router catches the flow (H_s, H_d) in its flow cache again, it shows that H_s has not stopped after the previous StopIt request. The router may immediately take the next step action, i.e., sends a StopIt request to the StopIt server, or retransmits an end-to-end StopIt request to H_s in case the previous one is lost.

Re-submissions of the filter replacement messages must be separated by the flow cache length T_f so that a router can trust that the flow (H_s, H_d) caught by its flow cache corresponds to new traffic from H_s to H_d. A router can enforce this interval by including a timestamp that specifies the time it sends the filter replacement message.

A router includes a keyed hash in a filter replacement message to make it unforgeable. The key is only known to the router itself and changes over time for improved security. A router sends a filter replacement message with high priority as the link from the router to a destination may be congested before an attack stops. Figure 4 shows the format of a filter replacement message. The message includes a $\#Retx$ field that records how many times a filter has been replaced, and the lower 24-bits of a router's local timestamp.

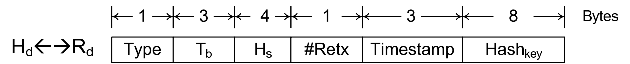

Figure 4: The format of a filter replacement message.

This filter replacement protocol ensures that a router can confirm an attack with bounded memory, because if a malicious source does not stop, a router will catch its attack flow in its flow cache when a host resubmits a filter replacement message. The router will proceed to the next step. If it takes k end-to-end StopIt request retransmissions to confirm an attack, then after at most $k-1$ re-submissions of a filter replacement message, a destination access router will confirm the attack and send a request to its StopIt server.

4.4.2 *Confirm Attacks by a Source or Source Router*

As described in § 3, a filter request initiated by a destination will eventually reach the source host's access router R_s. In the StopIt design, R_s also verifies that H_s has sent attacking traffic to H_d before it proceeds to block the flow (H_s, H_d). This verification prevents a malicious destination AS from wasting the source access router's filters. It uses the same flow cache mechanism as used by R_d to verify that H_s has sent some traffic to H_d. If it catches the attack flow (H_s, H_d) in its flow cache, it installs a filter and sends a router-host StopIt request to H_s. Similarly, when a source host H_s receives an end-to-end StopIt request, it verifies that it has sent some traffic to H_d using a local flow cache before it blocks its traffic to H_d.

4.4.3 *Bounded Flow Cache Memory*

In the StopIt design, a flow cache's size can be bounded because the cache only needs to last for a few seconds (T_f) to tolerate a round trip delay. Note that a destination H_d's attack detection module may take longer than T_f to identify an attack source H_s. After the detection, as soon as the destination receives a new packet from the attack source, it may immediately send a StopIt request. Therefore, as long as the flow cache lasts longer than the time it takes to forward a filter request from a destination to a source or a source's access router, the attack flow (H_s, H_d) will be found in a flow cache.

A flow cache can be implemented using a circular buffer of bloom filters, a technique also used in [29]. A bloom filter has no false negatives. A router can always catch an attack flow in its flow cache as long as the round trip delay is less than T_f. We are not concerned with the small percentage of a false positive, because it occurs rarely and randomly, and can only happen when a malicious host H_d wants to block its own traffic, and at most wastes one router filter. One can verify that it requires less than 100MB memory to implement a flow cache of 5 seconds on a gigabit link (See also [19]).

4.5 Manage Source Router Filters

In the StopIt design, an attack flow (H_s, H_d) is blocked at the access router R_s of the attack source. A key challenge it faces is how to block all attack flows without collateral damage if the router R_s has insufficient filters. This may happen, for instance, if a compromised host on the router's subnet attacks a large number of destinations, or a compromised destination AS sends a large number of StopIt requests to block a legitimate source host.

In this sub-section, we describe how we address this challenge. For clarity, we first describe the design assuming destination ASes are not compromised. We then describe how to cope with compromised destination ASes.

4.5.1 Aggregate Misbehaving Sources' Filters

In StopIt, if a router runs out of filters, it first reduces the number of filters for a misbehaving host H_s by aggregating them into per source and destination-prefix filters in the form of $(H_s, H_d/l)$. It may choose the length of the prefix l according to its available filters. This filter aggregation may harm misbehaving hosts' legitimate traffic to destinations that do not request to block them, providing incentives for users to patch their compromised machines.

StopIt uses either of the two following conditions to detect misbehaving hosts. First, a host H_s is considered misbehaving if it does not stop sending to a destination H_d after its access router R_s has installed a filter (H_s, H_d). This is because a compliant host will stop sending undesired traffic after R_s sends a StopIt request to it during filter installation (Step 5 of the StopIt protocol described in § 3). Second, a host H_s is considered misbehaving if its access router R_s receives a large number of StopIt requests to block its flows. This is because a legitimate host will comply to an end-to-end StopIt request sent by a destination's access router (Step 2 of the StopIt protocol described in § 3) and will not trigger excessive StopIt requests. StopIt uses two configurable parameters: the number of StopIt requests N_f received to block a source in a duration T_i, e.g., 10 million per day, to separate legitimate hosts from misbehaving ones. If a router R_s receives more than N_f StopIt requests in T_i to block a host H_s, H_s is considered misbehaving, and the router R_s may aggregate its filters.

4.5.2 Avoid Responding to Malicious ASes

If compromised destination ASes exist, a router may erroneously classify a legitimate host as misbehaving. For instance, a compromised AS may send a packet that triggers a reply (e.g., Ping, TCP SYN) to a legitimate host, and then send an inter-domain StopIt request without first sending an end-to-end StopIt request to the host. If compromised ASes successfully send more than N_f filter requests in the duration T_i, a router may mis-classify a legitimate host as misbehaving.

