Active Internet Traffic Filtering:
Real-Time Response to Denial-of-Service Attacks

Katerina Argyraki David R. Cheriton
Distributed Systems Group
Stanford University
{*argyraki, cheriton*}*@dsg.stanford.edu*

Abstract

This paper describes *Active Internet Traffic Filtering* (AITF), a mechanism for blocking highly distributed denial-of-service (DDoS) attacks. These attacks are an acute contemporary problem, with few practical solutions available today; we describe in this paper the reasons why no effective DDoS filtering mechanism has been deployed yet. We show that the current Internet's routers have sufficient filtering resources to thwart such attacks, with the condition that attack traffic be blocked close to its sources; AITF leverages this observation. Our results demonstrate that AITF can block a million-flow attack within seconds, while it requires only tens of thousands of wire-speed filters per participating router — an amount easily accommodated by today's routers. AITF can be deployed incrementally and yields benefits even to the very first adopters.

1 Introduction

We have recently witnessed a dramatic increase in the frequency and ferocity of distributed denial-of-service (DDoS) attacks. In December 2003, an attack kept SCO's web site practically unreachable for more than a day [6]; in June 2004, another attack flooded Akamai's name servers, disrupting access to its clients for 2 hours, including the Google and Yahoo search engines [7]; a month later, an attack flooded DoubleClick's name servers, disabling ad distribution to its 900 clients for 3 hours [8]. Considering that network downtime costs hundreds of thousands of dollars per hour [19], such incidents can translate into millions of dollars of lost revenue for the victim. Yet, the DDoS problem remains unsolved.

We recognize three (not all of them orthogonal) problems that render DDoS traffic hard to filter:

Source address spoofing: An attack source often uses multiple fake source IP addresses to send its traffic. As a result, the victim can neither identify the attack source nor specify a filtering rule (e.g., "block all traffic with source IP address S") that selectively blocks its traffic.

Large number of attack sources: Each hardware router has only a limited number of filters that can block traffic without degrading the router's performance (i.e., filters operating at wire speed). The limitation comes from cost and space. Wire-speed filters are typically stored in expensive TCAM (Ternary Content Addressable Memory), which they share with the router's forwarding table. Some of the largest TCAM chips available today accommodate 256K entries [4]. A sophisticated router linecard fits at most 1 TCAM chip [1], i.e., tens of thousands of filters per network interface. On the other hand, a large-scale attack can involve millions of attack sources [22]. So, even if source address spoofing were completely eliminated, i.e., even if the victim could identify each attack source and specify a filtering rule for it, the victim's firewall would not have enough filters to accommodate all the rules.

The straightforward solution to such a resource problem is aggregation: Don't install a separate filter for each attack source; instead, identify the IP prefixes that cover most attack sources and block all traffic from these prefixes. Unfortunately, filter aggregation does not work in most DDoS scenarios: Attack sources are typically worm-infected populations, highly distributed across the Internet. As a result, blocking the prefixes that correspond to the attack sources results in blocking most Internet prefixes, thereby causing severe collateral damage.

Pushing filtering into the Internet core does not scale: If the victim's firewall cannot block attack traffic by itself, the straightforward solution is to push filtering of attack traffic back into the Internet core: Identify the upstream peering networks that forward attack traffic and send them appropriate filtering requests. Unfortunately, this approach does not scale, because it introduces end-to-end filtering state into core routers. Consider a core router receiving filtering requests from 10 victims; each victim is under attack by a million sources. The core router can either block traffic from each attack source to each victim, or aggregate filtering rules and rate-limit all traffic going to the victims. The former requires 10 million filters; the latter requires only 10 filters, but sacrifices most good traffic going to the victims.

Yet, there *are* enough filtering resources in the Internet to block such large-scale attacks. An attack coming from thousands of different networks involves thousands of routers; assuming each router contributes a few thousand filters, there are millions of filters available to block attack traffic. And the closer we get to the attack sources, the larger the amount of filtering resources available per attack source — it is the victim's firewall and the Internet core that are the "filtering bottleneck". Unfortunately, today, the victim has no access to these resources, since there is no way for a DDoS victim to identify routers located close to the attack sources and make them block attack traffic.

In this paper, we present a DDoS filtering mechanism that overcomes these problems. Our source address spoofing solution is a hardware-friendly variant of the IP route record (RR) technique [20]. Although different from traditional packet marking techniques [21, 14] (which do not provide an explicit recorded route in each packet), our RR approach is not a radical departure from them, either. Our main contribution is Active Internet Traffic Filtering (AITF), a protocol that leverages recorded route information to block attack traffic.

An AITF-enabled receiver uses the routes recorded on incoming packets to identify the last point of trust on each attack path and causes attack traffic to be blocked at that point, i.e., as close as possible to its sources. We provide a way to do this securely — AITF prevents abuse by malicious nodes seeking to disrupt other nodes' communications. We show that our approach can selectively block a million attack sources, yet requires only tens of thousands of TCAM memory entries and a few megabytes of DRAM memory from each participating router; these numbers correspond to the specifications of real products [4, 1]. We also provide an incremental deployment scenario, in which even early adopters receive a concrete benefit; this benefit is compounded by further deployment.

The rest of the paper is organized as follows: Section 2 describes route record and how a receiver can use it to identify distinct traffic flows. Section 3 describes the AITF protocol in detail, naively assuming that no source address spoofing occurs. We remove this assumption in Section 4, where we describe how AITF deals with spoofing attacks. We estimate AITF performance in Section 5 and verify our estimates through simulation in Section 6. Section 7 analyzes deployment issues, Section 8 discusses additional attacks and potential defenses, and Section 9 presents related work. Section 10 concludes the paper.

2 Limiting spoofing

2.1 Route Record

A router that participates in a route record (RR) scheme writes its IP address on each packet it forwards. In our approach, only border routers participate in RR. As a result, each packet carries the identities of a sub-list of the border routers that forwarded it. For example, in Figure 1, border routers A_{gw}, X, Y, and V_{gw} are RR-enabled; each packet sent by host A to host V carries recorded route $\{A_{gw}\ X\ Y\ V_{gw}\}$ upon reaching its destination.

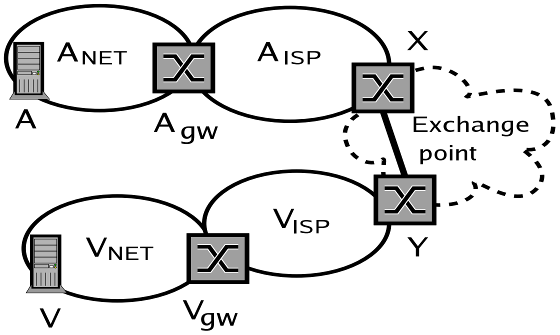

Figure 1: Packets sent by host A to host V carry recorded route $\{A_{gw}\ X\ Y\ V_{gw}\}$.

We implement RR functionality as a "shim" protocol between the IP and transport layers, i.e, the RR header is introduced as the beginning of the IP payload. We chose not to use the traditional IP RR option, because of its performance overhead — traditional IP RR packets are typically handled by routers off the fast path. The details of how each participating router adds its address to the RR header are described in the Appendix, Section A.1.

When a packet crosses an RR-enabled, "non-malicious" Internet area adjacent to its destination domain, its recorded route includes an authentic (non-spoofed) suffix. Specifically, the last n components of the recorded route are authentic, when the last n border routers crossed by the packet are RR-enabled, and there is no malicious node on the path that interconnects them. As we explain next, this enables a receiver to identify distinct incoming traffic flows in the face of source address spoofing.

2.2 Identifying Distinct Flows

We define the *recorded path* of a packet as the sequence of IP addresses that correspond to: the packet's source, the list of border routers specified in the RR header, and the packet's destination. We define a *flow* as the set of all packets that share a common recorded path suffix. For example, in Figure 1, all packets with path $\{A\ A_{gw}\ X\ Y\ V_{gw}\ V\}$ constitute a distinct flow; all packets with path $\{*\ A_{gw}\ X\ Y\ V_{gw}\ V\}$ also constitute a distinct (aggregate) flow. We use $F\{P\}$ to refer to flow F with recorded path P.

