RN/05/25
February 5, 2007

Research
Note

Deployable Filtering Architectures Against Denial-of-Service Attacks

Felipe Huici

Telephone: +44 (0)20 7679 0401
Fax: +44 (0)20 7679 1397
Electronic Mail: f.huici@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/staff/f.huici/

Abstract

The Denial-of-Service attack problem continues to grow despite the fact that many solutions have been proposed in the research literature and the commercial field. The problem is not that the solutions have not had technical merit, but that they have either not scaled to architectural levels or they have had serious deployment issues. As a result, there is still no comprehensive, architectural solution to the problem of large, distributed DoS attacks. This report presents three incrementally-deployable architectures against DoS that provide clear incentives for early adopters while improving in effectiveness as deployment progresses. The report also discusses completed research as well as a future experiment plan to show that the architectures would be able to cope with large attacks even when their components are built using cheap, off-the-shelf hardware.

Keywords

Internet Security Denial-of-Service Attacks Deployable Filtering Architecture Defense

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

Contents

1 Introduction

From its inception, the Internet was designed to efficiently transfer packets and to be flexible enough so that it could accommodate future applications. Dating back to its early days as the ARPA net, trust was placed on users not to abuse the network, and so security was not, until recently, of great concern. The Internet has evolved significantly since those early days, and people have become used to depending on it for a large variety of activities, from long distance voice and video communications to online banking, e-government and online shopping. However, this increase in activity, specially regarding economic transactions, has made the Internet attractive to the criminal sector, and so the assumption that users can be trusted no longer holds. While it would be impractical to completely revamp the current Internet to make it secure against attack, it is possible to design incrementally deployable solutions so that the Internet can continue to grow growth unhindered.

1.1 Definitions

In a Denial-of-Service (DoS) attack, an attacker tries to exhaust one or more resources from a victim so that it cannot service its legitimate clients. Resources that can be attacked range from a server's CPU and memory to its network link. Since it would be difficult for an attacker to exhaust a server's resources using only his system, he can use worms or other malicious software to take over the systems of other unsuspecting hosts, turning them into "bots" that do his bidding. Attacks involving such bots are called Distributed Denial-of-Service attacks (DDoS) and can be quite harmful depending on the size of the botnet used to perpetrate them. DDoS attacks are particularly effective at performing flooding attacks, in which the victim's network link becomes saturated by attack traffic. One variation on the DoS theme is called a reflection attack, in which the attacker sends a packet to a server but spoofs the source by putting the target's address instead of his in the header; the server then replies, but this traffic ends up going to the target rather than the attacker. By itself, this attack is not very effective, so it is usually coupled with an amplification attack, in which the server used sends replies that are substantially bigger than the requests that triggered them.

1.2 Problem Description and Motivation

Despite serious under-reporting, the number and size of flooding 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[61], Symantec reported an average of 6,000 attacks per day for the first half of 2006. Botnet size is also increasing: reports from 2005 indicate that Dutch bot-herders may have controlled as many as 1.5 million hosts[19]. And whereas early DoS attacks were perpetrated by script kiddies, recent years have seen a shift towards organized crime and extortion[50].

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 for several reasons. First, 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. Second, 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. Third, source address spoofing complicates matters further, making the task of tracing the sources of malicious traffic difficult. In addition, any DoS defense mechanism that does not take spoofing 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. Fourth, any proposed solution must be flexible enough to allow the development and deployment of new applications, one of the main reasons behind the Internet's amazing growth. Finally, any practical solution needs to provide clear incentives for initial deployment or it will fail regardless of technical merit.

It is clear that the DoS problem will not disappear but rather continue to grow. Steps must be taken to at least mitigate these attacks so that the Internet can continue to expand. While DoS attacks present a difficult problem, they are not unsolvable. The next section discusses in detail the solutions proposed by the research community, focusing on why most if not all of these have seen little real deployment; commercial solutions to the problem are also mentioned. The rest of the report then centers on the contribution that the thesis will make, including a description of the architectural solutions, the research progress made so far, and the future experimentation plan.

1.3 Scope

The contribution presented in this report will focus on DoS attack mitigation, not prevention. While attack prevention measures such as secure operating systems, anti-virus software and worm prevention form an important part of the fight against DoS, they are considered outside the scope of the report's contribution.

In addition, the work relies on an Intrusion Detection System to determine when an attack is happening and to request the necessary filters. The actual configuration of the IDS used to detect DoS attacks is outside the scope of the work presented in this report since it does not constitute research: currently operators can distinguish good sources from bad without too much trouble.

Finally, any BGP route distribution issues, specially as it pertains to section 3.1, will be largely ignored, since they do not constitute research.

2 Related Work

Despite several years of effort both in the research community and in the commercial field, Denial-of-Service attacks continue to grow in frequency, size and effectiveness. This section discusses in detail why the proposed solutions have failed to deal with the problem.

2.1 Research Community

While Denial-of-Service attacks have gained considerable notoriety recently, they have existed since the late 1990s. As a result, research in the field has produced numerous and varied schemes aimed at solving or at least mitigating the effects of these attacks, specially those of large, distributed DoS attacks. The discussion that follows is organized using a taxonomy of complete solutions or components that simplify the problem: while some of the proposed approaches do not cleanly fall into one of these categories, many do, and so this classification provides a useful organizational aid.

2.1.1 Traffic Policing and Filtering

Once a large attack has aggregated at a target, there is little the victim can do but drop the malicious packets. For sufficiently large attacks, the processing power needed to do so may exceed the capabilities of the target. Traffic policing aims to alleviate the burden on the victim, either by having network routers filter or rate-limit attack traffic before it becomes fully aggregated, or by preventing this traffic from entering the network in the first place.

Pushback [47] consists of a local mechanism designed to detect and control aggregates at a single router. The router begins by deciding whether or not it is congested by periodically monitoring each queue's packet drop rate to see if it exceeds a certain threshold (this threshold is policy-specific). An aggregate is defined as a collection of packets from one or more flows that have some property in common, like TCP SYN packets. When serious congestion is detected, the router attempts to identify the responsible aggregates by applying a clustering algorithm. If this algorithm fails to find a well-defined aggregate, no action is taken. Once an aggregate is identified, its rate is limited so that the combination of its new rate with the rates of all other (well-behaving) flows arriving at the router does not exceed a certain configured value. The congested router will then send a pushback message asking the immediate upstream routers that send the bulk of the traffic for an aggregate to rate-limit it; routers receiving this message can then recursively propagate it upstream. These pushback messages are then sent periodically as long as the attack persists. The downstream router now encounters the problem of deciding when to stop the request for pushback, since it cannot tell whether the high-bandwidth aggregate has stopped because of the upstream filtering or because the attack has ceased. To resolve this, the upstream routers send out feedback messages reporting how much traffic from an aggregate they are still seeing.

Another approach relies on server-centric router throttles [65]. An overloaded server installs appropriate throttling rates at distributed routing points such that, globally, server S exports its full capacity to the network, but no more. S wants to keep its load between two thresholds. If the load is too high S will increase the throttles and if the load is too low it will decrease them. The set of internal routers is denoted by R, and each keeps only a few bytes per victim: the victim's IP address and the throttle value. While this increases linearly with the number of victims, this should not present a scalability problem as DDoS attacks are the exception rather that the norm. The throttles are installed at a subset R(k) of the R internal routers, where k is the number of hops between a router and the server S, essentially forming a perimeter around S. Throttles are also installed at routers that are less than k hops away but directly connected to S. When S's load surpasses the upper threshold, S advertises a throttle increase to R(k), and each router in this set multiplies the current rate by a certain factor. Conversely, when S's load falls below the lower threshold, S advertises a throttle decrease to R(k), and each router in the set adds a factor to the current rate. This multiplicative decrease/additive increase mimics TCP's congestion control mechanism.

In D-WARD [48], routers at edge networks actively monitor TCP, ICMP and UDP flows and perform periodic comparison with normal flow models. The attack response is essentially a modification of TCP's congestion control mechanism. The system will apply fast exponential decrease to the rate limit of non-responsive flows; it will enforce slow recovery of rate-limited flows for a certain period; and it will allow fast recovery once a flow has proved that it is well-behaved.

Argyraki et al. [7] base their approach on the observation that while the resources of a victim's router are limited, the Internet routers have enough combined filtering capability to stop even very large DDoS attacks. Active Internet Traffic Filtering (AITF) assumes that an end host can identify when it is being attacked and can single out, with high probability, a border router close to the source of the attack traffic. The victim contacts its AITF-enabled gateway which installs a temporary filter on its behalf. The victim's gateway then contacts the attacker's gateway and asks it to filter the malicious flow. The attacker's gateway in turn asks the attacker to cease the flow or risk being disconnected. Should this gateway refuse to collaborate with the victim's gateway (perhaps because it does not have AITF deployed), the latter will either try to find an AITF-enabled gateway on the same path but closer to itself or decide to filter out the traffic itself. The mechanism resorts to this second option in the worst case when no AITF-enabled gateways exist.

Firebreak [26] forces packets to a protected host to travel through special routers called firebreaks that are deployed near sources of traffic; these firebreaks then tunnel the packets to the protected host. To deal with incremental deployment, all firebreaks advertise the complete set of firebreak addresses into the routing fabric, using IP anycast. As a result, packets addressed to any firebreak address are routed to a nearby firebreak.

Despite their merits, these solutions have their shortcomings. Pushback, for instance, requires a path of deployed routers between a victim and the sources of the attack, and unlikely scenario during initial deployment. Further, the routers determine that a flow causing congestion is malicious, which may not be the case. The server-centric router throttles approach suffers from a similar initial deployment issue, in that it assumes the availability of a perimeter of deployed routers. D-WARD has the advantage of placing control routers near potential sources of attack, but it is not clear what incentive a *source* ISP would have to deploy such a mechanism, nor is it trivial to determine whether a flow is behaving or not, specially for large, distributed attacks. AITF relies on middle-of-the-network routers (at least during initial deployment) to perform the actual filtering, and depends upon a variant of IP route record to determine where packets came from; as such, it faces severe deployment issues. Firebreak 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 it should deploy a firebreak in the first place if there is 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.

2.1.2 Service Differentiation and Capabilities

A characteristic of hosts on the Internet is that they do not decide which packets to receive. Schemes in this section aim to empower the receiver so that it will be in charge of deciding who is allowed to send it packets. In general, a client requests permission to send packets, and, if the server is not overloaded and decides that the client is legitimate, the server will respond with a capability token. The client then includes this token in all its subsequent packages and routers along the path to the server verify the validity of the token.

In the Stateless Internet Flow Filter (SIFF) approach [63], the process begins by dividing Internet traffic into privileged and unprivileged, where the former is always given priority over the latter. To set up a privileged channel a client begins by sending an unprivileged explorer (EXP) packet. Each router along the path calculates a z-bit long marking from a keyed hash function with four inputs: the IP addresses of the source and destination, that of the router's incoming interface and that of the previous hop's outgoing interface. The router then left-shifts all previous markings in the capability field of the EXP packet and adds its own marking to the least significant bits. Upon receipt of this EXP packet the server has to set up its half of the privileged channel, so it sends its own EXP packet back to the client, this time writing the markings that the routers provided into the optional capability reply field. Once this second packet arrives, the client copies the value of the reply field into the capability field of a privileged data (DTA) packet, and begins sending data to the server. Each router on the path now verifies that its marking matches that of the low-order bits in the packet's field: if it does not, the packet is dropped; if it does, the marking bits are moved to the right-end of the field and the packet is forwarded. A DTA packet will only arrive at the server if it passes the checks of all routers along the path.

Another form of capability mechanisms uses tokens [6]. Before being allowed to send packets, sources are required to obtain such tokens from the destination. Request-to-Send (RTS) servers distributed along the path between the source and destination aid sources in obtaining such tokens; these are coupled with Verification Points (VP) that enforce the capabilities specified by the tokens. A source contacts its local RTS server which in turn propagates the request all the way to the destination's RTS server, creating state in all the RTS servers in between. The destination then sends the token back to the source in a symmetric path with regards to the RTS chain, and the source includes the given token in all subsequent packets. Finally, the VP ensures that the source respects the given capability.

In [44], legitimate traffic is separated from attack traffic and protected during an attack. This process is done through cryptographic techniques, so that only the first connection packet will suffer a delay, while all the rest will be privileged. When a server is overloaded, each client (malicious or otherwise) is given a quota for high-priority traffic; when this quota is exceeded, the user will be considered an attacker and it will be rate-limited by a perimeter of routers forming a line of defense.