To address this problem, we design an algorithm for hosts to detect malicious ASes and refrain from responding to them. A host H_s can detect a malicious AS from the missing of legitimate end-to-end StopIt requests. If it repeatedly receives initial packets from an AS's address space that trigger reply packets, and then a router-host StopIt request from its access router to block a flow destined to the AS, it concludes that the AS is misbehaving. It can detect this pattern by caching the source addresses of the incoming packets to which it responds. When it receives a router-host StopIt request, it will find the malicious destination's address in this packet cache. A few missing end-to-end StopIt requests may be due to packet loss, but if it receives a large number (N_s) of router-host StopIt requests from an AS, it should stop responding to any initial packet from that AS. An AS can provide an address-to-AS mapping service to enable its hosts to associate an address with an AS.

An AS should set the parameters N_f, N_s, and T_i to accommodate a worst case estimate on the number of compromised ASes. The present Internet has less than 30K ASes. If we set $N_s = 1000$, $N_f = 10M$, and $T_i = 1$ day, a legitimate host will not be classified as misbehaving even if 10K ASes are compromised and intend to frame the host in one day, which is unlikely to happen in the near future. In addition, each AS can adjust N_f, N_s, and T_i to adapt to future attack scenarios.

4.5.3 Random Filter Replacement

Aggregating misbehaving hosts' filters only partially addresses the filter exhaustion problem. Routers may still run out of filters, because the StopIt request limit (N_f) for each host must be set to a large value to avoid misclassifying legitimate hosts. When this situation occurs, the StopIt design uses a random filter replacement policy to prevent a host from repeatedly attacking a destination. When a router receives a new StopIt request, it randomly replaces an old filter of a non-misbehaving host with the new one. A router does not replace a misbehaving host's filters, but aggregates them if it runs out of filters.

This random replacement algorithm ensures that a malicious host cannot repeatedly attack a destination without being caught as misbehaving. This is because right after a malicious host H_s attacks a destination H_d, it will trigger a StopIt request. To repeatedly attack the same destination without being caught, H_s must either wait for its filter to expire, which means it has stopped the attack, or exhaust the router's filters to make its filter (H_s, H_d) replaced. However, it can at most trigger N_f StopIt requests in a period T_i to exhaust router filters. Therefore, it will either be caught as misbehaving when it is caught sending to a filter, or when it exceeds its limit N_f. In either case, a router will not replace its filters any more.

We analyze how many times a malicious host may successfully attack a destination before it is caught or uses up its StopIt request limit. Suppose an access router R_s has F_s filters. After a source H_s attacks a destination H_d, the router installs a filter (H_s, H_d). To attack H_d again without being classified as misbehaving, H_s triggers N_a new StopIt requests. Suppose the router has run out of filters, and it performs N_a random filter replacements. The probability that a previously installed filter (H_s, H_d) is replaced is:

$$1 - (1 - 1/F_s)^{N_a} \quad (1)$$

This is the probability that the host H_s can attack H_d again without being caught as misbehaving.

After the host H_s attacks H_d again, the filter (H_s, H_d) will be re-installed. To repeat the attack, H_s must trigger new StopIt requests. For simplicity, suppose it triggers the same number of N_a requests before it attacks H_d again. Then the probability that H_s is not caught after attacking H_d for k times is $(1 - (1 - 1/F_s)^{N_a})^k$. Since the total number of StopIt requests it triggers must be less than N_f and $k \times N_a \leq N_f$, if a source chooses a smaller N_a, it has a higher probability being caught after one round of attack; if it chooses a larger N_a, the maximum number of rounds k it can attack is reduced. In § 7, we use both experiments and analysis to show that when F_s is 10M and N_f is 10M a day, a malicious source on average can only attack a destination less than three times a day. A similar analysis can be done if we assume the attack target is a destination prefix H_d/l, not a single destination H_d, or there are a few (a) colluding compromised hosts on a router's subnets. In the former case, the probability of being caught after attacking any previously attacked address in H_d/l is the same. In the latter case, the colluding hosts' total StopIt request limit is increased to $a \times N_f$, but the probability of any one being caught after attacking a previously installed filter is not changed.

If a router uses hardware filters for line speed filtering, its filters may be much less than 10M. High-speed routers often use Ternary Content Addressable Memory (TCAM) to filter, but a TCAM chip is limited to at most 256K filter entries [24]. This problem can be addressed using shadow filters, similar to the technique used in [5] except that we assume that a router also has limited slow memory. A router uses hardware filters to block misbehaving hosts' traffic, and shadow filters to catch misbehaving hosts. When a router receives a StopIt request, it installs and replaces hardware filters as

described above. In addition, it installs a shadow filter in its slow and large DRAM memory. When it receives a new StopIt request to block (H_s, H_d), and it finds the same flow (H_s, H_d) in its shadow filters and the attack flow in its flow cache, it concludes that H_s attacks H_d again after its hardware filter (H_s, H_d) is replaced. The router can classify the source as misbehaving. If a router receives two StopIt requests to block the same attack flow before a flow cache expires, it does not use the second StopIt request to indict the source, as it could be triggered by the same attack traffic.

A router randomly replaces a shadow filter when it runs out of slow memory. The above analysis on how often a source can attack a destination before it is stopped still holds. If a router uses 8 bytes to store an attack flow and 8 bytes to store the expiration and installation time of a shadow filter, it can store 10M shadow filters in less than 200MB memory.

One design detail worth mentioning is that StopIt makes a host explicitly acknowledge a router-host StopIt request. Otherwise, if a request is lost, a legitimate host may be misclassified as misbehaving. A router will not consider a host as misbehaving if it is caught to send traffic to an unacknowledged filter but will keep a maximum unacknowledged filter limit N_u to prevent malicious hosts from never acknowledging a StopIt request. If a host's unacknowledged filters exceed N_u, the router temporarily disconnects the host until all filters are acknowledged. Hosts need to keep their acknowledged but non-expired filters persistent across reboots, and query the router to acknowledge their unacknowledged filters when they are back online. With a reasonably small N_u, a router can keep all unacknowledged filters within bounded memory.

