Using Routing and Tunneling to Combat DoS Attacks

Adam Greenhalgh, Mark Handley, Felipe Huici
Dept. of Computer Science
University College London
Gower Street, London, WC1E 6BT
[a.greenhalgh|m.handley|f.huici]@cs.ucl.ac.uk

Abstract

Thorough defense against DoS attacks is extremely difficult without incurring significant changes to the Internet architecture. We present a series of changes aimed at establishing protection boundaries to reduce the effectiveness of most flooding DoS attacks against servers. Only minimal and local changes are required to current network architectures. We show that our scheme is highly beneficial even if deployed at a single ISP, with additional benefits arising from multiple-ISP deployment. Finally, we show that the changes can be implemented with off-the-shelf components.

1 Introduction

Denial-of-Service (DoS) attacks have become a serious threat to the future potential of the Internet. If the Internet is to become critical infrastructure, as advocated by Internet telephony and digital convergence proponents, then it must be robust to attack. This is not to say that it must be impossible to disrupt Internet systems, but that the probability of *persistent* disruption must be low. In practice, this means deterring attack through legal means, preventing attack by removing attack vectors, or enabling effective rapid response in face of attack. In reality, all three of these are likely to be needed.

In previous work[3], we examined possible changes to the Internet architecture aimed at increasing robustness. The goal was deliberately radical: to restrict the modes of operation of the network to those that are actually desired, greatly reducing the potential attack surface. We proposed separating the IP address space into "client" and "server" spaces, allowing communication between the two spaces, but not within each space, and we advocated path-based addresses for clients, rather than globally unique addresses.

The potential benefits would be significant: source address spoofing and reflection attacks would be eliminated, and worms would find it much more difficult to spread. Without spoofed addresses, reactive response mechanisms could be automated simply and safely.

Unfortunately the proposal required IPv6 to encode path-based addresses, HIP[7] to provide a stable end-system binding, and new inter-domain path-encoders to build path addresses. Peer-to-peer applications presented additional complexity. Even if there were global agreement on such an architectural change, implementation could not happen overnight.

Such practical constraints lead us to wonder how many of these potential benefits could be obtained without change to the end-systems, and with only minimal and local changes to the routing infrastructure.

The issue of local change is important. Altruism is rarely an important guiding principle for ISPs, so any changes must provide short-term benefits. More customers must be attracted, more money must be received from existing customers, or running costs must be lower if a change is to be made. Changes to the client-side of the Internet are likely to require very widespread deployment before benefits are seen by servers, and so they are unlikely to be deployed. The question then is whether there is a server-side-first deployment strategy that leads in the long run to a more robust Internet, has benefits for early adopters, and incremental benefits as more adopters come on board.

In this paper we will explore just such a strategy. The plan we will sketch out does not have all the benefits of our original architecture; there are, nonetheless, benefits at each step along the path for the ISPs doing the deployment. Most importantly, the solutions are still very general, so they do not trade off short-term expediency against the future ability to evolve the applications running on the Internet.

2 The Nature of the Problem

DoS attacks can target scarce resources at any layer in the stack, but in this paper we only address flooding attacks that target the IP and transport layers (e.g. TCP, UDP, DNS). As such we only present an IP layer solution rather than using an application layer overlay like the one presented by [5]. Attacks targeting higher-layer functionality are still important, but robust applications are of little use if data cannot reach them. Before examining solutions though, it is important to step back and evaluate why this is a hard problem.

There are a number of factors that complicate the picture. The simplest DoS attack consists of one host flooding traffic to another. If the recipient cannot keep up due to CPU, memory, network bandwidth, or other resource starvation, then the attack is likely to be successful. The obvious question is: why can't the recipient simply request that the traffic stop? The reasons are many and varied, and shed light on the nature of the problem.

The natural expectation is that an end-system could say to the network "don't send me any more traffic from X". There are essentially two technical issues: where in the network this would be enforced, and how to ensure that this mechanism can only be used legitimately.