The approaches presented in this section face important deployment issues: they require changes to both the end-hosts so that they will know to request permission to send and to the network so that it will police traffic travelling to a server. Further, these mechanisms do not provide a complete solution, since it is still possible to DoS the capability request channel.

2.1.3 Overlays

Solutions in this category, as their name indicates, consist of deploying a set of special nodes forming an overlay network protecting potential victims. More specifically, overlays aim to distribute the points of access to victims among a set of outer nodes: all packets must pass through one of these nodes before they are allowed into the overlay network. Should a particular flow misbehave, its corresponding access node can limit its rate or drop it. While any one of these nodes can be singled out and attacked, legitimate users can continue receiving service by connecting to another of these nodes.

In i3 [3, 2, 1], the authors argue that hosts would be much better placed to effectively respond to overload if they had control over which packets were dropped. For instance, a host running multiple services may give higher priority to some services than others. As a result, hosts, not the network, should be given control to respond to packet floods and overload. The use of the i3 overlay provides a method of identifying hosts without using IP addresses. Sources send packets to a logical *identifier* and receivers express interest in packets by inserting a *trigger* into the network. Packets are then of the form $(id, data)$ and triggers are of the form $(id, addr)$, where *addr* can be either an IP address or an identifier. Triggers, then, relay packets either to a final destination or to other triggers. This added level of indirection allows a host to stop receiving packets from a particular trigger by simply removing it.

In Secure Overlay Services (SOS) [39], a target or destination forming part of a Chord-based overlay [9] notifies other nodes called secret servlets of its presence, and has a filtering router drop any traffic that does not come from them. These secret servlets, in turn, compute, using several hash functions with the destination's IP address as input, the identity of other overlay nodes called beacons. The servlets then let the beacons know of their existence. When a client wants to send a message to the destination, it must first be authenticated by a Secure Overlay Access Point (SOAP). Upon success, the SOAP will securely forward the message to a beacon, who will in turn pass it to the secret servlet, and from there through the filtering router finally arriving at the destination. The levels of indirection and the self-healing nature of the Chord protocol make this architecture robust against any of its components failing or being compromised.

In COSSACK [31], each egress point has a component called a watchdog that monitors its own network and shares information with watchdog peers. Watchdogs have two main functions: detecting the onset of an attack and informing other watchdogs of the attack signature over a multicast channel. Watchdogs at attackers' networks then perform attack-packet filtering.

Centertrack [59] is an overlay that relies on using IP tunnels to re-route interesting packets directly from edge routers to special tracking routers. In this way a tracking router can determine whether a packet is malicious or not, and, if it is, easily follow the tunnel back to the ingress point that the packet took into the overlay.

While overlays have the important advantage of being deployable without affecting the existing network infrastructure, they do not provide a full solution to the DoS problem since they tend to operate above the network layer, assuming the presence of other mechanisms to protect it. Centertrack does operate at the network layer, but uses a central tracking router or small network of tracking routers, providing an obvious target for a DoS attack.

2.1.4 Source Routing

While source routing is not directly intended as a scheme against DoS attacks, its very nature makes it valuable against such attacks. The requirement that sources specify the path a packet should follow to the destination, coupled with the fact that routers along the way verify the path's validity, guarantees that sources cannot spoof their addresses. The availability of the entire path at the destination not only allows easy trace back of the source, but also enables effortless installation of filters close to the attacker.

Under the *New Internet Routing Architecture* [64], users specify the inter-domain path that a packet should follow, empowering them to select more optimal routes to destinations while fostering competition among ISPs. Another benefit of this architecture is that it forces sources to specify paths to destination that routers along the way can verify. As a result, source addresses cannot be spoofed, and malicious flows can be easily tracked back to their sources. The general process begins with each host knowing its route to the core of the Internet. When a source S wishes to send to a destination D, it requests that D send it its topology information (D's route to the core). Upon receiving this, S will combine its route to the core with D's route to the core to form a complete path from source to destination.

The *Wide-Area Relay Protocol* (WRAP) is a proposed new protocol running over IP and is a modified version of IP's *Loose Source and Record Route* (LSRR). WRAP involves the use of four header fields, two in the IP header and two in the WRAP header. IP's source address header is modified to always contain the IP address of the source-host or last relay (a WRAP-enabled router) that has forwarded the packet. Likewise, IP's destination header has the IP address of the destination-host or the next relay. WRAP's *reverse path* (RP) field contains a list of IP addresses of relays already traversed while its *forward path* has a list of IP addresses of relays yet to be traversed. End-hosts (or the edge-system) are in charge of computing and monitoring multiple paths to each potential destination in order to include a valid path in the FP field of the WRAP header. Relays then forward the packet according to the given path, as long as it is valid and ISPs along the path have agreements to carry the traffic. With this mechanism in place an end-host can mitigate a DDoS attacks by identifying the paths whose flows are malicious. If the attack is too large, however, the end-host or network will quickly become overwhelmed and will not be able to filter all of the malicious traffic. The authors suggest using a solution such as AITF to propagate filtering requests to upstream routers who can drop these packets closer to their sources.

The clear advantage of source routing in combating DoS attacks is that it provides an enabling mechanism for easy deployment of filters near the attack's source as well as easy trace back to the source, rendering packet marking schemes unnecessary. However, source routing requires wide deployment among routers in order to function. Further, NIRA introduces a new addressing scheme that would require significant changes not only to routers and providers but also to end-hosts. If the numbers of addresses per host is high, NIRA may also prove prohibitive to resource-limited devices. While WRAP indicates that no changes to hosts are needed if a "translating" function is added to edge routers, this scheme also demands changes in routers as well as significant packet remarking upon forwarding.

2.1.5 Proof-of-Work

Denial-of-Service attacks are possible because sources are allowed to send packets even if destinations do not want to receive them. Proof-of-work schemes rely on the destination or server issuing a computationally-intensive puzzle to its clients; it will then only accept packets from those clients that provide the correct answer to the puzzle. Since, in general, attackers send packets at a much higher rate than legitimate users, they will incur serious computational penalties while legitimate users remain largely unaffected.

To provide coverage across all applications and protocols, client puzzles must be placed at the layer that is common to all of them: the network layer. Feng et al. [25] introduce a new protocol that relies on ICMP source quench messages to issue puzzles and on IP options to reply with solutions. The authors consider four different types of puzzles to use in the protocol. Time-lock puzzles require clients to perform repeated squaring operations. While they provide an exact and fixed amount of computational work, it is expensive for the server to generate them. Hash-reversal puzzles forces clients to reverse a one-way hash given the original random input with n bits erased. The disadvantage of this approach is that the granularity of the puzzles' difficulty is coarse: an n bit puzzle is twice as hard as an n-1 bit puzzle. Multiple hash-reversal puzzles improve this granularity, but issuing an increasing number of puzzles to achieve this becomes prohibitive. Finally, hint-based hash-reversal puzzles provide the result of the hash along with a hint whose accuracy can be varied to regulate the puzzle's difficulty. It is this last type that is used in the implementation of the protocol.

Felten et al. [24] introduce a new method of outsourcing puzzles utilizing a distributed entity called a bastion that can be shared among several servers. Because of its distributed nature and its good scalability, the outsourcing mechanism is robust against DoS attacks. The authors present three methods for puzzle construction, time-lock, hash-reversal and "D-H" (Diffie-Hellman), focusing on this last one since it allows client to solve puzzles off-line. The current implementation is directed at TCP and it is left as future work to see if the techniques are efficient enough to work for lower-layer protocols such as IP.

In [54] clients are allowed to bid for service by computing puzzles with difficulty levels of their own choosing. The server under attack allocates its limited resources to requests carrying the highest priorities. Like the previous approach, this paper describes a TCP implementation, and leaves the possibility of a network-layer implementation as future work.

All of these solutions aim to diminish the imbalance between a server's resources and the combined resources of its clients. The fact that servers can generate and verify puzzles very efficiently while clients need to perform expensive calculations shifts some of the control over to the server. These schemes, however, suffer from several problems. First, if an attack is sufficiently distributed, each source of malicious packets (typically a zombie) will only have to spend a relatively insignificant amount of time solving the puzzle. Worse still, when zombies calculate solutions to puzzles on behalf of the attacker, it is the legitimate but unwitting owner of the system that incurs the performance degradation resulting from these. Further, puzzles do not take into account the possibility that several orders of magnitude in computing power may exist among a server's clients.

2.1.6 Anti-spoofing

As their name suggests, these schemes have the ultimate aim of distinguishing between genuine packets and those whose source address does not reflect where they actually originated. While they do not solve the DoS problem, they simplify the problem by allowing a victim to know where the attack is coming from. Anti-spoofing solutions are quite varied, differing even in where the filtering takes place. The following is a sample of some of these approaches.

Hop-count filtering [35] makes the observation that most spoofed IP packets do not carry hop-count values that are consistent with the IP address being spoofed. To determine whether or not a packet is spoofed, the hop-count is first computed. The system keeps a table containing a mapping of IP addresses to hop-counts, and if the hop-count of the incoming packet does not match the value found in the table for the IP address of the packet, then the packet is deemed spoofed. To populate this table initially, a node (presumably a server) collects traces of its clients that contain both IP addresses and the corresponding Time-to-Live (TTL) values. The hop-count value cannot be directly obtained from the IP header since no field in it contains that information. It can, however, be inferred from the TTL field: the hop-count is equal to the difference between the initial TTL value at the source and the final TTL value at the destination. The determination of the initial TTL value is based around the observation that while operating systems do not use a common value to initialize this field, the values used by most of them are only a few.

Another approach is called Route-Based Distributed Packet Filtering (DPF) [51] and it relies on placing anti-spoofing routers in the middle of the network (or adding functionality to existing routers). These routers must have knowledge of what IP address ranges are valid, and so the actual implementation of DPF would require at least an augmentation to BGP or the introduction of a new protocol that would disseminate source reachability information rather than destination reachability information.

Mirkovic et al. propose the *Source Address Validity Enforcement Protocol* (SAVE) [46] in which routers periodically exchange SAVE messages. These messages propagate valid source address information from the source location to all destinations, allowing each router along the way to build a table that associates each incoming interface of the router with a set of valid source address blocks. An incoming packet whose incoming interface and source IP address do not match any of the table entries of a router is dropped.

Approaches in the network, such as DPF and SAVE, have the advantage of stopping spoofed traffic near the source or before it has had a chance to aggregate, at the cost of difficulty of deployment. DPF requires that the underlying inter-domain routing protocol disseminates source reachability information. As a result, it faces the significant hurdle of either augmenting BGP or creating a separate routing protocol. Neither one of these suggestions seems feasible on the Internet. Further, a DPF node's ability to filter traffic may be hindered by the existence of multiple paths to a source. Also, one would have to evaluate whether the added overhead in keeping source reachability information in addition to destination information is justified by the benefits arising from doing so. Destination-based solutions such as Hop-Count Filtering are easily deployed since the victim has an obvious interest in doing so; unfortunately, the attacker could easily change the hop-count value of malicious packets, throwing off the mechanism.

2.1.7 Packet Marking

While not DoS solutions in themselves, schemes in this section aim to simplify the problem by providing a mechanism that allows a victim to determine where the malicious traffic came from. To achieve this, routers in the network inform the destination about the paths that packets follow before arriving; alternatively, routers can keep information about the packets they forward. Even if the attacker uses source address spoofing, with these mechanisms in place the victim can determine which flows are malicious and either filter them locally or trace them back towards their sources to filter them farther upstream.

In IP Traceback [56], as a packet travels from the source to its destination, each router along the path probabilistically marks it with partial path information. Since attacks are generally comprised of a large number of packets, this method ensures that, given enough packets, the victim will eventually receive a marking from each of the routers, allowing it to reconstruct the path from it to the attacker. Another approach similar to IP Traceback is proposed by Bellovin et al. [11] and is called ICMP Traceback. As packets flow through a router it generates, with low probability, an ICMP message to the destination containing the router's IP address. If a flow has enough packets, as is the case during an attack, the victim will receive an ICMP packet from each of the routers along the path and it will be able to trace the attack back to its sources.

A slightly different approach is called Hash-based IP Traceback [58]. This system allows tracing a single packet back to its origin, relying on the deployment near routers of components called Data Generation Agents (DGAs) that record information about packets as they are forwarded. Clearly storing complete packets at the DGAs would present an insurmountable storage capacity problem, so this scheme has DGAs use Bloom filters to reduce the amount of space needed to store each packet, with the added benefit of maintaining the privacy of the packet's contents.