4.6 Authenticate StopIt Requests

The StopIt design must prevent an attacker from blocking other legitimate hosts' traffic. To achieve this goal, the design enable each node that receives a filter request (H_s, H_d) to authenticate that the request is sent by the correct entity as described in § 3, and the entity owns the address H_d. We describe how each type of StopIt request is authenticated.

4.6.1 End-to-End StopIt Requests

A host H_s must verify that an end-to-end StopIt request is sent from the IP address H_d before it blocks its traffic to H_d. The StopIt design uses address-based authentication to verify that the request is from H_d: if the source IP address of an end-to-end StopIt request is H_d, then H_s trusts that it is sent from the IP address H_d. Address-based authentication is a weak authentication scheme, but it suffices for this purpose, because StopIt is based on a secure source authentication architecture that ensures any node outside the source or destination AS can not spoof the IP address of H_d. Only compromised routers inside a source or a destination AS may spoof a StopIt request with the source IP address H_d. In this case, the source AS or the destination AS is considered as compromised. The StopIt design does not intend to provide non-interrupted communication between a pair of hosts if either host's access AS is compromised, as a compromised access AS may cause more harm such as discarding its hosts' traffic. Note that a compromised host in a source or a destination's AS cannot spoof a StopIt request because we assume a compliant AS can prevent internal source address spoofing (§ 2.2).

A compromised AS on the path from a destination AS to a source AS may replay an end-to-end StopIt request, as the source authentication system that StopIt uses only prevents attackers not on the forwarding path of a packet from re-injecting the packet from other network locations. If an on-path AS attacker replays an old end-to-end StopIt request to block the flow (H_s, H_d) after the block period has expired, it can block the communication from H_s to H_d longer than what H_d desires. We are not concerned much with this type of attack, because an on-path attacker can always discard the packets from H_d to H_s (e.g., TCP SYN/ACK, or the capability return packets in a capability-based system) to disrupt their communications. But our design includes a timestamp field in an end-to-end StopIt request (Figure 3) to mitigate this attack. A source may optionally verify the timestamp and discard very old StopIt requests, e.g., older than a few hours. If a source (or a destination) has a completely out of sync clock, the source may erroneously discard an end-to-end StopIt request, but this error at most triggers an inter-domain StopIt request to stop the source. Note that an on-path compromised AS cannot modify an end-to-end StopIt request because the integrity of the first eight bytes of a packet's payload is ensured by Passport [18].

4.6.2 Inter-domain StopIt Requests

A StopIt server S_s that receives a request to block (H_s, H_d) from another StopIt server S_d must verify that the request is sent by S_d, and the IP address H_d is in S_d's AS's address space. The StopIt design uses cryptographic authentication for this purpose because address-based authentication is insufficient for several reasons. A compromised AS on the path may modify the content of inter-domain StopIt Requests. A source AS's StopIt server may not trust that a destination AS has prevented source address spoofing in its network, and it does not wish to waste its filters if a malicious node in a destination AS can spoof its StopIt server's address.

A StopIt server S_s obtains the same pair-wise secret keys that an AS's border routers obtain using Passport [18]. With a shared key, two servers can authenticate messages from each other using a standard cryptographic scheme (more details can be found in [19]). After S_s verifies that a request to block (H_s, H_d) is from S_d, it further verifies whether S_d and H_d belong to the same AS. S_s can verify this using the address-prefix-to-AS mapping obtained from its BGP feeds. If S_d and H_d belong to the same AS, S_s considers the StopIt request valid, and forwards it to the access router of the indicted source H_s.

4.6.3 Intra-domain StopIt Requests

As we assume that an AS can secure its intra-domain communications (§ 2.2), intra-domain StopIt requests, including router-server, server-router, router-host, host-router requests, can be authenticated using any local security mechanism such as address-based authentication, or a cryptographic authentication scheme. Due to space constraints, we omit the details. They can be found in [19].

5. FAIL-SAFE

The previous section describes how we design StopIt to combat various attacks that prevent filters from being installed. However, regardless of how hard we try, filters may fail to install when a source AS is compromised and ignores filter requests, or during a link flooding attack when destinations of the flood fails to initiate StopIt requests.

For simplicity and feasibility, the StopIt design uses hierarchical fairness, the same mechanism used in a capability-based system [37], to gracefully degrade when filters are not installed. A router either uses two-level hierarchical weighted fair queuing to allocate its bandwidth among ASes and then among hosts within the same AS queue, or uses a two-level rate limiters. For the first level resource allocation, as there are less than 30K ASes on the present Internet, it is feasible for routers to maintain per-AS state. For the second level allocation, if a router has insufficient queues or rate limiters to separate every host in an AS, it randomly hashes

different hosts from the AS into the same queue or rate limiter, as in stochastic fair queuing [22], except that a StopIt server's traffic is always separated from an AS's other hosts' traffic, and may be given a larger share. Legitimate hosts in an AS that harbors compromised hosts may suffer from collateral damage, but we think such damage could incentivize an AS to clean up its network.

Another approach to fail-safe when a compromised source AS does not respond to StopIt requests is to install Pushback-style filters. A filter request is propagated from an access router to a border router, and from a destination AS to its upstream provider, and so on. We assume that ASes are much less likely to be compromised than hosts. Therefore, the benefit of avoiding per-flow filter state in the network outweighs the disadvantage of not completely blocking compromised ASes but limiting them to their fair shares of bandwidth. However, if in practice it is desirable to entirely block a compromised AS, StopIt can be extended to support Pushback filters. We defer the detailed design of this extension to future work.

6. DEPLOYMENT

The StopIt design aims to facilitate incremental deployment and incentivize early adoption. Each AS can independently deploy StopIt and benefit from it. To deploy StopIt, an AS needs to upgrade its border routers to use Passport for source authentication as described in [18], upgrade its access routers to support StopIt, and install a StopIt server. It also needs to enable a hierarchical per-AS and per-host resource allocation scheme at its congested links.