A DDoS victim feeds recorded paths into a local policy module, which classifies incoming traffic in distinct flows, decides which ones are undesired and forms filtering requests against them. The operation of the policy module depends on the specific service run by the victim and is outside the scope of this paper. We call it a "policy module", because it determines a "defense policy", i.e, which flows must be blocked. AITF (described in the next section) is the mechanism that enforces the chosen policy.

It is up to the policy module to classify incoming traffic in multiple "flow levels", in order to identify undesired flows in the face of source address and path spoofing. For example, consider that in Figure 1 attack source A is sending high-rate traffic to victim V. If network A_{NET} prevents source address spoofing, V can easily identify $F_1\{A\ A_{gw}\ X\ Y\ V_{gw}\ V\}$ as a high-rate flow and, thus, undesired. If A is able to spoof multiple source IP addresses, V can only identify $F_2\{*\ A_{gw}\ X\ Y\ V_{gw}\ V\}$ as the undesired flow.

Once the policy module identifies an undesired flow, it sends a filtering request to the local AITF process. AITF does not assume that the recorded path of an undesired flow coincides with its real path. I.e., if the policy module identifies $F\{*\ A_{gw}\ X\ Y\ V_{gw}\ V\}$ as an undesired flow, this only means that the victim does not want to receive any more packets with this recorded path; it does not necessarily mean that these packets are indeed forwarded by A_{gw}, X, and Y .

3 Basic AITF Protocol

We start with an overview of the protocol (Section 3.1) and terminology (Section 3.2); we describe our algorithm incrementally, through Sections 3.3, 3.4, 3.5, and 3.6; we discuss appropriate values for its parameters in Section 3.7. To simplify description, we naively assume that no source address/path spoofing occurs — we remove this assumption in the next section.

3.1 Overview

Upon identifying an undesired flow, the victim sends a filtering request to its gateway (V_{gw} in Figure 1). The victim's gateway temporarily blocks the undesired flow and identifies the border router located closest to the attack source(s) — call it the *attack gateway* (A_{gw} in Figure 1). Then, the victim's gateway initiates a "counter-connection" setup with the attack gateway, i.e., an agreement not to transmit certain packets — the opposite of a TCP connection setup, which is an agreement to exchange packets. As soon as the counter-connection setup is completed, the victim's gateway can remove its temporary filter. If the attack gateway does not cooperate, the victim's gateway can *escalate* the filtering request to the next border router closest to the attack gateway (X in Figure 1). Escalation can continue recursively until a router along the attack path responds and a counter-connection setup is completed. If no router responds, attack traffic is blocked locally by the victim's gateway. However, as we will see, AITF both assists and motivates routers close to the attack source(s) to help block attack traffic.

3.2 Terminology

The recorded path P of an undesired flow has form $\{A\ A_{gw}\ ...\ V_{gw}\ V\}$, where

- A is the "attack source", i.e., the node thought to be generating the undesired traffic; if $A = *$, all traffic through A_{gw} is undesired.
- A_{gw} is the "attack gateway", i.e., the border router thought to be closest to A.
- V_{gw} is the "victim's gateway", i.e., the border router closest to the victim.
- V is the victim.

We assume that the only node affected by the attack is V; e.g., if this is a flooding attack, the only part of the network that is congested is the tail-circuit from V_{gw} to V. If V_{gw} were also affected, it itself would be the "victim", and its closest upstream border router would be the "victim's gateway".

3.3 Blocking Close to the Attack Source

As shown in Figure 2, AITF involves 4 entities:

1. The victim V sends a filtering request to V_{gw}, specifying an undesired flow F.
2. The victim's gateway V_{gw}:
   (a) Installs a temporary filter to block F for T_{tmp} seconds.
   (b) Initiates a *3-way handshake* with A_{gw}.
   (c) Removes its temporary filter, upon completion of the handshake.
3. The attack gateway A_{gw}:
   (a) Responds to the 3-way handshake.
   (b) Installs a temporary filter to block F for T_{tmp} seconds, upon completion of the handshake.
   (c) Sends a filtering request to the attack source A, to stop F for $T_{long} \gg T_{tmp}$ minutes.

(d) Removes its temporary filter, if A complies within T_{tmp} seconds; otherwise, it disconnects A.

4. The attack source A stops F for T_{long} minutes or risks disconnection.

All filtering requests are rate limited. I.e., the victim's gateway accepts a limited rate of requests from each alleged victim. Similarly, the attack gateway (i) accepts a limited rate of requests (handshake initializations) from each alleged victim gateway and (ii) sends a limited rate of requests to each alleged attack source.

The reason for the temporary filter on the victim's gateway is to immediately protect the victim until the attack gateway takes responsibility. The reason for directly contacting the attack gateway is to avoid creating a filtering bottleneck in the Internet core. Finally, the reason for the 3-way handshake is to enable the attack gateway to verify that the requester of the filter is indeed on the path to the alleged victim; the handshake is further explained next.

3.4 Securing Edge-to-edge Communication

The 3-way handshake is depicted in Figure 2: V_{gw} sends to A_{gw} a request to block F; A_{gw} sends to V a message that includes F and a nonce; V_{gw} intercepts the message and sends it back to A_{gw}.

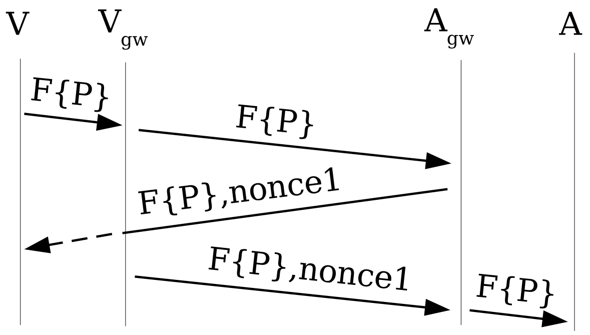

Figure 2: AITF entities and message exchange.

V_{gw} proves its location on the path to V by intercepting the nonce sent to V. This prevents malicious node M, located off the path from A_{gw} to V_{gw}, from causing a filter to be installed at A_{gw} and block traffic to V. By picking a sufficiently large and properly random value for the nonce, it can be made arbitrarily difficult for M to guess it (see Section 5.4).

To avoid buffering state on incomplete 3-way handshakes, A_{gw} computes the nonce as follows:

$$nonce1 = hash_{key}(F)$$

where key is a local key and $hash$ is a keyed hash function. To verify the authenticity of a completion message, A_{gw} just hashes the flow included in the message and compares the result to the nonce included in the message (similar to the TCP SYN-cookie technique [13]).

3.5 Identifying Liars

With what we have described so far, there are two entities that can lie: (i) An attack source can pause an undesired flow (to avoid disconnection) and resume as soon as the attack gateway has removed its temporary filter. (ii) An attack gateway can pause an undesired flow and resume as soon as the victim's gateway has removed its temporary filter. To catch such liars, we introduce the *shadow filtering table*.

Every time a gateway removes a temporary filter from its TCAM, it creates a copy in DRAM that expires after T_{long}. This "shadow filter" helps check whether the corresponding undesired flow is released prematurely (before T_{long}) by its source. For example, suppose attack gateway A_{gw} has already told attack source A to block F; now suppose A_{gw} receives a new filtering request against F and installs a new temporary filter; if the new filter catches F traffic, A_{gw} checks its shadow filtering table, finds out that a shadow filter for F already exists (i.e., A has already been told to stop once) and disconnects A.

The victim's gateway uses the same technique to check whether the attack gateway keeps the undesired flow blocked for T_{long} minutes. The only difference is that the attack gateway has to be caught violating the filtering agreement twice to be classified as "lying" — the first time could be due to a lying attack source, so the attack gateway is given the benefit of the doubt once.

3.6 Dealing with Non-Cooperative Gateways

An attack gateway can deal with a non-cooperative attack source by disconnecting it. This is possible because the attack gateway is the border router providing Internet connectivity to the attack source. The victim's gateway can obviously not deal with a non-cooperative attack gateway the same way, since they belong to separate (not even peering) administrative domains. To address this, we introduce *escalation*.

