Scalable Network-layer Defense Against Internet Bandwidth-Flooding Attacks

Katerina Argyraki and David R. Cheriton

***Abstract*—In a bandwidth-flooding attack, compromised sources send high-volume traffic to the target with the purpose of causing congestion in its tail circuit and disrupting its legitimate communications. In this paper, we present Active Internet Traffic Filtering (AITF), a network-layer defense mechanism against such attacks. AITF enables a receiver to contact misbehaving sources and ask them to stop sending it traffic; each source that has been asked to stop is policed by its own Internet service provider (ISP), which ensures its compliance. An ISP that hosts misbehaving sources either supports AITF (and accepts to police its misbehaving clients), or risks losing all access to the complaining receiver—this is a strong incentive to cooperate, especially when the receiver is a popular public-access site. We show that AITF preserves a significant fraction of a receiver's bandwidth in the face of bandwidth flooding, and does so at a per-client cost that is already affordable for today's ISPs; this per-client cost is not expected to increase, as long as botnet-size growth does not outpace Moore's law. We also show that even the first two networks that deploy AITF can maintain their connectivity to each other in the face of bandwidth flooding. We conclude that the network-layer of the Internet can provide an effective, scalable, and incrementally deployable solution against bandwidth-flooding attacks.**

I. Introduction

In a distributed bandwidth-flooding attack, a large number of compromised sources send high-volume traffic to the target in order to create congestion and packet loss in its tail circuit; as a result, the target's communication to legitimate sources deteriorates. It has been shown that such attacks can exploit the behavior of legitimate TCP sources (which back off in the face of packet loss) to dramatically reduce their throughput or, in the case of long-lived flows, drive it to zero [1].

Real-life reports complement such analysis: The first well-documented incident we are aware of is the 2001 attack against the Gibson Research Corporation (GRC) web site. To block the flood, GRC analyzed the undesired traffic, determined its sources, and asked from their Internet service provider (ISP) to manually install filters that blocked traffic from these sources; in the meantime, their site was unreachable for more than 30 hours [2]. More recent attacks are less well documented (the victims are increasingly unwilling to reveal the details), but hint that botnet sizes have increased beyond thousands of sources, while undesired traffic is harder to identify—an article on a 2003 attack against an online betting site reports that the undesired traffic came from more than 20 000 sources, its rate ranged from 1.5 to 3 Gbps, and it was addressed at routers, DNS servers, mail servers, and web sites [3]. Despite the magnitude of the problem and the indications that it is getting worse, no effective solution has been deployed yet.

There are two basic steps in stopping a bandwidth-flooding attack: (1) identifying undesired traffic and (2) blocking it; this paper addresses the latter. To prevent undesired traffic from causing legitimate-traffic loss, it must be blocked before entering the target's tail circuit, for example, inside the target's ISP. The first solution that comes to mind is to automate the approach followed by GRC: one can imagine an ISP service, in which a flooding target sends *filtering requests* to its ISP, and, in response, the ISP installs wire-speed filters (i.e., filters that do not affect packet-forwarding performance) in its routers to satisfy these requests; each filtering request specifies traffic from one undesired-traffic source to the target.

The problem with this approach is that it requires more resources than ISPs can afford: Wire-speed filters in routers are a scarce resource, and this is not expected to change in the near future. Modern hardware routers forward packets at high rates that allow only few lookups per forwarded packet; to reduce the number of per-packet lookups, router manufacturers store filters—as well as any state that must be looked up per packet, e.g., the router's forwarding table—in TCAM (ternary content addressable memory), which allows for parallel accesses. However, because of its special features, TCAM is more expensive and consumes more space and power [4] than conventional memory; as a result, a router linecard or supervisor-engine card typically supports a single TCAM chip with tens of thousands of entries. For example, at the time of writing, the Catalyst 4500, a mid-range switch, provides a 64 000-entry TCAM to be shared among all its interfaces (from 48 to 384 100-Mbps interfaces); Cisco 12 000, a high-end router used at the Internet core, provides 20 000 entries that operate at line-speed per linecard (each linecard has up to 4 1-Gbps interfaces). So, depending on how an ISP connects its clients to its network, each client can typically claim from a few hundred to a few thousand filters—not enough to block the attacks observed today and not nearly enough to block the attacks expected in the near future [5].

One could argue that, if an ISP does not have enough filters to block traffic from each undesired-traffic source to each targeted client, it can aggregate filtering rules, i.e., use one filter to block traffic from multiple sources. The problem is that such filter aggregation can lead to *collateral damage*: imagine a scenario where 1000 AOL clients flood a public-access site and, in response, the target's ISP blocks all traffic from AOL to the target, including traffic from AOL's legitimate clients; such a measure that sacrifices a significant portion of the target's legitimate traffic can be more damaging to the target's business than the attack itself.

Manuscript received December 25, 2007.

K. Argyraki is with the School of Computer and Communication Sciences at EPFL, Switzerland (e-mail: see http://people.epfl.ch/katerina.argyraki).

D.R. Cheriton is with the Department of Computer Science at Stanford University, CA, USA (e-mail: see http://www.stanford.edu/˜cheriton).

If an ISP does not have enough filters to block undesired traffic to its clients, it can appeal to other ISPs—a distributed attack coming from hundreds of domains necessarily involves hundreds of routers, which means that millions of filters are available to help block undesired traffic. However, any filtering mechanism that involves inter-ISP cooperation faces two major challenges. The first one is securing the mechanism itself against attacks: once an ISP starts accepting filtering requests to block traffic against alleged flooding targets, a malicious entity can pose as the ISP of a flooding target and send filtering requests to disrupt communication between end hosts or even ISPs. The second challenge is motivating other ISPs to help: without an incentive, an ISP is unlikely to spend its resources helping some flooding target located in a foreign domain.

In this paper, we present Active Internet Traffic Filtering (AITF), a network-layer filtering mechanism that enables a receiver to explicitly deny tail-circuit access to misbehaving sources, while addressing these challenges. We show that:

- AITF enables a receiver to preserve on average more than 80% of its tail circuit in the face of a SYN-flooding attack that exceeds the target's tail-circuit capacity by a factor of 10 (§V).
- AITF requires an amount of per-client resources affordable for today's ISPs; the cost of these resources is not expected to increase with time, as long as botnet-size growth does not outpace Moore's law (§VI).
- AITF does not require any pre-configured inter-ISP relationships or any public-key infrastructure; it is incrementally deployable, in the sense that even the first two ISPs that deploy it can maintain their connectivity to each other in the face of bandwidth flooding (§IV).

We conclude that the network layer of the Internet can provide an effective, scalable, and incrementally deployable solution against bandwidth-flooding attacks.

The rest of the paper is organized as follows: After stating our assumptions (§II), we describe AITF in two steps: first the core of the protocol (§III), then certain extensions that shield it against non-cooperative or malicious behavior (§IV). Then we evaluate AITF: we compute its effectiveness in protecting receivers (§V) and the resources required by participating providers (§VI), then demonstrate our results through simulation (§VII). Finally, we discuss limitations and open issues (§VIII), present related work (§IX), and conclude (§X).

II. Assumptions

To block undesired traffic, we assume that receivers have the capability to identify it, while routers have the capability to filter it. We now justify these assumptions, some of which already hold in the current Internet, while others can be satisfied through existing research proposals.

1) Path Identification: We define the *domain-level path* of a received packet as the sequence of border routers that forwarded the packet; a *border router* is a router that interconnects different administrative domains. We assume that there exists a (not necessarily globally deployed) path-identification mechanism that enables participating domains to associate the packets they forward with some form of identity, such that the receiver of a packet can combine these identities and reconstruct part of the packet's domain-level path. E.g., in an early deployment scenario, as in Fig. 1, where only S_{NET} and R_{NET} have deployed path identification, the receiver R can identify $\{S_{gw}\ R_{gw}\}$ as part of the domain-level path.

Researchers have already proposed ways to provide path identification via *record route*, i.e., by enabling routers to mark forwarded packets, such that a packet's path is specified inside its headers: NIRA [6], WRAP [7], and the Points of Control approach [8] all provide sufficient path identification for AITF. Whatever the underlying record-route mechanism, we do not assume that it is globally deployed; the only domains that have to deploy it are the ones that also deploy AITF.

2) Undesired-traffic Identification: We define a *packet flow* as a sequence of packets with a common source IP address, domain-level path specification, and destination IP address; we use notation {*source domain_level_path destination*} to specify a flow. For instance, in Fig. 1, traffic with source IP address S, domain-level path specification $\{S_{gw}\ R_{gw}\}$, and destination IP address R constitutes a flow, denoted by $\{S\ S_{gw}\ R_{gw}\ R\}$. We assume that a receiver can run an "undesired-flow identification system," which takes as input incoming traffic and outputs specifications of undesired flows; a flow is classified as *undesired* once the receiver decides it does not want to receive it for a certain amount of time. We base this assumption on the fact that existing technology already identifies undesired flows in terms of their source and destination prefixes (and potentially other header fields in use today) [9], [10]; once the domain-level path is specified inside a packet's headers, it should be possible to extend this technology to take it into account.

3) Path-based Wire-speed Filtering: We assume that a router that runs the AITF protocol (which, as we will see, is necessarily a border router) can install a wire-speed filter that blocks all traffic matching a certain flow specification. We base this assumption on the fact that modern routers already use wire-speed filters to block packets based on their IP and transport-layer headers; once the domain-level path is specified inside a packet's headers, it should be possible to use the same technology to filter the packet based on its domain-level path.

4) Provider-client Message Authentication: We assume that a provider can verify the authenticity of messages sent from its own clients, and a client can verify the authenticity of messages sent from its own provider. This can be achieved with message authentication codes or three-way handshakes.

