An Edge-to-Edge Filtering Architecture Against DoS

Felipe Huici, Mark Handley
Dept. of Computer Science, University College London
London, United Kingdom
f.huici@cs.ucl.ac.uk, m.handley@cs.ucl.ac.uk

ABSTRACT

Defending against large, distributed Denial-of-Service attacks is challenging, with large changes to the network core or to end-hosts often suggested. To make matters worse, spoofing adds to the difficulty, since defenses must resist attempts to trigger filtering of other people's traffic. Further, any solution has to provide incentives for deployment, or it will never see the light of day. We present a simple and effective architectural defense against distributed DoS attacks that requires no changes to the end-hosts, minimal changes to the network core, is robust to spoofing, provides incentives for initial deployment, and can be built with off-the-shelf hardware.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: [Network Architecture and Design]

General Terms

Design, Security

Keywords

Denial-of-service, Internet Architecture

1. INTRODUCTION

Despite serious under-reporting, the number and size of DoS attacks are growing at an alarming rate. ISPs are under constant pressure to upgrade access speeds, often leaving little time to implement security measures: it is precisely these high-speed yet largely unmonitored links that provide a powerful base from which to launch attacks. This, combined with the continuous stream of exploits, ensures that attacks will keep growing both in number and size. In its biannual report[32], Symantec reported as many as 1,400 DoS attacks per day, an increase of 51% since the previous report. Botnet size is also increasing: reports from 2005 indicate that Dutch bot-herders may have controlled as many as 1.5 million hosts[10]. And whereas early DoS attacks were perpetrated by script kiddies, recent years have seen a shift towards organized crime and extortion[26].

As DDoS attacks grow, it becomes increasingly clear that fighting them at or near the victim is largely ineffective, since the traffic can aggregate sufficiently to saturate even well-provisioned links. Designing defenses against these attacks is intrinsically difficult. While the core of the network would seem like the perfect location to place defense mechanisms, ISPs there have few if any direct incentives to deploy new measures. Further, change in the core is likely to require hardware change to core routers. Even if simply reconfiguring routers is all that is needed, operators are generally reluctant to make modifications without clear incentives. Defenses implemented at the client hosts are just as problematic. Unless such defenses were hardware-based, it is likely they could be broken or exploited by an attacker. Source address spoofing complicates matters further. Any DoS defense mechanism that does not take this into account can be subverted by an attacker into denying service to an unsuspecting victim, turning the mechanism into a DoS tool in its own right. Finally, any practical solution needs to provide clear incentives for initial deployment or it will fail regardless of technical merit.

In this paper we present a new and rather simple architectural defense against distributed Denial-of-Service attacks. It requires minimal changes to the core network, no changes to the end clients, it is robust to spoofing, it provides initial deployment incentives for all parties involved and can be implemented using currently-deployed or off-the-shelf hardware.

2. PROPOSED SOLUTION

Our aim is to tackle one of the main enablers of DoS attacks in the current Internet: hosts are allowed to send traffic regardless of whether the receiver wants it or not. The solution allows hosts under attack to request that the network stop traffic from specified sources before it can aggregate significantly. Three basic mechanisms are required: *marking*, so that the network can know where malicious packets are coming from; *filtering*, so that undesired packets are dropped; and *routing*, so that traffic traveling towards a destination is forced to traverse this filtering.

In previous work [15], we described an architecture using similar mechanisms. The approach consisted of diverting traffic going to protected servers so it traversed control points. These control points would IP-in-IP encapsulate the traffic, sending it to a decapsulator near the server. The server could then tell which control point a malicious flow had traversed, and request it be shut down at this boundary. However, there were some shortcomings. The distribution of routes relied on BGP, putting a burden on global routing, and generally requiring disaggregation of these routes. The marking and filtering of packets happened fairly close to the destination, so that each control point had to be able to handle potentially large amounts of traffic. Initial deployment was also only effective for larger ISPs with many edge

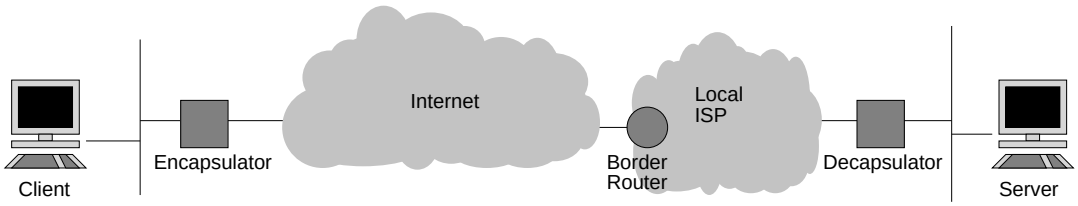


Figure 1: Encapsulation architecture.

routers, so that the incoming attack could be diffused across several of these control points.

These shortcomings, while not insurmountable, can be avoided with a different encapsulation strategy, described in this paper. Our new architecture implements the three mechanisms mentioned earlier, but does so in a very different way. The most important change has to do with the location of the control points that mark and filter packets: whereas previously they were placed at the edges of the ISP hosting the server being defended, these are now deployed as close to the clients of the server as possible.

This change has significant benefits. First, it means that these control points never have to handle large traffic aggregates and so can be built and deployed inexpensively. Second, their placement at the edges of the network makes it difficult for an attacker to indirectly DoS a server by attacking the encapsulator. Finally, no filtering or marking is required from the middle of the network, simplifying the deployment story.

The architecture presented in this paper also improves over the previous one by distributing *path-agnostic* routes using a separate and extremely robust peer-to-peer protocol, relieving any burden from BGP and removing any disaggregation issues. Finally, it has a simpler initial deployment story, and provides better incentives for it.

But with these advantages come one major problem: it is now possible for some attackers at legacy ISPs to spoof the encapsulation. A key contribution of this paper is to show that this is not a showstopper; some additional mechanism is required to be robust, but the complexity is not excessive. The following sections explain the full solution and these issues in greater detail.

2.1 Marking and Filtering

As source IP addresses can be spoofed, if we want to shut down malicious traffic close to its source, a marking mechanism is needed so packets can be traced back to their origin. While many solutions have been proposed for this, a simple mechanism already exists that is just right for the job: IP-in-IP tunneling. The idea is simple: encapsulate all packets near their sources and decapsulate them near their destinations, using the encapsulation header to record the origin network. Two additional boxes are needed for this: an "encapsulator" near the source and a "decapsulator" near the destination; these boxes will also collaborate to filter unwanted traffic. We believe they can be implemented using off-the-shelf hardware.