An attack gateway is classified as "non-cooperative", if it does not respond to the handshake or responds, but is caught violating the filtering agreement twice. In that case, the victim's gateway can "escalate" the filtering request to the border router that follows the non-cooperative attack gateway on the flow's path. The new attack gateway is requested to block all traffic from the last non-cooperative attack gateway to the victim. For example, in Figure 1, V_{gw} first contacts A_{gw} asking it to block all traffic from A to V. If A_{gw} does not cooperate,

V_{gw} can contact X asking it to block all traffic from A_{gw} to V. Simply said, an attack gateway either cooperates and blocks traffic from its misbehaving client(s) to the victim, or risks losing its connectivity to the victim overall. This is a strong incentive to cooperate, especially when the victim is a popular public-access site like eBay or Amazon.

There are cases in which escalation is good, and cases in which it is bad. E.g., if eBay is losing most of its good traffic to a severe flooding attack, it makes sense to block access from non-cooperative attack gateways in order to preserve connectivity to the rest of the world. On the other hand, if eBay wants to get rid of a couple of annoying AOL clients, it does not make sense to block the entire AOL, even if its gateway does not cooperate. Deciding whether escalation is a good or a bad idea is the responsibility of the policy module. The policy module communicates its decision by declaring (or not) an undesired flow "escalatable".

The victim's gateway escalates a filtering request if and only if (i) the corresponding flow is escalatable and (ii) local filter utilization has exceeded a pre-configured threshold. Otherwise, the undesired flow is blocked locally for T_{long} minutes.

A smart attack source can introduce multiple fake components in the beginning of the RR header and make the victim's gateway escalate multiple times (as many as the fake components), before it actually contacts an authentic border router. To avoid such abuse, a DDoS victim can simply ignore the first components of "suspiciously" long paths when classifying flows. For example, 95% of Internet domains are no more than 6 hops apart [17]. So, a DDoS victim can consider only the last 6 components of the RR header when classifying flows. This limits the number of unsuccessful escalation attempts to $6 - n$, where n is the number of RR/AITF-enabled, cooperative border routers on the attack path. However, this also sacrifices the traffic of good sources collocated with bad sources, in networks that are more than 6 domains away from the victim. The policy module must decide whether keeping this traffic is worth the unsuccessful escalation attempts.

3.7 Filter Timeout Values

The goal of a temporary filter on the victim's gateway is to block an undesired flow until the corresponding handshake is complete. Considering that Internet round-trip times range from 50 to 200 msec, a "safe" value is $T_{tmp} = 1$ sec.

The choice of the long-term filter timeout T_{long} involves the following trade-off: An attack source A is typically an "innocent" end-host compromised through a worm. A large T_{long} of, say, 30 minutes guarantees that the victim V will not receive any undesired traffic from the corresponding attack source A for at least 30 minutes; on the other hand, it also guarantees that V will not receive any traffic at all from A, even if A is appropriately patched before T_{long} expires. Section 5 explains how exactly the value for T_{long} affects AITF performance.

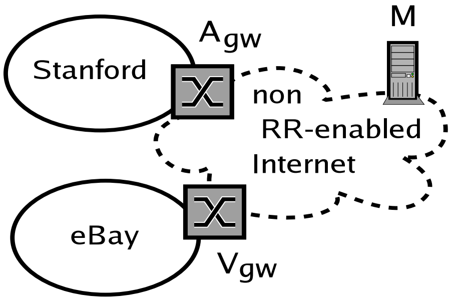

Figure 3: Malicious node M pretends to be at Stanford and sends undesired traffic to eBay; its packets carry (spoofed) recorded path $\{* \; A_{gw} \; V_{gw} \; eBay\}$.

4 Adding Resistance to Spoofing

As presented thus far, AITF is effective only if deployed in a significant portion of the Internet, adjacent to the victim's network. Otherwise, a malicious node can use path spoofing and abuse AITF to disrupt other nodes' communications. We illustrate with an example.

Figure 3 illustrates an early deployment stage, where only eBay and Stanford have deployed RR and AITF. Traffic sent by Stanford hosts to eBay has path $\{* \; A_{gw} \; V_{gw} \; eBay\}$. Malicious node M spoofs this path and sends to eBay high-rate traffic that appears to be coming from Stanford. V_{gw} requests from A_{gw} to stop; although A_{gw} agrees, V_{gw} continues to see high-rate traffic from Stanford; it falsely concludes that A_{gw} is non-cooperative and blocks all traffic from Stanford to eBay.

To prevent this abuse, we augment the recorded path with randomized components. Now each RR-enabled border router that forwards a packet writes on the packet (i) its IP address and (ii) a random value that depends on the packet's destination. By picking a sufficiently large and properly random value, it can be made arbitrarily hard for malicious off-the-path nodes to guess it (see Section 5.4). For example, in Figure 3, traffic sent by Stanford to eBay, has recorded path $\{* \; A_{gw}{:}R_1 \; V_{gw}{:} R_2 \; eBay\}$, where R_1 and R_2 are random values inserted by A_{gw} and V_{gw} respectively. To avoid keeping per destination state, each router computes the random value to insert in each packet as follows:

$$R = hash_{key}(D)$$

where key is a local key, $hash$ is a keyed hash function and D is the packet's destination.

When A_{gw} receives a request to block undesired flow $F\{P\}$, it checks whether it indeed forwarded F, i.e., whether P includes the correct random value. If yes, A_{gw} commits to filter F by responding to the handshake as described in Section 3.4. Otherwise, A_{gw} responds with the "authentic" path P', i.e., the path that includes the correct random value. So, if the victim's gateway sends a filtering request with a spoofed path, the handshake consists of just two messages, depicted in Figure 4.

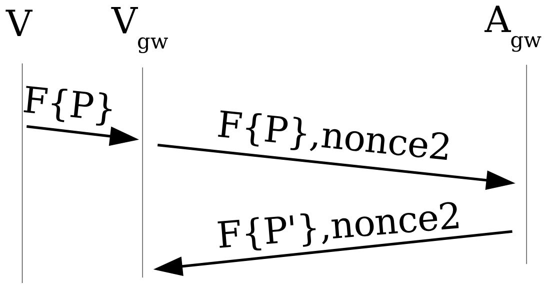

Figure 4: V_{gw} sends a filtering request against spoofed flow $F\{P\}$ that appears to be coming from A_{gw}; A_{gw} responds with the authentic path P', which includes the correct random value inserted by A_{gw} in all packets addressed to V. We added one more nonce, to enable V_{gw} to verify that the response with the authentic path is indeed coming from A_{gw}.

The victim's gateway uses the authentic path P' to recognize spoofed traffic that claims to be coming from A_{gw} and block it. For example, in Figure 3, V_{gw} blocks everything that appears to be forwarded by A_{gw} but does not include A_{gw}'s correct random value R_1, i.e., V_{gw} blocks traffic with path $\{*\ A_{gw}{:}!R_1\ V_{gw}\ eBay\}$. Note that in the specific example, V_{gw} can only block this traffic locally — escalation is not an option, since no other domain has deployed AITF.

We defer answering the following questions to later sections: (i) Exactly how large must each random component be? (ii) A router computes its random values based on a local key; how often does this key expire? (iii) When the local key changes, all random values communicated to victim gateways as part of the handshake become invalid; does the router notify the corresponding victim gateways?

5 Evaluation

In this section, we use back-of-the-envelope calculations to show that (i) a victim can have an undesired flow blocked within milliseconds; (ii) the victim's gateway can block a certain number of undesired flows with two orders of magnitude fewer filters than flows and a reasonable amount of DRAM; and (iii) the probability of abusing AITF can be made arbitrarily low.

5.1 Filtering Response Time

We define *filtering response time* T_{fr} as the time that elapses from the moment the victim sends a filtering request against an undesired flow, until the victim stops receiving the flow. With AITF, this is equal to the one-way delay from the victim to the victim's gateway plus the negligible overhead of processing the filtering request and installing the corresponding temporary filter. I.e., filtering response time is a few milliseconds.