Another scheme called Pi [62] deterministically marks packets so that those that follow the same path will share a Path Identifier. The packet marking is formed piecemeal by the routers on the path from the attacker to the victim, overloading the 16-bit IP Identification field to do so. While this means that the path identifier may not be globally unique, this is not a requirement to providing DDoS protection. Once the markings are in place, filtering can be deployed at the victim or preferably a dedicated machine like a firewall placed on the attack path to the victim. It can be simple, dropping all packets with a certain path identified, or more complex, based on configurable thresholds.

The main advantage of packet marking schemes is that they allow victims to trace packets back to their origin even in the presence of source address spoofing. In addition, this information can be derived post-mortem, after the attack has subsided (in the case of the hash-based approach, the victim would have to query the DGA relatively soon after the router forwarded the packet). However, packet marking has two serious obstacles. First, it requires significant deployment in order for the victim to receive markings from all (or most) of the routers along a packet's path. Even Pi, with its less stringent requirements, needs around 50% deployment to guarantee deterministic markings. While the schemes can be implemented using existing equipment, there is little incentive for deployment: the farther upstream a router is from the victim, the less it will be affected by an attack, since distributed attack traffic aggregates at the victim and far away routers are unlikely to be seriously affected by it. Second, packet marking, specially the probabilistic kind, depends upon a large number of packets flowing through a path, an assumption that does not always hold for distributed attacks. The hash-based approach does not have this problem; however, despite the use of efficient Bloom filters, it will eventually encounter storage space limitations since network bandwidth increases faster than memory access speeds.

2.1.8 Others

In a rather radical approach [34], the authors state that a characteristic of DoS attacks is that a small number of bad clients can deny service to a far larger number of legitimate clients. Further, they observe that in many cases bad clients are bandwidth-limited, otherwise they would simply send at higher rates to increase the attack's effectiveness. The solution consists of a server under attack asking legitimate clients to send at a higher rate, so that the percentage of the link used up by malicious traffic is decreased. Despite its merits, operators will likely be reluctant to deploy a DoS protection scheme that increases traffic during an attack.

2.1.9 Further Reading

General Reading

Inferring Internet Denial-of-Service Activity[49].
Internet Denial of Service: Attack and Defense Mechanisms[23].
Towards Network Denial of Service Resistant Protocols[8].

Anti-spoofing Mechanisms

Detecting Spoofed Packets[45].
MULTOPS: A Data-Structure for Bandwidth Attack Detection[28].

Packet Marking

Advanced and Authenticated Marking Schemes for IP Traceback[53].
An Algebraic Approach to IP Traceback[22].
An IP Traceback Technique Against Denial-of-Service Attack[17].
Efficient Packet Marking for Large-scale IP Traceback[30].
Enhanced ICMP Traceback with Cumulative Path[43].
IP Traceback-based Intelligent Packet Filtering: A Novel Technique for Defending against Internet DDoS Attacks[60].
On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack[42].
Tradeoffs in Probabilistic Packet Marking for IP Traceback[4].

Traffic Policing and Filtering

Guaranteeing Access in Spite of Service-Flooding Attacks[29].
Implementing Pushback: Router-Based Defense Against DDoS Attacks[12].
Mitigation of DoS Attacks Through QoS Regulation[27].

Service Differentiation and Capabilities

Mitigating Distributed Denial-of-Service Attacks with Dynamic Resource Pricing[13].

Overlays

Mayday: Distributed Filtering for Internet Services[5].
Websos: Protecting Web Servers from DDoS Attacks[21].

Miscellaneous

Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites[36].
Tracing Anonymous Packets to Their Approximate Source[14].

2.2 Commercial Solutions

Commercial DoS solutions exist from companies like Cisco, Arbor and Mazu. However, these do not inter-operate and are meant to provide point or site protection against DoS attacks, not scale to architectural levels. In addition, if an attack is large enough it can still bring down a site protected by this type of hardware. Finally, these boxes are expensive, and a large number of ISPs and server owners cannot afford them. An alternative are scrubbing centers, in which traffic is sanitized before reaching a victim. Both of these options are expensive, and a cheaper solution based on commodity hardware would be of great help.

2.3 Conclusions

While many of the research solutions discussed above show promise, they tend to place new requirements on the network or end clients, resulting in difficult initial deployment issues. From the more radical source routing approaches to packet marking and capabilities schemes, a requirement is placed to implement significant changes on the network and sometimes even the end-hosts, rendering these solutions impractical in the current Internet. While overlays do present a more feasible deployment story, they do not provide a full solution to the DoS problem since they tend to operate above the network layer.

Another common problem among research solutions is deployment incentives. Often these are completely overlooked or are misaligned: those parties that must deploy in order for the schemes to work are the ones that see the least benefit from initial deployment. Consequently, the approaches, while technically sound, have little chance of seeing the light of the day on the Internet.

Commercial solutions do not provide a perfect solution either. These rely on buying special boxes or redirecting traffic through scrubbing centers, both expensive options that only larger sites can afford. In addition, it is possible for a very large attack to bring even these defenses down, since the boxes' lack of inter-operability means that their defense cannot be made to scale to architectural levels.

Despite promising work both in the research and commercial fields, there is still no comprehensive, architectural solution to the problem of large, distributed DoS attacks. While several proposals have been put forth, they all suffer from initial deployment and incentives issues. The next chapter presents a series of architectural solutions to the DoS problem that aim to address these issues while requiring only off-the-shelf hardware to be implemented.

3 Proposed Contribution

As mentioned previously, the goal is to design and implement an architecture to combat large, distributed Denial-of-Service attacks. In addition, the architecture has to be incrementally deployable and provide clear incentives for early adopters or it will never see the light of day. Indeed, many of the solutions presented in the related work solve the DoS problem to various degrees, but they fail to do so in an incrementally deployable manner, and, as a result, the DoS problem continues to grow. Further, the solution needs to take spoofing into account or it may be turned into a DoS tool in its own right. While currently most attacks do not spoof, this is certain to change if an anti-DoS mechanism is put in place.

The proposed contribution consists of three architectures against DoS and an evaluation to show that they would be able to cope with large, distributed attacks under full deployment. The architectures are incrementally deployable, providing clear incentives for early adopters while improving in effectiveness as deployment continues. Each of the components of the architectures will be built on different types of cheap, off-the-shelf-hardware and their performance evaluated on a network testbed to see how they cope when dealing with a load similar to the one they would experience under full deployment while enduring a large attack. The testing will also aim to understand which extreme conditions would make a particular component fail. The ultimate goal and contribution will be to show that the architectures presented would be effective against large DoS attack on the Internet.

The work in the first architecture stemmed from a paper by Handley and Greenhalgh [32] describing seven steps that could be taken to protect the Internet against DoS attacks. The paper's approach was purposely radical, ignoring deployment issues. The aim of the architecture described in section 3.1 is, consequently, to retain as many of the advantages described in the paper while keeping deployability in mind. The second architecture (section 3.2) makes changes to the first one, resulting in important improvements in terms of deployability and allowing the proposal to be implemented using off-the-shelf hardware. The final architecture, described in section 3.3, eliminates a significant amount of complexity from the previous ones, making it easier to deploy on the Internet.

3.1 Routing and Tunneling Architecture

Ideally we would like to protect all Internet hosts, but realistically it is usually servers that present the biggest target. We propose, consequently, to provide a protected virtual net for those servers that wish to be better defended. An ISP providing such protection could charge a premium fee for the service, giving a clear incentive for deployment. In [59] the author presents a similar approach to the trace back problem that works for small DDoS attacks in the context of a single ISP. Our solution is intended to extend to a network of many collaborating ISPs, but in this section we first examine how our solution might initially be deployed at a single ISP, before examining how multiple ISPs might cooperate.

3.1.1 Single ISP Architecture

The first step is to designate certain subnets of the IP address space as server subnets: these will receive additional protection from attack. We refer to these subnets collectively as the *server-net*; conceptually they are within a protection boundary ringed by control points. Traffic from the public Internet must traverse one of these control points on its way into the server-net.

A condition of being a server-net host is not being permitted to send directly to other server-net hosts. This constraint prevents hosts inside the server-net from attacking other hosts inside the server-net, and prevents server-net hosts being exploited as relays in reflection attacks on other server-net hosts. It also helps slow the spread of worms within the server-net boundary.

The basic functions of a server-net boundary control point are *encapsulation* and *filtering*. At an encapsulator, packets destined for a server are encapsulated IP-in-IP, and sent to a decapsulator located in the server's co-location facility. An ISP must have at least one encapsulator, but maximum benefit will be gained with one encapsulator associated with each PoP or peering link.

The principal advantage of this architecture is that when a server is attacked, the decapsulator knows precisely which encapsulators the malicious traffic traversed. As a result, it can ask them to filter traffic, stopping the attack some distance upstream of the victim. We note that encapsulation isn't the only technique by which this could be achieved. In particular, MPLS tunneling might also be used for this purpose. However IP-in-IP encapsulation has advantages over MPLS. First, the address of the encapsulator can be directly obtained by the decapsulator, rather than needing additional mechanisms to reverse map the MPLS labels, but perhaps more importantly it is much harder to extend an MPLS solution inter-domain, which is our eventual goal.

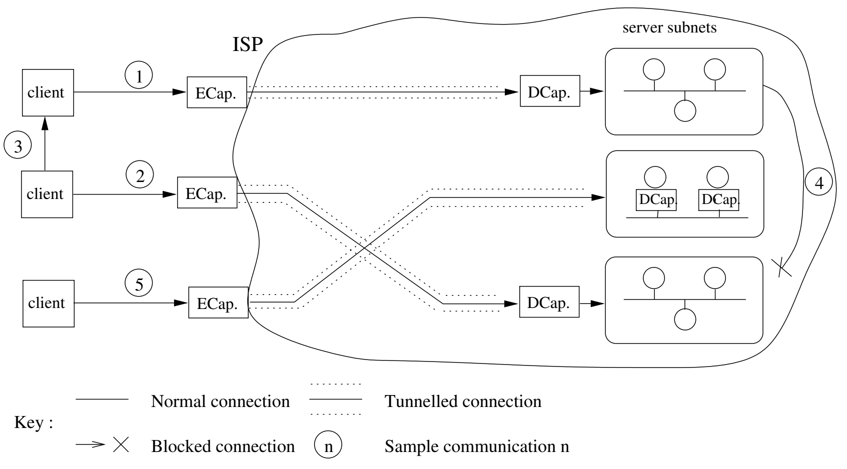


Figure 1: Scenarios for single ISP architecture.

Causing incoming traffic, even traffic originating from within the local ISP, to traverse an encapsulator requires careful control of routing. Routes to the server-net subnets should only be advertised from the encapsulators themselves, to ensure that there is no way to bypass them and send directly to the servers. In addition, the decapsulator addresses should be taken from the ISP's infrastructure address space, which (according to best current practice) should never be advertised outside of the ISP's own network. This prevents an attacker directly flooding a decapsulator associated with a server.

Possible communication paths within this architecture are illustrated in Figure 1. Flows 1 and 2 show the typical scenario with packets from a client passing through an encapsulator, being tinselled to a decapsulator, and finally arriving at the servers. Flow 3 is client to client and is unaffected. Flow 4, from one protected server to another, is disallowed to prevent reflection attacks and the spread of worms. Finally, flow 5 shows a protected server choosing to perform decapsulation itself.

Server→client traffic could be sent via the reverse of the incoming tunnel, or it could be forwarded natively. Either reverse path for the traffic is feasible with the proposed architecture, it should be an operation decision as to which is used. Tunneling has the benefit that a smart encapsulator can view both directions of a flow, allowing it to monitor and filter traffic more intelligently. However, this requires forwarding state at the decapsulator that is set up based on observed incoming traffic, and this state might be vulnerable to DoS if source address spoofing is used.

One solution is for the server to switch dynamically from a native to a tunneled reverse path once a connection is fully established and is therefore known not to be spoofed. Traffic with a tunneled reverse path can then be forwarded from encapsulator to decapsulator with higher diffserv priority, which might lessen the effect of flooding attacks that spoof source addresses.

The goal of a server-net when deployed at a single ISP is to allow automated filtering of unwanted traffic at the ingress point of that ISP. To perform such automated filtering requires a detection infrastructure in place, located so that it can monitor traffic to the server. Possible locations for this would be in the decapsulator, in the server, or on the path between the two.