In the existing Internet, there are not many places where such packet filters could be installed. To be effective, filtering must be installed upstream of any bottleneck on the path to the recipient, and ideally should be close to the source of the attack. Unfortunately, tracing back the path taken from the attacker to the victim is not trivial, given that wide-area Internet paths are almost always asymmetric. As a result, to automatically find an appropriate place to install such a filter requires mechanisms that do not currently exist. Even if such mechanisms did exist, it is far from obvious what incentive the ISP at the appropriate place for a filter would have for installing it, since an ISP close to the attacker is unlikely to have a direct business relationship with the victim. Finally there is the issue of authenticating that the request to install such a filter really did come from the victim. No appropriate authentication infrastructure currently exists, and without good authentication, any process that automates the installation of filters may be exploited as a DoS mechanism in its own right.

All these problems are made worse by *Distributed* Denial-of-Service attacks (DDoS), where many hosts are compromised and flood a victim in unison. With DDoS it becomes very important that any filters are as far upstream as possible. The sheer number of filters that the victim might need to be installed could also be a problem with existing hardware.

Any automated filter mechanism that installs filters for a specific {source, destination} pair of IP addresses will be useless against an attacker that spoofs the source address of packets. Not only can the attacker simply change address to avoid existing filters, but he might be able to spoof a legitimate machine's source address in "attack" packets, with the goal of causing a filter to be installed that prevents legitimate communication.

Recent data[2] from security vendors indicates that source address spoofing is rarely used today in DDoS attacks since it is not needed for them to be effective. Botnets are relatively easy to create, and since there is no automatic way to silence compromised hosts, there is little incentive to spoof. In fact, just enough edge-sites employ ingress filtering on outgoing traffic that a botnet actually has slightly greater firepower if address spoofing is *not* used. However if an automatic filtering mechanism were deployed, the prevalence of address spoofing would no doubt increase significantly. At present, an interesting vicious cycle exists:

1. Attacks do not source address spoof because there is no automated filtering mechanism.
2. There is relatively little deployment of ingress filtering, because attacks do not source address spoof.
3. There is no automated filtering mechanism because attacks could source-address spoof to bypass it.

One way to break such a cycle would be to deploy a mechanism to effectively defend against non-spoofed DoS attacks, while not being vulnerable to abuse by spoofed traffic. Such a mechanism would not *by itself* defend against spoofed traffic, but it would shift the balance. There would be more incentive for ingress filtering, although this seems unlikely to ever be ubiquitous. Most importantly, it opens the door for additional mechanisms to be designed that can detect the difference between spoofed and non-spoofed traffic.

In this paper we sketch out one such solution. It is not intended to be a panacea, but rather to raise the bar somewhat, and to do so with off-the-shelf technologies.

2.1 Points of Control

There are two important points in defending against DoS: the *point of detection* of an attack, and the *point of control* where the effects of an attack can be mitigated.

The process of attack detection is not simple. Some attacks such as SYN-flooding are easy to detect, at least at the recipient. Other attacks such as making a large number of HTTP requests to a web server are harder, since they require knowledge of what normal traffic looks like, and the purpose normally served by that traffic. Detection, however, is not the focus of this paper. Products from vendors such as Arbor Networks are available that can perform this sort of analysis, so we will assume that a detection infrastructure exists at important servers.

Our focus is on establishing *points of control*, once an attack has been detected. The current Internet has no real concept of a point of control, beyond what can be done using traditional firewalls and routing protocols. Generally firewalls are too close to the victim to be useful against DoS, and destination-based routing does not provide sufficient discrimination of traffic flows.

The requirement is for a victim to be able to communicate with a point of control upstream of local bottleneck links, and to be able to request instantiation of filters that prevent hostile traffic reaching its subnet. Such filters must be installed quickly, without human inter-

vention. In addition, the point of control must be able to validate the filter request, so filters cannot be installed maliciously by third parties.

None of this is terribly radical, and what remains is engineering rather than science. How can the victim rapidly and reliably discover the existence of a point of control on the path from the attacker? How can it do this when there are many thousands of attackers? How can the point of control validate the filter request?

Our solution answers these questions through segmentation of the address space, careful control of routing information, and encapsulation. These are the three key IP-level building blocks we have to work with in the Internet today.

3 Proposed Solution

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 [10] the author presents a similar approach that 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 options how multiple ISPs might cooperate.

3.1 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.