5) Non-compromised Path: We assume that, in order for a source-receiver pair to be able to communicate, the network elements (typically routers) that are on the path between them must not be compromised. Our rationale is that, once a router gets compromised, all the communications served by it are at its mercy: the router can drop their traffic or hijack their TCP connections. Of course, if a source-receiver pair can communicate over multiple paths, and at least one of them is not compromised, they should be able to maintain their communication—akin to how multi-path communication between access points and clients increases attack resilience in the Stateless Multipath Overlays approach [11]. Combining multi-path with AITF is part of our future work.

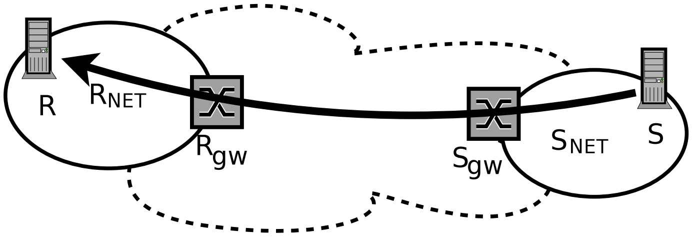


Fig. 1. Source S sends undesired traffic to receiver R through routers S_{gw} (in S's domain) and R_{gw} (in R's domain). S_{NET} and R_{NET} have deployed AITF (and the underlying path-identification mechanism). R identifies $\{S\ S_{gw}\ R_{gw}\ R\}$ as an undesired flow.

III. The Basic AITF Protocol

We now describe the basic elements of the Active Internet Traffic Filtering (AITF) protocol. For simplicity, we initially assume all domains that deploy AITF to be honest and well behaved, then relax these assumptions in §IV.

A. Players

AITF involves four players per undesired flow, illustrated in Fig. 1:

- The *receiver* R is the target of the undesired flow.
- The *source* S is the node generating the undesired flow.
- The *receiver's gateway* R_{gw} is a border router located in R's ISP, on the path from S to R, before R's tail circuit. Note that R_{gw} is not significantly affected by the attack; if it were (i.e., if its own tail circuit were congested), R_{gw} itself would be the "receiver," while the role of the receiver's gateway would be played by another router upstream.
- The *source gateway* S_{gw} is a border router located in S's ISP, on the path from S to R.

These four players communicate through AITF messages, which include one or more *filtering requests*. Each filtering request includes the specification of an undesired flow F and the amount of time W_f (called the *filtering window*) for which the requester does not want to receive F.

For simplicity, we make three temporary assumptions that we relax in §IV:

1) The source gateway S_{gw} cooperates with the receiver's gateway R_{gw} to help the receiver.

2) Filtering requests are not malicious, i.e., they indeed originate from the specified undesired-traffic receiver R and correspond to traffic indeed sent from the specified source S.

3) The receiver can trust the path specified inside each received packet, i.e., it knows the true source S and the true source gateway S_{gw} for each undesired flow.

B. Algorithm Overview

Once a receiver R identifies an undesired flow F, it contacts the corresponding source S and asks it to stop sending F for an amount of time W_f. R's request is propagated through R_{gw} and S_{gw}, which temporarily block F to immediately protect R until S complies. The parameters of the protocol are defined in Table I.

More specifically, R sends a filtering request to its gateway R_{gw} to block F for W_f. In response, R_{gw} installs a temporary filter that blocks F for time $T_{dr} \ll W_f$ and forwards the request to the source gateway S_{gw}; once S_{gw} satisfies the request, R_{gw} removes its temporary filter. Similarly, S_{gw} installs a temporary filter that blocks F for time $T_{ds} \ll W_f$, logs the request for W_f, and forwards the request to S; once S satisfies the request, S_{gw} removes its temporary filter.

If S does not cooperate (i.e., continues to send F), S_{gw} classifies S as *non-cooperating* and blocks all S-originated traffic. If S "pretends" to cooperate (i.e., stops sending F, but resumes before W_f has elapsed), the following takes place: R sends a second filtering request against S; upon receiving this second request, S_{gw} checks its log, detects that S has already been told to stop sending F, classifies S as non-cooperating, and blocks all S-originated traffic.

C. Details and Rationale

We now discuss and justify the three key elements of the algorithm outlined above.

1) Temporary Filters: AITF-enabled routers install wire-speed filters only temporarily, in order to allow for efficient filter reuse: The receiver's gateway installs a filter to selectively block each undesired flow for $T_{dr} \ll W_f$; once the source gateway has taken over filtering this flow, the receiver's gateway can reuse its filter to satisfy another filtering request. Similarly, the source gateway installs a filter to selectively block each undesired flow for $T_{ds} \ll W_f$; once the source stops sending the flow, the source gateway can reuse its filter to satisfy another filtering request. Of course, the source gateway must still keep a log on each filtering request for W_f, in order to ensure that the corresponding source is cooperating. However, keeping a log on a filtering request for tens of minutes is significantly less expensive than filtering the corresponding undesired flow for the same amount of time; this is because filters are stored in TCAM, whereas logs can be stored in conventional DRAM, accessed off the router's fast path. So, AITF does not reduce the amount of router state necessary to block an undesired flow, but "moves" it off the fast path, i.e., from an expensive, physically-limited state store to a commoditized one.

2) Selective vs Aggregate Filtering: A source gateway uses two different ways to block a source's undesired traffic: first, it uses multiple selective temporary filters to block each undesired flow; second, if the source is classified as non-cooperating, the source gateway by default uses a single aggregate long-term filter to block all its traffic, until the source's owner fixes the vulnerability that caused it to send undesired traffic and contacts her provider. Selective filtering preserves the source's legitimate traffic, but requires one filter per undesired flow; hence, it is reserved for well behaved sources, which quickly cooperate, allowing the source gateway to reuse its filters for other purposes. Aggregate filtering requires a single filter, but sacrifices the source's legitimate traffic. We choose this, admittedly draconian, default policy against non-cooperating sources, because it minimizes the amount of filtering resources that a provider spends on misbehaving clients. However, a provider is free to implement

Parameter	Meaning	Example value
Outbound filtering request rate (REQ_{out})	R_{gw} honors this rate of filtering requests from R.	1000 req/sec
Inbound filtering request rate (REQ_{in})	S_{gw} honors this rate of filtering requests against S.	1000 req/sec
Filtering window (W_f)	After W_f, S is allowed to send F traffic again.	10 minutes
R_{gw} deadline (T_{dr})	R_{gw} expects S_{gw} to block F within T_{dr} from the moment it sends the filtering request to it.	1 sec
S_{gw} deadline (T_{ds})	S_{gw} expects S to stop F within T_{ds} from the moment it sends the filtering request to it.	10 msec

TABLE I
THE PARAMETERS OF THE AITF PROTOCOL

more lenient per-client policies, where it blocks progressively larger aggregates from a non-cooperating client (rather than immediately block all its traffic), as long as it can afford the necessary resources (we discuss this in §VIII-B).

3) Filtering Window: Each filtering request includes a filtering window W_f, i.e., the amount of time for which the specified undesired-traffic source is asked to stop sending traffic to the receiver, which is on the order of minutes. Choosing a value for W_f involves the following trade-off: The larger W_f is, the longer each source is forced to stop sending traffic to the receiver—otherwise, it gets all its traffic blocked. However, undesired-traffic sources are typically infected hosts that, once patched and brought back online, should be able to send (legitimate) traffic to the receiver; the larger W_f is, the longer each former undesired-traffic source must wait before being able to send legitimate traffic to the receiver once it has been patched. We show how W_f affects the effectiveness of AITF in §V; we show how it affects its cost in §VI.

D. Controlling Resource Consumption

There are three knobs for controlling resource consumption:

1) The receiver's gateway has the notion of a per-client *outbound filtering request rate* (REQ_{out}), which is the maximum rate of filtering requests from this client that R_{gw} honors; if the client exceeds this rate, its requests get dropped. In this way, a single undesired-traffic receiver cannot exhaust/monopolize its provider's resources.

2) The source gateway has the notion of a *maximum filtering window* W_{fmax}: even if the filtering window specified in a filtering request exceeds W_{fmax}, the source gateway logs the filtering request only for W_{fmax}.

3) The source gateway also has the notion of a per-client *inbound filtering request rate* (REQ_{in}), which is the maximum rate of filtering requests against this client that S_{gw} honors; requests against the client beyond this rate are dropped.

E. Legacy Traffic

When the traffic addressed to an AITF receiver exceeds a pre-configured threshold, the receiver's gateway gives priority to packets carrying path information; in this way, undesired legacy traffic can only affect other legacy traffic, but not the traffic coming from other AITF-enabled domains.

IV. OPERATION IN NON-COOPERATIVE ENVIRONMENTS

In a non-cooperative environment, the three assumptions made in §III-A do not always hold, namely, source gateways may not cooperate, filtering requests may be malicious, and path specifications may be spoofed. We now remove these assumptions and show how AITF operates in non-cooperative, potentially malicious environments.

A. Non-cooperating Source Gateways

A source gateway may not cooperate for three reasons: first, it may not have the resources to satisfy every filtering request; second, it may be compromised and controlled by the same attacker that controls the undesired-traffic source(s); third, and most important, a provider has no incentive to block traffic from (and potentially dissatisfy) its own clients, even if they are misbehaving, in order to help some complaining host located in a foreign administrative domain.

To deal with non-cooperating source gateways, AITF offers the option of *escalation*: a receiver R can ask its gateway to block *all* traffic from a non-cooperating source gateway S_{gw} to R. Hence, a source gateway either cooperates and blocks undesired traffic from its misbehaving clients or risks losing all connectivity to the complaining receiver.