Once a flow has been identified as hostile, the decapsulator needs to be informed, and it in turn informs the encapsulator, which installs the appropriate filter. The use of infrastructure addresses between decapsulator and encapsulator is the first line of defense against subversion of the filtering capabilities, as it should simply not be possible for normal Internet hosts to send filtering requests directly to an encapsulator. Behind this first line of defense, the signalling channel between the decapsulator and the encapsulator should be secured. Simple nonce exchange may be sufficient to protect against off-path attacks, given that a compromised router on the path can already cause DoS. Public-key-based solutions are, of course, also possible.

The benefits are clearly greatest for large ISPs hosting server farms and running a large number of encapsulators. Such ISPs will typically have many peering points with other large ISPs, and these peering points will be both geographically and topologically distributed. This traffic from a large distributed attack will be spread across many encapsulators because it will be entering the network from many neighboring ISPs, and so it will be stopped closer to its origin, before it has aggregated to the point where it can cause serious damage.

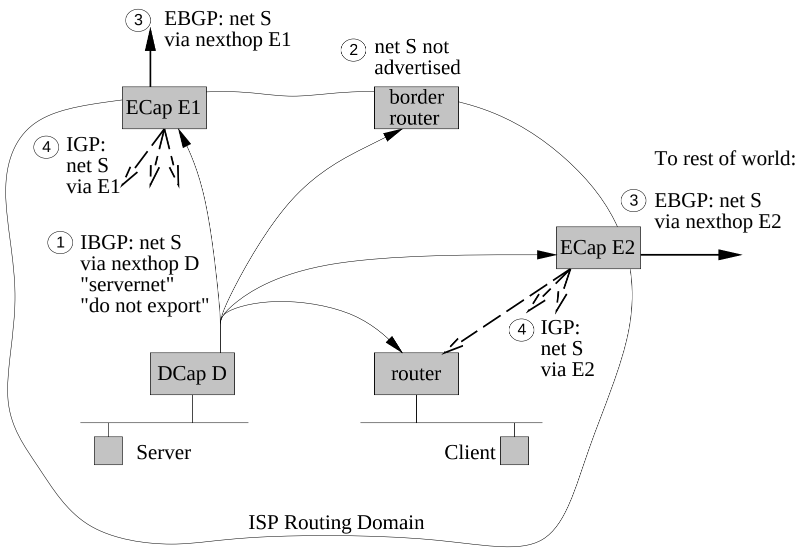


Figure 2: Route propagation for single ISP server-net

Smaller ISPs still benefit because their customers can control their degree of exposure, but a large enough attack is likely to overwhelm all incoming links. Nothing an ISP can do *by itself* will help in such circumstances.

3.1.2 Inter-ISP Communication

The benefits of a server-net increase as ISPs co-operate. Extending the protection boundary of the server-net to include the server-nets of co-operating ISPs moves the control points nearer to the sources of the DoS traffic. As we extend the protection boundary, distributed attacks become less concentrated at any particular control point, since traffic from each attacking host enters the server-net through its local encapsulator. Traffic that would previously have traversed the peering link uncontrolled now traverses the peering link encapsulated, within the server-net control boundary.

The general idea is that traffic enters the server-net at the server-net ISP closest to the traffic source, and is then tunneled from that ISP's encapsulator directly to a decapsulator at the destination ISP border. At the destination ISP, the traffic is decapsulated and re-encapsulated to get it to the final decapsulator near to the server. In principle it would be possible to tunnel direct from the remote ISP to the server subnet, but this assumes a degree of trust between ISPs that seems unlikely and unnecessary.

Traffic originating within an immediate neighbor ISP is forwarded natively to the border of the destination ISP where it is encapsulated as in the single ISP case.

3.1.3 Single ISP Routing

The basic requirements of routing to implement a server-net within a single ISP are:

- Server-net addresses must only be advertised to the rest of the Internet from the encapsulators.
- The encapsulators need to learn which subnets are in the server-net space, and which decapsulator is associated with each.
- The addresses used by the encapsulator and decapsulator must not be advertised to the outside world.
- A server-net subnet must not be advertised to server-net hosts on other server-net subnets.

We believe these requirements can be satisfied by current BGP[55] implementations on current router hardware.

The server-net could be manually configured, but in a large ISP this will probably be unfeasible. There are many ways to do this dynamically, but one possible solution is illustrated in Figure 2 and elaborated below.

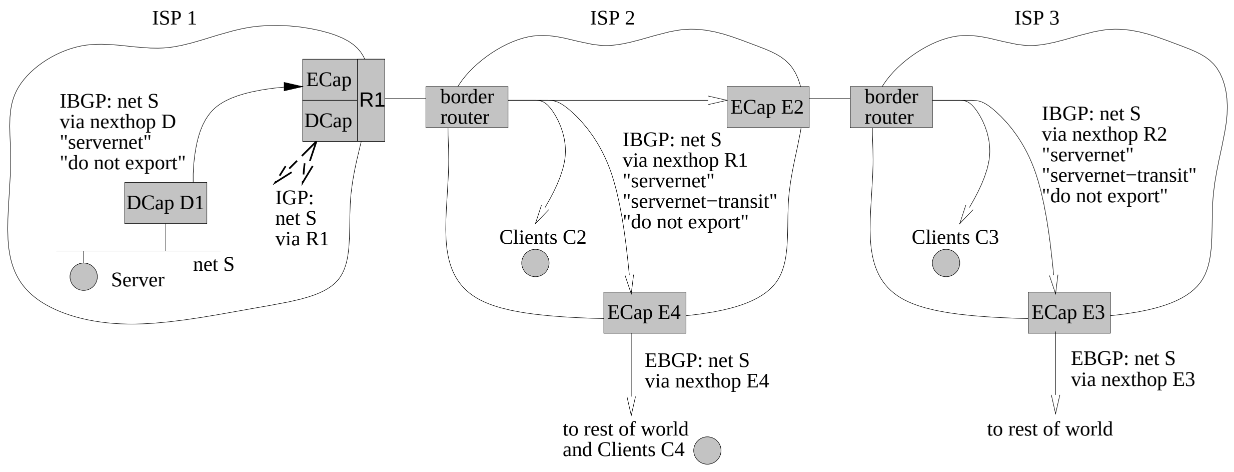


Figure 3: Route propagation for multi-ISP server-net

1 The decapsulator associated with a server-net subnet advertises that subnet into I-BGP[1]. It advertises its decapsulation address as the BGP next-hop router address. The route is tagged with a special BGP *server-net community*[2]. The route is also tagged with the "*do not export*" community that the ISP normally uses to indicate internal infrastructure routes that should not be exported to peers.

2 All border routers in the domain are already configured to not propagate routes to their external peers that have been tagged with the *do not export* community, so server-net routes will not leak to the outside world.

3 All encapsulators in the domain receive the server-net routes, and match on the server-net community. They then remove the *do not export* community and the *server-net community* from matching routes, and re-advertise them to external peers with the next-hop rewritten to be their own address. This will draw external traffic to the encapsulators.

4 The encapsulators also re-advertise any routes tagged with the *server-net community* into their IGP routing[3]. IGP routes are generally preferred over I-BGP routes on the basis of *administrative distance*[18], so this will cause traffic from clients within the ISP's domain to be drawn to the encapsulators rather than directly to the decapsulators.

5 The decapsulator routers also receive routes containing the *server-net community* that have been advertised by other decapsulators. The decapsulator installs a black-hole route for these subnets to prevent server-net to server-net communication.

Using routing in this way should satisfy our requirements. It is likely that slightly simpler solutions are possible if we can add BGP Path Attributes, but this is probably best done in the light of deployment experience.

3.1.4 Multiple ISP Routing

To protect its own server subnets, each ISP joining a multi-ISP server-net first runs the mechanisms described for a single-ISP server-net. To peer *within* a multi-ISP server-net, the following changes are needed:

1 Instead of an encapsulator at the border of the ISP, as shown in figure 2, a router with both encapsulation and decapsulation capability is used.

2 The encap/decap router does not remove the *server-net* and *do not export* communities when transmitting the route to a cooperating neighboring ISP[4].

If the number of server subnets in the multi-ISP server-net were small, then the neighboring ISP can do exactly as described in the single ISP solution. However, in a large server-net, the number of server subnets is likely to exceed

[1]I-BGP: Internal BGP - using BGP to carry routing information between routers within a domain.
[2]A *community* is a locally defined BGP routing tag. The actual value of community to use can be locally decided by the ISP.
[3]IGP: Interior Gateway Protocol - the intra-domain routing within an ISP, typically OSPF or IS-IS.
[4]This assumes that both ISPs use the same community values to indicate the server-net and do-not-export. If not, it will have to translate to the neighboring ISPs values.

the number of routes that can be safely redistributed into the IGP. In addition, even cooperating ISPs are likely to be wary of providing another ISP with a way to inject routes into their IGP. Thus we need to modify the mechanism a little, as shown in figure 3.

On receipt at the neighboring ISP, the border router will match on the *server-net* community, and add an additional *server-net-transit* community to these routes, distinguishing them from locally originated server-net routes. Routes with this additional community will *not* be redistributed into the IGP routing by encapsulators.

The result is that traffic from clients within the neighboring ISP's network (C2 in Figure 3) will reach the first encapsulator in the destination ISP using native forwarding. On the other hand, traffic from clients that would transit the neighboring ISP (C4 in Figure 3) will be encapsulated by the neighboring ISP's encapsulator and tunneled to the decapsulator at the destination ISP, before being immediately re-encapsulated and sent on to the destination server subnet's decapsulator.

Where more than two ISPs peer within a server-net, routes distributed onward to other server-net ISPs are sent with the BGP next-hop unchanged. Thus traffic will only ever be encapsulated and decapsulated twice:

- Encapsulated at the first encapsulator in the nearest server-net domain to the client.
- Decapsulated and immediately re-encapsulated at the first encapsulator in the destination ISP.
- Decapsulated at the destination subnet.

There is one issue remaining with regards to figure 3. There is no guarantee that traffic from clients C3 will traverse encapsulator E2 on its way to R1, and hence be encapsulated. To ensure this does happen requires controlling the inter-domain distribution of routes for R1. In fact the requirement is that routes for such decapsulators only transit between ISPs at the same peerings that server-net routes transit. A very similar use of an additional community tag can be made to preserve this congruence. The principal difference is that such routes are never propagated outside the server-net boundary.

3.1.5 Encapsulation and Filtering

IP-in-IP encapsulation is a standard feature on most routers. Juniper Networks ship a hardware tunnel interface module[37] capable of encapsulation at 10Gb/s that supports 8,000 tunnel virtual interfaces; other vendors no doubt have similar products. Thus current hardware is capable of performing fast tunneling to enough destinations to satisfy even large server-nets.

Most backbone routers also support packet filtering capability. It seems likely that they support sufficient filter rules to cope with attacks on the scales currently seen. In a multi-ISP server-net, any one attack is spread across many encapsulators, making it even harder for an attacker to saturate the filtering capability.

A cheaper solution is to use PC hardware. 400 Mb/s forwarding was possible in 2000[40] with minimum sized packets, limited largely by the PCI bus. A modern PC with PCI-Express is likely to be capable of in excess of 1 Gb/s with a very large number of hashed filter rules.

No standard exists for automated pushback of filters, but one would likely emerge if server-nets were widely deployed. In the meantime, a signalling channel would have to work around what is currently available.

Bro[52] can enable filter rules via the command line interface of Cisco routers. This is clunky, but works. In a similar manner, a server-net could use buddy-hosts co-located with each encapsulator. A buddy host would validate that a filter request came from the correct server-net decapsulator address by checking the BGP routing table, and then performing a handshake to prevent spoofing. It would install the filter rule in its local encapsulator using the command line interface. In the long run, we expect routers would directly support such a signalling channel.

The minimum filter granularity would likely be {source address, destination prefix} to prevent the targeting of other hosts on a victim's subnet. Also possible is {source prefix, destination prefix} to avoid an attacker spoofing multiple hosts on the same source subnet.