During normal operation (see Figure 1), a packet from a client will reach a local encapsulator on the path to the Internet. The encapsulator looks up the IP address of the decapsulator, IP-in-IP encapsulates it, and sends the packet to the decapsulator. The source address in the encapsulation header serves to tell the decapsulator which encapsulator forwarded the packet. At the decapsulator, the outer encapsulation header is removed, and the packet is forwarded on to the server.

When a protected server comes under attack, it sends a request to its local decapsulator to filter traffic it deems unwanted[1]. On receipt of a filtering request for a particular source, the decapsulator waits for the next packet from that source and notes down which encapsulator it came through. While this requires the decapsulator to keep state and monitor the traffic going through it, this state is only temporary and lasts only until the filtering request is sent.

Once the decapsulator figures out which encapsulator to talk to, it sends out the actual filtering request. Finally, the encapsulator installs the filter, blocking the unwanted traffic. In this way, the server under attack can ask the network to stop sending it undesired traffic, an impossibility in the current Internet.

Of course the full story is not quite as simple as sketched out above, and we will now look at what would be required for this general idea to be viable.

2.2 Routing

To perform edge-to-edge encapsulation, the encapsulator needs to know how to map the destination IP address from a data packet to the address of the relevant decapsulator. Essentially this is a routing problem, and this information could in principle be conveyed by BGP. However, using BGP would be far from ideal, as the routers in the core of the Internet do not need to know this information. Indeed, the encapsulator does not care about the precise path, but only which decapsulator to tunnel the packet to. Thus, we would be burdening BGP with a great deal of additional information for no good reason. The mapping from network prefix to decapsulator address or addresses is relatively static, so we suggest that a separate dissemination channel be used to distribute these mappings.

As this information is not path-specific, any simple flooding algorithm can be used for this distribution, so long as the data itself is secured. Since no policy is involved, decapsulation routes can be flooded to all of an ISP's neighbors, making this distribution much more robust than conventional inter-domain routing. We will discuss possible routing designs in Section 4.4.

How large would the decapsulator routing table held at each encapsulator be? This clearly depends on how widely decapsulators are deployed, but in the extreme case where every routed network is associated with one or more decapsulators, this table is likely to be at least as large as the backbone BGP routing tables, currently in the region of 200,000 routes. The size of these decapsulator tables would increase further if decapsulation is performed close to the destina-

[1]Detection mechanisms are beyond the scope of this paper, but commercial detection boxes may be suitable for this role.

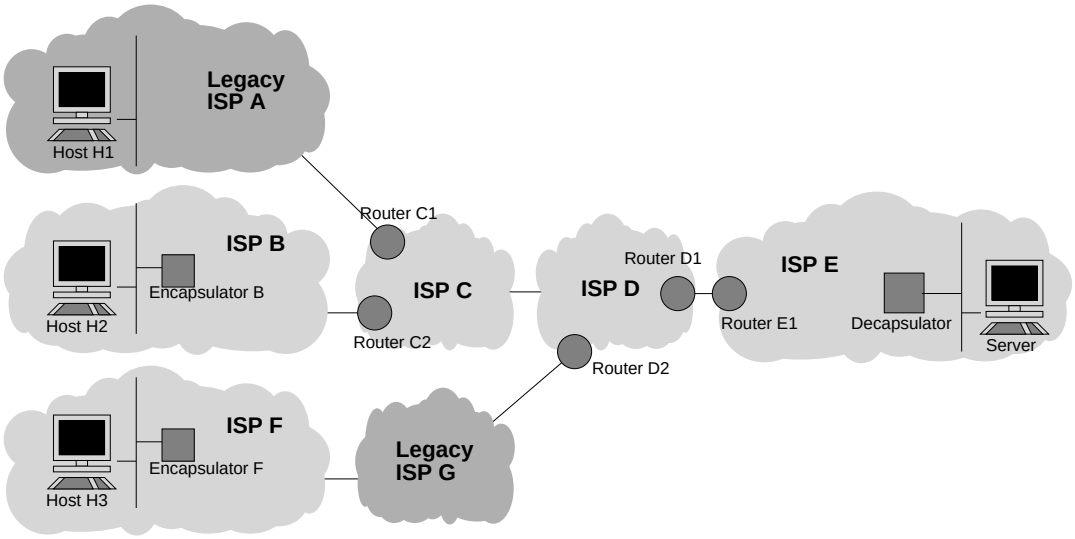


Figure 2: Partial deployment scenarios.

tion subnets for routes which are advertised via BGP to the public Internet as aggregate routes. We might imagine a near-worst-case scenario where every /24 subnet were advertised with a different decapsulator. At the present time, with the IPv4 unicast address space roughly half-filled, this would require about eight million prefixes. This seems like a lot, but the problem is rather different from that faced by backbone routers running BGP. A BGP router must maintain full routing information from many BGP peers, must calculate the best route as paths change, and must perform longest prefix match on the resulting forwarding table of best routes. The decapsulator tables are different, as there is only one routing table used by all encapsulators worldwide, and it does not change as paths through the Internet change. If these routes were maintained as a flat hash-table, rather than requiring longest prefix match, there should be no problem doing look-ups on this table at high speed. As the mappings are relatively static, distributing and maintaining this data should not cause a problem either.

Making an encapsulator check eight million digital signatures on startup might be problematic. In reality though, the number of *signatures* required is not related to the number of decapsulators, but rather to the number of origin Autonomous Systems in the BGP routing tables. Each AS can sign as a group all the prefix-to-decapsulator bindings for routes that it advertised, reducing the number of signatures to around 20,000 or so in the current Internet. In addition, there is no need for the edge encapsulators to directly check the signatures themselves. Instead a hierarchy within an ISP can be established, whereby one or more servers receive the routes from neighboring ISPs, check the signatures and only then pass them on to the encapsulators. Thus we believe that signed mappings from destination address prefixes to decapsulator addresses are technically viable and economically feasible, even in the extreme case of a different decapsulator for every /24 subnet in the Internet.

2.3 Legacy ISPs

If the mechanism deployed above were ubiquitous, then source-address spoofing by end-systems would be ineffective, and all unwanted flooding attacks could be stopped close to their sources. However, such a scheme must also work with partial deployment, which leaves open the possibility of DoS attacks by end-systems at legacy ISPs that do not deploy encapsulators.