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 tunneled 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

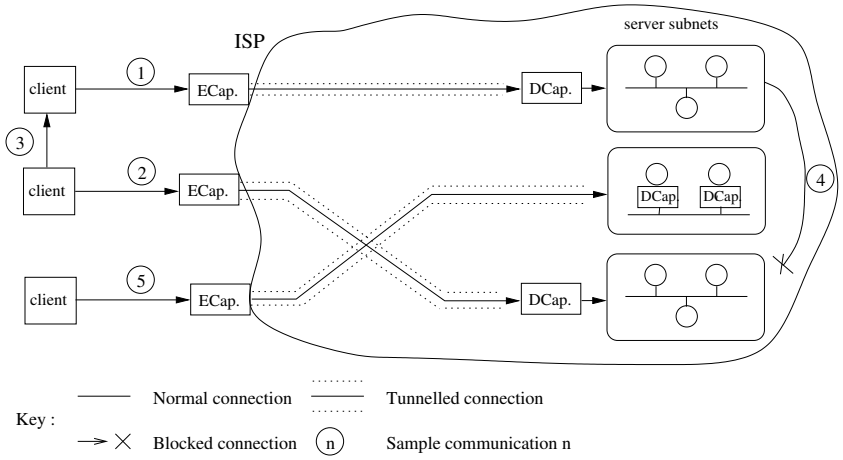

Figure 1: Scenarios for single ISP architecture.

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 signaling 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.

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.

3.2 Inter-ISP communications

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.

4 Implementation Details

To achieve DoS resistance for the server-net, we need careful control over the propagation of routing information. We also need appropriate filtering capability, a filter request channel, and sufficient encapsulation capacity. We will discuss first the single ISP scenario, then explore the options for extending this to multiple ISPs, and finish by discussing feedback and encapsulation.

4.1 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.

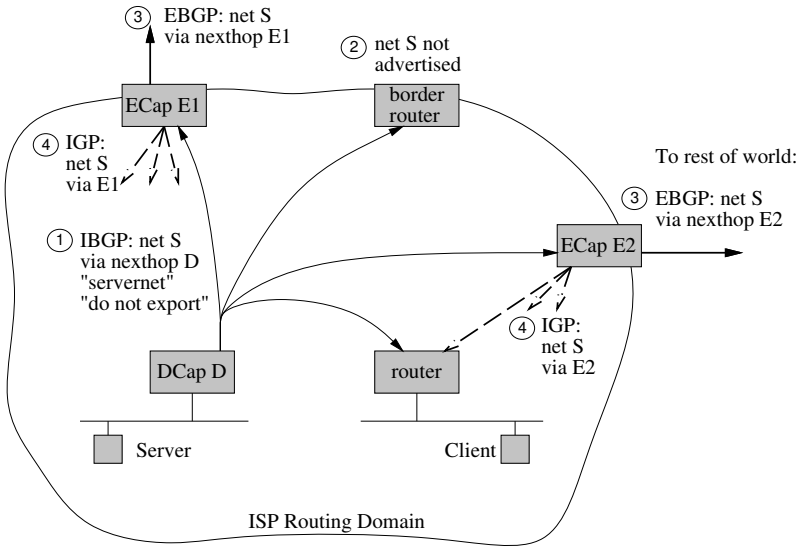

Figure 2: Route propagation for single ISP server-net

- 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[9] 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.

1. The decapsulator associated with a server-net subnet advertises that subnet into I-BGP[1]. It advertises its decapsulation address as the BGP nexthop 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 nexthop 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*[1], 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.

4.2 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-*

[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.