An AS that deploys StopIt can block attack traffic from ASes that also deploy StopIt. It can also authenticate the source addresses of the traffic from Passport-enabled but not StopIt-enabled ASes and queue or rate limit their traffic separately. Attack traffic from a Passport-enabled but not StopIt-enabled AS only congests the traffic from the same AS, incentivizing the AS to adopt StopIt.

A StopIt-enabled AS cannot authenticate the source addresses of the traffic from ASes that do not deploy Passport. It should queue or rate limit the traffic from all non-upgraded ASes as one traffic aggregate. Attack traffic from those non-upgraded ASes may overwhelm legitimate traffic from those ASes, providing incentives for ASes to adopt both source authentication and StopIt.

A transit AS that is unlikely to originate attack traffic only needs to deploy Passport to authenticate source addresses and implement the hierarchical resource allocation scheme at its congested links. We think it has incentives to deploy these mechanisms to protect the traffic from Passport-enabled customers, because otherwise, DoS flooding attacks will inflict damage on all its transit traffic.

A server host that wishes to stop undesired traffic needs to upgrade to support StopIt. A client host does not need to upgrade to support StopIt, if it is unlikely to be attacked or compromised to attack other hosts. However, a router may aggregate a non-upgraded client's filters if it does not stop sending undesired traffic after a destination requests to block it. When this happens, a client will notice and should upgrade to support the StopIt protocol and source authentication. An upgraded host will not respond to an end-to-end StopIt request with a demoted or without a Passport header, because the source address of this request might be spoofed.

7. IMPLEMENTATION

We implement a prototype of the StopIt design on Linux using Click [14] and test its performance using Deterlab [9]. This evaluation aims to answer the following questions: 1) Can StopIt stop multimillion-node attacks with bounded router filters? 2) How long does it take for StopIt to stop such attacks? 3) What is the processing overhead of StopIt requests?

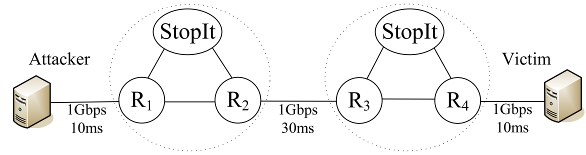

Figure 5: The network topology used in our experiments.

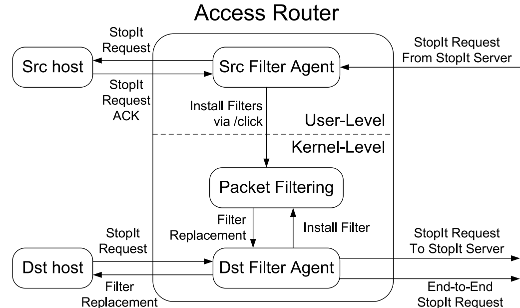

Figure 6: Access Router Prototype

For quick prototyping, we implement the StopIt protocol on top of UDP. A StopIt server is implemented as a user-level application. As shown in Figure 6, the access router's packet filtering function, the destination-side logic, including the host-router StopIt request processing and the secure filter replacement protocol are implemented using Click in the Linux kernel for better performance. The source-side logic, including the server-router StopIt request processing and router-host StopIt request processing are implemented as a user-level application. We have not implemented flow caches using bloom filters in this prototype, but this simplification should not affect the results, because as we will soon explain, the bottleneck in our experiments is not the access router's kernel processing module. The Click implementation modifies the IPRouteTable element. The authentication function in an inter-domain StopIt request or in a filter replacement message is implemented using UHASH, AES and the first UMAC construction as described in [15].

In the first experiment, we evaluate whether StopIt can stop large-scale DoS attacks when a destination's access router has a bounded number of filters, and if it does, how long it takes to stop an attack. The experiment topology is shown in Figure 5. This topology emulates an attacker's AS and a victim's AS. R_2 and R_3 emulate border routers, and R_1 and R_4 emulate access routers that implement the StopIt protocol. Each AS is also configured with a StopIt server that processes inter-domain StopIt requests. The single attacker machine emulates 1 to 10 million attackers by sending packets with source addresses distributed within a /8 address prefix. Attack packets are all destined to the victim. The victim either sends a fresh StopIt request to R_4, or resubmits a filter replacement message after a flow cache expiration interval $T_f = 5$ seconds. R_4 sends three end-to-end StopIt requests to confirm an attack before it sends a StopIt request to its local StopIt server. The border routers R_2 and R_3 should be performing the source authentication task as described in [18]. We have not integrated this part with the StopIt implementation, but source authentication is not the bottleneck in our experiments: traffic volume through R_2 and R_3 is less than 300kpps, which is much lower than the source authentication throughput according to [18].

The victim's access router R_4 is configured with 256K filters, emulating a limited number of hardware filters. Each emulated attacking source stops after it receives a StopIt request from R_1. Thus

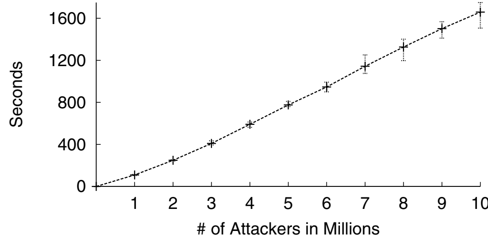

Figure 7: The time it takes for a victim to block various number of attackers.

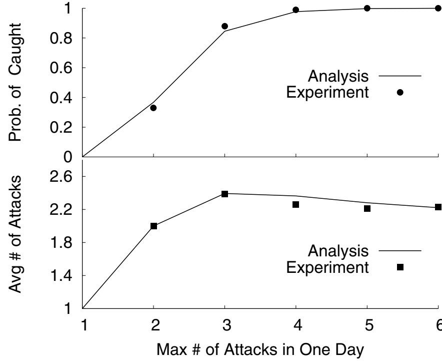

Figure 8: This figure shows the probability that a misbehaving host will be caught if it does not stop attacking a destination, and the average number of times it can attack the destination in one day. The x-axis is the maximum number of times it can attack the destination before it triggers more than N_f StopIt requests. The daily StopIt request limit is $N_f = 10$ million, and the router's filter limit is $F_s = 10$ million.