In a full-deployment scenario, R escalates by sending out a filtering request that specifies all traffic from S_{gw} as the undesired flow and the next border router on the path from S_{gw} to R as the source gateway—i.e., the non-cooperating source gateway becomes an undesired-traffic source, while the role of the source gateway is now played by another border router. In an early deployment scenario, where the only AITF-enabled routers are S_{gw} and R_{gw}, this is not an option; in this case, R sends out filtering requests against S_{gw} every T_{dr}, such that all traffic from S_{gw} to R remains blocked at R_{gw}.

Escalation is reminiscent of aggregate filtering of a misbehaving source's traffic: using multiple selective filters to block each undesired flow from a non-cooperating source gateway would be too expensive; instead, the receiver's gateway uses a single aggregate filter to block all traffic from the source gateway to the receiver. We choose this as the default policy toward non-cooperating source gateways, because it minimizes the amount of filtering resources that a receiver consumes due to non-cooperating networks that refuse to deal with their misbehaving clients. Of course, a receiver is free to implement more lenient policies, as long as it can afford the necessary resources (we discuss this in §VIII-B).

B. Malicious Filtering Requests

A filtering request has three possible outcomes: all traffic from the source S to the receiver R is blocked; all traffic from the source S is blocked; or, all traffic from the source gateway S_{gw} to R is blocked (if S_{gw} does not cooperate). Hence, there are three ways in which a malicious node M may try to abuse filtering requests to disrupt legitimate communications:

- It may try to disconnect a source S from a receiver R: pretend to be R's gateway and ask from S to stop sending it traffic.

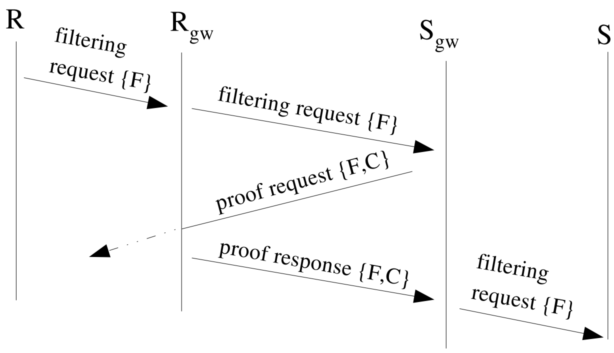


Fig. 2. AITF message exchange: The receiver R sends a filtering request to its gateway R_{gw}, specifying an undesired flow F; R_{gw} propagates the request to the source gateway S_{gw}; S_{gw} responds with a proof request sent to R that includes the undesired-flow specification and a cookie C; R_{gw} intercepts the proof request and generates a proof response with the undesired-flow specification and the same cookie C; S_{gw} receives the proof response, verifies that it includes a valid cookie, and propagates the filtering request to the undesired-traffic source S.

- It may try to disconnect a source S from its network: send bogus filtering requests against S, so that S is misclassified as non-cooperating and blocked.
- It may try to disconnect a source gateway S_{gw} from its receivers: send it lots of bogus filtering requests, so that S_{gw} cannot respond to legitimate requests by honest receivers and loses access to these receivers.

Next, we describe how AITF prevents each type of abuse.

1) Verifying the Origin of Filtering Requests: The source and receiver's gateways perform a three-way handshake, illustrated in Fig. 2: when receiver gateway R_{gw} sends a filtering request to source gateway S_{gw}, the latter responds with a *proof request*, addressed to the alleged receiver R; by intercepting this proof request and sending it back (as a *proof response*) to S_{gw}, R_{gw} proves that it is indeed on the path to R.

To prevent a malicious node M that is not on the path from R_{gw} to S_{gw} from guessing the contents of the proof request, S_{gw} includes in it a *cookie*, which is computed as follows:

$$cookie = hash_{ck}(R)$$

where $hash$ is a one-way keyed hash function, ck is a periodically regenerated secret known only to S_{gw}, and R is the IP address of the alleged receiver. The cryptographic properties of the hash function guarantee the following: first, M cannot practically guess the cookie included in the request, unless it knows the secret ck; second, a node that *is* on the path of a certain proof request and observes the included cookie cannot practically compute from it the value of the secret or of cookies that correspond to other receivers. Note that this stateless-handshake approach protects S_{gw} from filtering-request floods the same way TCP SYN cookies protect servers from SYN floods [12].

2) Verifying Non-cooperation Claims: Before classifying a source S as non-cooperating, its gateway S_{gw} first monitors S's traffic to verify that it is indeed misbehaving; by "monitoring" we mean that S_{gw} keeps a record of the destinations S has sent traffic to within a certain period of time W_m (called the *monitoring window*), which is on the order of a few seconds. Whenever a new filtering request arrives against S, S_{gw} checks its record; if the request does not concern traffic sent by S within the last monitoring window, it is dropped. As a result, S is never misclassified and blocked due to a false claim.

To allow the source gateway to verify non-cooperation claims in this manner, each receiver gives it two chances to cooperate: Suppose receiver R has sent a filtering request against undesired flow F generated by source S. If F reappears before W_f has elapsed, R sends a second filtering request against it; in response, S_{gw} starts monitoring traffic from S. If F appears for a third time, R sends a third filtering request against it; now S_{gw} can check its monitoring log, verify that S is not cooperating, and block all its traffic. In the worst-case scenario, S_{gw} does not cooperate and F appears for a fourth time; as a result, R escalates and blocks all traffic from S_{gw}.

3) Taming Filtering-request Floods: The same technique is used to prevent a set of compromised nodes from exhausting a source gateway's resources with bogus filtering requests (including non-cooperation claims): whenever a source gateway receives an unusually high rate of requests from an alleged receiver's gateway, it starts monitoring traffic to that gateway; if it turns out that its requests are bogus (i.e., do not correspond to traffic actually sent to it during the last monitoring window), the receiver's gateway is classified as "malfunctioning" and all its filtering requests are dropped. An honest receiver's gateway can prevent its clients from causing it to be classified as malfunctioning by regulating the rate of outbound filtering requests per targeted source gateway.

C. Spoofed Addresses and Paths

The path specified inside a received packet may not correspond to the actual path followed by the packet because of source-address or path spoofing.

1) Source-address Spoofing: AITF does not detect source-address spoofing nor handle it specially. If the source specified in an undesired flow is S, the corresponding source gateway is asked to block all traffic from S to the receiver, even if S is not the true identity of the undesired-traffic source. A malicious node can abuse this to disrupt communications between a receiver and a source located behind the same gateway with the malicious node. It is up to the provider of the malicious node to detect this activity or prevent it by taking anti-spoofing measures. I.e., if a provider hosts potential undesired-traffic sources, it is in the provider's best interest to prevent source-address spoofing within its network, in order to protect its own clients from each other.

2) Path Spoofing: Consider a partial deployment scenario, where two AITF-enabled domains are interconnected through legacy domains, as in Fig. 1. In this case, a malicious node located in a legacy domain can generate packets that appear to be coming from S_{NET} and are addressed to R. As a result, S_{NET} is asked to block traffic it does not generate; if it just drops the request, R misclassifies S_{NET} as non-cooperating and potentially blocks all its traffic.

We can prevent such abuse in the following way: Each source gateway S_{gw} can (1) mark outgoing packets with a *stamp* that depends on the packet's destination domain and (2) communicate that stamp to the corresponding receiver's gateway R_{gw}; the latter can then drop all incoming traffic that claims to be coming from S_{gw} but has an invalid stamp. Such packet stamping can be initiated by S_{gw}, in response to filtering requests against traffic that it never sent; in this case, S_{gw} piggy-backs the stamp on the corresponding three-way handshake. Alternatively, R_{gw} can preventively ask for packet stamping from all its source gateways, before any attack takes place; in this case, it pays the cost of more incoming traffic (because all packets are augmented with a stamp), but drops all spoofed traffic, making it easier for its clients to identify undesired flows.

To prevent a malicious node M that is not on the path from S_{gw} to certain destination from guessing the corresponding stamp[1], S_{gw} computes stamps as follows:

$$stamp = hash_{sk}(dst\_prefix)$$

where $hash$ is a one-way keyed hash function, sk is a periodically regenerated secret known only to S_{gw}, and dst_prefix is the destination prefix of the packet. The cryptographic properties of the hash function guarantee the following: First, M cannot practically guess the stamp included in a packet, unless it knows the secret sk. Second, a node that *is* on the path of a certain packet and observes the included stamp cannot practically compute from it the value of the secret or of stamps that correspond to other destination domains.

D. Summary

To deal with non-cooperating gateways, AITF offers the option of escalation: ISPs that host attack sources either cooperate and police their misbehaving clients, or risk losing all access to the complaining receiver(s). The origin of a filtering request is verified through a three-way handshake between the two involved networks, while the claim of a filtering request is verified by monitoring the alleged source. Finally, to prevent path spoofing, source networks mark outgoing packets with hard-to-guess, destination-dependent stamps.

V. Effectiveness

In this section, we describe the different flooding strategies that can be used against an AITF-enabled receiver, then compute an upper bound on the damage each strategy can inflict on the receiver's tail circuit. We omit the straightforward proofs and justify these bounds intuitively; for more details, we refer the reader to [13].

A. Attack Model

Each attack consists of a certain number of undesired flows N_{uf}; each undesired flow corresponds to one source. Different sources may send at different rates, but, for simplicity, we assume that each source i sends at one rate r_i.[2] The highest rate of undesired traffic arrives at the receiver's tail circuit when all sources send at the same time; we call this the *aggregate undesired-traffic rate* and it is equal to $R_{ut} = \Sigma_{\forall i} r_i$, $i \in [0, N_{uf})$. This model corresponds to an attack in which the botnet master turns different bots on and off, but does not vary the rate at which each bot sends when it is on.

The receiver identifies undesired flow i after receiving b_{id}^i bits from it. Hence, the total number of unidentified bits that the receiver gets before identifying all undesired flows is $B_{id} = \Sigma_{\forall i} b_{id}^i$, $i \in [0, N_{uf})$; we call this the *identification overhead*. The receiver uses the same filtering window W_f in all filtering requests it sends throughout the attack; W_f is larger than the amount of time it takes to receive B_{id} bits.