3.2 Edge-to-Edge Filtering Architecture

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

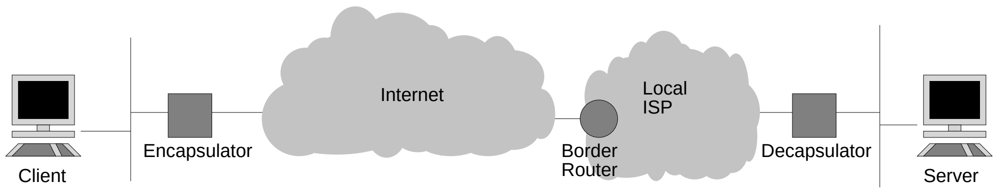


Figure 4: Encapsulation architecture.

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 travelling towards a destination is forced to traverse this filtering.

In the previous section, we described an architecture using similar mechanisms. The approach consisted of diverting traffic going to protected servers so it traversed control points. 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 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 section. This 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 section 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. What follows is an explanation of the full solution and these issues in greater detail.

3.2.1 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 4), 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 will send a request to its local decapsulator to filter traffic it deems unwanted[5]. On receipt of a filtering request for a particular source, the decapsulator will wait for the next packet from that source and note 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 will last only until the filtering request is sent.

[5]The detection mechanism is beyond the scope of this paper, but a commercial detection box may be able to perform this role.

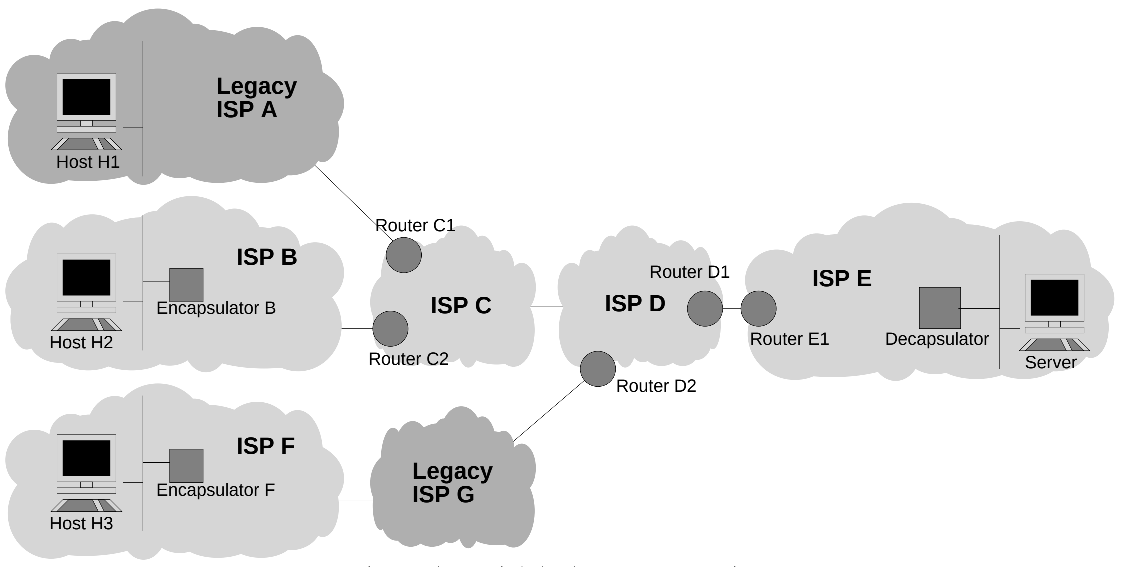


Figure 5: Partial deployment scenarios.

Once the decapsulator figures out which encapsulator to talk to, it will send out the actual filtering request. Finally, the encapsulator will install 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.

3.2.2 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 3.2.8.

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 destination 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.

3.2.3 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 the previous section. 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 would be 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" [10]. ISPs deploying our scheme would install a simple filter at all 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 choose to prioritize traffic that has come via a path where all the ISPs perform encapsulation.

Figure 5 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 upstream providers) so that their clients will not be placed in a lower priority queue.

3.2.4 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:

Spoofed C	Spoofed E	A same E as C	Comments
×	×	×	No spoofing
×	×	√	No spoofing
×	√	×	See figure 8:1
×	√	√	Impossible
√	×	×	See figure 8:2
√	×	√	See figure 8:3
√	√	×	See figure 8:4
√	√	√	Impossible

Figure 6: 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 8:5
√	√	A can drop request

Figure 7: Table of direct attacks.

- 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 6 and 7.

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.

We will now examine each of the indirect attack scenarios in Figure 6. The first two entries do not constitute an attack and are there merely for completeness.

The third entry (figure 8.1) 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 state-holding 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 8.2) 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

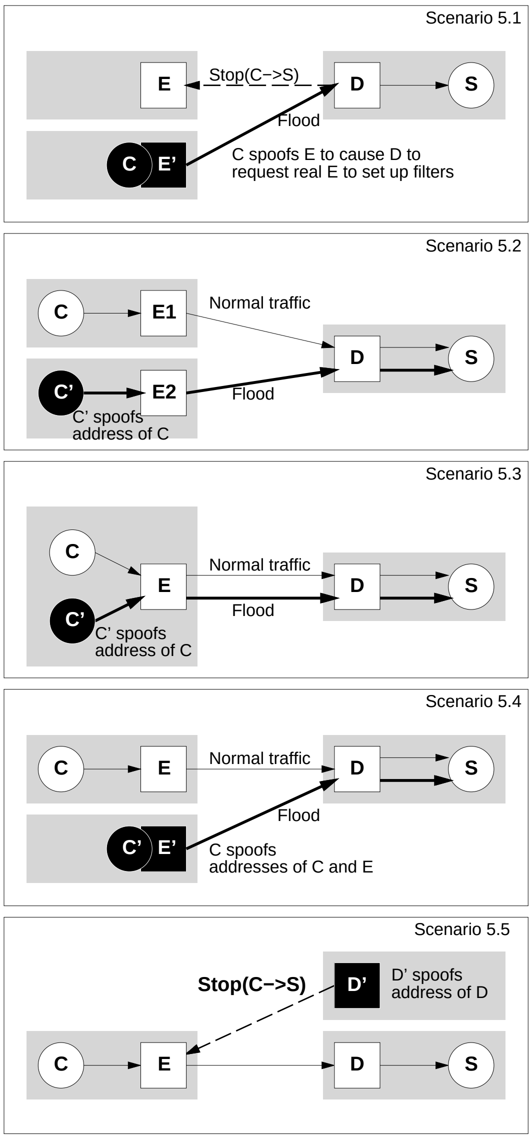


Figure 8: Potential abuse scenarios

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.

In the sixth entry in the table (figure 8.3), 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 8.4) 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 6 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 7. 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 8.5). 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 5) 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 in order to enable it.

3.2.5 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 [41]. 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.

3.2.6 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 sub-section 3.2.4, 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 wild-card source address, whereby the encapsulator will filter all traffic going to that destination; and wild-card 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 sub-section 3.2.4 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 8.5. 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 8.4.
- 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 8.2.

3.2.7 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 5 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.

3.2.8 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 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

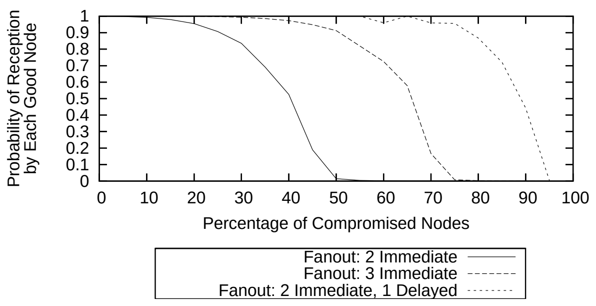


Figure 9: Robustness of routing infrastructure to compromised nodes.

not dissimilar to that of NNTP[38] or BitTorrent[20], 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[33]. 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.

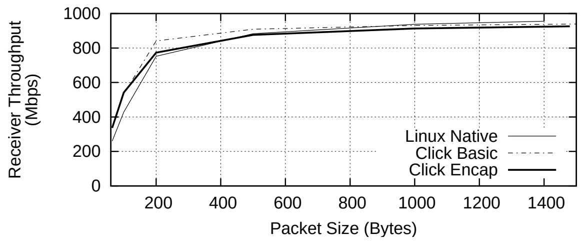
As the bindings are signed, every node receiving one should check the signature before passing it on to its peers. Thus there is no possibility for a malicious node to inject bad routing data unless the certificate chain itself has been compromised. Any node receiving bad data from a peer knows that 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 9, taken from [33], 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.

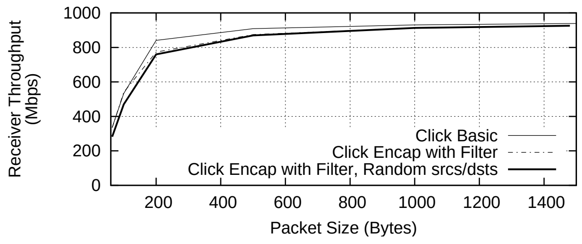
Our conclusion is that the distribution of the routes and the checking of signatures on these routes is not only feasible, but also relatively easy and very robust.

3.2.9 Performance

In the long run, we would expect encapsulation and filtering to become normal router or Ethernet switch functionality. Indeed, Juniper [37] already ships a hardware tunnel interface module capable of encapsulation at 10Gb/s; the missing part is mostly control software.



(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 10: Encapsulator performance

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 [40] 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 10(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 [57, 15, 16] depending on where the monitor was located and when the 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 10(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 10(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 labelled "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 look-up 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 10(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.

3.3 Terminus Architecture

Considering the size of botnets currently being reported, it is virtually impossible to defend against large, distributed flooding DoS attacks near the destination, even if the victim is connected to well-provisioned links. If a defense is to be effective against such attacks, it needs to filter the malicious traffic as close to its sources as possible, before it has had a chance to aggregate. The architecture we propose, called Terminus, enables a victim to request that certain traffic be stopped close to its sources. While the architecture described in the previous section achieved this goal, Terminus does so while greatly simplifying a lot of the mechanisms.

3.3.1 Edge Filtering

Terminus places special control points called border patrols (BP) in ISPs, as close to the sources of traffic as possible. The deploying ISP configures its network so that traffic from these sources is forced through border patrols. In this way, each BP can later be asked to install filters for traffic going through it, and, since it is close to the sources, the total aggregate throughput it forwards should be manageable.

With this mechanism in place, a victim or server would now have to know which BP the traffic came through. In a perfect world, this would be as simple as looking at the source IP address of the malicious traffic and deriving a mapping between this and the correct BP. Unfortunately, because of spoofing, the information in this field cannot be trusted.

The problem is not that most of the sources on the Internet spoof, but that a receiver cannot tell the difference between a spoofed packet and a non-spoofed one. All is not lost however, and we can use the border patrols mentioned above along with some added mechanism to combat this problem.

The idea is to use one bit, called the true source bit, in the IP header to mark whether the source IP address field in a packet is in fact the address of the host that originated the packet. As the packet travels from source to destination, ISPs who have deployed Terminus have their ingress edge routers set or unset this bit depending on whether the packet came from a peering link to a deployed or legacy ISP; the routers would know this through contractual agreements. In this way, if a packet traverses only deployed ISPs on its way from source to destination, it will arrive with its true source bit set, and its source IP address can be trusted. Of course, deployed ISPs are assumed to perform ingress filtering, either at their routers or their border patrols.

Figure 11 gives a couple of different scenarios illustrating this mechanism. Packets originating at ISP A and going to the server hosted by ISP G will arrive with the true source bit set. Packets from ISP B, on the other hand, will have this bit unset by router E2, since it knows that its link connects to a legacy ISP. Finally, any packet from ISPs C and D will arrive with the bit unset, since router G2 knows ISP F to be a legacy ISP.

Thanks to the border patrols and the true source bit, a victim can now know whether the source IP address in a packet is valid or not. Naturally, the server will only be able to make this assertion for packets that traversed a deployed ISP-only path, but as deployment progresses this will become the common case.