To protect against attacks from such hosts, an ISP protecting a server can involve the border routers at its ingress points. One possible solution would be to deploy a diffserv classifier that prioritizes IP-in-IP traffic destined for the local decapsulators, reducing the effectiveness of bandwidth flooding attacks that attempt to saturate links. Attacks using unencapsulated traffic will then have minimal effect on traffic from networks deploying encapsulators.

Of course the astute attacker will then simply perform encapsulation directly from his attacking end-systems, so that his traffic is prioritized too. By spoofing the encapsulation header, such an attacker at a legacy ISP can make his traffic appear to be coming from many encapsulators. One way to avoid spoofed encapsulation would be to restrict the distribution of routes for the decapsulator addresses, so that the decapsulators are simply unreachable from legacy ISPs. If all the ISPs deploying encapsulators formed a connected graph under normal BGP routing, then this would effectively prevent such attacks from succeeding. However, it is unlikely that such a graph would be connected, at least early in the deployment process, so this solution is not ideal.

Another option would be for all ISPs deploying our scheme to also run an encapsulator for all traffic arriving from legacy neighbors; this is close to the solution described in [15]. However, performing such encapsulation and filtering on high-speed peering links would be more costly than performing it at edge-links since it could no longer be done with off-the-shelf PCs; this would present an unnecessary deployment hurdle.

Instead, our preferred solution is to use a single bit in the packets to indicate that traffic destined to a protected server has traversed a legacy ISP. Such a bit is inspired by Bellovin's "evil bit" [5]. ISPs deploying our scheme install a simple filter at border routers connecting to legacy ISPs, setting the evil bit in packets arriving from these neighbors. If along the path a packet traverses any ISP that does not perform encapsulation, then it is classed as potentially evil. ISPs hosting servers can then prioritize traffic that has come via a path where all the ISPs perform encapsulation.

Figure 2 illustrates this mechanism. ISPs A and G are legacy ISP while the other ISPs have deployed encapsulation. Routers C1, C2, D1, D2, and E1 are conventional border routers configured with simple packet classifiers.

A spoofed encapsulated packet from host H1 destined for the server will traverse ISP A unchanged. Through contractual agreements ISP C will know whether ISP A is a legacy ISP or not; since it is, when the packet reaches ISP C, Router C1 will set the packet's "evil bit", since it knows that the packet came from a legacy neighbor. Router E1 later uses the evil bit to put the packet in a lower priority diffserv class.

A similar packet from host H2 would reach E1 with its evil bit *cleared*, since ISPs B, C and D have all deployed encapsulation; as a result, it is classified as higher priority. However, should the server deem packets from this source to be malicious, it can request that the encapsulator drop them. Even if the host is spoofing, this mechanism will be able to filter the traffic as far as the encapsulator, at which point the problem becomes a local one.

Finally, an encapsulated packet from host H3 would have its evil bit set by ISP D, since although ISP F has deployed encapsulation, provider G has not. Thus all packets arriving at router D2 get the evil bit set, and so receive lower priority at E1. While this may at first seem harsh, it provides the right incentive: customer ISP F will put pressure on ISP G to deploy the scheme (or even switch providers) so that their clients will not be placed in a lower priority queue.

3. PREVENTING ABUSE OF DEFENSES

Providing a mechanism whereby the recipient of traffic can request that traffic cease is an effective way to defend against all but the most subtle flooding attacks. However, care must be taken to avoid an attacker using our mechanism to deny service to legitimate traffic. We divide the spectrum of possible attacks into two independent vectors:

- Direct attacks that send filtering requests.
- Indirect attacks that generate spoofed traffic with the aim of triggering the defense mechanism to take inappropriate action.

For both cases it is possible to exhaustively enumerate the options open to an attacker, based on what a bot can spoof under differing circumstances and on where the bot is located relative to the systems under attack. These are shown in figures 3 and 4.

For direct attacks the attacker's options are fairly limited, and we only need to consider spoofed decapsulators in various locations. For indirect attacks there are more cases to consider:

- A bot at a legacy ISP may be able to spoof the encapsulator header, or the client address (so long as a full TCP connection is not needed for the attack) or both.
- A bot at an encapsulating ISP may still be able to spoof the client address if ingress filtering is not performed.
- A bot may be located at the same ISP as the legitimate client he wishes to deny service to, or at a different ISP.

For indirect attacks, spoofing the decapsulator provides no advantage, so we do not consider this case.

Spoofed C	Spoofed E	A same E as C	Comments
×	×	×	No spoofing
×	×	√	No spoofing
×	√	×	See figure 5:1
×	√	√	Impossible
√	×	×	See figure 5:2
√	×	√	See figure 5:3
√	√	×	See figure 5:4
√	√	√	Impossible

Figure 3: Table of indirect attacks. The letter A stands for the attacker, C the client, E the encapsulator and D the decapsulator.

Spoofed Request	A on $E \leftrightarrow D$ Path	Comments
×	×	No attack
×	√	A can drop request
√	×	See figure 5:5
√	√	A can drop request

Figure 4: Table of direct attacks.

We will now examine each of the indirect attack scenarios in Figure 3. The first two entries do not constitute an attack and are there merely for completeness.

The third entry (figure 5.1) consists of an attacker C located at a legacy ISP who spoofs encapsulator E, but uses his own address C as the client address. The reason for doing this might be that he needs a full TCP connection to trigger a response from the server's defenses. As a result of the spoofing, if the server determines the traffic of C to be malicious, the decapsulator D will ask E to install a filter. This attack is rather harmless unless the attacker possesses a very large number of bots. In such cases it could represent a stateholding attack on E. However, so long as E knows which subnets are legitimate client addresses, this attack will not succeed, as E can simply decline to install the filters.

The fourth entry in the table represents an impossible situation, since the attacker would not be able to spoof an encapsulator from a non-legacy ISP.

In the fifth entry (figure 5.2) attacker C' sits behind legitimate encapsulator E2 and spoofs traffic from client C, which sits behind encapsulator E1. When D receives a request to block traffic "from C to S", it will wait for the next packet from C to S to arrive and note which encapsulator it came from. However, it is possible that this packet will come from the legitimate C through E1, while the packets that caused the detection mechanism to request the filter may have come through E2; if we asked E1 to install the filter, the attacker would succeed in denying service to C. In essence, D does not know whether to send the filtering request to E1 or E2. The simplest solution would be for E2 to block this spoofed traffic, as C is not a customer address for E2's ISP. However, this is not always possible, so we would like our solution to work even in the absence of ingress filtering. If every ISP deploying an encapsulator also deploys a decapsulator (a likely scenario) then, using the signed decapsulator routing tables, D would contact C's decapsulator requesting a list of C's encapsulators. As E2 is not on this list, D can safely request that E2 block all traffic from C. The same effect could also be achieved by disseminating the encapsulator list in the decapsulation routing table.