A source S can inflict different types and amounts of damage to the receiver R's tail circuit depending on how it behaves when asked to stop sending undesired flow F. Ideally, S stops sending F and never resumes—we do not discuss this further, as it is the best case for our mechanism. We distinguish three other cases:

- *Deaf* sources ignore filtering requests and are immediately identified and blocked.
- *Lying* sources stop sending undesired traffic when so requested, but resume after the corresponding source gateway has removed its temporary filter, before W_f has elapsed. As a result, they manage to send multiple rounds of traffic before they are identified and blocked.
- *On-off* sources cooperate with filtering requests, but resume sending undesired traffic after the requests have expired. As a result, they avoid punishment, yet force the receiver to re-detect their undesired traffic and send new filtering requests against them every filtering window.

To describe the short- and long-term damage inflicted by these flooding strategies, we use the following metrics:

- The *initial overhead* is the number of undesired bits received until all undesired flows are identified and blocked for the first time.
- The *tail-circuit capacity loss* is the fraction of the receiver's tail circuit that is consumed by undesired traffic throughout the attack, computed at the granularity of a filtering window.

We express our results in terms of the AITF parameters defined in Table I and the receiver and attack parameters defined in Table II.

B. Result Summary

The initial overhead inflicted by deaf sources is bounded according to Eq. 1; this gives the maximum number of undesired bits received until the receiver identifies all undesired flows and sends a filtering request against each one, and until these requests take effect.

Lying sources can inflict a bigger initial overhead, bounded according to Eq. 2, at the cost of having all their traffic

[1] In [13] we compute the probability of an off-path attacker guessing the stamp as a function of its size. As an example, in the current Internet, a 128-bit stamp could be guessed with probability $1.3 \cdot 10^{-25}$, while it would introduce roughly 4% bandwidth overhead.

[2] The model could be easily adjusted to capture the case where each source sends at more than one rate, if we break the traffic sent by each source into multiple flows in such a way that each flow has only one possible rate.

Metric	Description	Units
Tail-circuit capacity (C_{tc})	The capacity of the bottleneck link between the receiver and its gateway.	bps
Tail-circuit RTT (RTT_{tc})	The round-trip time between the receiver and its gateway.	seconds
Aggregate undesired-traffic rate (R_{ut})	The maximum rate at which undesired traffic arrives at the receiver's tail circuit.	bps
Average undesired-flow rate ($\bar{r}_i$)	The average rate at which each undesired flow arrives at the receiver's tail circuit.	bps
Aggregate identification overhead (B_{id})	The total number of unidentified bits that the receiver must get before identifying all undesired flows.	bits
Identification time (T_{id})	$T_{id} = \frac{B_{id}}{R_{ut}}$ A measure of the amount of time it takes to identify an undesired flow. It corresponds to the average identification overhead divided by the average undesired flow rate.	seconds
Number of undesired flows (N_{uf})	The total number of different undesired flows sent to the receiver during the attack. Each undesired flow corresponds to a single source.	
Request time (T_{req})	$T_{req} = \frac{N_{uf}}{REQ_{out}}$ The amount of time it takes to send filtering requests against all undesired flows.	seconds

TABLE II
PARAMETERS USED TO QUANTIFY AITF EFFECTIVENESS

blocked. The factor of 4 in Eq. 2 captures the fact that lying sources force the receiver to send multiple (up to four) filtering requests in order to block each undesired flow.

On-off sources inflict the same maximum initial overhead with deaf sources. Moreover, they periodically resume their attack and, hence, re-inflict up to the same overhead during each filtering window; the resulting tail-circuit capacity loss is bounded according to Eq. 4.

To give some concrete numbers, consider a receiver with tail-circuit capacity $C_{tc} = 100$ Mbps, RTT_{tc} a few milliseconds, filtering window $W_f = 10$ minutes, and outbound filtering request rate $REQ_{out} = 1000$ requests/second. Suppose this site is under a SYN-flooding attack by $N_{uf} = 100\,000$ on-off sources; each source sends 10 Kbps, so undesired traffic arrives at the site's tail circuit at ten times its capacity, i.e., $R_{ut} = 1$ Gbps. Suppose it takes 10 Kbits to identify each undesired flow (roughly 20 SYN packets), so $T_{id} = 1$ second. Given these numbers, $\lambda \leq 0.19$, which means that the target preserves on average more than 80% of its tail circuit in the face of an attack that exceeds its capacity by a factor of ten.

C. Deaf and Lying Sources

Consider a host R receiving N_{uf} flows from deaf sources. R incurs initial overhead

$$B_o \leq B_{id} + C_{tc}\,T_{req} + R_{ut}RTT_{tc} \qquad (1)$$

The bound is derived by breaking down the impact of deaf sources on the receiver's tail circuit into three components: The *identification overhead* (B_{id}) consists of undesired traffic received before the corresponding flows are identified as undesired. The *request overhead* ($C_{tc}\,T_{req}$) consists of identified undesired traffic received before R sends filtering requests against the corresponding flows: it takes $T_{req} = \frac{N_{uf}}{REQ_{out}}$ seconds for R to send filtering requests against all N_{uf} flows; the maximum number of bits that R can receive within this interval is $C_{tc}\,T_{req}$. The *blocking overhead* ($R_{ut}RTT_{tc}$) consists of identified undesired traffic received after R has sent filtering requests against the corresponding flows.

Now consider a host R receiving N_{uf} flows from lying sources. R incurs initial overhead

$$B_l \leq B_{id} + 4(C_{tc}\,T_{req} + R_{ut}RTT_{tc}) \qquad (2)$$

The difference between deaf and lying sources is that the latter manage to send undesired traffic up to four times before being identified and blocked (see §IV-B2)—hence, R incurs up to four times the maximum request and blocking overhead caused by deaf sources.

Eq. 1 and 2 may overestimate initial overhead in two ways: First, they assume that the receiver *first* identifies all flows and *then* starts sending filtering requests against them; this may not be the case, for instance, in a simple SYN-flooding attack, where the receiver can identify flows and send filtering requests in parallel. Second, the equations assume that the receiver's tail circuit is flooded until all undesired flows have been blocked; this may not be the case, as there may not be enough undesired flows to consume 100% of the receiver's tail circuit. So, the upper bounds of the two equations are reached only in sophisticated attacks, where (1) the receiver must first process all undesired flows before identifying them and (2) there are enough undesired flows to consume all of the receiver's tail circuit until they are all blocked.

D. On-off Sources

The maximum number of on-off sources that an AITF-enabled receiver can keep blocked is

$$N_{fmax} = REQ_{out} \cdot W_f \qquad (3)$$

The gist of the equation is that a receiver cannot block more flows than can be blocked within a filtering window W_f: once W_f elapses, previously blocked flows reappear, and the receiver must spend its resources re-blocking these recurring flows rather than block new ones. For instance, a receiver with $REQ_{out} = 1000$ flows/second and $W_f = 10$ min cannot handle more than $600\,000$ on-off sources.

Consider a host R receiving $N_{uf} \leq N_{fmax}$ flows from on-off sources. R incurs initial overhead B_o, bounded according to Eq. 1, and tail-circuit capacity loss

$$\lambda \leq \frac{R_{ut}}{C_{tc}} \cdot \frac{T_{id} + RTT_{tc}}{W_f} + \frac{T_{req}}{W_f} \qquad (4)$$

The bound is derived by assuming that R receives B_o undesired bits during every filtering window W_f.

The first term of Eq. 4 captures the identification and blocking overhead, i.e., the fact that the receiver must identify and block each undesired flow during every filtering window; it depends on how the undesired-traffic rate (R_{ut}) compares to the receiver's tail-circuit capacity (C_{tc}), and how the amount of time for which each undesired flow is received

$(T_{id} + RTT_{tc})$ compares to the amount of time for which it is blocked (W_f). The second term captures the request overhead, i.e., the fact that the receiver must send filtering requests against all undesired flows during every filtering window; it depends on how the amount of time it takes to send filtering requests (T_{req}) compares to the amount of time for which undesired flows are blocked (W_f).

E. Limits

When the aggregate undesired-traffic rate R_{ut} is so large that the bound given by Eq. 4 exceeds 1, then the receiver's tail circuit is flooded despite AITF. For instance, consider again the example of Section V-B, where a receiver with tail-circuit capacity $C_{tc} = 100$ Mbps is under attack by $N_{uf} = 100\,000$ on-off sources. We said that, if $R_{ut} = 1$ Gbps, AITF enables this receiver to preserve more than 80% of its tail-circuit capacity. However, if the same receiver comes under a $R_{ut} = 50$-Gbps attack, then its tail circuit is flooded despite AITF.

On the other hand, if an attack is sending 50 Gbps to a certain site, it is most likely flooding not only the site's connection to its ISP, but also the ISP link that carries the undesired traffic to the site—today, such an attack would be enough to flood the Internet core. This means that the "tail circuit" is not the site's 100 Mbps connection, but the congested ISP link (which most likely has a capacity of hundreds of Mbps), and the "receiver" of the undesired traffic is not the site, but the ISP router at the end of the congested ISP link. So it is now up to the affected router to identify undesired flows and send filtering requests against them.

VI. Deployment Cost

In this section, we look at AITF deployment cost. For completeness, we first discuss what it takes to identify undesired traffic today—we do not provide a complete solution, merely give a rough sense of the kind of tools and the amount of time it requires (§VI-A). Then, we compute the amount of resources required to deploy AITF today (§VI-B) and examine how their cost is expected to evolve as the Internet grows (§VI-C).