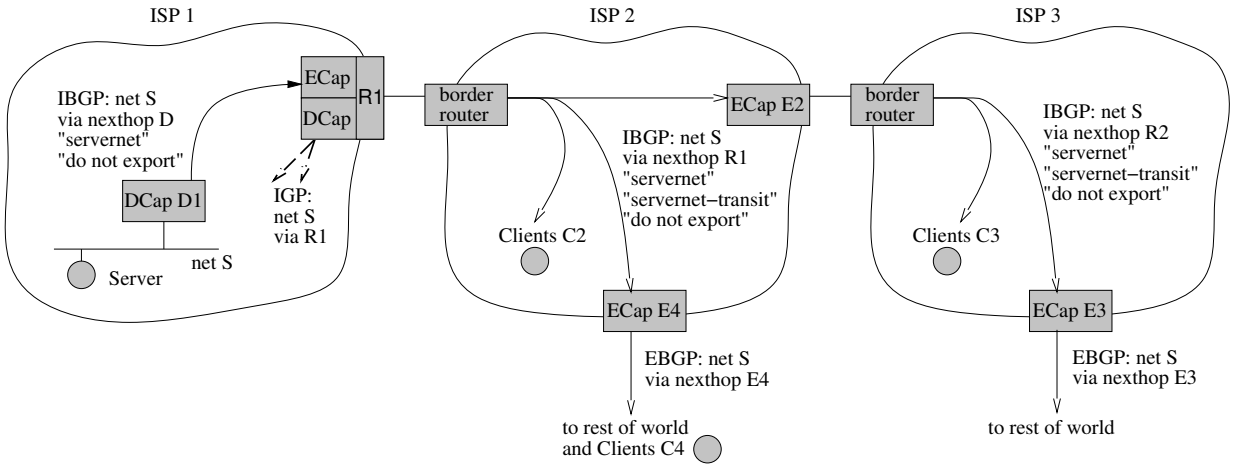

Figure 3: Route propagation for multi-ISP server-net

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 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 nexthop 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.

4.3 Encapsulation and Filtering

IP-in-IP encapsulation is a standard feature on most routers. Juniper Networks ship a hardware tunnel interface module[4] 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[6] 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

[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.

deployed. In the meantime, a signaling channel would have to work around what is currently available.

Bro[8] 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 signaling 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.

5 Future Work

Our initial plans are to work with ISPs to evaluate the feasibility of deploying a trial server-net in the wild. Based on this as well as any problems experienced, we will investigate whether it is worth standardizing additional BGP Path Attributes to simplify deployment.

A server-net neutralizes whole classes of attack, and attackers will adapt. Attacks requiring a full connection will have to become more subtle to avoid detection, and brute-force attacks will need to employ spoofing. To provide full protection, a server-net clearly needs to tackle spoofing; in the short term, servers might signal encapsulators to prioritize bidirectional flows.

In the longer term, server-net control points might be enhanced to validate source addresses. A stateful control point could perform a nonce-exchange with a source before allowing packets downstream. More specifically, in response to a data packet, a control point might send an ICMP nonce packet back to the source. The source would then need to echo the nonce in the data packet to allow the packet to pass and to instantiate state. For IPv6, the nonce could be carried in a data packet using a destination option. For IPv4, this is harder, as IP options are virtually useless. However, the IPSEC authentication header might be generalized into a form similar to the IPv6 destination option. Although such extensions are a long-term proposal, there might be incentive for deployment: a server-net under attack would give much higher priority to connection setup packets that were resent with the correct nonce echo.

6 Conclusions

A server-net is a protected region of the Internet that can provide upstream filtering capability for servers under DoS attack. We have sketched out how to build such a capability using current off-the-shelf router hardware and software components. The scheme we presented provides a workable solution using tunneling and careful control of routing to provide points of control in the network; filters may then be installed at these points to drop DoS traffic close to its source. Further work is needed to test real deployment of the scheme at ISPs, to understand the economics of deployment, and to limit source address spoofing.

This work is not intended to be a complete solution to DoS attacks in the Internet. It only protects servers, and only those that do not need to communicate frequently with other similarly-protected servers. In its current form, it only protects against non-spoofed attacks. However, this is still a significant improvement over unprotected directly-connected servers, especially when defending against attacks on higher-level functionality which cannot be spoofed because they require a full TCP connection to be setup. More complete solutions are likely to require significant architectural change to the Internet, and so will take very much longer to come to consensus and deploy.

References

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

[2] Communications Innovation Institute. Summary of the initial meeting of the DoS-resistant Internet working group. http://www.thecii.org/dos-resistant/meeting-1/summary.html, January 2005.

[3] 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.

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

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

[6] 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.

[7] R. Moskowitz, P. Nikander, and P. Jokela. Host Identity Protocol. draft-moskowitz-hip-09.txt, work-in-progress, IETF, February 2004.

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

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

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