If there are compromised routers on the flow's path, they can agree to filter the flow and later break the agreement — recall that each attack gateway is given two chances to cooperate. This results in the victim receiving "spikes" of the undesired flow after T_{fr}. Spikes are spaced out by at least T_{tmp} seconds; the number of spikes is bounded from above by $2 \times n$, where n is the number of compromised routers on the flow's path. The effect of these spikes on the victim is insignificant, as we demonstrate with later simulation results.

5.2 Filtering Rate, Capacity and Gain

In this section, we examine three basic AITF properties: how much attack traffic it blocks, how fast it does so, and at what cost. To quantify these properties, we use three simple metrics: the *filtering rate* of a router is the number of flows that the router can block every second; the *filtering capacity*, is the number of flows that the router can keep blocked simultaneously; the *filtering gain* is the number of blocked flows divided by the required number of filters:

$$G = \frac{N_{flows}}{N_{filters}}$$

For example, $G = 1$ flow/filter means that the router uses 1 filter to keep 1 flow blocked.

Victim's gateway — best-case scenario: Assume all attack gateways cooperate and all attack sources comply with their gateways to avoid disconnection. In this case, every time V_{gw} satisfies 1 filtering request, it spends 1 filter for T_{tmp} seconds and causes 1 flow to be blocked for T_{long} minutes. Thus, if V_{gw} uses $N_{filters}$ filters, it achieves filtering rate $\frac{N_{filters}}{T_{tmp}}$ flows/sec, filtering capacity $\frac{N_{filters} \times T_{long}}{T_{tmp}}$ flows, and filtering gain

$$G = \frac{T_{long}}{T_{tmp}}$$

For example, assume $T_{tmp} = 1$ sec and $T_{long} = 10$ min. With $10,000$ filters, V_{gw} blocks $10,000$ flows/sec and keeps $6,000,000$ flows blocked simultaneously.

If certain attack gateways and/or attack sources do not cooperate, undesired flows occur more than once. Suppose each undesired flow occurs on average n times. In this case, V_{gw} spends n temporary filters to cause 1 flow to be blocked for T_{long} minutes. Thus, filtering rate, capacity and gain drop by a factor of n. For example, if all attack sources lie to their gateways, each undesired flow occurs twice. Then, with $10,000$ filters, V_{gw} blocks $5,000$ flows/sec and keeps $3,000,000$ flows blocked simultaneously.

Victim's gateway — worst-case scenario: Now assume that none of the attack gateways cooperates, filter utilization exceeds its threshold, V_{gw} escalates all filtering requests, and all escalation attempts fail. In this situation, V_{gw} blocks traffic from all attack gateways to the victim locally, which means that with 1 filter, V_{gw} keeps 1 flow blocked. I.e., V_{gw} achieves filtering gain $G = 1$.

So, the maximum number of filters ever needed on V_{gw} to protect a single victim equals the total number of potential attack gateways. Note that this holds even if all undesired flows are spoofed — V_{gw} ends up *accepting* one flow from each alleged attack gateway, so it still needs as many filters as there can be attack gateways.

Surprisingly, the number of potential attack gateways is only a few tens of thousands. According to BGP data from Route Views [3] (retrieved in February 2005), there are currently $19,230$ Autonomous Systems (ASes); of these, roughly 90% correspond to edge domains, while the rest are Internet Service Providers (ISPs). We processed the data with Gao's algorithm for inferring AS relationships [15] to get the number of providers per edge domain. Assuming a separate border router per customer-provider pair, there are only tens of thousands of border routers that could act as attack gateways.

To summarize, there may be millions of attack sources, but there are only tens of thousands of edge domains to host them. A router cannot accommodate a million filters, but it does accommodate tens of thousands. So, if eBay is under attack, eBay's gateway may be unable to locally block each attack source individually, but it *is* able to locally block each edge domain individually — i.e., each edge domain that does not cooperate to block its own misbehaving clients.

Attack gateway: Every time A_{gw} completes a handshake, it spends 1 filter for T_{tmp} seconds and causes 1 flow to be blocked. If the attack source complies (to avoid disconnection), the flow remains blocked for T_{long} minutes, as requested. So, with $N_{filters}$ filters, A_{gw} blocks $\frac{N_{filters} \times T_{long}}{T_{tmp}}$ flows, i.e., A_{gw} achieves filtering gain $G = \frac{T_{long}}{T_{tmp}}$. For example, assume $T_{tmp} = 1$ sec and $T_{long} = 10$ min. Suppose A_{gw} provides connectivity to $64,000$ hosts (a class B network). With $256,000$ filters, A_{gw} blocks $2,400$ flows from each one of its clients.

If an attack source does not comply, the corresponding flow recurs before T_{long} minutes, A_{gw} completes a second handshake and spends again a temporary filter for T_{tmp} seconds. However, the attack source gets disconnected, which means that it does not come back online, unless it has been cleaned and patched. Thus, if attack sources do not comply, the filtering gain of the attack gateway actually increases, because infected hosts get disconnected, and A_{gw} does not have to filter their traffic again.

5.3 DRAM Requirements

Shadow memory: Both V_{gw} and A_{gw} keep a shadow filter for each flow that has been blocked. So, the maximum number of used shadow filters equals the maximum number of flows simultaneously blocked (i.e., the filtering capacity). Each shadow entry has form $\{A\ A_{gw} : R\ ...\ V_{gw}\ V\}$. Assuming R is 64 bits long (see Section 5.4) and flows are classified/filtered based on the last 6 components of their path, each shadow entry is 320 bits wide (2×32 bits for the IP source and destination, 6×32 bits for the non-random RR path components, and 64 bits for R). So, to keep a million flows blocked, V_{gw} needs about 40 MB of DRAM.

Note that DRAM is not the resource bottleneck; if it were not for the limited number of wire-speed filters, 40 GB of shadow memory would be enough to keep 1 billion undesired flows blocked.

Long-term filters on attack source: An attack source is not necessarily a malicious or compromised node; it is simply the sender of a traffic flow deemed undesired by its recipient. An "innocent" host classified as an attack source needs long-term filters to remember which receivers do not want its traffic (and avoid disconnection). Specifically, to satisfy n requests/sec by its provider, the host needs $n \times T_{long}$ filters. However, as opposed to wire-speed filters on routers, software filters on end-hosts are not a scarce resource.

Note that A does not risk disconnection for not satisfying requests beyond the agreed rate — a correctly functioning provider does not overload a customer with filtering requests and then disconnect the customer for failing to satisfy them.

5.4 Probability of Abuse

One potential attack against AITF is to try to guess the randomized RR header. For example, consider the topology of Figure 3 and suppose a set of malicious nodes (like M) spoof Stanford addresses and send high-rate traffic to eBay. V_{gw} contacts A_{gw}, gets the random value recorded by A_{gw} on all packets to eBay, and blocks all spoofed traffic. However, by sending a sufficiently large number of messages to eBay, the malicious nodes can try to guess the correct random value by brute force. If they succeed, they can successfully pretend to be Stanford hosts and potentially cause all traffic from Stanford to eBay to be classified as undesired and, thus, blocked.

To limit the probability of such abuse, A_{gw} changes the process by which it computes its random values every T_{change} minutes. If the random value is N bits long, to guess it with probability p, the malicious nodes must send $p \times 2^N$ messages. The amount of time it takes to send that many messages to eBay is bounded from below by eBay's maximum packet reception rate B. So, if the random value is N bits long, the malicious nodes can guess it during one T_{change} interval with probability $p = \frac{T_{change} \times B}{2^N}$. The probability that the malicious nodes guess one random value in T_{guess} minutes is the probability that they guess the random value in any of the $\frac{T_{guess}}{T_{change}}$ intervals:

$$P_{guess} = 1 - (1 - \frac{T_{change} \times B}{2^N})^{\frac{T_{guess}}{T_{change}}}$$

For example, if eBay is connected through a 1 Gbps link, it receives up to 1.95 million packets/sec.[1] Suppose $T_{change} = 10$ minutes and $N = 64$ bits. The probability that the malicious nodes guess one random value in a month is 2.74×10^{-7}.