3.3.2 Filtering Requests

We now have control points deployed (the border patrols), routing injected by the deploying ISP that forces traffic through them on their way to the destination, and a way for a victim to determine where a packet came from (the

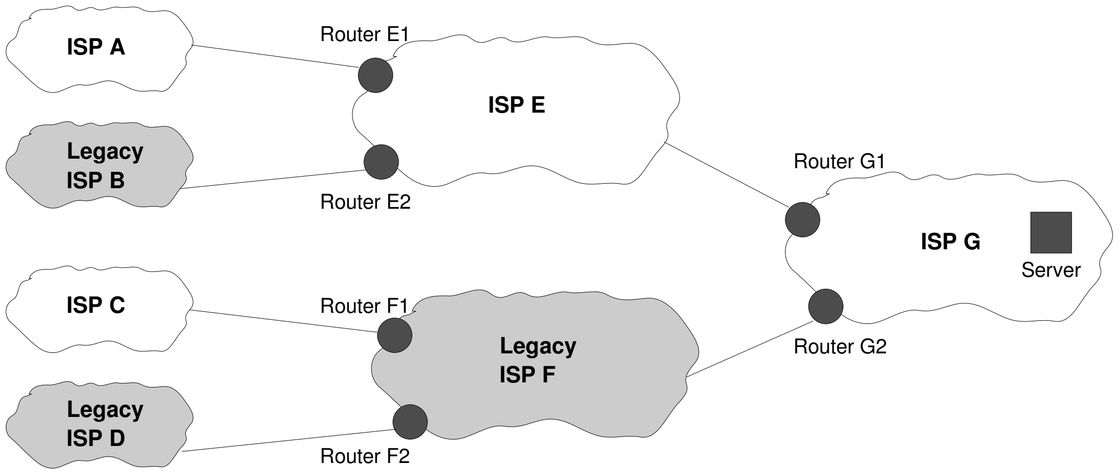


Figure 11: True source bit scenarios.

combination of the true source bit and the IP source address field). [6]

The next step is clearly to send filtering requests. While it would be possible to have the victim, typically a server, handle this itself, we would ideally not want to require more work from a potentially overloaded host. An alternative would be to have the IDS send these requests, but this would again place undue burden on a loaded system, and it is not clear that it would be easy to add the functionality we need to a commercial IDS box. Perhaps more importantly, it is not necessary to be on the fast path between a server and its clients to send filtering requests, so this functionality is a perfect candidate for being off-loaded to a separate, off-the-path box.

Such a box, which we will call a filter manager (FM), will typically receive requests from the IDS to filter sources. A FM will need some way to map a source IP address to the address of the border patrol that the packet came through, and a couple of possibilities exist. One such possibility would be to form a peer-to-peer network of nodes, where each deployed AS would install one node containing the necessary mapping. When a FM receives a filtering request, it could contact this network, asking to retrieve the mapping for a particular source IP address. While this requires no work from nodes that are not sourcing attacks, it does require work and adds delay for the FM trying to find all the necessary mappings. A better solution is to use a peer-to-peer network but distribute the mappings before-hand, so that they are ready when needed. The size of this "routing" table would certainly be manageable: only one entry would be required per AS, or about 20,000 entries in the current Internet.

What would one such entry look like? One possibility would be for an entry to have several mappings, each consisting of the address of a border patrol and the prefix for the clients it handles. This would mean that if an ISP decided to change the BP that traffic from a client goes through, the entire entry would have to be updated. An improvement over this, then, would be for an entry to consist of an IP address and a set of prefixes, aggregated as much as possible, representing the clients of the ISP that sit behind border patrols. In this way, switching clients from one BP to another would not result in having to update the mapping, and each entry is actually smaller.

As a result of this mechanism, the FM now knows the IP address of the host that it needs to contact for any source of traffic it wishes to filter. Such host, called a border manager (BM), needs to forward the filtering request to the appropriate border patrol (in a small deployment, the BM could be the same host as the BP; the architecture places no requirements on how this should be implemented). There are a couple of possibilities as to how this is actually done. First, the BM could keep track of the mapping between clients and BPs. This would present the same issue described in the previous paragraph, although we have now turned it into a local problem and changing these mappings would be trivial. Second, it would be possible for the BM to send all the filtering requests for the AS to each of the BPs. Each BP could then wait to see traffic for the filter before installing it, or perhaps install the filter right away but retire it if does not get sufficient hits. A final possibility would be for the BM to query each of the BPs, asking whether they are seeing traffic for a particular filter, and only installing the filter if they answer yes.

We now have all the basic elements needed to filter a DoS attack. Traffic from sources is forced through border patrols and arrives at the server, where a nearby IDS detects the attack, determines the malicious sources, and sends a filtering request to its ISP's filter manager. The FM, in turn, uses the mapping of source IP address to border manager obtained via the peer-to-peer network to send the necessary filtering requests to the appropriate border managers. These, in turn, ensure that the requests arrive at the appropriate border patrols that the traffic is actually going through, where

[6]This assumes that a detection mechanism such as an IDS is in place as a bump-on-the-wire next to the server to determine that a DoS attack is happening and to decide which sources are malicious; such a mechanism is beyond the scope of this architecture.

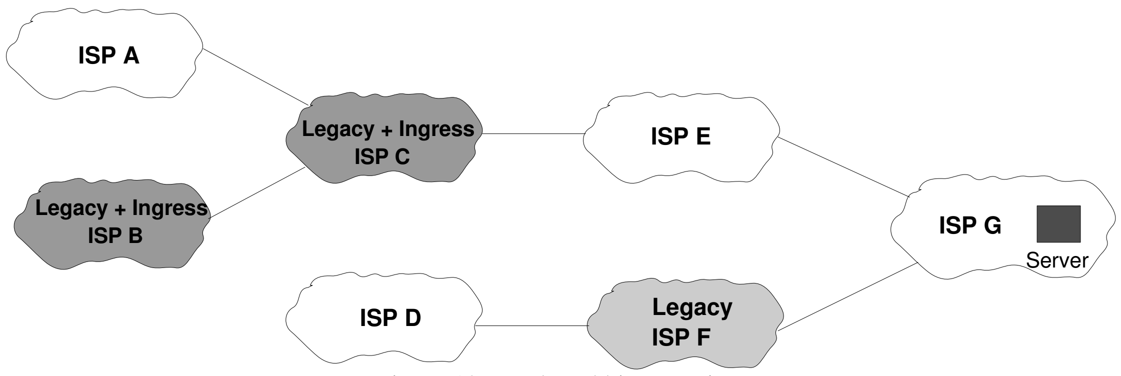


Figure 12: Deployed bit scenarios.

the malicious traffic is finally filtered.

3.3.3 Protecting Against Legacy ISPs

If Terminus was fully deployed, it would be possible to filter large DoS attacks, even if there still remained a few legacy ISPs. However, during initial deployment the story is likely to be different, with legacy ISPs being the norm rather than the exception. As a result, we need to provide some level of protection for a victim being attacked by sources hosted by legacy ISPs.

To this end, we can make use of the true source bit already described. This bit not only denotes that the IP source address is valid, but it also says that the packet originated and has traversed deployed ISPs. It makes sense to reward ISPs that have deployed the architecture, and so we can install classifiers at the edge routers of the destination ISP, sending packets that have the true source bit set to a higher priority queue. As a result, packets from legacy ISPs will always get lower priority and, during an attack, potentially little or no service.

3.3.4 Legacy ISPs Performing Ingress Filtering

What about legacy ISPs who perform ingress filtering? While a server would not be able to request filters from them, it would still be useful to know that the source of the traffic in the IP header is valid. All that we would require is that they set/unset the true source bit like deployed ISPs do. In this way, we would have three types of packets, each requiring a different action:

1 Deployed ISP packets that only traverse deployed ISPs or legacy ingress filtering ISPs (A→C→E→G in Figure 12)

2 Legacy ISP with ingress filtering packets that only traverse deployed ISPs or legacy ingress filtering ISPs (B→C→E→G)

3 Packets that either originate at legacy ISPs (F→G) or traverse them (D→F→G)

Clearly an extra bit, which we'll call the deployed bit, would be needed to distinguish between cases 1 and 2 (case 3 is already distinguished by the true source bit). The result is summarized in figure 13, along with the action to take for each.

3.3.5 Validating Filtering Requests

As described so far, it would be trivial for an attacker to contact a border manager and request a malicious filter. To prevent this, the BM will require that the IP source address of the packet containing the request is equal to the IP

TS Bit	D Bit	Action
0	0	Possible spoofing, no action
0	1	Possible spoofing, no action
1	0	Source valid, manual filtering
1	1	Source valid, filtering via BPs

Figure 13: Table of all possible states for true source (TS) and deployed (D) bits.

address of the filter's destination. The attacker will now of course proceed to spoof the source IP address, so we need added mechanism against this. It turns out that a simple nonce will do: the BM sends a nonce to the source IP address (a filter manager) and only installs the filter when it gets the nonce back. This will not stop an attacker on the path between the border manager and the filter manager; such a scenario is highly unlikely, since routers are much harder to compromise than hosts, but in the worst case the filtering requests could be digitally signed.

How would the reply to the nonce actually take place? The server could handle this, but we would rather not burden it with this task. Alternatively, it would be possible to have the IDS sniff the nonce request and reply to it, configuring the server's firewall to drop any such nonces. A variation on this theme would for the IDS to forward the nonce request to the filter manager, so that it could handle the actual reply. However, either one of these options would require extra work from the IDS, and we would like to off-load this work if possible.

What we would like is for the border manager to know that a particular filter manager is authorized to request filters on behalf of a set of servers, in essence a mapping of server prefixes to the IP address of the FM. The same peer-to-peer network described in the previous section could be used for this purpose, with the added requirement that each AS digitally signs its mappings to prevent attackers from injecting false mappings (see sub-section 3.2.8 for a detailed description of how a peer-to-peer network can be used to distribute signed mappings in a robust way).

With this in place, upon receiving a filtering request the border manager will inspect its source IP address. If the mapping between this address and the destination address of the actual filter exists in the set of mappings distributed using the peer-to-peer network, then the BM will issue a nonce. This nonce will reach a filter manager, which will echo it if it had, in fact, issued a filtering request. Finally, the nonce will be returned to the BM, who will contact the appropriate border patrols to block the unwanted traffic.

3.3.6 Triggering Requests Through Spoofing

Even though the architecture ensures that all filtering requests come from legitimate parties, it might still be possible for an attacker to spoof client traffic to trigger a legitimate filter to be installed against an unsuspecting victim. The list of possible attack scenarios have to do with the location of the attacker with regards to the victim. Only five such scenarios exist:

1 The attacker is in a legacy ISP that permits spoofing.

2 The attacker is in a legacy ISP that performs ingress filtering.

3 The attacker is in a different deployed ISP as the victim.

4 The attacker is in the same ISP as the victim but is behind a different BP.

5 The attacker is behind the same border patrol as the victim.

In the first scenario, the attacker will be able to spoof the client's address. However, the attacker's packets will arrive with their true source bit unset and, consequently, the filter manager will know not to issue any filtering requests as a result of them.

The next two scenarios are impossible: the attacker simply cannot spoof the address of a client in a different ISP from an ISP that performs ingress filtering. The attack described in the fourth scenario is easily preventable by either performing ingress filtering at the border patrols or by ensuring that the ISP has the same mechanism deployed in its network.

In the last scenario, the border patrol cannot tell traffic from the attacker and the victim apart. In essence, the problem is a local one, and the ISP can use ingress filtering or other measures to combat it.

3.3.7 Protecting BPs, BMs and FMs

In order for the architecture to be successful, all of its components must be robust against attack. Attacking border patrols, for instance, could deny traffic from clients from reaching a server. Under full deployment, there will be a significant number of BPs, and so DoSing the server by stopping client traffic would prove very difficult at best. During initial deployment, however, there might only be a few BPs deployed and the attack might be effective. The solution is simple: do not advertise the BPs' addresses in BGP. In this way, they will not be externally reachable and not susceptible to attack.

Border managers, on the other hand, do have to be externally visible in order to receive filtering requests. However, these boxes are not on a fast path, and so are likely to be able to handle substantial amounts of traffic before falling over. The hope is that as deployment progresses, it will be harder to cause serious harm by attacking BMs because of their sheer number. Another form of attack would be to use a border manager as a reflector, since it issues nonces in response to filter installation requests. This can be mitigated by designing the filtering protocol such that the filtering request is larger than the nonce, thus denying the attacker any possibility of amplification.

One final element of the architecture that could be attacked is the filter manager. Again, this box is not on a fast path, and so it is likely to be able to handle a high traffic load before falling over. More importantly, should it find itself the target of a DoS attack, it could simply drop all incoming traffic and concentrate on the filtering requests being issued by the IDS (care would have to be taken, of course, not to drop the nonce requests arriving from border managers).