A. Undesired-traffic Identification

Accurate identification requires anti-spoofing measures, so the first question is whether spoofing is still an issue today. Beverly and Bauer's Spoofer project currently shows that close to a fifth of Internet addresses and a quarter of Autonomous Systems allow the corresponding hosts to spoof [14]. So, even though we do not know to what extent bandwidth-flooding sources use spoofing (victims do not typically release such information), we do know that many of them can still do it. Hence, a provider that chooses to offer its clients the capability of quick identification (and, hence, blocking) of undesired traffic, must pay the cost of preventive packet stamping (§IV-C).

Next, we look at how a receiver can identify (non-spoofed) undesired flows. We first consider a simple SYN-flooding attack, where 100 000 sources flood the receiver's 100 Mbps tail circuit with SYN packets—we choose these values, because, at the time of writing, botnet sizes are reported to be on the order of tens of thousands, while a Web search for "unmetered servers" shows that 100 Mbps is the state-of-the-art tail-circuit capacity for online services. Suppose the receiver classifies traffic from a source as an undesired flow, as soon as it has received 20 SYN packets (about 10 Kbits) from the source. In this case, the receiver needs 1 Gbit of undesired traffic to identify all undesired flows; assuming legitimate TCP flows consume a small fraction of the tail circuit and quickly back off in the face of the attack, it takes about 10 seconds for the receiver to identify all undesired flows.

An attacker can make things harder by emulating a flash crowd, i.e., flood the receiver with legitimate-looking traffic, such that the receiver cannot tell legitimate from undesired flows. One way to deal with such an attack is to classify a flow as undesired when it is not generated by a human user. Online services already use reverse Turing tests to identify flows that are not generated by humans; researchers have showed how to use them to identify denial-of-service traffic: the receiver responds to each SYN packet with a reverse-Turing test challenge; traffic from sources that keep sending false responses (or new SYN packets) is classified as undesired [15]. In this way, an attacker cannot use bots to emulate legitimate behavior and can at best resort to a simple SYN-flooding attack as described above.

B. A Sample Configuration and its Cost

Table III shows the number of per-client filters and the amount of per-client memory that a provider needs to deploy AITF as a function of its parameters. Note that the provider's routers act both as receiver's and source gateways for its clients, hence, the provider needs resources to satisfy both outbound and inbound filtering requests.

To use the equations of Table III, we must first derive a sample AITF configuration—if a provider deployed AITF today, what would be reasonable values for the parameters of Table I?

The **receiver-gateway deadline** (T_{dr}, the amount of time for which the receiver's gateway blocks each undesired flow until the source gateway takes over) must be long enough to accommodate the three-way handshake between the receiver's and the source gateway; given that, at the time of writing, Internet round-trip times are on the order of hundreds of milliseconds [16], a conservative value would be 1 second. Similarly, the **source-gateway deadline** (T_{ds}) must be long enough to accommodate a round-trip time between the source gateway and the source; a conservative value for a modern network would be 10 milliseconds.

The maximum **outbound filtering-request rate** (REQ_{out}) that the receiver's gateway accepts from a certain receiver is bounded by the number of filters F_{out} that the provider can devote to that receiver. In §I, we argued that, at the time of writing, it is reasonable to assume from a few hundred to a few thousand filters per client; for instance, if $F_{out} = 1000$ filters and $T_{dr} = 1$ second, the provider can accept $REQ_{out} = 1000$ requests/second from the receiver (Table III). When source

Resource	Amount	Usage	Explanation
Filters	$F_{out} = REQ_{out} \cdot T_{dr}$	Block unwanted traffic to the client.	The corresponding receiver's gateway accepts request rate REQ_{out} from the client and blocks each specified flow for T_{dr}.
	$F_{in} = REQ_{in} \cdot T_{ds}$	Block unwanted traffic from the client.	The corresponding source gateway accepts request rate REQ_{in} against the client and blocks each specified flow for T_{ds}.
DRAM slots	$L = REQ_{in} \cdot W_{fmax}$	Log filtering requests against the client.	The corresponding source gateway accepts request rate REQ_{in} against the client and logs each request for W_{fmax}.

TABLE III
PER-CLIENT RESOURCES REQUIRED BY AN AITF-ENABLED PROVIDER

gateways cooperate, this rate is enough to block traffic from up to $600\,000$ sources (§V-D, Eq. 3). In the worst-case scenario (none of the source gateways cooperate, and none of the other border routers on the path support AITF), the receiver's gateway locally blocks traffic from each source gateway to the receiver (§IV-A); in that case, a rate of 1000 requests/second is enough to block traffic from 1000 source gateways.

When source gateways do not cooperate and the receiver is not granted a large enough REQ_{out} to have traffic from each one of them blocked, it must somehow choose the "worst" source gateways to block. Chen et al. recently observed that compromised sources tend to be clustered—in their DShield.org trace, about 80% of the sources were concentrated in the same 20% of the IP address space [17]. At the time of writing, assuming one domain per Autonomous System (AS), a rate of 6000 requests/second would be enough to block traffic from about a fifth of Internet domains.

The maximum **inbound filtering-request rate** (REQ_{in}) that the source gateway accepts against a certain source is bounded by the number of filters F_{in} that the provider can devote to this task; assuming $F_{in} = 10$ filters and $T_{ds} = 10$ milliseconds, the provider can accept $REQ_{in} = 1000$ requests/second against the source (Table III).

The **filtering window** (W_f) must be long enough so that the receiver has time to identify all the undesired flows and send filtering requests against them before they start reappearing—hence, the right value depends on the nature of the attack. For instance, consider a SYN-flooding attack (like the one described in §VI-A), where $100\,000$ sources generate 1 Gbps of undesired traffic against the receiver's 100 Mbps tail circuit: it takes about 10 seconds for the receiver to identify all undesired flows and, assuming $REQ_{out} = 1000$ requests/second, at most another 10 seconds to send filtering requests against them; in this case, a filtering window of $W_f = 10$ minutes preserves more than 80% of the receiver's tail circuit (§V-D, Eq. 4).

The filtering window determines the amount of memory used at the source gateway to log incoming filtering requests; assuming a maximum filtering window of $W_{fmax} = 10$ minutes and $REQ_{in} = 1000$ requests/second, a provider needs enough memory to log $L = 600\,000$ requests per client (Table III); assuming about 20 bytes per request (enough to fit the receiver and receiver-gateway's addresses and a timestamp), this corresponds to 12 MB of DRAM per client.

To summarize, our sample configuration requires a few thousand filters and a few MB of DRAM per client; today, such resources would be enough to protect a significant fraction of each receiver's tail circuit, even when undesired-traffic rate exceeds its tail-circuit capacity by several factors.

C. Evolution of AITF Cost

AITF guarantees limited tail-circuit capacity loss in the face of flooding attacks, as specified by Eq. 4, repeated here in an equivalent form:

$$\lambda \leq \frac{\bar{r}_i}{C_{tc}} \cdot \frac{N_{uf}}{W_f} \cdot (T_{id} + RTT_{tc}) + \frac{N_{uf}}{W_f} \cdot \frac{1}{REQ_{out}}$$

where all parameters are defined in Tables I and II. We now examine what the receiver must do in order for this guarantee to remain the same as the Internet grows.

Assuming T_{id}, RTT_{tc}, and REQ_{out} remain stable, we expect that: (1) $\bar{r}_i$ (the average undesired-traffic rate per source) will grow at the same rate with the average tail-circuit capacity of Internet hosts; to keep λ stable, the receiver must increase C_{tc} at the same rate with $\bar{r}_i$—i.e., the receiver's tail-circuit capacity must keep up with the average tail-circuit capacity in the Internet. (2) N_{uf} (the number of undesired flows per attack) will grow as botnet sizes increase; to keep λ stable, the receiver must increase W_f at the same rate with N_{uf}—i.e., as the number of undesired flows grows, each individual flow must be blocked for a longer period of time.

According to Table III, if receivers increase their filtering windows, the amount of per-client DRAM required to log filtering requests will increase accordingly—the intuition is that, as filtering windows grow, a provider that hosts undesired-traffic sources must be able to remember each undesired flow for a longer period of time, which means that it needs more memory for logging filtering requests. Hence, the amount of per-client DRAM must increase at the same rate with the number of bots attacking the receiver. This means that the evolution of AITF cost depends on two factors: botnet growth and the fall of DRAM price; as long as the former does not outpace the latter, AITF cost is not expected to rise.

DRAM price has consistently been dropping to half every 18 months for the last 30 years. Assuming it continues to fall at this rate, AITF cost is not expected to increase, unless botnet-size growth outpaces Moore's law—i.e., in 15 years from now, there are botnets consisting of tens of millions of hosts. In this unfortunate situation, either the cost of AITF will rise, or receivers will have to aggregate undesired traffic more aggressively, at the cost of sacrificing a certain amount of legitimate traffic.

VII. SIMULATIONS

In this section, we use simulation to analyze the effect of undesired traffic on AITF-enabled receivers and illustrate the effectiveness of AITF against bandwidth flooding.

A. Simulation Framework

The goal of our simulation was to illustrate not only the tail-circuit capacity loss (which is computed at the granularity of a filtering window), but also the burstiness of the undesired traffic *within* each filtering window. To capture the short-term dynamics of undesired traffic, we needed our simulation to work at the granularity of individual sources and bottleneck links. None of the existing network-simulation packages allowed us to simulate attacks from tens to hundreds of thousands of sources at such granularity, which led us to create our own framework.

In the beginning of each simulation, we create a set of interconnected (core and edge) border routers—to create a realistic topology, we used BGP routing tables from Route Views [18] and applied to them Gao's algorithm for inferring inter-AS relationships [19]. We also create a set of sources randomly distributed behind edge routers and one receiver. We interconnect neighbor domains through OC-48 and OC-192 full-duplex links; we also connect each edge router to its hosts, through Fast (100 Mbps) or Thin (10 Mbps) Ethernet full-duplex links. End-to-end round-trip times average 200 milliseconds, while host-to-edge router round-trip times average 10 milliseconds.