R_1's filter table size does not affect the results. Figure 7 shows the time it takes to stop an attack with various number of attacker sources. Each attack repeats 10 times, and the error bars show the standard deviations of the stopping time. As can be seen, StopIt is able to stop all attack flows. The router R_4 has successfully confirmed up to 10 million attack flows with only 256K filters. The time it takes to stop an attack grows proportionally with the number of attackers. Note that this time does not include the attack detection time, as we assume attack detection is a separate design module (§ 2.2).

The blocking rate roughly corresponds to 6000 attackers per second. This rate is primarily limited by the victim, because it receives and sends filter replacement messages and StopIt requests in user-space, and at the same time, receives flooding packets. We notice that the StopIt agent on the victim can send out roughly 28K messages per seconds. As in our implementation, the router R_4 retransmits an end-to-end StopIt request up to three times before it sends a request to its StopIt server, and the victim needs to send one StopIt request and resubmit three filter replacements to stop one attacking source. Thus, it can stop at most 28/4=7K attackers per second. At the same time, it also discards some of the filter replacement messages from the router R_4, because it is receiving attack packets at the same time and cannot keep up with the incoming packet rate. Despite the low blocking rate, a victim can still stop a 10-million node attack in less than 30 minutes.

Next, we evaluate whether an access router can catch a misbehaving source if the source does not stop attacking a destination. In these experiments, we let the attacker machine first attack the victim, and then attack a large number of fresh destinations to exhaust its router R_1's filters. R_1 is configured to have 10 million filters. To save experiment times, we install a packet capture agent at R_1 that intercepts the attack packets to fresh destinations and immediately installs filters at R_1. R_1 implements the random filter replacement algorithm as described in § 4.5.3. We pre-populate all R_1's 10 million filters to emulate a filter exhaustion attack, e.g., other compromised hosts and ASes have exhausted the router's filters by colluding with the attacker. The daily StopIt request limit N_f is set to 10 million. We first choose k, the maximum number of times an attacker can possibly attack the victim before it triggers more than N_f StopIt requests, and then choose the number of fresh destinations N_a that an attacker should attack in one round to minimize its probability of being caught at the end of the attacks. Each run finishes if either R_1 catches the attacker as mis-behaving, or the attacker has triggered N_f StopIt requests. For each run i, we record a binary variable Y_i. It is set to 1 if an attacker is caught before it triggers more than N_f requests. Otherwise, $Y_i = 0$. We also record the number of times S_i that the attacker can successfully attack the victim before one experiment finishes.

Figure 8 shows the probability that a misbehaving source is caught in the above attacks for various values of k. The lines are plotted using the analysis in Eq 1. Each point is obtained using the results from 100 runs for each k value. For each k, the probability of being caught is calculated as $\sum_i Y_i/100$, and the average number of attacks is calculated as $\sum_i S_i/100$. As can be seen, the experimental results match well with the analysis. When $k = 2$, after an attacker attacks a victim once, even after it triggers 10 million StopIt requests, it still has more than 35% probability to be caught if it attacks the victim again. In contrast, if we use a deterministic first-installed-first-replaced policy, the attacker will have zero probability to be caught. When $k = 3$, the average number of times it can attack a victim before it is caught is maximized to 2.40, but it will be caught more than 85% of the times.

We have also benchmarked the processing overhead of various StopIt requests. Due to space constraints, we omit the results in this paper but they are available in [19]. As StopIt messages only involve light-weight cryptography operations, their processing overhead is low, and a router or a server's CPU is unlikely to be the bottleneck resource.

8. COMPARING EFFECTIVENESS

To gain insights on how effective StopIt performs relatively to other DoS defense systems, we compare StopIt with two well-known capability-based DoS defense systems TVA [37] and Portcullis [25], and two existing filter systems: AITF [5] and Pushback [20]. We implement StopIt and other systems in ns-2, and simulate how effectively each system combats various DoS flooding attacks on large topologies. Note that this section does not simulate filter exhaustion attacks. They are studied in part in the previous section.

8.1 Methodology

Ideally, we would like to simulate various systems on an Internet-scale topology and vary the number of attackers to millions. Unfortunately, our simulator is incapable of simulations at this scale. Instead, we generate AS-level topologies from BGP table dumps, and simulate each AS as one node. We randomly mark an edge AS as hosting attackers or not. If a node is marked as hosting attackers, it floods the aggregate attacking traffic of all attackers in the AS.

Our results only measure the performance of hosts in legitimate ASes that do not have attackers. We believe this is a valid performance metric because a solution that sacrifices the performance of hosts in "clean" ASes to improve that in ASes that harbor attackers does not reward ASes that maintain a clean network, and is less desirable. We assume that attackers are not uniformly distributed

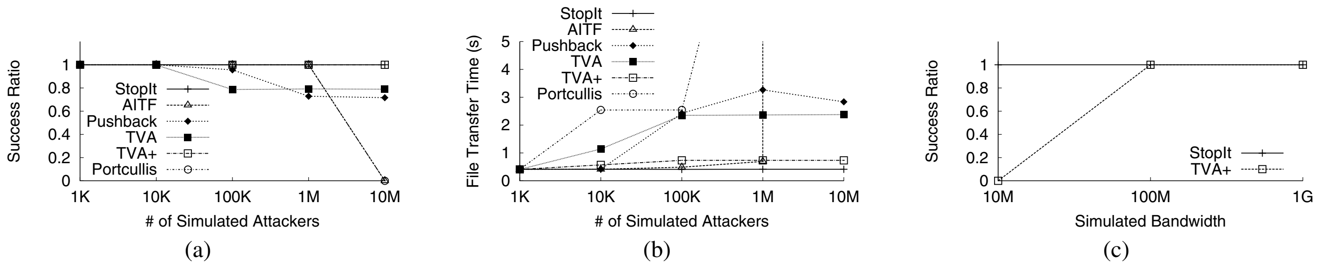

Figure 9: Destination flooding attacks. In 9(a) and 9(b), the simulated bandwidth is 1Gbps. Both TVA+ and StopIt can finish all TCP transfers. In 9(c), the simulated bottleneck bandwidth is varied from 10Mbps to 1Gbps. Only StopIt can finish for slower links, demonstrating the advantages of the filter approach.