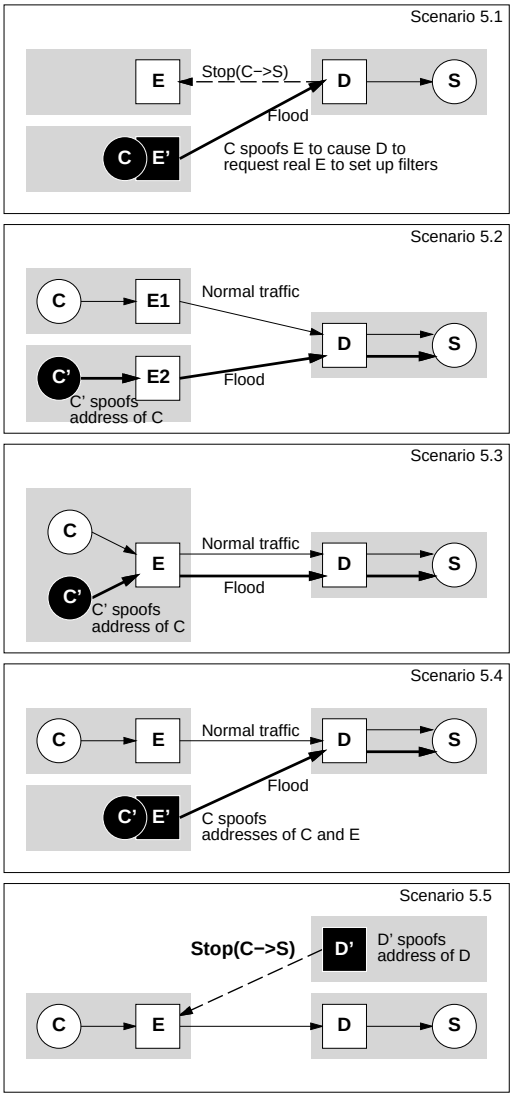


Figure 5: Potential abuse scenarios

In the sixth entry in the table (figure 5.3), both C and C' sit behind encapsulator E. Clearly, neither D nor S can distinguish between traffic from C and C'. S will request E to filter traffic from C (spoofed or not), so C' is successful in denying service to C, although he cannot deny service to other clients of S. In effect the problem has become one that is local to C's ISP, and there are a range of existing solutions available to tackle this, including enabling ingress filtering.

The seventh entry (figure 5.4) describes the most subtle attack to defend against. An attacker C' at a legacy ISP spoofs a traffic flood so that it seems to come from C, encapsulated by E. In this way, it could cause S to request a filter at E that would block legitimate traffic from C. If the packets claiming to come from C arrived with the evil bit cleared, we would know that the outer header is valid, a fact that can be propagated to S and the detection system it uses. As a result, S could distinguish between packets arriving on the real path $C \rightarrow E \rightarrow D$ and the spoofed path $C' \rightarrow E' \rightarrow D$ by observing the evil bit. S can now determine whether the unspoofed traffic via E is hostile (in which case it should request a filter) or only the traffic spoofing E is hostile (in which case it should take no action, and rely on prioritization to limit the effects of the attack). What if the path $C \rightarrow E \rightarrow D$ contained legacy ISPs, so that the evil bit is set on all packets? In this case, traffic from both paths would be indistinguishable. To remedy this, upon receiving the filtering request from S, D can provide E with a random number to use to mark subsequent packets (the encapsulation header can contain this). Now D can once again distinguish between the two paths, and all it needs do is to propagate this distinction downstream to S. D can use a diffserv code point to do this, so S's detector can again take the right action to ensure that no legitimate traffic from C is blocked.

The final entry in figure 3 presents an impossible scenario, since the attacker cannot spoof an encapsulator from a non-legacy ISP.

We will now proceed to discussing the direct attack cases described in figure 4. The first entry does not represent an attack. In the second and fourth entries the attacker sits on the path between the encapsulator and the decapsulator. These cases are hard to defend against, since the attacker can blackhole legitimate filtering requests. In the fourth case, unless requests are required to be digitally signed by the decapsulator, the attacker can also spoof them to shut down legitimate flows. Such digital signatures should be a mechanism of last resort, reserved only for the case where an on-path attacker is suspected, as they would greatly increase the CPU load on both the encapsulator and decapsulator. However, the encapsulator does already have the relevant key chain to validate such requests, as this is needed to validate prefix-to-decapsulator bindings. In reality though, this is a case we are not greatly concerned about - on path attackers should be rare (the normal bot infection techniques don't apply to routers) and in any event a compromised on-path router has so many other ways to deny service, including simply dropping the packets, that abuse of our mechanism is not likely to make the problem worse.

The third and only remaining entry presents an attack where attacker D' spoofs the address of decapsulator D and requests that encapsulator E install a filter blocking legitimate traffic from client C to server S (figure 5.5). This is trivially solved without digital signatures by requiring a three-way handshake, whereby a nonce sent from E to D must be echoed back to E before a filter request will be honored. In this way, even though D' can send a malicious filtering request, it will not be able to respond to E's nonce, since it is not on the path between E and D and will therefore not see it.

One final attack that does not rely on spoofing nor abusing the filtering request consists of flooding the link between the destination's ISP and that ISP's upstream provider. Since the prioritization of packets happens only at the destination's edge router, it may be possible to attack servers by flooding the link. Many ISPs may be able to prevent this by moving the place where diffserv categorization and prioritization occurs from E1 (in figure 2) to the upstream provider's edge router D1. As this categorization is static, it requires no active intervention on the part of the upstream ISP, but it does require their cooperation to enable it.

4. IMPLEMENTATION ISSUES

In this section we discuss in more detail how the encapsulators and decapsulators should be implemented, how the

filtering and evil bit might work, and provide some basic performance figures to show that off-the-shelf hardware is up to the task.

4.1 When to Encapsulate?

If packets from a client to a server are encapsulated, should the reverse path traffic also be encapsulated? If it is encapsulated, should the forward-path encapsulator serve as the reverse-path decapsulator?