For our simulation scenarios, we use the parameters of Table IV unless otherwise noted. We assume that preventing stamping is used to drop all spoofed traffic (§IV-C), while source gateways cooperate and block misbehaving sources.

B. Initial Overhead

1) Deaf Sources: We first simulate a 1-Gbps SYN-flooding attack from 100 000 deaf sources; the receiver identifies each source after receiving one second's worth of traffic from it (about 20 SYN packets). Fig. 3 shows the outcome: all undesired flows are blocked within 100 seconds, at which point the target has received about 9.5 Gbits of initial overhead (1 Gbit of identification overhead and 8.5 Gbits of request overhead). In this case, the initial overhead does not reach the upper bound of Eq. 1 for the two reasons stated in §V-C: first, the receiver identifies undesired flows and sends filtering requests in parallel; second, about 90 seconds after the start of the attack, there aren't enough undesired flows left unblocked to flood the receiver's tail circuit.

2) Lying Sources: Next, we simulate an attack with the same parameters, but coming from lying sources. Fig. 4 shows the outcome: each source sends three rounds of undesired traffic—one round in the beginning of the attack, a second round after the corresponding source gateway has removed its temporary filter for the first time, and a third round after the gateway has removed its temporary filter for the second time. Once the source has violated the filtering request against it twice, it is identified as lying and blocked by its gateway. Until all sources are blocked, the target receives about 29 bits of initial overhead (1 Gbit of identification overhead and 28 Gbits of request overhead). Note that each source sends three rounds of undesired traffic (rather than four, as dictated by Eq. 2); this is because the equation was derived assuming the worst-case scenario, where source gateways do not cooperate, whereas, in this case, the source gateways do cooperate and block lying sources once these are exposed.

Parameter	Value
Tail-circuit capacity	$C_{tc} = 100$ Mbps
Round-trip time across tail circuit	$RTT_{tc} = 10$ milliseconds
Outbound filtering request rate	$REQ_{out} = 1000$ requests/second
Filtering window	$W_f = 10$ minutes
Number of sources	$N_{uf} = 100\,000$ flows
Aggregate undesired-traffic rate	$R_{ut} = 1$ Gbps
Identification overhead	$B_{id} = 1$ Gbit
Identification time	$T_{id} = 1$ second

TABLE IV
DEFAULT SIMULATION PARAMETERS

C. Tail-circuit Capacity Loss

1) Non-coordinated On-off Sources: Our next scenario is an attack with the same parameters, but coming from on-off sources. These sources are "non-coordinated": they cooperate with filtering requests, and each one resumes sending undesired traffic as soon as the filtering request against it has expired. The outcome is shown in Fig. 5: during the first filtering window, the attack looks exactly the same with the one from deaf sources; after that, undesired flows reappear every filtering window and are re-blocked, wasting $\lambda = 0.00167$ of the target's tail-circuit capacity.

An interesting point is that the undesired traffic received during the first filtering window is more than the undesired traffic received during subsequent windows. The explanation is the following: In the beginning of the attack, undesired flows arrive at the receiver's tail circuit in bursts; the receiver's filtering-request rate is not enough to block each undesired flow as soon as it is identified—it takes 100 seconds to block all identified flows; in the meantime, the receiver incurs significant request overhead. After the first filtering window, however, sources resume as soon as the corresponding filtering requests have expired, which means that undesired flows reappear at the rate at which they were blocked; as a result, the receiver does have enough filtering-request quota to block each undesired flow as soon as it is identified; hence, it avoids the request overhead and incurs only the identification and (negligible) blocking overhead. In this way, the burstiness of the attack is diluted after the first filtering window.

2) Number of Sources and Burstiness: Given a certain aggregate undesired-traffic rate (R_{ut}) and identification overhead (B_{id}), the burstiness of an attack depends on how the undesired traffic is distributed among different flows: the higher the amount of per-flow traffic, the higher the burstiness of the attack. To demonstrate this, we simulate a flooding attack of the same rate and identification overhead as in the previous scenario, but involving fewer (10 000) flows, each one sending at a higher rate (100 Kbps). Fig. 6 shows the outcome: because there are less flows, it takes less time (10 seconds) to have all of them blocked; however, because each flow has a higher rate, the target's tail circuit is flooded in the beginning of every filtering window. So, relative to the previous scenario, the receiver incurs the same tail-circuit capacity loss (0.00167); however, all the overhead occurs within 10 seconds, making this a burstier attack.

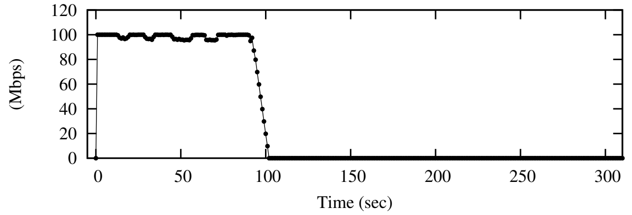


Fig. 3. Flooding attack from deaf sources. $N_{uf} = 100\,000$ flows; $R_{ut} = 1$ Gbps. The first $10\,000$ flows arrive at the tail circuit at $t = 0$; every time a flow is blocked, a new one takes its place, until all flows are blocked. The target identifies each flow after receiving it for 1 second, hence, it sends its first filtering requests at $t = 1$ second. It takes $T_{req} = \frac{N_{uf}}{REQ_{out}} = 100$ seconds to send filtering requests against all flows, hence, the attack is blocked at $t = 101$ seconds. The total amount of undesired traffic received is $B \approx 9.5$ Gbits, of which 1 Gbit is identification overhead and 8.5 Gbits are request overhead; the blocking overhead is negligible.

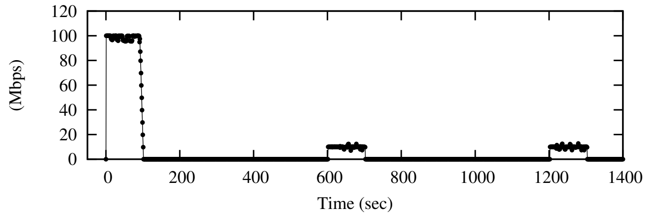


Fig. 5. Flooding attack from non-coordinated on-off sources. $N_{uf} = 100\,000$ flows; $R_{ut} = 1$ Gbps. As in Fig. 3, at time $t = 101$ seconds all undesired flows have been blocked for the first time. Each source resumes exactly $W_f = 10$ minutes after it is blocked; as a result, undesired flows reappear every $W_f = 10$ minutes at rate $REQ_{out} = 1000$ flows/second, and get blocked at that same rate. After the first filtering window, the receiver incurs only the 1-Gbit identification overhead every $W_f = 10$ minutes; the tail-circuit capacity loss is $\lambda = 0.00167$.

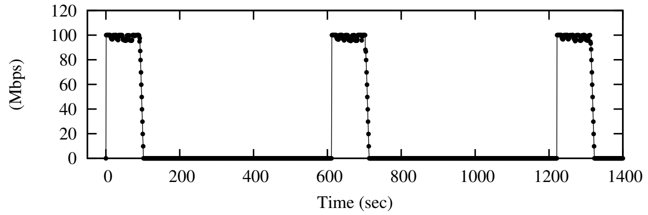


Fig. 7. Flooding attack from coordinated on-off sources. $N_{uf} = 100\,000$ flows; $R_{ut} = 1$ Gbps. As in Fig. 3 and 5, at time $t = 101$ seconds all undesired flows have been blocked for the first time. The first $10\,000$ sources resume as soon as the first $10\,000$ filtering requests have expired, which happens 10 seconds after the first filtering request expires, i.e., at $t = 611$ seconds; the rest of the sources resume as early as they are allowed. As a result, undesired flows recreate the initial attack pattern every 611 minutes. The tail-circuit capacity loss is $\lambda = 0.17$.

3) Coordinated On-off Sources: We simulate, once again, a 1-Gbps SYN-flooding attack from $100\,000$ on-off sources. These sources, however, are "coordinated": they cooperate with filtering requests, then resume their attack *in groups*, such that each group can send enough to flood the receiver's tail circuit. As a result, they periodically re-inflict the initial overhead (including the request overhead). Fig. 7 shows the outcome: the sources recreate the initial attack burst every 610 seconds; tail-circuit capacity loss is $\lambda = 0.17$.

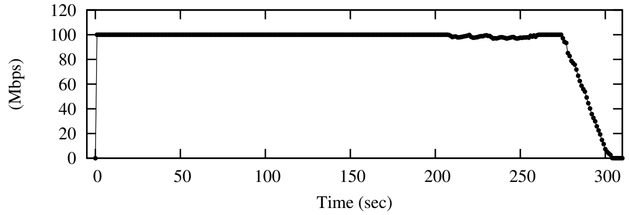


Fig. 4. Flooding attack from lying sources. $N_{uf} = 100\,000$ flows; $R_{ut} = 1$ Gbps. As in Fig. 3, at time $t = 101$ seconds each undesired flow has been blocked at least once. Each source resumes sending twice—until it is classified as lying and blocked. Hence, the receiver ends up sending $3N_{uf} = 300\,000$ filtering requests, which takes 300 seconds. The total amount of undesired traffic received is $B \approx 29$ Gbits—1 Gbit of identification overhead and 28 Gbits of request overhead.