Changing the process that computes the random value creates the following problem: Suppose A_{gw} has communicated random values to a number of victim gateways; the moment it changes the process, all communicated random values become invalid. To avoid this problem, when A_{gw} responds to a handshake, it communicates to the victim's gateway, not only its current random value, but also the next $\frac{T_{long}}{T_{change}}$ random values and when they will be valid. By choosing $T_{change} = T_{long}$, A_{gw} must pre-compute one random value.

We should note that the size of the random value cannot be chosen solely based on the desired probability of abuse; it also affects the bandwidth overhead introduced by RR (because each RR-enabled border router adds a random value to each forwarded packet). We discuss RR bandwidth overhead in the Appendix, Section A.4.

Another potential attack against AITF is to try to compromise the 3-way handshake. For example, consider the topology of Figure 1 and suppose a set of malicious nodes pretend to be V's gateway and initiate 3-way handshakes with A_{gw}, asking it to block all traffic to V. Assuming the malicious nodes are not on the path from A_{gw} to V, they do not see A_{gw}'s responses and the included nonce, necessary to complete the handshake. However, by sending a sufficiently large number of messages, they can guess the correct nonce by brute force. Following similar rational as above, the probability to guess one nonce in T_{guess} minutes depends on A_{gw}'s maximum packet reception rate. For example, if A_{gw} is connected through a 10 Gbps link, it receives up to 19.5 million packets/sec. Suppose $T_{change} = 10$ minutes and the nonce size is 64 bits. The probability that the malicious nodes guess one nonce in a month is 2.74×10^{-6}. For a 128-bit nonce, the probability becomes 1.3×10^{-25}. Note that, unlike the random value, the nonce is not included in every packet forwarded by A_{gw}; hence, the incentive to keep it small is less relevant.

5.5 Good Traffic Lost to Escalation

Escalation blocks all traffic from a non-cooperative attack gateway A_{gw} to the victim; clearly, this can lead to loss of good traffic. The decision to escalate or not is made by the victim, because the victim is the only one who can quantify the value of A_{gw}'s good traffic versus the damage caused by A_{gw}'s attack traffic. For example, suppose eBay is under attack by a million attack sources, all connected through AOL; at the same time, it is serving $1,000$ legitimate AOL clients. If the AOL gateway does not cooperate, only eBay can decide whether serving the $1,000$ good AOL clients is worth sustaining the $1,000,000$ bad ones.

Whether a flow is "escalatable" or not depends on the policy module. Hence, we cannot compute a general estimate of the percentage of good traffic lost to escalation. However, we do implement a simple policy module in our simulation, and show how its decisions affect the victim's good traffic.

6 Simulation Results

We use real Internet routing table data to build a realistic simulation topology. We simulate different attack scenarios, where multiple attack sources (up to a million) attack a single victim, connected through a 100 Mbps link; the victim's gateway uses up to $10,000$ filters to protect the victim. For each scenario, we plot the bandwidth of the attack traffic that reaches the victim as well as the victim's goodput as a function of time, i.e., we show how fast attack traffic is blocked and how much of the victim's goodput is restored.

6.1 Framework

We built our simulator within the Dartmouth Scalable Simulation Framework (DaSSF) [2]. To create our topology, we downloaded Internet routing table data from the Route Views project site [3]. We map each AS and each edge network to a separate AITF domain — we derive AS topology and peering relationships by applying Gao's algorithm for inferring inter-AS relationships [15] to the Route Views data; we derive edge network topology by roughly creating one edge network per advertised class A and class B prefix. Each AITF domain is represented by one AITF router. AITF routers are interconnected through OC-192 (9.953 Gbps) and OC-48 (2.488 Gbps) full-duplex links. End-hosts are connected to their

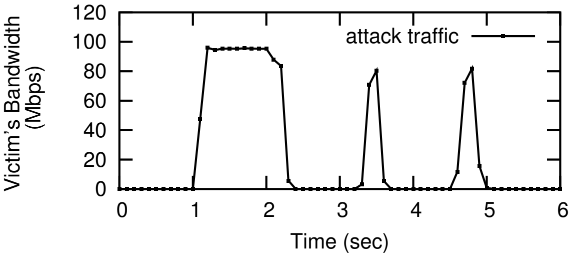

Figure 5: $t = 1$ sec: attack starts; $t = 2 - 3$ sec: V detects the attack and sends $10,000$ filtering requests to its gateway; $t = 3 - 4$ sec: flows recur for the first time; $t = 4 - 5$ sec: flows recur for the second time, and V_{gw} blocks all traffic from the compromised gateways.

routers through Fast (100 Mbps) and Thin (10 Mbps) Ethernet full-duplex links. Internet round-trip times average 200 msec. Host-to-router round-trip times average 20 msec. In all scenarios, $T_{tmp} = 1$ sec and $T_{long} = 2$ min.

6.2 Filtering Response Time

Our first experiment demonstrates that AITF achieves a filtering response time equal to the one-way delay from the victim to its gateway, i.e., on the order of milliseconds. It also demonstrates that "lying" gateways are quickly detected and blocked; the worst each lying gateway can do is cause up to two "spikes" spaced out by at least T_{tmp} seconds.

Scenario 1: The victim receives a flooding attack by $10,000$ attack sources, each behind its own attack gateway. The bandwidth of the attack (before defense) is 1 Gbps. All attack gateways are lying, i.e., they agree to block their undesired flows and then break the agreement.

Figure 5 illustrates that V_{gw} blocks attack traffic within milliseconds from the moment the attack is detected; attack traffic recurs twice and is completely blocked within 2 seconds. V_{gw} gives two chances to each attack gateway to honor its filtering agreement; when the attack gateways break their agreements twice, all their traffic to V is blocked. We run the experiment only for $10,000$ undesired flows (which allows the victim to have all of them blocked within 1 sec), so that the "spike" effect due to the recurring flows is visible.

6.3 Filtering Gain

The next two experiments demonstrate that the victim's gateway can achieve filtering gain on the order of hundreds, i.e., the victim's gateway blocks two orders of magnitude more flows than it uses filters.

Scenario 2: The victim receives a flooding attack by $100,000$ attack sources. The bandwidth of the attack (before defense) is 1 Gbps. The victim's goodput (before the attack) is approximately 80 Mbps. All attack gateways cooperate.

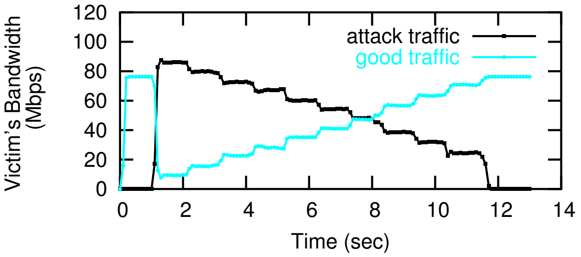

Figure 6: $t = 0-1$ sec: V is receiving $\sim$ 80 Mbps of goodput; $t = 1 - 2$ sec: attack drops V's goodput to 12% of original; $t = 2$ sec: V starts sending $10,000$ filtering requests/sec to its gateway; $t = 12$ sec: V's goodput is restored to 100% of original.

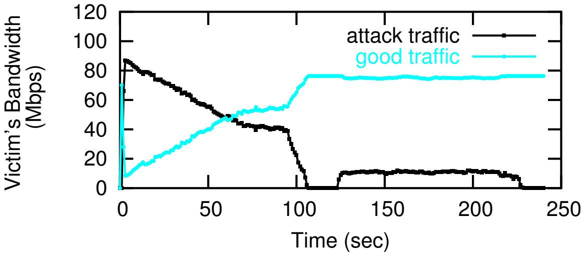

Figure 7: $t = 0-1$ sec: V is receiving $\sim$ 80 Mbps of goodput; $t = 1 - 2$ sec: attack drives V's goodput to 12% of original; $t = 2$ sec: V starts sending $10,000$ filtering requests/sec to its gateway; $t = 104$ sec: V's goodput is restored to 100% of original; $t = 122$ sec: filtering requests start expiring, undesired flows are released and re-blocked, $10,000$ at a time.