The architecture does not require either, but there are advantages if both are true. If traffic is always bidirectional through the encapsulator, it can pro-actively limit malicious traffic, as described in [22]. Short of mandating symmetry, which seems excessively inflexible, we can still gain these benefits if the encapsulator knows which decapsulators will enforce symmetry; this can be advertised using the route distribution mechanism.

Should encapsulation be an always-on feature, or only enabled when under attack? The latter would essentially mean that the decapsulator publicly advertises being under attack, potentially inviting other attackers to cause further harm. Our performance numbers, discussed later, lead us to believe that the best solution is to always encapsulate. This also serves to publicly advertise which ISPs are being good network citizens and which are not.

4.2 Filtering Protocol and Filters

To handle communication between decapsulators and encapsulators we need a new signalling protocol. The primary purpose is to allow a decapsulator to request a filter from an encapsulator, but as we discussed in Section 3, there needs to be more to it than that.

The most important operation is the installation of filters. The protocol should allow the decapsulator to specify what the filter should match, the type of filter, what action to take when a packet matches the filter, and an expiration time.

So what should the format of the actual filters be? From an architectural point of view, a filter in an encapsulator can be anything that stops unwanted traffic with minimal side effects. It is in both sides' interest to install the most specific filters that actually suffice to block the traffic, so long as the encapsulator can maintain sufficient state. In the case of attacks which require a connection to be established, spoofing is not an issue, so specifying both source and destination IP addresses is desirable. In the case of spoofed attacks, the worst case is a flooding attack on a link, where the source addresses can be spoofed and the destination address can be any address beyond that link. In such a case, the decapsulator may be prepared to accept some collateral damage, and request that all traffic from an encapsulator to the decapsulator should be blocked. In between these extremes, we can envisage uses for various combinations of source address prefixes and destination address prefixes.

While the protocol should be flexible enough to accommodate different types of filter, a good starting point would be to initially support three types of filters: a prefix-based IP source address along with a specific destination address; a specific destination address with wildcard source address, whereby the encapsulator will filter all traffic going to that destination; and wildcard source and destination addresses, to address the link flooding attack above. We believe these would cover most of the requirements that might be encountered in the early stages of deployment, and that other policies can be implemented with reasonable performance in terms of these three.

A filtering request should also specify the action to take when a packet matches a filter: besides dropping the packet, the signalling protocol should be expressive enough to at least support rate-limiting based on a token bucket, even though such capabilities may not be available in all encapsulators.

Finally, filters should be soft-state - they should expire if not refreshed to avoid the filter being orphaned if the decapsulator loses state. Additional ways to expire filters, perhaps based on traffic levels, could be envisaged too, but they are not strictly required to satisfy our basic requirements.

The remaining functions of the signalling protocol, as discussed in Section 3 are:

- Nonce exchange. The signalling protocol cannot use TCP, as this would require too much connection state at a decapsulator under attack. Thus an encapsulator cannot blindly trust the IP address of a decapsulator that requests a filter, as shown in Figure 5.5. Instead it validates the decapsulator address by sending a random nonce, and requiring the decapsulator to return that nonce before it will install the filter.

- Request traffic marking. The decapsulator should be able to specify a random number for the encapsulator to include in the encapsulation header, so as to cope with Figure 5.4.

- Request encapsulator list. The decapsulator should be able to ask another decapsulator for the list of encapsulators corresponding to a specific client address handled by that decapsulator. This allows for defense against the scenario in Figure 5.2.

4.3 Evil Bit

Should the "evil bit" apply to all packets, or just to encapsulated ones? This impacts the type of filter needed at the border routers of the server's ISP. If the evil bit is applied only to encapsulated packets, router C1 in Figure 2 would have to first separate encapsulated packets from native ones, and then set the evil bit based on the peer the packet came from. Border router E1 would classify all non-encapsulated packets destined for the server to a lower priority traffic class, but it would also classify encapsulated packets as lower priority if they have the evil bit set. The alternative seems simpler: C1 sets the evil bit on *all* packets from ISP A, and E1 installs a single filter based solely on this bit.

Regarding where this bit would be implemented, the second solution requires it to be in the regular IP header, whereas the first is more flexible as it allows the evil bit to be in the encapsulation header, which can be largely of our own design. However, it is unclear whether current backbone routers have sufficient flexibility to set a bit in such a header, or classify based on it. On balance, the simplest solution may be to use a diffserv code point in the regular IP header to signal that the "evil bit" is set.

4.4 Routing Protocol Design

We need a routing protocol to distribute the binding between network prefixes and decapsulator addresses to encapsulators worldwide. But this is not a routing protocol in

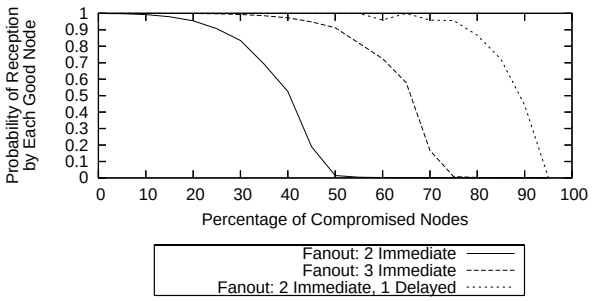


Figure 6: Robustness of routing infrastructure to compromised nodes.

the traditional sense, as it is agnostic to the path that the encapsulated traffic takes to get to the decapsulator. Thus these bindings are much more static than conventional routing information and moreover, the same encapsulation table should be distributed to every encapsulator worldwide. Even through these tables may be very large if our scheme is deployed globally, the actual size of the data is well within the capability of cheap commodity hardware. In terms of a distribution mechanism, the requirement is not dissimilar to that of NNTP[19] or BitTorrent[11], which are both capable of distributing much larger volumes of data relatively reliably to large numbers of hosts worldwide. Basically the requirement is simply for each domain to inject its own prefix-to-decapsulator bindings into the routing system, and for the routing system to then flood those bindings worldwide in a reliable manner.

Obviously it would be simple to hijack traffic for a site if an incorrect mapping could be distributed, so each binding needs to be secured using a digital signature. To do so, some form of public key infrastructure is needed to establish a trust hierarchy. One possibility is to use a hierarchy rooted at ICANN and delegated via the regional registries to ISPs. An alternative would be to use a multiply-rooted hierarchy anchored at the Tier-1 ISPs, delegated via Tier-2s, and so on along pre-existing provider-customer relationships. A third alternative would be to use a model similar to that used for SSL, using arbitrary certification authorities as trust anchors that are unrelated to the routing or address delegation hierarchies. All three are technically viable, but they have different political consequences; which would be best is really outside the scope of this paper. However, we note that the hierarchy rooted at the Tier-1 ISPs has the advantage that the trust chain matches the existing trust chain of the underlying routing system, making anomalies easier to detect. The disadvantage is that incremental deployment would be harder than the SSL-like model which does not require initial buy-in from the Tier-1 ISPs. In reality a good protocol design might permit any of these options, and the solution used might evolve as incremental deployment progresses.