4 Validation of Contribution

Performance is clearly a concern when dealing with DoS attack mitigation. A solution which may seem reasonable on paper is useless if it cannot cope with the load of an attack. Since performance is paramount, experimentation must happen on real hardware and not in simulation. As a result, a network testbed will be used to validate the architectures proposed. Since it would be extremely hard to create a large and realistic attack scenario on the testbed, the experiments focus on testing the performance of the individual elements of the architectures. The aim is to show that each of these elements would be able to cope with the load that a large attack would put on it, and that the sum of these results will demonstrate the scalability of the architectures as a whole. The following sections describe the testbed, the metrics used, the research progress, and the future experiment plan.

4.1 Testbed

Over the past two years, UCL's Networks Research Group has put a significant amount of work into building HEN, the Heterogeneous Experimental Network. HEN consists of about 80-100 computers of various speeds, each containing at least four network interfaces and all connected to one large, non-blocking gigabit switch. Consequently, the testbed allows for easy configuration of network topologies using different types of hardware. In addition, since computers on HEN are net-booted, it is very simple to migrate an experiment from one type of hardware to another. This setup is ideal for testing the architectures proposed in this report, and has already been used to derive the results mentioned below in section 4.3.

4.2 Metrics

The dominant metric used to measure performance will be the number of packets per second (or alternatively bytes per second) that the forwarding plane of a particular element of the architecture can process. Using this metric it can be shown that such an element can perform its task even while under attack. For an encapsulator or border patrol, for instance, it can be used to show that the forwarding performance of the box does not greatly decrease even when several filters are installed and traffic is hitting them.

A secondary metric will be the time between when a filtering request is issued and the filter is actually installed. Tests will be conducted to ensure that this figure is negligible compared to the round-trip delays experienced on the Internet, even when a large number of filtering requests are being issued.

4.3 Research Progress

The design of most of the elements of the three architectures has been completed and was presented in the previous section of this report. Regarding implementation and testing, a first version of an encapsulator and decapsulator have been written using the Click modular router platform. All tests used gigabit cards, varied packet size from minimum to maximum and measured forwarding throughput in megabits per second. Wherever filters were used, these consisted of a simple source IP address / destination IP address pair. The tests conducted were as follows:

- Basic IP forwarding to form baselines figures. This showed that despite using a cheap computer, throughput was excellent for most packet sizes (800 Mbps for packet sizes above 200 bytes) and very respectable for minimum sized packets (350 Mbps, a result of the limitations of the Ethernet hardware and/or driver).
- A test performing basic IP-in-IP encapsulation with a routing table with 200,000 entries, approximately the size of backbone BGP routing tables. This resulted in no significant degradation of performance.
- A test performing filtering in addition to encapsulation. The encapsulator kept the 2,000 filters in a hash of approximately 1,000 buckets (the 2,000 figure was chosen to obtain a relatively loaded hash). All traffic arriving at the encapsulator was destined for a non-filtered address, resulting in a negligible degradation of performance.
- A similar test as the previous one but adding another source of traffic, so that traffic from one source was filtered while traffic from the other was not. No degradation of performance was seen for packets larger than 200 bytes, and a minor performance hit to the "good" traffic for smaller packet sizes.
- A test where one of the hash's buckets contained 20 filters and all incoming traffic was made to traverse all 20 filters and miss. This again presented favorable figures, showing that hash collisions do not seriously degrade performance.

- A test where the hash was loaded with 2,000 filters and each packet in the incoming flow was made to hit a different bucket in the hash. The good numbers resulting from the test demonstrated that CPU cache trashing does not seriously hurt performance.

To summarize, these results show that using cheap, off-the-shelf hardware, it is possible to encapsulate and forward at almost line rate for most packet sizes, and at a reasonable rate for minimum sized packets. In addition, the hash-based filter is efficient even when fairly loaded, performs well even for buckets with long chains, and does not suffer greatly from cache trashing when packets hit different buckets.

4.4 Future Experiment Plan

As mentioned earlier, the experiments focus on the performance of individual elements of the architectures. Throughout this discussion, it is worth keeping in mind that several elements are common among the architectures, so that it will not be necessary to validate each element of each architecture separately. For instance, the encapsulators in the first and second architectures are the same, and are equivalent to the border patrols in the Terminus architecture. Further, decapsulators are common to the first two architectures, and the filtering protocol to all three. Taking this into account, the sections that follow describe the validation testing that will be done for each of these common elements.

It is also worth noting that for the most part the experiments described below will be conducted on several types of computers. HEN provides at least four types of computers, each with significantly different specs. The aim is to see what effect the varying kinds of CPU, memory and other components have on the results. Finally, Click is used to both generate and count packets, since it outperforms alternatives such as Iperf and Netperf, specially for small packet sizes.

4.4.1 Baselines

Before conducting experiments on the elements of the architectures, it is important to determine the basic performance of the computer hardware being used. While some baseline testing was done while deriving the results mentioned above, a more thorough examination is necessary. The following experiments will all be conducted once for each of the different types of computers used to test the components of the architectures. It is also worth noting that generally each type of computer has a different type of network interface on it.

Experiment 1

- **Experiment objective**: Determine how fast Click can generate and count packets on a box.
- **Procedure**: Use Click configuration to send packets as fast as possible and measure the rate. Repeat for varying packet sizes. See effect on results when the box generates packets out of more than one interface. Similarly, to test counting performance, measure throughput when receiving on one interface, then on more than one. Note that counters on HEN's main switch can be used to verify these results.
- **Network topology**: A traffic generator computer connected to a counter box, then additional interfaces on that same computer connected to other counter boxes. To test counting performance, use the same initial setup and then connect additional interfaces to other generator boxes.
- **Expected results**: Click can generate minimum-sized packets at a rate of around 330 Mbps, which is probably due to a limitation on the transmit rate of the network card. This experiment will help either confirm or deny this, as well as to uncover other hardware limitations.

Experiment 2

- **Experiment objective**: Determine basic forwarding performance for Linux and Click.
- **Procedure**: Use static routing with one route on a basic IP router both in Linux and in Click. Measure the forwarding performance (in Mbps) and repeat for different packet sizes. Also vary the number of incoming and outgoing interfaces.
- **Network topology**: A traffic generator computer connected to the IP router box, whose next hop is a traffic counter box. Additional generator and counting boxes will be used for experiments where the router has more than two interfaces.
- **Expected results**: The router forwards at line rate for large packets (in the case of one incoming interface and one outgoing interface) and forwards at around 330 Mbps for minimum-sized packets.

4.4.2 Encapsulator / Border Patrol

The results presented in section 4.3 help demonstrate the possibility of real deployment on the Internet. The total number of routes that an encapsulator might have to maintain has to do with the total number of decapsulators under full deployment. In the worst case scenario, it could be expected that every /24 subnet might have a decapsulator. With the IPv4 unicast address space roughly half-filled, this would equate to about eight million decapsulators. The number might seem daunting, but the address of each decapsulator could be kept in a hash table, to allow for fast look-ups. While the 200,000-route figure used in the experiments so far is still an order of magnitude off, it goes some way towards showing the scalability of an encapsulator. Still, further testing is needed to show that an encapsulator can cope with a larger number of decapsulator addresses without suffering a large performance hit.

Experiment 3

- **Experiment objective**: Prove the scalability of the encapsulator as the number of decapsulators increases.
- **Procedure**: Vary the number of installed routes and measure forwarding performance. Repeat for different packet sizes.
- **Network topology**: A traffic generator computer connected to the encapsulator, whose next hop is a traffic counter box.
- **Expected results**: For large packet sizes, throughput is the same as the one obtained in previous experiments. For smaller packet sizes, only a small performance hit is seen.

What about the maximum number of filters that an encapsulator might have to keep? The worst case results from full deployment and when fine-granularity filters (a source IP and destination IP pair) are used, as was the case with the mentioned experiments. The first variable in this equation is clearly the number of hosts sitting behind the encapsulator. While this figure will depend on ISPs, it is possible that an encapsulator might have to service thousands of hosts. In terms of filters, however, it is very unlikely that all or most of these will form part of a DoS attack at any one point in time. As a result, the number of filters needed to weed out malicious sources will be significantly smaller. In terms of destinations, only a certain number of hosts are under attack at any one point in time; Symantec [61] indicates that about 8,000 attacks happen per day, and so we can expect the number of hosts under attack at any one point to be perhaps one order of magnitude below this. The number of destinations under attack multiplied by the total number of hosts behind an encapsulator yields the maximum number of filters that an encapsulator might need to cope with. The 2,000 filter figure used so far is likely to be smaller than this, and so further testing is needed to ensure scalability in terms of numbers of filters. Also, the encapsulator should have a maximum number of allowed filters and a mechanism whereby filters are expired to make room for new ones.

Experiment 4

- **Experiment objective**: Prove the scalability of the number of filters that an encapsulator can have installed at any one point in time.
- **Procedure**: Vary the number of installed filters and measure forwarding performance. Vary the ratio of total number of buckets in the hash to the number of filters. Vary whether traffic is hitting the filters or not. Vary whether traffic is hitting different buckets. For all of these, measure forwarding throughput. Repeat for different packet sizes.
- **Network topology**: A traffic generator computer connected to the encapsulator, whose next hop is a traffic counter box. A second generator computer can be connected to the encapsulator to send malicious traffic.
- **Expected results**: Similar figures as in the 2,000 filter experiment. In addition, the expiration mechanism should not hurt performance.

In all experiments conducted so far, traffic was either all legitimate or half legitimate and half malicious. While realistically it is expected that most traffic will be legitimate, it is worth performing experiments to make sure that the encapsulator will cope well as the ratio of good to bad traffic varies.

Experiment 5

- **Experiment objective**: Prove that the encapsulator will perform well under various mixtures of legitimate and malicious traffic.
- **Procedure**: Vary the ratio of good to bad traffic (each in packets per second), and measure the effect of filtering on the good traffic (the forwarding throughput). Repeat for different packet sizes.
- **Network topology**: A traffic generator computer connected to the encapsulator, whose next hop is a traffic counter box. A second generator computer connected to the encapsulator to send malicious traffic.
- **Expected results**: The encapsulator should be able to process the same number of packets per second regardless of what the ratio of good to bad traffic is.

The discussion so far has focused on a simple filter consisting of a source IP address / destination IP address. While this provides a fine level of granularity giving victims the power to separate traffic from malicious hosts from that of legitimate ones, this is likely to be insufficient. Victims of large attacks, for instance, might not want to have to install a potentially very large number of filters one by one. Instead, it would be desirable to be able to filter using an IP prefix filter or by using a destination IP only, meaning that all traffic going through a particular encapsulator and going to the victim should be filtered.

It is not clear at this point what sort of mechanism and data structures to use so that an encapsulator will accept both prefix and destination-based filters. In the case of source IP address / destination IP address filters, the data structure was a hash, which of course does not extend well to the prefix case. One possibility would be to only accept certain prefixes, such as /32 and /24, but this would require two look-ups (and yet another look-up would be needed for the destination-only filters). Further work is needed to design possible solutions to this problem and test their performance.

While border patrols have not been explicitly mentioned, they are essentially a simpler version of an encapsulator, since all they have to do is filter traffic and forward it to their next hop router. Consequently, any filtering performance results obtained for an encasulator will apply to a border patrol.

4.4.3 Decapsulator / Filter Manager

A simple decapsulator has been built and a few very simple tests have been run using it. These essentially showed that the box can decapsulate packets without any noticeable degradation in forwarding performance. While the decapsulator was actually able to receive filtering requests from protected servers and forward these to the relevant encapsulators, this mechanism was only tested for correctness.

When a decapsulator receives a request to filter traffic from a particular source, it must wait until another packet from that source arrives and note down the address of the encapsulator it came through. For a DoS attack, this waiting period is likely to be minimal, so the state that the decapsulator might have to maintain at any one point might not be considerable. However, this assumes that a decapsulator issues a request as soon as it can and forgets about it. In reality, a decapsulator will have to keep state about a filtering request longer than this to respond to nonces from encapsulators and to receive confirmation from them that the filter has indeed been installed. As a result, it is worth conducting experiments to ensure that the decapsulator's forwarding performance does not suffer significantly when a large number of filtering requests are issued.

Experiment 6

- **Experiment objective**: Ensure that the forwarding performance of the decapsulator does not degrade when it has to keep state for outstanding filtering requests.
- **Procedure**: Vary the number of outstanding filtering requests and measure the forwarding performance.
- **Network topology**: A computer used to generate filtering requests connected to the decapsulator, whose next hop is a traffic counter box.
- **Expected results**: The decapsulator should be able to forward packets at similar rates as those obtained when it is not receiving any filtering requests.