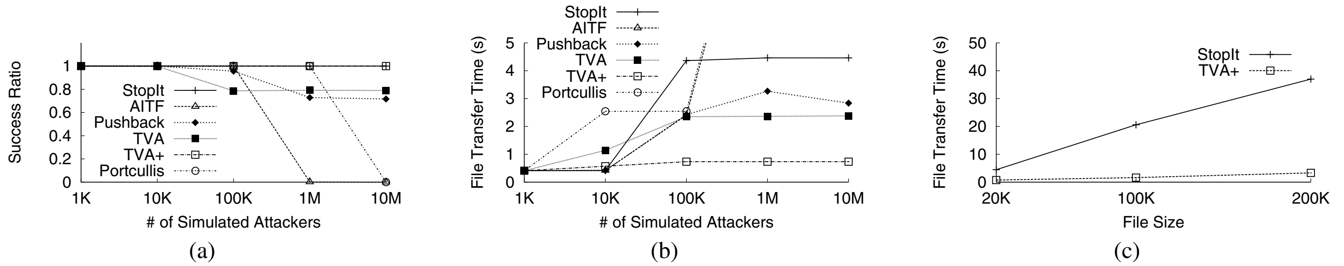

Figure 10: One-way Link Flooding Attacks. The simulated bandwidth is 1Gbps.

among all ASes based on data shown in [26, 35]: [35] shows that in a six month period, only half of the ASes on the Internet are observed to host bot machines; and [26] shows that Bobax drones are concentrated on a few IP address ranges. One plausible explanation is that unpatched pirated Windows operating systems might be unevenly distributed.

Topologies: We use a realistic simulation topology from the BGP table dump obtained on Aug 1st, 2007 from a RouteViews server. Realistic topologies are desirable because TVA, AITF, and Pushback's performance are path dependent. We construct a directed graph using the reverse AS paths seen from the RouteViews server. This topology contains about 26K nodes and is still too large for ns-2 simulations. We randomly sample a branch that has less than 2000 ASes, which is roughly the maximum size we can simulate. We refer to this topology as the sampled topology. The sampled topology has 1691 ASes, approximately 1/20 of the size of an Internet-scale topology.

Attacks: We simulate both destination flooding attacks and link flooding attacks. In a destination flooding attack, we connect a victim to the sampled topology via a bottleneck.

We intend to simulate scenarios that 1K $\sim$ 10M compromised hosts on the Internet attacking a victim behind a bottleneck ranging from 10Mbps to 1Gbps. Since we only have 1/20 of the topology, we scale the number of attackers to 1/20 of the number we intend to simulate: 50 $\sim$ 500K, and similarly, the bottleneck bandwidth to 500Kbps $\sim$ 50Mbps. We set the ratio of ASes that have attackers to be at most 2/3 among the edge ASes in the sampled topology. This corresponds to at most 1000 attacker ASes. If the number of attackers x in a simulation is less than 1000, we randomly sample x edge ASes from the sampled topology to be ASes that host attackers (i.e., attacker ASes). If x exceeds 1000, the maximum number of attacker ASes, we randomly sample 1000 edge ASes as attacker ASes, and let each attacker AS originate the attack traffic for $x/1000$ attackers. To make our simulations finish within a reasonable time, we bound the maximum total attack traffic to be 100 times the bottleneck bandwidth. Therefore, each attacker sends at 10Kbps (except for Portcullis, which we will soon explain). If an attacker AS simulates $x/1000$ attackers, its aggregate sending rate is 10Kbps times $x/1000$.

We also simulate one-way and two-way link flooding attacks. In the one-way attack, we have a sink node on the same side of the bottleneck as the victim. Attackers on the sampled topology may send attack traffic to the sink node without being detected. In the two-way attack, we have a large number of colluding attackers on the same side of the bottleneck as the victim. Attackers on the other side of the bottleneck may send attack traffic to those colluders without being detected. At the same time, the colluders may send reverse direction attack traffic to the other attackers. We simulate 500 $\sim$ 5M colluders on an Internet-scale topology with 25 $\sim$ 250K attackers on our sampled topology.

Implementations: AITF is implemented as described in [5]. A victim uses the last six router addresses in the recorded path option to describe an attack flow. Pushback is already officially included in ns-2 and is implemented as described in [20]. Routers recursively sends rate limiting requests to their upstream routers if a downstream bottleneck is congested. TVA is implemented as in [37]. TVA uses path identifiers to approximate an unspoofable source identifier and hierarchically queues capability request packets on path identifiers. We also implement an enhanced version TVA+ that uses the same source authentication system that StopIt uses to prevent source address spoofing on its request channel and uses a two-level (per-AS and then per-source) hierarchical queue on its request channel. We compare TVA+ and TVA to show the benefit of unspoofable source addresses. Source authentication in a capability-based system is only required for the slow request channel. Therefore we think it is feasible to combine capabilities and source authentication. Portcullis is implemented as in [25]. Different from StopIt, Pushback, TVA, and TVA+, Portcullis uses computational puzzles to implement per-host fairness, rather than per-network fairness. A capability request packet that solves a more difficult puzzle is forwarded with higher priority. In our simulations, a Portcullis attacker does not send constant rate flooding traffic. Instead, it adjusts its sending rate and puzzle level based on the total number of attackers involved in an attack and the bottleneck bandwidth. For instance, if 10M attackers attack a 100Mbps bottleneck, each attacker only needs to send 0.5 bits/s to congest the 5% request channel of the link. An attacker will solve a 640-second puzzle and sends it with a 40-byte packet. Our implementation

assigns a packet's priority based on the per-bit puzzle difficulty. Otherwise, an attacker can send at an even slower rate.