Scenario 3: Similar to scenario 2, but the victim receives a flooding attack by $1,000,000$ attack sources.

Figures 6 and 7 show that, using $10,000$ filters, V_{gw} blocks $100,000$ flows in 10 seconds and $1,000,000$ flows in 100 seconds. Without AITF, a router needs a million filters to block a million flows; these experiments demonstrate that, with AITF, V_{gw} needs only ten thousand filters to block a million flows. Hence, AITF reduces the number of filters required to block a certain number of flows by two orders of magnitude — a critical improvement, since routers typically accommodate tens of thousands of filters, whereas DDoS attacks can easily consist of millions of flows. Figure 7 also reveals what happens after $T_{long} = 2$ minutes. We assume that attack sources are "smart", i.e., they pause sending an undesired flow when so requested (to avoid disconnection) and they restart after $T_{long} = 2$ minutes.

6.4 Escalation and Lost Goodput

The last two experiments illustrate the trade-off involved in policy decisions regarding non-cooperative gateways. The victim's gateway may decide to escalate and lose

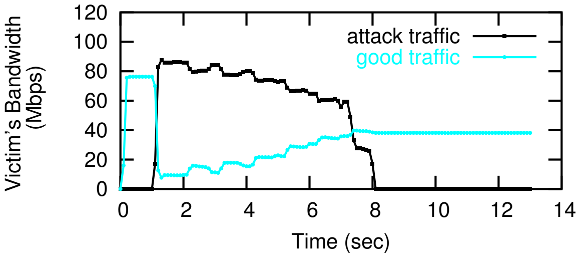

Figure 8: $t = 0{-}1$ sec: V is receiving $\sim$ 80 Mbps of goodput; $t = 1 - 2$ sec: attack drives V's goodput to 12% of original; $t = 2$ sec: V starts sending $10,000$ filtering requests/sec to its gateway; $t = 2 - 8$ sec: half of the attack gateways do not cooperate, so V_{gw} escalates and blocks all their traffic; $t = 8$ sec: V's goodput is restored to 50% of original.

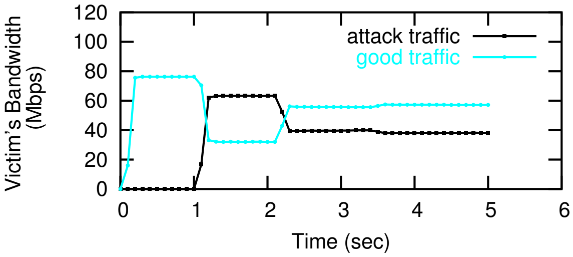

Figure 9: $t = 0{-}1$ sec: V is receiving $\sim$ 80 Mbps of goodput; $t = 1 - 2$ sec: attack drives V's goodput to $\sim$ 50% of original; $t = 2$ sec: V starts sending $10,000$ filtering requests/sec to its gateway; none of the attack gateways cooperate, so V_{gw} blocks locally as many flows as possible and lets the rest through; $t = 2 - 3$ sec: V's goodput is restored to $\sim$ 75% of original.

all traffic from a non-cooperative gateway to the victim; alternatively, it may decide to locally block as many undesired flows as possible and let the rest through. Although we did not implement a complete policy module, we choose two "extreme" scenarios, each favoring a different decision, implement the best policy for each scenario, and show the results.

Scenario 4: The victim receives a flooding attack, but, this time, half of the attack gateways are non-cooperative. Good and bad sources are collocated, i.e., evenly distributed behind the same gateways. The attack comes from $100,000$ attack sources; attack bandwidth (before defense) is 1 Gbps. The victim's goodput (before the attack) is approximately 80 Mbps.

Figure 8 shows that, using $10,000$ filters, V_{gw} restores 50% of the victim's goodput in 6 seconds. In this scenario, attack traffic has 10 times the rate of good traffic. The policy module chooses to block all traffic from non-cooperative attack gateways. This cannot make things any worse, since most good traffic is being dropped anyway due to the flood; on the contrary, it allows good traffic from cooperative gateways to get through.

Scenario 5: In this scenario, all attack gateways are non-cooperative, and good and bad sources are collocated. However, the attack comes from fewer attack sources $(20,000)$ and consumes lower bandwidth (160 Mbps, before defense). The victim's goodput (before the attack) is approximately 80 Mbps.

Figure 9 shows that, using $10,000$ filters, V_{gw} restores 75% of the victim's goodput in a few milliseconds from the moment the attack is detected. This scenario is trickier than the previous one, because attack traffic has only twice the rate of good traffic. In this case, escalating would be disastrous — it would drop goodput to 0, because good and bad sources are collocated. The policy module chooses to block as many flows as possible locally and let the rest through. This results in half the attack traffic being dropped and half getting through, which allows only 75% of the victim's goodput to get through. I.e., the victim prefers to sustain some attack traffic from non-cooperative attack gateways, rather than sacrificing all their good traffic.

7 Deployment

This section describes how AITF can be deployed in today's Internet. We also consider the incentives for early adopters and compatibility with legacy hosts.

7.1 Model

Rather than requiring every Internet router to support AITF, it is sufficient for the border routers between administrative domains to support it. We introduce the notion of an *AITF domain* as an administrative domain whose border routers are AITF-enabled.

An AITF domain has a *filtering contract* with each local end-host and peering domain. Such a contract specifies a maximum filtering request rate, i.e., the maximum rate at which the AITF domain can send/receive requests to block undesired flows to/from each end-host and peering domain. An AITF domain enforces the specified rates and indiscriminately drops messages from an end-host/domain when that party exceeds the agreed rate.

In a way, an AITF domain is the "dual" of a BGP Autonomous System (AS): ASes exchange routing information, which communicates their willingness to relay certain packets. Similarly, AITF domains exchange filtering information, which communicates their *unwillingness* to receive certain packets. However, an AITF domain differs from an AS, in that it exchanges messages with other AITF domains that are not adjacent to it — recall that the victim's gateway talks directly to the attack gateway. It seems natural for every AS to map to a separate AITF domain. Popular public-access sites, like eBay or Amazon, may even deploy AITF on their corporate firewalls, to avoid sharing a victim's gateway with other customers of their AS.

Our position is that the filtering contract should be part of the Service Level Agreement (SLA) signed between the customer and the service provider. In this way, when a domain agrees to provide a certain amount of bandwidth to a customer, the provider also agrees to satisfy a certain rate of filtering requests coming from that customer. At the same time, the customer agrees to satisfy a certain rate of filtering requests coming from the provider. The customer-provider pair can be an end-host and an edge network, or an edge network and an ISP, or even a small ISP and a backbone network.

7.2 Incentive for Initial Deployment

It makes sense for AITF deployment to begin at the edges: An edge network that hosts potential DDoS victims (e.g., a web hosting domain) deploys AITF to protect its clients; an edge network that hosts potential attack sources (e.g., a campus network or a dialup provider) deploys AITF to maintain its connectivity to public-access sites, even when these sites are under attack.

For example, in Figure 3, we depicted an early deployment stage, where only eBay and Stanford have deployed AITF. Suppose a worm compromises millions of attack sources, uniformly distributed across the Internet, and commands them to send high-rate traffic to eBay. Without AITF, there is nothing eBay can do; almost all its good traffic is lost in the flood. With AITF, eBay identifies the undesired flows coming from Stanford; eBay's gateway exchanges a handshake with Stanford's gateway, which agrees to block its undesired flows to eBay; eBay's gateway accepts all traffic from Stanford and drops (or rate-limits) all the rest — most good traffic to eBay is lost anyway due to the attack, so dropping traffic from AITF-unaware domains cannot make things worse. I.e., Stanford is the only domain that cooperates with eBay and, hence, the only domain to maintain its connectivity to eBay throughout the attack.