A filter manager is a much simpler decapsulator, since it does not need to perform any decapsulation nor be on the fast path to a protected server. Indeed, it does not need to forward any traffic, just service filtering requests. To do so, a filter manager needs to know which border manager to contact for the source of malicious traffic that needs to be filtered. As mentioned before, only one entry would be required per AS, or about 20,000 entries in the current Internet. As a result, a test is needed to demonstrate that a filter manager box can deal with a large number of filtering requests while maintaining this mapping state.

Experiment 7

- **Experiment objective**: Show that a filter manager can handle a large number of filtering requests while holding state for all the source hosts to border manager mappings.
- **Procedure**: Load the filter manager with the state, send varying amounts of requests as fast as possible, and measure how long it takes to install all filters (including the time to reply to nonces).
- **Network topology**: A computer used to generate requests connected to a filter manager. The filter manager will also be connected to two other hosts which will act as border managers (these will be dummy border managers that will have the only task of issuing nonces).
- **Expected results**: The filter manager should be able to issue requests fast enough so that the delay is dominated by round-trip times on the Internet.

4.4.4 Border Manager

A border manager needs to listen for filtering requests, ask for nonces and forward the requests to the appropriate border patrols. Before issuing any nonces, it needs to ensure that the filter is appropriate for the BPs it manages. The border manager, then, needs to keep a small amount of state that maps prefixes of source hosts to the IPs of border patrols. If the request satisfies this check, the BM needs to figure out which filter manager corresponds to the destination IP address of the filter. The amount of state needed is similar to that of a filter manager, around 20,000 entries. Finally, the border manager needs to receive the replies to the nonces and resolve which border patrol to forward the request to. Testing is needed to show that a border patrol can perform these actions efficiently even when a large number of requests are arriving.

Experiment 8

- **Experiment objective**: Show that a border manager can handle a large number of filtering requests while holding state for the filter manager-to-destination IP address mappings.
- **Procedure**: Load the border manager with the state, send varying amounts of requests as fast as possible, and measure how long it takes to install all filters (including the time to send and receive nonces).
- **Network topology**: A computer used to generate requests connected to a border manager. The border manager will also be connected to two other hosts which will act as border patrols.
- **Expected results**: The border manager should be able to issue requests fast enough so that the delay is dominated by round-trip times on the Internet.

4.4.5 Architectural Experiments

If time permits, experiments will be conducted that will combine the individual elements in order to test the architectures as a whole. While it was previously mentioned that it would be very difficult to come up with a realistic DoS attack scenario on a testbed, the purpose of these tests would not be to show the performance of the architecture. Rather, they will aim to prove their correctness (including the route distribution and filtering protocols) in a complete, albeit small, setup. The ultimate goal is that the good performance of individual elements combined with the correctness of the overall architecture results in a strong validation of the claim that the proposed solutions would be effective against large DoS attack on the real Internet. Should time be of the essence, this type of experiment would be carried only for the Terminus architecture, since it shows the greatest deployability potential.

A Proposed Thesis Outline

B Work Plan

This section gives a description of the order and the time that each of the remaining tasks leading to thesis submission will take. Please refer to Figure 14 for a summary of this work plan.

B.1 Baseline Experiments

The objective is to have basic performance figures to compare against those derived from later experiments.

- Click packet generation performance **(1/2 weeks)**
- Click packet counting performance **(1/2 weeks)**
- Linux basic forwarding performance **(1/2 weeks)**
- Click basic forwarding performance **(1/2 weeks)**

Total time: 2 weeks

B.2 Encapsulator / Border Patrol Experiments

The object is to prove that encapsulator and border patrols will be able to perform well while dealing with DoS attacks under full deployment scenarios, and that they will be able to do so while accepting different types of filters.

- Test scalability of encapsulator as number of decapsulators increases **(1 weeks)**
- Test scalability of number of filters than an encapsulator can hold **(2 weeks)**
- Test performance when receiving various mixtures of malicious and legitimate traffic **(1 weeks)**
- Design and test encapsulator that accepts different types of filters **(4 weeks)**

Total time: 8 weeks

B.3 Filtering Protocol

The design of the filtering protocol is almost finished. It has been designed so that it can be shared among the architectures.

- Finish the design of the filtering protocol **(1/2 weeks)**
- Implement the filtering protocol **(2 1/2 weeks)**

Total time: 3 weeks

B.4 Decapsulator / Filter Manager Experiments

The objective is to show that decapsulators and filter managers are able to deal with a high rate of filtering requests, and, in the case of the former, to be able to do so while still providing good forwarding throughput.

- Test performance of decapsulator when dealing with filtering requests **(2 weeks)**
- Test performance of filter manager when dealing with large number of filtering requests and holding state for all the source hosts to border manager mappings **(3 weeks)**

Total time: 5 weeks

B.5 Border Manager Experiments

The objective is to show that border managers can cope with a large rate of filtering requests.

- Test performance of border manager when dealing with large number of filter installation requests **(2 weeks)**

Total time: 2 weeks

B.6 Architectural Experiments

The following tests will only be conducted if time permits. The aim is to show the correctness of the architectures as a whole, and to integrate their various components.

- Test full Edge-to-Edge architecture **(5 weeks, time permitting)**
- Test full Terminus architecture **(5 weeks, time permitting)**

  Total time: 10 weeks

B.7 Thesis Write-up

- Approximate time to completion will be **12 weeks**

  Total time: 12 weeks

B.8 Time Table

# Weeks	**Dates**	**Task Description**
2	3rd wk Feb - 4th wk Feb	Baseline experiments.
8	1st wk March - 4th wk April	Encapsulator and border patrol experiments.
3	1st wk May - 3rd wk May	Filtering protocol.
5	4th wk May - 4th wk June	Decapsulator and filter manager experiments.
2	1st wk July - 2nd wk July	Border manager experiments.
10	3rd wk July - 4th wk Sept	Architectural experiments.
2	1st wk Oct - 2nd wk Oct	Buffer time.
12	3rd wk Oct - 2nd wk Jan 08	Thesis write-up.
1	3rd wk Jan 08	Thesis submission.

Figure 14: Time table for activities up to thesis submission. Periods include the ending week.

References

[1] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Brief announcement: towards a secure indirection infrastructure. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 383–383. ACM Press, 2004.

[2] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. *SIGCOMM Comput. Commun. Rev.*, 34(1):45–50, 2004.

[3] D. Adkins, S. Shenker, I. Stoica, S. Surana, and S. Zhuang. Internet Indirection Infrastructure. In *Proceedings of the ACM SIGCOMM 2002 Conference*, August 2002.

[4] M. Adler. Tradeoffs in probabilistic packet marking for IP traceback. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 407–418. ACM Press, 2002.

[5] D. Andersen. Mayday: Distributed filtering for internet services. In *Proceedings of the 4th USENIX Sysmposium on Internet Technologiesand Systems*, March 2003.

[6] 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.

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

[8] T. Aura, J. Leiwo, and P. Nikander. Towards network denial of service resistant protocols. In *Proceedings of the IFIP TC11 Fifteenth Annual Working Conference on Information Security for Global Information Infrastructures*, pages 301–310. Kluwer, B.V., 2000.

[9] H. Balakrishnan, F. Dabek, F. Kaashoek, D. Karger, D. Liben-Nowell, R. Morris, and I. Stoica. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

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

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

[12] S. Belloving and J. Ioannidis. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the Network and Distributed System Security Sympo sium*, 1775 Wiehle Ave., Suite 102, Reston, VA 20190, February 2002. The Internet Society.

[13] C. Boyd, Michael Frentz, R. Krishnan, D. Mankins, and J. Zao. Mitigating distributed denial-of-service attacks with dynamic resource pricing. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.

[14] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of the 14th Systems Administration Conference*, December 2000.

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

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

[17] Z. Chen and M. Lee. An IP traceback technique against denial-of-service attack. In *Proceedings of the 19th Annual Computer Security Applications Co nference*, December 2003.

[18] Cisco Systems. What is administrative distance? http://www.cisco.com/warp/public/105/admin_distance.html.

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

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

[21] D. Cook, A. Keromytis, V. Misra, W. Morein, and D. Rubenstein. Websos: Protecting web servers from DDoS attacks. In *Proceedings of the 11th IEEE International Conference on Networks*, September 2003.

[22] D. Dean, M Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *ACM Trans. Inf. Syst. Secur.*, 5(2):119–137, 2002.

[23] S. Dietrich, D. Dittrich, J. Mirkovic, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, January 2005.

[24] E. Felten, A. Halderman, A. Juels, and B. Waters. New client puzzle outsourcing techniques for DoS resistance. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 246–256. ACM Press, 2004.

[25] W. Feng, W. Feng, and A. Luu. The design and implementation of network puzzles. In *Proceedings of the 24th INFOCOMM Conference*, March 2005.

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

[27] A. Garg and N. Reddy. Mitigation of DoS attacks through QoS regulation. In *Proceedings of the 10th IEEE International Workshop on Quality of Service*, pages 45–53, May 2002.

[28] T. Gil and M. Poletto. MULTOPS: A data-structure for bandwidth attack detection. In *Proceedings of the 10th USENIX Security Symposium*, pages 23–38, 2001.

[29] V. Gligor. Guaranteeing access in spite of service-flooding attacks. In *Proceedings of the Security Protocols Workshop*, 2003.

[30] M. Goodrich. Efficient packet marking for large-scale IP traceback. In *Proceedings of the 9th ACM conference on Computer and communicati ons security*, pages 117–126. ACM Press, 2002.

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

[32] Mark Handley and Adam Greenhalgh. Steps towards a DoS-resistant Internet architecture. In *Workshop on Future Directions in Network Architecture (FDNA 2004).* ACM SIGCOMM, September 2004.

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

[34] Michael Walfish Hari. Dos: Fighting fire with fire.

[35] C. Jin, H. Wang, and K. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41. ACM Press, 2003.

[36] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In *Proceedings of the eleventh international conference on World Wide Web*, pages 293–304. ACM Press, 2002.

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

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

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

[40] 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.

[41] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using Packet Symmetry to Curtail Malicious Traffic. In *Proceedings of the 2005 HotNets Workshop*, 2005.

[42] H. Lee and K. Park. On the effectiveness of probabilistic packet marking for IP traceba ck under denial of service attack. In *Proceedings from INFOCOMM 2001*, pages 338–347, April 2001.

[43] H. Lee, V. Thing, and J. Zhou. Enahnced ICMP traceback with cumulative path. In *Proceedings of the 61st IEEE Vehicular Technology Conference*, May 2005.

[44] 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.

[45] K. Levitt and S. Templeton. Detecting spoofed packets, 2003.

[46] J. Li, J. Mirkovic, M Wang, P Reiher, and L. Zhang. SAVE: Source address validity enforcement protocol, 2001.

[47] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3), July 2002.

[48] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *Proceedings of the IEEE ICNP Conference*, November 2002.

[49] D. Moore and G. Voelker. Inferring internet denial-of-service activity. In *Proceedings of the USENIX Security Conference*, June 2001.

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

[51] K Park and H Lee. On the effectiveness of Route-Based packet filtering for distributed DoS attack prevention in Power-Law internets. In *Proceedings of ACM SIGCOMM*, pages 15–26, 2001.

[52] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks, 31(23-24)*, pages 2435–2463, December 1999.

[53] A. Perrig and D. Song. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of IEEE Infocomm 2001*, April 2001.

[54] M. Reiter and X. Wang. Defending against denial-of-service attacks with puzzle auctions. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 78. IEEE Computer Society, 2003.

[55] Y. Rekther and T. Li. A border gateway protocol (BGP-4). http://www.ietf.org/rfc/rfc1771.txt, March 1995.

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

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

[58] A. Snoeren, C. Partridge, A. Sanchez, Jones C., F. Tchakountio, S. Kent, and T. Strayer. Hash-based IP traceback. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[59] Robert Stone. Centertrack: An ip overlay network for tracking dos floods. http://www.eecs.umich.edu/ msim/Bibliography/DoSAttack/centertrack.pdf.

[60] M. Sung and J. Xu. IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks. In *ICNP '02: Proceedings of the 10th IEEE International Conference o n Network Protocols*, pages 302–311. IEEE Computer Society, 2002.

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

[62] 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.

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

[64] X. Yang. NIRA: a new internet routing architecture. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 301–312. ACM Press, 2003.

[65] D. Yau, J. Lui, and F. Liang. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *Proceedings of the IEEE IWQoS Conference*, May 2002.