Once we have signed bindings, we need a distribution mechanism for them. The problem is quite similar to that which we proposed for pushing DNS data out worldwide[16]. The basic idea is to build a peer-to-peer distribution mechanism, built from infrastructure nodes such as our encapsulators, perhaps supplemented by additional distribution nodes. Peerings between nodes need not follow the normal routing adjacencies; they can simply be configured between any two ISPs that have a business relationship, or who decide to peer just for this purpose. The only requirement is that the resulting network is connected. These configured peerings would then be supplemented by additional learned peerings which would make the overlay topology richer, so that the overall peer-to-peer network has small-world properties resulting in rapid distribution of data.

As the bindings are signed, every node receiving one should check the signature before passing it on to its peers. Thus there is no possibility for a malicious node to inject bad routing data unless the certificate chain itself has been compromised. Any node receiving bad data from a peer knows that that peer is malicious. Such networks are extremely robust to attack by insiders. The only real attack that is possible is for a malicious node to receive a message but refuses to pass it on. Figure 6, taken from [16], shows how robust such overlay networks are. In this simulation, each node passes on a message to two peers; three peers; or first two peers, then (after a short delay) one additional peer who has not yet received the message. This illustrates that the probability of correct delivery is very high right up until the majority of nodes within the network have been compromised.

The major cost in such a distribution network is to check signatures. Although this could be done in the encapsulators themselves, it is also possible to offload this checking to a fast server at each ISP. Such a server can also cache which signatures it has checked in the past, so only truly new bindings need to be checked, even after a reboot.

Our conclusion is that the distribution of the routes and the checking of signatures on these routes is not only feasible, but also relatively easy and very robust.

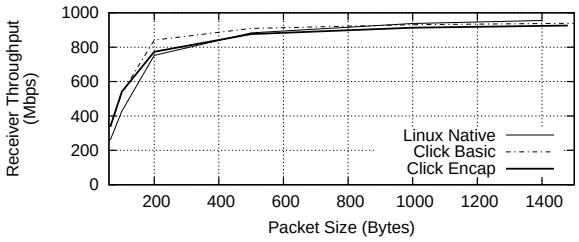
4.5 Performance

In the long run, we would expect encapsulation and filtering to become normal router or Ethernet switch functionality. Indeed, Juniper [18] already ships a hardware tunnel interface module capable of encapsulation at 10Gb/s; the missing part is mostly control software.

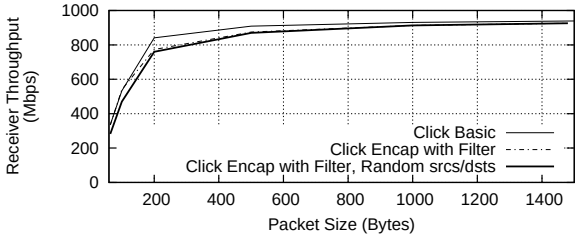
However, vendor support will be patchy early in the deployment process, and cheap solutions will always be required. To this end, we have begun building encapsulators and decapsulators based on very inexpensive 1U rackmount servers. The results below use two-year old Opteron 250 processors running at 2.4 GHz. Similar performance rackmount machines currently cost under $1,000. We use the Click [21] modular router software as the basis for the forwarding plane of our implementation.

When addressing DoS issues, performance obviously matters a great deal, so we have done extensive performance testing to demonstrate feasibility for these low-cost boxes. For reference purposes, Figure 7(a) shows how Click slightly outperforms Linux native forwarding; it also illustrates the clear relationship between the size of packets being forwarded and the throughput that is possible. Throughput is excellent (above 800Mb/s) for all packet sizes larger than 200 bytes. Only with very small packet sizes does performance degrade, with minimum sized packets achieving about 350 Mb/s, primarily due to limitations of the Ethernet hardware and/or driver.

What packet sizes is an encapsulator likely to see? Different measurement studies give different results [29, 7, 8] depending on where the monitor was located and when the



(a) Click outperforms Linux native forwarding. IP-in-IP encapsulation yields negligible performance hit



(b) Adding a hash-based filter on the fast path results in a negligible performance hit. Causing each packet to hit a different bucket yields similar results.

Figure 7: Encapsulator performance

study was performed. The packet size distribution appears to have strong modes at 40 bytes and 1500 bytes, with a range of values in between. Mean packet sizes appear to range from 200 to 600 bytes, which bodes well for our implementation. In the worst case, if all the traffic were 40 byte TCP Acks, we could only handle about 350Mb/s outgoing. However this would typically correspond to an incoming TCP data rate of upwards of 5Gb/s, so our basic forwarding performance seems more than adequate for many deployment scenarios.

Figure 7(a) further shows that adding IP-in-IP encapsulation to the forwarding path results in no significant degradation of performance. This curve was obtained using a small, static routing table, but a similar test using a large routing table of 200,000 routes (approximately the size of backbone BGP routing tables) yielded almost identical results.

Besides forwarding packets, the other main function of our encapsulators is filtering. Figure 7(b) shows performance figures for filtering. The encapsulator uses a hash-based filter, with filters consisting of source and destination IP address pairs. The "Click Basic" curve is for reference. The curve labeled "Click Encap with Filter" is generated with a filter table consisting of 2,000 filters, but with all the traffic destined for a single non-filtered address. This shows that adding such a filter bank to the forwarding path has an almost negligible impact on performance.

With hash functions, we always need to be careful that performance does not degrade unacceptably if we get hash collisions. A further test (not shown) which forced the forwarding path to always traverse a 20-node chain resulted in minor degradation of performance.

The final curve attempts to probe the worst case in terms of CPU cache locality. We loaded the hash table with 2,000 random filters and tweaked the hash function to emulate traffic to random destinations so that the filter lookup for each forwarded packet hit a different hash bucket. The 2,000 number was chosen to create a heavily loaded hash, since the hash table contained approximately 1,000 buckets. The concern was that CPU cache thrashing could seriously degrade performance, but even in such an extreme scenario the encapsulator performed rather well, with the curve closely resembling the previous curve where all traffic hit the same hash bucket.