Metrics: We use legitimate hosts' TCP transfer performance to measure the effectiveness of a DoS defense system. During an attack, each legitimate AS has one user that sends 20KB TCP transfers one by one to the victim. A TCP transfer is aborted if it cannot finish within 25 seconds, simulating an application timeout. This timeout is also necessary to make the simulations finish in a reasonable amount of time. We use the ratio of completed transfers and the transfer time averaged over the completed transfers as the performance metrics. TCP SYN retransmission timeouts are limited to 1 second to speed up the simulations. One run finishes when all legitimate ASes have tried three transfers. We adjust the number of simultaneously active legitimate ASes based on the simulated bottleneck bandwidth to avoid congestion among legitimate ASes.

8.2 Destination Flooding Attacks

Figure 9 shows the results for the destination flooding attacks. The results for StopIt are steady state results after the attack traffic is blocked. In Figure 9(a) and 9(b), the simulated bandwidth is 1Gbps. AITF cannot finish after the number of attackers exceeds 1M, as the three-way handshake messages to install filters are lost due to the DoS flooding attack. After the number of attackers exceeds 1M, Portcullis does not finish within 25 seconds. Legitimate users may eventually finish if they wait longer and retransmit their request packets with increasing puzzle difficulties, but they timeout in our simulations. TVA and Pushback have similar results, because both ensure per-path fairness. TVA hierarchically queues on path identifiers on its request channel, while Pushback recursively sends rate-limiting messages to a router's upstream routers. A longer path may get a smaller bandwidth share. Therefore legitimate users that are far away from the victim may not finish their TCP transfers.

Both TVA+ and StopIt can finish all TCP transfers, outperforming other solutions. TVA+ does well because legitimate users are isolated from attacker ASes via hierarchical fair queuing, and have sufficient request channel bandwidth. In Figure 9(c), we vary the simulated bottleneck bandwidth from 10Mbps to 1Gbps, and compare TVA+ with StopIt. Only StopIt can finish all TCP transfers, because the attack traffic is completely blocked, demonstrating the advantages of a filter approach. With TVA+, each attacker can still send request packets. When the number of attackers is large, it is sufficient to congest a slow link's request channel.

8.3 One-Way Link Flooding Attacks

Figure 10 shows the results for one-way link flooding attacks. In our simulations, attackers launch the maximal-damage attack. That is, if their traffic to the victim is blocked, they send attack traffic to the sink node on the same side of the bottleneck as the victim. Otherwise, they attack the victim directly. As can be seen, the performance of StopIt is affected, but other systems' performance remains unchanged. With StopIt, TCP transfer times increase to 4 seconds, because filters are not installed and the attack traffic competes for bandwidth with the legitimate traffic. The transfer time does not increase after the number of simulated attackers exceeds 100K, because at this number, we have populated all ASes that can have attackers. Other schemes have similar performance in destination flooding attacks and one-way link flooding attacks.

Figure 10(c) compares different file size transfer times to show the difference between TVA+ and StopIt. Although TVA+ does not entirely block the attack traffic either, the attack traffic only competes for the request channel bandwidth. The authorized traffic is not affected. For large files, the file transfer time is significantly shorter than that in StopIt.

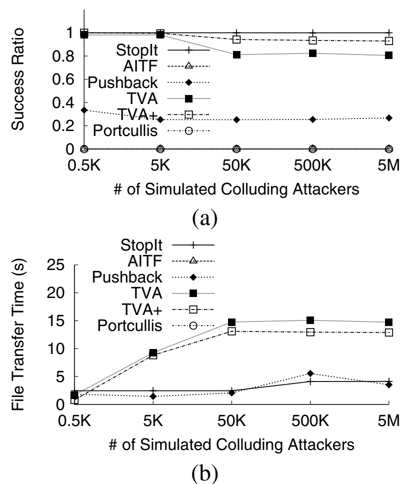

Figure 11: Two-way link flooding attacks.

8.4 Two-Way Link Flooding Attacks

In these experiments, active attackers are on both sides of the bottleneck. We refer to attackers on the same side of the bottleneck as a victim *colluding attackers* or *colluders*, and those on the opposite side as the left-side attackers. Similarly, left-side attackers and colluders attempt to launch the maximal-damage attack. In our simulations, left-side attackers send attack traffic to both the victim and their colluders. If their traffic to the victim is blocked, they use all their attack bandwidth to send attack traffic to their colluders to congest the link. At the same time, the colluders also send reverse flooding traffic. We fix the simulated left-side attackers to 5M, and vary the colluders from 500 to 5M.

Figure 11 shows the simulation results. StopIt's performance is not affected by the attack, because it queues packets based on source addresses. Colluders do not affect a legitimate user's bandwidth. Pushback's performance degrades significantly such that less than 40% of the TCP transfers finish. This is because Pushback installs destination-based rate limiters and attempts to fairly allocate a destination's bandwidth share among all senders. In the presence of y colluders and x left-side attackers, the victim only obtains $1/y$ fraction of the bottleneck bandwidth. This bandwidth is further divided among all legitimate users and x left-side attackers, each obtaining less than $\frac{1}{xy}$ fraction of the bottleneck bandwidth. The finished transfers have a short transfer time due to the on-off behavior of rate limiters: if a legitimate user is silent for a while, a router cancels its rate limiter temporarily, and its TCP transfers can finish quickly. TVA+ and TVA's performance degrade as well, because they queue authorized traffic based on destination addresses. The victim now only obtains $1/y$ fraction of the bottleneck bandwidth. But this bandwidth is shared by legitimate users only. Thus, most of their TCP transfers still finish with increased transfer times. Portcullis cannot finish its TCP transfers because the left-side attackers' puzzle level exceeds 25 seconds. AITF cannot finish because three-way handshake messages are lost and filters are not installed.