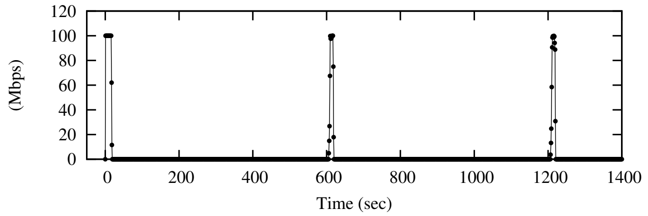


Fig. 6. Burstier flooding attack from non-coordinated on-off sources. $N_{uf} = 10\,000$ flows; $R_{ut} = 1$ Gbps. The target identifies each flow after receiving it for 10 seconds, hence, it sends its first filtering requests at $t = 11$ seconds. It takes $T_{req} = \frac{N_{uf}}{REQ_{out}} = 10$ seconds to send filtering requests against all flows, hence, the attack is blocked for the first time at $t = 21$ seconds. After the first filtering window, the receiver incurs the 1-Gbit identification overhead every $W_f = 10$ minutes. Compared to Fig. 5, the recurring overhead is the same ($\lambda = 0.00167$), but it is inflicted in 10 (rather than 100) seconds.

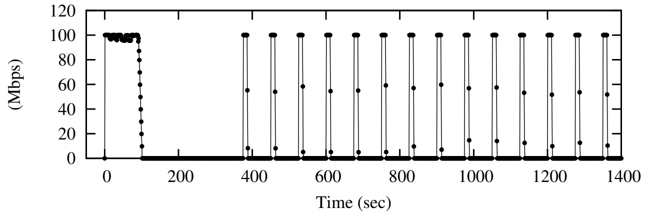


Fig. 8. Flooding attack from coordinated on-off sources with distributed filtering window. $N_{uf} = 100\,000$ flows; $R_{ut} = 1$ Gbps. The filtering window is uniformly distributed from 5 to 15 minutes. Sources resume in groups of $10\,000$, as soon as enough filtering requests have expired, i.e., every 75 seconds. It takes $T_{req} = \frac{N_{uf}}{REQ_{out}} = 10$ seconds to have $10\,000$ flows blocked, hence, each group periodically induces a 10-second spike. Compared to Fig. 7, the recurring overhead is the same ($\lambda = 0.17$), but it consists of 10-second (rather than 100-second) spikes.

4) Varying Filtering Window: An intuitive conclusion from the last simulation is that, if an attack can cause certain damage to the receiver's tail circuit once, it can inevitably do so again, as long as the participating sources wait long enough so they can attack with the same burstiness. This led us to the following observation: during an attack from coordinated on-off sources, what matters to the attacker is not how early each individual flow can resume, but how early each *group* of flows can resume; if the receiver associates a different window with each filtering request (so the average is W_f), then the participating sources must on average wait longer in order to achieve the same level of burstiness.

To demonstrate this, we simulate an attack with the same parameters as in the previous scenario, during which the receiver uses a uniformly distributed filtering window with an average of $W_f = 10$ minutes. Fig. 8 shows the outcome of the attack when sources wait until they can resume in groups of $10\,000$ (just enough to flood the target's tail circuit): not surprisingly, the tail-circuit capacity loss is the same as in the previous scenario ($\lambda = 0.17$), but the overhead is distributed in 10-second spikes every 75 seconds (rather than a 100-second outage every 610 seconds).

We do not address the issue of determining the optimal filtering-window distribution or the worst type of attack, but we believe that it depends on the particular type of service offered by the receiver. For example, a web server may handle the 10-second spikes better: each spike consumes more than 80% of the tail circuit roughly for two seconds; TCP retransmission may allow the short HTTP flows to recover from losses incurred during these two seconds. On the contrary, a server that expects its legitimate flows to exceed 75 seconds (e.g., a movie database), may handle the 100-second outages better: if it knows when to expect the next outage, it can refuse to serve any requests expected to complete during or after the outage and prompt its clients to retry a few minutes later.

VIII. Discussion

A. Request-channel Flooding

Any solution to bandwidth flooding that involves a "request channel" faces the challenge of becoming itself a bandwidth-flooding target. In our case, the request channel is the path from the receiver to the source gateway; we now discuss to what extent an attacker can flood this path and interfere with AITF operation.

1) Upstream-channel Flooding: So far, we have considered the scenario where an attacker floods receiver R's *downstream* connection, while R still controls its *upstream* connection and can send filtering requests. However, if the attacker controls a host located close to R (e.g., behind the same Ethernet hub), it can also flood R's upstream connection to its gateway, preventing R from sending filtering requests (or any traffic). This scenario is possible only if the attacker has access to R's upstream connection to its gateway—unlikely if R is a professional public-access site, but plausible if it is a server residing in a home or campus network. To handle such "insider attacks," AITF would have to be adapted to work at the intra-domain level: the router that controls the flooded bottleneck link from R to R_{gw} should be able to detect the flood and make the corresponding internal source stop.

2) Source-gateway Flooding: An attacker may try to flood the tail circuit to source gateway S_{gw}, in an effort to prevent it from receiving legitimate filtering requests, so that it gets classified as non-cooperating, potentially losing its communications. This attack corresponds to bandwidth-flooding an Internet border router, so it requires more resources from the attacker's side than flooding a simple receiver. In any case, AITF handles this like any other bandwidth-flooding attack: S_{gw} becomes the complaining receiver and sends (to its own gateway) filtering requests, which are eventually propagated to the undesired-traffic sources.

3) Receiver-gateway Flooding: Finally, an attacker may try to flood the tail circuit to the receiver's gateway R_{gw}, in an effort to prevent it from completing any three-way handshakes with source gateways. Indeed, if an attacker causes congestion on R_{gw}'s tail circuit, R_{gw} cannot operate correctly as a receiver's gateway. However, AITF is based on the principle that the *highest* upstream entity affected by the attack acts as the "receiver"—so, if R_{gw}'s tail circuit is flooded, R_{gw} acts as the "receiver" and sends filtering requests to its own "receiver's gateway" upstream.

B. Filtering Costs Versus Collateral Damage

Filtering undesired traffic per source and destination address in the long term (i.e., for the duration of the attack, which may last hours or days) is not a sustainable solution: as botnet sizes increase, each receiver may get undesired traffic from hundreds of thousands, even millions of sources; and as attacks become more sophisticated, each source may send undesired traffic to equally large numbers of receivers. Hence, whether the filtering is done at the receiver's or the source's network, we expect long-term selective filtering to become increasingly expensive—certainly beyond the capabilities of current networks with a few thousand filters per client.

AITF avoids long-term selective filtering by blocking traffic from non-cooperating entities (sources or networks) with *aggregate* filters that block multiple undesired flows at a time; the catch is that aggregate filters may affect the legitimate traffic generated by the non-cooperating entities. This approach may annoy users/administrators that will now lose part or all of their network connectivity until they clean up their compromised equipment; on the other hand, it provides a strong incentive to them to keep their equipment clean or else risk reduced network connectivity.

By default, AITF blocks all traffic from a non-cooperating source with a single aggregate filter (this policy minimizes the resources spent on misbehaving clients). However, each provider can define its own policy, e.g., it can agree to filter up to N aggregates from each non-cooperating client. A harsh policy (a small N) is likely to dissatisfy the owners of compromised machines; a lenient policy (a large N) is more client-friendly, but also more expensive, as it commits multiple filters to each non-cooperating client; finally, an "indifferent" policy (i.e., ignoring filtering requests) is both client-friendly and inexpensive, yet it bears the risk of losing connectivity to the complaining receivers and dissatisfying the legitimate clients that were communicating with them. Similar trade-offs are involved when a receiver chooses a policy toward non-cooperating source gateways.

We do not explore these trade-offs as part of this work, but we believe that they should be resolved separately for each receiver and/or provider, taking into account the type of their business, potentially differentiating between more and less important (business-wise) clients and/or domains.

IX. Related Work

Bandwidth flooding belongs to the wider topic of denial of service (DoS), which covers source-address spoofing, attack detection, undesired-traffic identification, and application-level attacks that target server resources like memory or CPU; even though all that work is related to ours (in the sense that they complement each other), we do not discuss it here, in favor of a deeper comparison of AITF to more closely related work.

Overlay-based solutions protect a receiver from flooding by restricting its communications to a set of authorized sources; to prevent the authentication process from becoming itself a DoS target, it is outsourced to a set of access points connected to the receiver through an overlay network [11], [20], [21]. In contrast, AITF addresses the more general problem of protecting public-access sites—receivers that do not know in advance which sources they want to receive traffic from, while sources can become compromised at any point in time and start sending undesired traffic. Moreover, AITF does not require protected receivers to trust any entities other than the routers that are already on the path of their communications.

Pushback enables a receiver to identify the last-hop routers that forward to it high-rate traffic and ask them to rate-limit all traffic addressed to it; this process is repeated recursively at each router, so that rate-limiting of undesired-traffic is eventually pushed closer to its sources [22]. Rate limiting prevents congestion on the target's tail circuit (and, hence, protects traffic addressed to other receivers sharing the same tail circuit), but does not protect the target's legitimate traffic during distributed attacks: when the rate limiting happens at interfaces receiving a mix of legitimate and undesired traffic addressed to the target (as is expected to happen during distributed attacks), legitimate traffic still ends up competing with undesired traffic for the target's tail-circuit capacity, even if this "competition" is pushed outside the target's tail circuit—and we have already mentioned that legitimate TCP flows fare poorly in such situations. To our knowledge, Pushback offers the best result that can be achieved without assuming any anti-spoofing or undesired-traffic identification mechanisms.

Similar to AITF, the Points of Control approach (developed concurrently) selectively blocks undesired traffic before the receiver's tail circuit, at ISP boundaries [8]; its fundamental difference from AITF is that traffic is blocked in the long term by wire-speed filters (the issue of an ISP not having enough filters is not addressed). In general, the Points of Control proposal focuses more on the issues of providing a separate address space for publicly addressable servers and performing wire-speed encapsulation, whereas this paper focused on a filtering protocol with well characterized scalability properties.