What about the encapsulator's performance when traffic is actually being dropped by the filter? For this purpose we conducted one further test using two simultaneous sources, one whose traffic was dropped and another one whose traffic was forwarded. For packets larger than 200 bytes, the unfiltered traffic saw no degradation in performance; for minimum-sized packets, the encapsulator was able to process packets at a rate of over 390 Mb/s.

For the decapsulator, preliminary testing shows that Click can decapsulate packets without a significant performance hit, giving results similar to the "Click Encap" curve from figure 7(a). We will conduct further implementation and testing to investigate the performance of the decapsulator when it has to keep temporary state while locating the encapsulators that malicious flows are coming through, but we expect the results to be similar to those for filtering.

5. INCENTIVES FOR DEPLOYMENT

In a typical deployment scenario, an ISP might offer DoS protection as a premium service and a host (commonly a server) would be placed behind a decapsulator. Other ISPs with hosts wanting to connect to the server would then deploy encapsulators; in exchange, traffic from these ISPs to the server would be given priority at the border routers of the server's ISP at all times.

Server-side ISPs

Since a server will pay a premium for protection against DoS attacks, server-side ISPs such as ISP E in figure 2 have a clear incentive for deployment. Such deployment would be neither difficult nor expensive. In most cases decapsulation can be performed at sufficient speed for a busy server in an inexpensive 1U rack-mount server running our software. A large data center may need multiple such decapsulators, but their cost is very small compared to other data center costs. ISP E also needs to obtain a security certificate from someone higher up in the trust hierarchy, and uses this to sign the mapping between its servers' address prefixes and the decapsulator address. The decapsulator machine then needs to be configured with at least one peer (although this does not need to be at an immediately neighboring ISP), so it can advertise the network prefix-to-decapsulator bindings to encapsulators in the rest of the Internet. Finally, E installs classifiers at its edge routers (in this case E1) so that incoming packets that do not have the evil bit set get sent to a higher priority diffserv class.

Client-side ISPs

Customers wanting to connect to the protected servers will pressure their ISPs to deploy encapsulators. Having a legacy provider will mean that a client receives lower priority than one connected through a non-legacy ISP. Not only may delays be longer during normal operation, but also little or no service will be given if the server is attacked. However, we are not convinced that such customer pressure will be sufficient, especially in the early stages of deployment, to

entice ISPs to deploy additional equipment. The costs may be small, but they are there nonetheless.

Another incentive for ISPs deploying encapsulators is a potential for a reduction in tech-support costs. Encapsulators should have no false positives - if a receiver does not want traffic, only then it is shutdown. The one case where there might be false positives is Scenario 5.3 in Figure 5, and this is not an issue for the many ISPs that perform ingress filtering. In the absence of false positives, encapsulation filters automatically isolate the bad behavior of a compromised client host, and should require less human intervention that current, mostly manual, mechanisms for dealing with compromised hosts. The hope is that deploying an encapsulator allows the ISP to deal with a bot being used in a DoS attack at their leisure, reducing operational costs.

This client-side deployment is even simpler than on the server-side: configure a fast PC to act as an encapsulator, configure it to receive decapsulation routes from one or more peers, and have it clear the "evil bit" in packets it encapsulates. Should the encapsulation infrastructure gain support from ISPs, it is likely that this functionality would quickly become a normal feature of access routers, and so the deployment cost would be close to zero.

Transit ISPs

There probably are no *pure* transit ISPs (most have direct customers), but the deployment for any such ISP would be trivial: simply configure border routers to set the evil bit for packets that come from legacy peers. If the evil bit is implemented as a diffserv codepoint, then this can be done in all current backbone routers without requiring any upgrades.

5.1 Early Deployment

It is easy to see that if such an architecture started to be deployed successfully, the incentives are such that increasing deployment increases the incentive to deploy. The question then is how to persuade potential early adopters to deploy.

Any site that is attacked has incentives to deploy decapsulators, but only if they allow the attack to be reduced. However this requires that sufficient traffic traverses an encapsulator, and there is little incentive for that to happen until many decapsulators are deployed. One option would be for a large ISP or Internet Exchange Point to offer an encapsulation service. The site under attack could then subscribe to this service, and all their incoming unencapsulated traffic would be routed there to be encapsulated. This would bootstrap the deployment of decapsulators and then, sometime later, there would be incentive to deploy encapsulators directly at customer-facing ISPs.

6. RELATED WORK

Various techniques have been proposed in the literature to counter DoS attacks. Approaches based on capabilities [2, 35, 24] generally rely on clients having to ask the server for permission to send packets. If the server decides to allow the connection, it replies with a capability token, which the client includes in subsequent packets and which the network polices. Such schemes, while technically feasible, face daunting initial deployment issues, since they depend on changes to the end clients as well as the network. Other solutions are based around constructing overlays to protect victims [20, 23, 14], but these tend to operate above the network layer, assuming the presence of other mechanisms to protect it. Pushback [17] relies on identifying misbehaving aggregates and iteratively asking upstream routers to filter them. Unfortunately, this scheme relies on paths where every router has implemented Pushback and is effective only when the routers are close to the sources of the traffic.

In [4], the authors employ a solution that has a few similarities with this paper, but its success depends upon the core of the network (at least until full deployment is achieved) to perform the actual filtering. In addition, the solution relies on a variant of IP route record to mark where packets came from, so that intermediate points in the network would need modification. In contrast to this, the architecture presented in our paper does not require participation from middle-of-the-network nodes.

Firebreak [13] also suggests placing control points called firebreaks near the edges so that traffic can be stopped near its sources. Unfortunately, it suffers from several shortcomings. First, the firebreaks use IP anycast to advertise all of the addresses of protected targets. Not only is large-scale IP anycast not well understood nor widely deployed, but advertising in this fashion is likely to present scaling problems. Second, the paper points out that, thanks to IP anycast, traffic from clients whose ISPs do not have firebreaks will be redirected to an ISP who does have firebreaks. However, it is quite unclear what incentive this latter ISP has for accepting this traffic nor why should it deploy a firebreak in the first place if there will be the possibility of having to deal with another ISP's traffic. Finally, the scheme suggests stopping traffic from clients connected to legacy routers from going directly to the target (without first traversing a firebreak) by only advertising the route to deployed ISPs. While this will indeed prevent a legacy router from forwarding packets to a protected target, it assumes that there will be no legacy ISP in the path between deployed ISPs and the target ISP. If there were, the route would not be advertised to the legacy ISP and any deployed ISP behind it would be oblivious to the fact that a route exists. This presents a significant problem, especially during initial deployment.