To conclude, the first domains to deploy AITF benefit, because they preserve their connectivity to each other in the face of DDoS attacks. As AITF deployment spreads, the benefit of AITF-enabled domains grows, because they maintain their connectivity to larger and larger portions of the Internet.

7.3 Compatibility with Legacy Hosts

AITF involves a rather draconian measure against attack sources: Either they stop sending an undesired flow or they get disconnected. One could argue that malicious node M can abuse this measure to disconnect legacy host L: M sends a filtering request to L's gateway to block all traffic from L to M; L's gateway sends a similar request to L (which is ignored, since L is AITF-unaware); then M tricks L into sending it traffic (e.g., sends an ICMP request); as a result, L's gateway disconnects L.

This scenario cannot happen. Recall that an AITF domain has a filtering contract with each of its customers; the contract specifies the maximum rate at which the domain sends filtering requests to the customer. An AITF-unaware customer by definition has agreed to rate 0; hence, its provider never sends filtering requests to it.

A domain that deploys AITF either forces its end-hosts to deploy AITF as well, or accepts the cost of filtering their undesired flows; for example, L's gateway locally blocks all traffic from L to M for T_{long} minutes. The second option is more incrementally deployable, but requires more filtering resources from an attack gateway — namely, as many filters as undesired flows generated by its end-hosts. As a compromise, a provider can charge legacy customers that do not support AITF, for the potential cost induced by their inability to block their undesired traffic themselves.

8 Discussion of Additional Attacks

Malicious nodes collocated with the victim: The handshake between the victim's gateway and the attack gateway protects only the communication between these two entities; it does not protect the communication between the victim and the victim's gateway. A malicious end-host located on the same LAN with end-host V can clearly spoof V's address, send filtering requests as V, and disrupt V's communications.

An AITF domain can easily avoid such abuses by either preventing source address spoofing in its own network or authenticating local filtering requests. Note that the latter does not require any public key infrastructure; it only requires from each border router of an AITF domain to share a secret with each of the domain's customers.

DDoS against AITF: One could argue that a set of malicious nodes can launch the following attack against router A_{gw}: First, pretend they are victim gateways and send a high rate of filtering requests to exhaust A_{gw}'s filters. Once A_{gw}'s filters are exhausted, a malicious node located behind A_{gw} is commanded to start an undesired flow against, say, Google; Google asks from A_{gw} to stop, A_{gw} has no filters left and gets disconnected from Google.

The first thing to note is that the attack gateway uses the 3-way handshake to verify the "authenticity" of a filtering request, i.e., that the requester is on the path to the alleged victim. Once a request is deemed authentic, the identity of the requesting victim gateway is established, and the attack gateway can accept or drop the corresponding request based on that. The attack gateway satisfies up to a certain rate of requests from each victim gateway to avoid exhausting all its filters to satisfy a few demanding domains.

Now consider a set of malicious nodes seeking to attack router A_{gw}. One way is to send to A_{gw} a high rate of authentic filtering requests; the only thing they accomplish is to exhaust the quota of their own domains.[2] The other way is to send to A_{gw} a high rate of spoofed filtering requests; now their requests are dropped. The only harm the malicious nodes can do is launch a SYN-flood-style attack, i.e., flood A_{gw} with fake requests, hoping to exhaust the processing cycles devoted to 3-way handshakes.

There are two steps to dealing with such attacks. The first one is to implement the part of the attack gateway algorithm that deals with the 3-way handshake in the fast path — it just involves hashing certain contents of a received message and comparing the result to a value included in the message. The second step is to let A_{gw} act as the victim: if a set of malicious nodes manage to flood A_{gw}'s AITF (or any other) hardware module, then A_{gw} uses the RR headers of incoming traffic to identify undesired flows and asks from its own gateway to have them blocked.

Malicious on-the-path nodes: The handshake between the victim's gateway and the attack gateway prevents malicious node M from installing a filter on router A_{gw} to block certain traffic to router V_{gw}, as long as M is off the path from A_{gw} to V_{gw}. A malicious node on the path from A_{gw} to V_{gw} can clearly forge filtering requests and disrupt A_{gw}-V_{gw} communication.

However, a malicious node on the path from A_{gw} to V_{gw} can only be an Internet core router; such a malicious router can disrupt A_{gw}-V_{gw} communication anyway, e.g., by blocking all their traffic. I.e., if a core router gets compromised, thousands of domains that connect through that router are at its mercy — it makes little difference whether the router is AITF-enabled or not.

9 Related Work

Packet Marking: The alternative to route record is probabilistic packet marking (PPM)[21, 14]: Each participating router marks each forwarded packet with certain probability p; a DDoS victim combines marks from multiple packets to identify the routers that forward attack traffic. Instead of using a separate header, PPM uses a lightly utilized IP header field (typically the 16-bit IP identifier); doing so facilitates deployment and decreases packet overhead. Although a promising and incrementally deployable traceback technique, PPM is less useful in actually blocking attack traffic. Identifying the routers that forward attack traffic is not enough; the victim's gateway V_{gw} must identify *their traffic* in order to block it. Even if attack gateway A_{gw} agrees to block an undesired flow, V_{gw} must still identify A_{gw}'s traffic in order to verify that A_{gw} is honoring the agreement. Therefore, V_{gw} must be able to identify the path followed by each incoming packet at wire speed. The only way to perform path-based wire-speed filtering is to have each packet's path deterministically recorded on the packet.

One could certainly argue for "compressing" the path — no need to record one full 32-bit address per border router! In Path Identifier (Pi) [23], each participating router deterministically marks the forwarded packets, so that each packet obtains a "fingerprint" that reflects the entire path followed by the packet. Indeed, this approach enables the victim (or its gateway) to locally block undesired flows in the face of source address spoofing. However, it does not meet the two following requirements: (i) V_{gw} must know the addresses of the border routers on an undesired flow's path; otherwise, it must locally block all attack flows itself, which is typically beyond its capabilities. (ii) V_{gw} must be able to block attack traffic at different granularities, e.g., block all packets with path $\{* \; A_{gw} \; ... \; V_{gw} \; V\}$.

Filtering: The Pushback scheme [18] uses hop-by-hop filter propagation to push filtering of undesired traffic away from the victim: The victim identifies the upstream routers that forward attack traffic to it and sends them filtering requests; the routers satisfy the requests, potentially identify the next upstream routers that forward attack traffic to the victim, and send them similar requests. Each Pushback router installs a single filter per victim and rate-limits all traffic to each victim. The main benefit of this approach is that its does not require knowledge of the attack paths, which makes it deployable without packet marking. However, when attack traffic and good traffic share common paths, and attack traffic is of much higher rate than good traffic (increasingly the case, nowadays), rate-limiting all traffic to the victim sacrifices most of the victim's good traffic.

The Stateless Internet Flow Filter (SIFF) [9] divides all Internet traffic into privileged and non-privileged. A client establishes a privileged channel to a server through a capability exchange handshake; the client includes the capability in each subsequent packet it sends to the server; each router along the path verifies the capability and gives priority to privileged traffic. The main advantage of SIFF is that it does not require any filtering state in the routers. However, once a server is under attack, a new client must contend with attack traffic to establish a connection — because channel establishment involves an exchange of non-privileged packets. It also requires counter-measures to prevent malicious nodes from establishing privileged channels between themselves and flood the network with privileged traffic.

Secure Overlays: The set of AITF-enabled border routers can be viewed as a "filtering overlay". Filtering overlays have also been suggested as a way to prevent

DoS against critical applications (like Emergency Services) that are meant to be accessed only by authorized users [16, 12]. In that context, the overlay nodes perform client authentication and relay traffic to a protected server, whose IP address is unknown outside the overlay.

10 Conclusion

We presented Active Internet Traffic Filtering (AITF), a mechanism for filtering highly distributed denial-of-service attacks. We showed that AITF can block a million undesired flows, while requiring only tens of thousands of wire-speed filters from each participating router — an amount easily accommodated by today's routers. It also prevents abuse by malicious nodes seeking to disrupt other nodes' communications.

More specifically, we showed the following:

1. AITF offers filtering response time equal to the one-way delay from the victim to the victim's gateway. I.e., a victim can have an undesired flow blocked within milliseconds.