More recent, clean-slate proposals suggest that receivers directly contact undesired-traffic sources and ask them to stop; they rely on sources being enhanced with uncompromisable functionality (e.g., running on NIC firmware) that verifies and satisfies such requests [23], [24]. The Accountable Internet Protocol, moreover, equips all packets with self-certifying, unspoofable addresses [24], which enables elegant bandwidth-flooding solutions—no need for extra measures against source-address, path, or filtering-request spoofing. This reduced complexity, however, relies on clean-slate elements (a new Internet protocol, new NICs), whereas our proposal aims for incremental deployment on top of the current Internet.

Network capabilities enable a receiver to deny by default all traffic and explicitly accept traffic from identified legitimate sources [25]. The key feature of capability-based filtering, introduced in the SIFF proposal [26], is that it is *stateless* and, hence, obviates the need for wire-speed filters in routers and inter-ISP filtering agreements (because no filtering state is explicitly exchanged between ISPs). On the other hand, capability-based solutions face a significant challenge: to protect the capability-setup channel itself from flooding [27]. This challenge brings to mind public-key infrastructures, where the greatest deployment issue has proved to be not the encryption of the data, but the management and distribution of the keys.

One proposal for protecting the capability-setup channel is to fair-queue capability requests per incoming router interface [28]. This approach faces a similar challenge with Pushback: when the fair queuing happens at interfaces receiving a mix of legitimate and undesired capability requests, legitimate requests end up competing with (and losing to) undesired ones, even if this "competition" is pushed away from the target's tail circuit [27]. Another proposal is to combine capabilities with stateful filtering, i.e., explicitly block capability requests from specific sources [29]; in contrast, AITF was designed to introduce as few new mechanisms as possible—if capabilities alone are not enough and we have to use stateful filtering anyway, why not design a protocol that uses *only* stateful filtering? A third proposal is Portcullis, where capability distribution is regulated through special "puzzles," distributed over the Domain Name Service (DNS). This approach relies on the assumption that the DNS infrastructure is itself protected from flooding through over-provisioning and/or other dedicated infrastructure [30]; in contrast, AITF consists of a single mechanism, suitable for protecting any bandwidth-flooding target—including the DNS infrastructure.

At a higher level, network capabilities take the "connection-oriented" approach, where the network only allows (or gives priority to) traffic that belongs to explicitly established connections. This approach has been showed to work in the context of a single administrative domain (e.g., an enterprise or campus network), where connection authorization can be performed centrally, based on predefined access policies [31]. However, in the Internet context, where receivers from one domain are expected to authorize sources from another, an important missing piece in evaluating the connection-oriented approach is answering the following fundamental question: what is a reasonable number of bytes and a reasonable amount of time to allow an unknown source, which could become compromised at any point in time? While it is worth investigating this question, it is also worth considering the alternative, "datagram" approach, where a receiver explicitly denies undesired traffic, while accepting, by default, all other traffic; to the best of our knowledge, the work we presented here is the first that proposes a datagram-based filtering solution that requires a credible, bounded amount of resources from participating ISPs and addresses the security issues that arise from filter propagation across different administrative domains.

X. Conclusions

We have presented Active Internet Traffic Filtering, a network-layer filtering mechanism that preserves a significant fraction of a receiver's tail circuit in the face of bandwidth flooding, while requiring a reasonable amount of resources from participating ISPs.

We have showed that: (1) AITF allows a receiver to preserve on average 80% of its tail circuit in the face of a SYN-flooding attack that has ten times the rate of its capacity. (2) Each participating ISP needs a few thousand filters and a few megabytes of DRAM per client; the per-client cost is not expected to increase, unless botnet-size growth outpaces Moore's law. (3) The first two AITF-enabled networks can maintain their communication in the face of flooding attacks, as long as the path between them is not compromised.

The feasibility of AITF shows that the network-layer of the Internet can provide an effective, scalable, and incrementally deployable solution to bandwidth-flooding attacks.

References

[1] A. Kuzmanovic and E. Knightly, "Low-rate Targeted TCP Denial-of-service Attacks (The Shrew vs. The Mice and Elephants)," in *Proceedings of the ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003.

[2] S. Gibson, "The Strange Tale of the Denial of Service Against GRC.com," http://www.grc.com/dos/grcdos.htm.

[3] S. Berinato, "Online Extortion," http://www.csoonline.com/read/050105/extortion.html.

[4] B. Agrawal and T. Sherwood, "Modeling TCAM Power for Next Generation Network Devices," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, USA, March 2006.

[5] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in Your Spare Time," in *Proceedings of the USENIX Security Symposium*, San Francisco, CA, USA, August 2002.

[6] X. Yang, "NIRA: A New Internet Routing Architecture," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, Karlsruhe, Germany, August 2003.

[7] K. Argyraki and D. R. Cheriton, "Loose Source Routing as a Mechanism for Traffic Policies," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, Portland, OR, USA, August 2004.

[8] A. Greenhalgh, M. Handley, and F. Huici, "Using Routing and Tunneling to Combat DDoS Attacks," in *Proceedings of the USENIX Workshop on Steps Towards Reducing Unwanted Traffic in the Internet (SRUTI)*, Cambridge, MA, USA, July 2005.

[9] "Mazu Enforcer," http://www.mazunetworks.com/products/mazu-enforcer.php.

[10] "Peakflow X Data Sheet," http://arbornetworks.com/downloads/Arbor_Peakflow_X_Data_Sheet.pdf.

[11] A. Stavrou and A. Keromytis, "Countering DoS Attacks With Stateless Multipath Overlays," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, USA, November 2005.

[12] D. Bernstein, "SYN Cookies," http://cr.yp.to/syncookies.html.

[13] K. Argyraki, "Scalable Defense Against Internet Bandwidth-Flooding Attacks," Ph.D. dissertation, Stanford University, December 2006.

[14] R. Beverly and S. Bauer, "The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet," in *Proceedings of the USENIX Workshop on Steps Towards Reducing Unwanted Traffic in the Internet (SRUTI)*, Cambridge, MA, USA, July 2005.

[15] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Botz4Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds," in *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, May 2005.

[16] A. Markopoulou, F. Tobagi, and M. Karam, "Loss and Delay Measurements of Internet Backbones," *Elsevier Computer Communications (Special Issue on Measurements and Monitoring of IP Networks)*, vol. 29, pp. 1590–1604, June 2006.

[17] Z. Chen, C. Ji, and P. Barford, "Spatial-Temporal Characteristics of Internet Malicious Sources," in *Proceedings of the IEEE INFOCOM Mini-conference*, Phoenix, AZ, USA, April 2008.

[18] "University of Oregon Route Views Archive Project David Meyer," http://archive.routeviews.org/oix-route-views/.

[19] L. Gao, "On Inferring Autonomous System Relationships in the Internet," in *Proceedings of the Global Internet Symposium*, San Francisco, CA, USA, November 2000.

[20] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *Proceedings of the ACM SIGCOMM Conference*, Pittsburgh, PA, USA, August 2002.

[21] D. G. Andersen, "Mayday: Distributed Filtering for Internet Services," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, WA, USA, March 2003.

[22] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Schenker, "Controlling High Bandwidth Aggregates in the Network," *ACM Computer Communications Review*, vol. 32, no. 3, pp. 62–73, July 2002.

[23] M. Shaw, "Leveraging Good Intentions to Reduce Unwanted Internet Traffic," in *Proceedings of the USENIX Workshop on Steps Towards Reducing Unwanted Traffic in the Internet (SRUTI)*, San Jose, CA, USA, July 2006.

[24] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Holding the Internet Accountable," in *Proceedings of the ACM Workshop on Hot Topics in Networking (HotNets)*, Atlanta, GA, USA, November 2007.

[25] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet Denial-of-Service with Capabilities," in *Proceedings of the ACM Workshop on Hot Topics in Networking (HotNets)*, Cambridge, MA, USA, November 2003.

[26] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2004.

[27] K. Argyraki and D. R. Cheriton, "Network Capabilities: The Good, the Bad, and the Ugly," in *Proceedings of the ACM Workshop on Hot Topics in Networking (HotNets)*, College Park, MD, USA, November 2005.

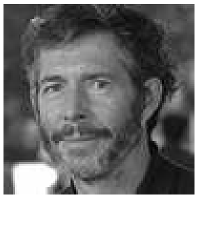
[28] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting Architecture," in *Proceedings of the ACM SIGCOMM Conference*, Philadelphia, PA, USA, August 2005.

[29] M. Casado, A. Akella, P. Cao, N. Provos, and S. Shenker, "Cookies Along Trust-Boundaries (CAT): Accurate and Deployable Flood Protection," in *Proceedings of the USENIX Workshop on Steps Towards Reducing Unwanted Traffic in the Internet (SRUTI)*, San Jose, CA, USA, August 2006.

[30] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks," in *Proceedings of the ACM SIGCOMM Conference*, Kyoto, Japan, August 2007.

[31] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the ACM SIGCOMM Conference*, Kyoto, Japan, August 2007.

Katerina Argyraki is a researcher with the Operating Systems group in the School of Computer and Communication Sciences, EPFL, Switzerland. She works on network architectures and protocols with a focus on denial-of-service defenses and accountability. She received her undergraduate degree in Electrical and Computer Engineering from the Aristotle University, Thessaloniki, Greece, in 1999, and her Ph.D. in Electrical Engineering from Stanford University, in 2007.



David R. Cheriton has been a Professor of Computer Science and Electrical Engineering at Stanford University since 1981. His research spans the areas of distributed systems, object-oriented software structuring, Internet architecture and protocols, and hardware-software interaction, particularly at the operating-system level. He was a co-founder of Granite Systems (acquired by Cisco), Kealia (acquired by Sun), and, most recently, Arastra, and a technical advisor with Google, VMware, Cisco, Sun, and a number of startup companies. He received his Ph.D. in Computer Science from the University of Waterloo in 1978.