In [12], the use of Diffserv is suggested to distinguish traffic from well-managed sites that protect their hosts against penetration. Our scheme also uses diffserv, but to indicate a site will respond to filtering requests.

A more imaginative and radical approach to combatting DoS attacks [33] consists of increasing the ratio of good traffic to bad traffic arriving at a victim by asking normal clients to send at a higher rate. Despite its merits, operators will likely be reluctant to deploy a DoS protection scheme that increases traffic during an attack. Commercial solutions generally involve buying special boxes [3, 25, 9] or redirecting traffic through scrubbing centers during attack [1, 27], both costly options.

CenterTrack [31] proposed solving the problem of tracing sources with an overlay network based on IP-layer tunnels, linking edge routers to a central tracking router or small network of tracking routers. This would be an obvious target for DoS attacks, so it is unclear how effective this approach would be, nor is it clear what initial deployment incentives exist under this scheme. Traceback mechanisms [6, 30, 34, 28] consist of marking a small percentage of packets as they travel from sources to server; the server can then use this information to reconstruct the path they travelled. MPLS can also be employed for this task, though deploying it across AS boundaries may present an insurmountable obstacle.

7. CONCLUSIONS

We have presented a novel and conceptually simple architectural solution to the problem of large, distributed Denial-of-Service attacks. We finally provide a way for a receiver to simply ask the network to stop sending unwanted traffic. Great care must be taken with such mechanisms to avoid opening up new channels for attack, and we believe we have addressed these issues. We have shown that our mechanisms are largely robust to spoofing via both direct and indirect attacks, and have exhaustively enumerated potential attack vectors, showing how they can be overcome with minor extensions to the architecture. Thus we believe our solution is safe, even in incrementally deployed scenarios, where we cannot control all the players.

While our end-network mechanisms should eventually become standard functionality on edge routers, we have shown that the mechanisms can also be implemented on current low-cost off-the-shelf hardware, running efficiently even under heavy load. In the network core, our scheme makes use of already-deployed routers without requiring any additional functionality from them. No changes are needed to any end systems.

Further, our architecture does route distribution using a peer-to-peer mechanism that is both simple and extremely robust to attack. This imposes no additional burden on the existing inter-domain routing protocols.

Finally, the proposed architecture is not only incrementally deployable from a technical point of view, but also provides incentives for both ISPs of victims and those of sources or attackers.

8. REFERENCES

[1] S. Agarwal, T. Dawson, and C. Tryfonas. DDoS mitigation via regional cleaning centers. Technical report, Sprint ATL Research Report RR04-ATL-013177, August 2003.

[2] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proc. ACM SIGCOMM 2nd Workshop on Hot Topics in Networks*, November 2003.

[3] Arbor Networks. Arbor peakflow: Network security management and intrusion detection.

[4] K. Argyraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Usenix Annual Techical Conference*, April 2005.

[5] S. Bellovin. The Security Flag in the IPv4 Header. http://www.ietf.org/rfc/rfc3514.txt, April 2003.

[6] S. Bellovin, M. Leech, and T. Taylor. The ICMP traceback message, October 2001.

[7] CAIDA: Cooperative Association for Internet Data Analysis. Packet Sizes and Sequencing. http://learn.caida.org, 1998.

[8] CAIDA: Cooperative Association for Internet Data Analysis. Packet Length Distributions. http://www.caida.org, 2000.

[9] Cisco Systems. Cisco Guard DDoS mitigation appliances - products and services.

[10] CNET. Bot herders may have controlled 1.5 million pcs. http://news.com.com/, October 2005.

[11] Brad Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, May 2003.

[12] S. Crocker. Protecting the internet from distributed denial-of-service attacks: a proposal. *Proceedings of the IEEE*, 92(9), 2004.

[13] Paul Francis. Firebreak: An IP Perimeter Defense Architecture. 2004.

[14] R. Govindan, A. Hussain, R. Lindell, J. Mehringer, and C. Papadopoulos. COSSACK: Coordinated suppression of simultaneous attacks. In *Proc. IEEE DISCEX*, April 2003.

[15] A. Greenhalgh, M. Handley, and F. Huici. Using Routing and Tunneling to Combat DoS Attacks. In *Proceedings of the 2005 Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2005.

[16] Mark Handley and Adam Greenhalgh. The case for pushing DNS. In *Proc. ACM HotNets IV*, November 2005.

[17] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. Network and Distributed System Security Symposium, San Diego*. ISOC, Reston, VA., February 2002.

[18] Juniper Networks. Tunnel services PIC datasheet. http://www.juniper.net/products/modules/tunnel_pic.html.

[19] B. Kantor and P. Lapsley. Network news transfer protocol, a proposed standard for the stream-based transmission of news. RFC 977, IETF, February 1986.

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

[21] E. Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, August 2000.

[22] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using Packet Symmetry to Curtail Malicious Traffic. In *Proc. ACM HotNets 2005*, 2005.

[23] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP packet flooding attacks. In *Proc. ACM HotNets 2003*.

[24] W. Lee and J. Xu. Sustaining availability of web services under distributed denial of service attacks. *IEEE Transactions on Computers*, 52(3):195–208, 2003.

[25] Mazu Networks. Network behavior analysis for the perimeter.

[26] Network World. Extortion via DDoS on the rise. http://www.networkworld.com/, May 2005.

[27] Prolexic Technologies. Prolexic Technologies Home Page.

[28] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. *SIGCOMM Comput. Commun. Rev.*, 30(4):295–306, 2000.

[29] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet Packet Size Distributions: Some Observations. http://netweb.usc.edu/~rsinha/pkt-sizes, 2005.

[30] A. Snoeren, C. Partridge, A. Sanchez, Jones C., F. Tchakountio, S. Kent, and T. Strayer. Hash-based IP traceback. In *Proc. ACM SIGCOMM 2001*.

[31] R. Stone. CenterTrack: An IP overlay network for tracking DoS floods. In *Proc. 9th USENIX Security Symposium*, August 2000.

[32] Symantec Corporation. Internet Security Threat Report Volume IX. http://www.symantec.com/enterprise/threatreport/index.jsp.

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

[34] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 93. IEEE Computer Society, 2003.

[35] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *Proc. IEEE Security and Privacy Symposium*, May 2004.