9. RELATED WORK

The design of StopIt is motivated by the criticisms on capabilities [6] and an earlier filter design AITF [5]. StopIt and AITF employ a few common design mechanisms, such as filters at edge ASes, flow cache, and filter aggregation of non-cooperating sources. But the novelty of StopIt lies in the carefully designed control channel

(§ 3) that enables filters to be installed during DoS flooding attacks, the source authentication mechanism that enables precise filtering based on source and destination addresses despite source address or path prefix spoofing attacks (§ 4.1), the filter exhaustion prevention mechanism that enables routers with a few hundred megabytes of memory to defeat strategic attacks from multimillion-node botnets (§ 4.4, § 4.5), and the fail-safe mechanism that does not involve filter installation in the core of the network, nor blocks all traffic from a source AS that fails to respond to filter requests (§ 5). To the best of our knowledge, AITF does not achieve these goals under similar attacks.

Pushback [20] uses rate limiters to reduce the attack traffic to its fair share, but it does not completely block it. Other proposals use special host hardware [27] to install filters or use a new Internet addressing architecture [3] to prevent source address spoofing attacks. StopIt does not require host hardware upgrade, and preserves the Internet's hierarchical addressing architecture. The blackholing method [12] discards attack traffic as well as legitimate one.

This work is our first step towards building a DoS-resistant network architecture that can protect anyone on the Internet, and differs in goals from other work [2, 8, 10, 13, 21, 28, 31–33].

10. CONCLUSION

This works aims to understand the effectiveness of filters and capabilities in battling DoS attacks. In the paper, we present the design and evaluation of StopIt, a filter-based DoS defense system. StopIt enables a receiver to install a network filter that blocks the undesired traffic it receives. Its design uses a novel *closed-control* and *open-service* architecture to battle strategic attacks that aim to prevent filters from being installed and to provide the StopIt service to any host on the Internet. We implement the design and evaluate its performance using both simulations and emulations. We then compare its performance with other capability-based and filter-based DoS defense systems. Our evaluation shows that StopIt outperforms existing filter-based designs, and is highly effective in providing non-interrupted communications under a wide range of DoS attacks. However, we discover that it does not always outperform a capability-based system. If the attack traffic does not reach a victim, but congests a link shared by the victim, a capability-based design is more effective. From this study, we conclude that both filters and capabilities are highly effective DoS defense mechanisms, but neither is more effective than the other in all types of DoS attacks. It is our future work to study how to build a DoS-resistant network architecture using the most cost-effective combination of various DoS defense mechanisms.

Acknowledgement

We thank Junfeng Yang, Michael Sirivianos, Ang Li, the anonymous SIGCOMM reviewers, and our shepherd Nick Feamster for their helpful feedback. This work is supported in part by the NSF Grant CNS-0627787 and Grant CNS-0627166.

11. REFERENCES

[1] IEEE Standard 802.1X. http://www.ieee802.org/1/pages/802.1x.html, 2001.

[2] D. Andersen. Mayday: Distributed Filtering for Internet Services. In *3rd Usenix USITS*, 2003.

[3] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Holding the Internet Accountable. In *ACM HotNets-VI*, 2007.

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

[5] K. Argyraki and D. R. Cheriton. Scalable Network-layer Defense Against Internet Bandwidth-Flooding Attacks. *To appear in ACM/IEEE ToN*.

[6] K. Argyraki and D. R. Cheriton. Network Capabilities: The Good, the Bad and the Ugly. In *ACM HotNets-IV*, 2005.

[7] J. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. *IEEE/ACM ToN*, 5(5), 1997.

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

[9] Deterlab. http://www.deterlab.net/.

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

[11] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, 2000.

[12] K. Foster. Application of BGP Communities. *The Internet Protocol Journal*, 6(2), 2003.

[13] A. Keromytis, V. Misra, and D. Rubenstein. SOS: An Architecture for Mitigating DDoS Attacks. *IEEE JSAC*, 22(1), 2004.

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

[15] T. Krovetz. Software-Optimized Universal Hashing and Message Authentication. UC Davis Ph.D. Dissertation, 2000.

[16] E. Larkin. Storm Worm's Virulence may Change Tactics. http://www.networkworld.com/news/2007/080207-black-hat-storm-worms-virulence.html, 2007.

[17] R. Lemos. Bots Surge Ahead in March. http://www.securityfocus.com/brief/466, 2007.

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

[19] X. Liu, X. Yang, and Y. Lu. StopIt: Mitigating DoS Flooding Attacks from Multi-Million Botnets. Technical Report 08-05, UC Irvine, 2008.

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

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

[22] P. McKenny. Stochastic Fairness Queueing. In *IEEE INFOCOM*, 1990.

[23] J. Nazario. Estonian DDoS Attacks - A Summary to Date. http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date/, 2007.

[24] K. Pagiamtzis and A. Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. *IEEE Journal of Solid-State Circuits*, 41(3), 2006.

[25] 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.

[26] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[27] M. Shaw. Leveraging Good Intentions to Reduce Malicious Network Traffic. In *USENIX SRUTI*, 2006.

[28] 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.

[29] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-Based IP Traceback. In *ACM SIGCOMM*, 2001.

[30] K. Spiess. Worm 'Storm' Gathers Strength. http://www.neoseeker.com/news/story/7103/, 2007.

[31] A. Stavrou and A. Keromytis. Countering DoS attacks with stateless multipath overlays. In *ACM CCS*, 2005.

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

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

[34] D. Wendlandt, D. G. Andersen, and A. Perrig. FastPass: Providing First-Packet Delivery. Technical report, CMU-CyLab, 2006.

[35] R. Wesson. Botnets and the Global Infection Rate: Anticipating Security Failures. http://www.stanford.edu/class/ee380/Abstracts/070606-slides.pdf, 2007.

[36] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Symposium on S&P*, 2004.

[37] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting Network Architecture. In *IEEE/ACM Transactions on Networking (to appear)*, 2009.