2. AITF offers filtering gain on the order of hundreds of blocked flows per used filter. I.e., a router can block two orders of magnitude more flows than it has wire-speed filters. For example, suppose eBay is receiving a million undesired flows; with $10,000$ filters, eBay's gateway can have all flows blocked within 100 seconds. In the worst-case scenario, eBay's gateway blocks all traffic from each domain that hosts attack sources and refuses to filter their traffic, which (in today's Internet) requires a few tens of thousands of filters.

3. A set of malicious nodes can practically not abuse AITF to disrupt communication from node A to node B, as long as they are not located on the path from A to B. This holds even during initial deployment, where most Internet domains are AITF-unaware.

The idea behind AITF is that the Internet *does* have enough filtering capacity to block large amounts of undesired flows — it is just that this capacity is concentrated close to the attack sources. AITF enables service providers to "gain access" to this filtering capacity and couple it with a reasonable amount of their own filtering resources, in order to protect their customers in the face of increasingly distributed denial-of-service attacks.

11 Acknowledgments

We would like to thank Daniel Faria, Evan Greenberg, Dapeng Zhu, George Candea, Nagendra Modadugu, Costa Sapuntzakis, Tassos Argyros and our shepherd, Mema Roussopoulos, for their constructive feedback.

A Appendix: Route Record

A.1 Header Update

The RR header consists of three fields: the *path* is a list of (initially empty) slots; the *size* is the total number of slots in the path; the *pointer* points to the first empty slot in the path. The first RR-enabled border router on a packet's path (i) inserts an RR header in the packet; (ii) writes its own IP address and random value in the first slot; and (iii) sets the pointer to point to the second slot. Each subsequent RR-enabled border router that ingresses the packet into a new AS (i) reads the pointer; (ii) writes its own IP address and random value in the indicated slot; and (iii) increments the pointer to point to the next slot. If there is no room left in the RR header, the router drops the packet.

A.2 Hardware Support

Each RR-enabled border router updates the RR headers of forwarded packets as described in Section A.1. RR-header update requires (i) computing a keyed hash function on the packet's destination, (ii) reading and modifying the RR pointer, and (iii) writing the router's address and random value on the indicated slot. So, although the RR header has a variable length, its update requires reading/modifying a 4-bit pointer and a 96-bit path component (assuming 64-bit random values). Computing a 64-bit hash (like HMAC-SHA1) per forwarded packet can be easily done today at wire-speed [11].

As mentioned in Section A.1, RR headers are not inserted by end-hosts; they are inserted by the first border router on each packet's path — call it the packet's "gateway". So, when an edge network deploys RR, it upgrades its border routers to support RR-header insertion. A border router determines the required RR header size for each packet, by looking up the AS path length to the packet's destination domain as communicated through BGP. If the real AS path turns out to be longer than the communicated AS path, the packet gets dropped; the packet's gateway receives an ICMP message, increases the RR header size and retransmits the packet — i.e., AS path length discovery is similar to MTU discovery using the "Don't Fragment" IP-header flag. Once a router discovers the real AS path length to a destination domain, it caches its value to avoid future retransmissions.

RR-header insertion is similar to packet encapsulation, a well-studied operation, for which multiple hardware implementations already exist. Note that, as stated in Section 7.2, RR/AITF deployment starts at the edges; i.e., it is edge routers that insert RR headers, not core routers connected to Internet backbones. Edge routers are connected at best through OC-48 (2.488 Gbps) links; the Cisco 10000 Series OC-48c linecard already supports encapsulation.

A.3 Compatibility with Legacy Hosts

Before inserting an RR header in an outgoing packet, the packet's gateway must make sure that the receiving domain has deployed RR; otherwise, the receiver will not recognize the RR header and drop the packet. Hence, each packet gateway keeps track of the destination ASes to which it forwards packets, and asks them whether they care to receive RR headers.

To avoid introducing latency, a packet gateway forwards packets without inserting RR headers and starts doing so as soon as it concludes that the destination domain is RR-enabled. To avoid querying the destination domain on each packet, it periodically queries potential destination domains (e.g., once per hour) and caches their response. Note that the number of potential destination domains can be no bigger than the number of Internet ASes — $19,230$ for the current Internet.

A.4 Bandwidth Overhead

RR bandwidth overhead depends on the average number of ASes per packet path. Although we do not know this number, we can approximate it with the average number of AS-level hops between Internet ASes, which is close to 4 [17, 10]. Assuming an average of 4 ASes per Internet path and 64-bit random values, route record introduces on average 49 extra bytes per packet. For an average packet size of 500 bytes [5], this leads to 10% bandwidth overhead.

Bandwidth overhead can be reduced to 5% by using the following twist: Instead of each router adding a random value to the recorded path, only one router does so; the first border router that forwards the packet into an RR-unaware domain. For example, consider the topology in Figure 1; suppose only A_{NET}, A_{ISP} and V_{NET} have deployed AITF. If a packet is sent from A to V, the only router that adds a random value to the packet's header is X. This reduces bandwidth overhead, at the cost of restricting the routing of filtering requests: If V sends a filtering request against A's traffic, that request must be routed through X, so that X verifies the authenticity of the recorded path.

Notes

[1] To make our analysis conservative, we assume that eBay responds to all the messages sent by the malicious nodes. In reality, if a set of nodes send such high-rate traffic to eBay, eBay will consider their traffic undesired and use AITF to have it blocked.

[2] An AITF-enabled border router can prevent its own clients from pretending to be victim gateways and exhausting its quota, simply by dropping all outgoing filtering requests — recall that no other node but the border router itself is supposed to send filtering requests outside the local domain.

References

[1] Access list configuration in Cisco's Gigabit Ethernet Interface. http://www.cisco.com/en/US/products/hw/switches/ps5304/prod_configuration_guides_list.html.

[2] Dartmouth scalable simulation framework. http://www.crhc.uiuc.edu/~jasonliu/projects/ssf/.

[3] Route Views Archive. http://archive.routeviews.org/oix-route-views/.

[4] SiberCore SCT1842 Features. http://www.sibercore.com/products_siberCAM.htm#3.

[5] Sprint backbone data. http://ipmon.sprint.com/packstat/packetoverview.php.

[6] SCO Offline from Denial-of-Service Attack. http://www.caida.org/analysis/security/sco-dos/, December 2003.

[7] Attack downs Yahoo, Google. http://news.zdnet.co.uk/internet/security/0,39020375,39157748,00.htm, June 2004.

[8] DDoS Attack Knocks Out DoubleClick Ads. http://www.eweek.com/article2/0,1759,1628340,00.asp, July 2004.

[9] SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Security and Privacy*, May 2004.

[10] BGP table data. http://bgp.potaroo.net/index-bgp.html, February 2005.

[11] Personal communication with Fusun Ertemalp, Distinguished Engineer at Cisco Systems, February 2005.

[12] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *USITS*, March 2003.

[13] D. J. Bernstein. SYN Cookies. http://cr.yp.to/syncookies.html, 1996.

[14] D. Dean, M. Franklin, and A. Stubblefield. An Algebraic Approach to IP Traceback. *NDSS*, February 2001.

[15] L. Gao. On Inferring Autonomous System Relationships in the Internet. In *Global Internet*, November 2000.

[16] A. D. Keromytis, V. Misra, and D. Rubenstein. Secure Overlay Services. In *ACM SIGCOMM*, August 2002.

[17] D. Magoni and J. J. Pansiot. Analysis of the Autonomous System Network Topology. *ACM CCR*, 31(3):26–37, July 2001.

[18] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Schenker. Controlling High Bandwidth Aggregates in the Network. *ACM CCR*, 32(3):62–73, July 2002.

[19] D. A. Patterson. A simple way to estimate the cost of downtime. In *USENIX Systems Administration Conference*, November 2002.

[20] J. Postel. RFC 791 - Internet Protocol.

[21] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, August 2000.

[22] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *USENIX Security*, August 2002.

[23] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend against DDoS Attacks. In *IEEE Security and Privacy*, May 2003.