Preventing Internet Denial-of-Service with Capabilities

Tom Anderson
University of Washington

Timothy Roscoe
Intel Research at Berkeley

David Wetherall
University of Washington

Abstract

In this paper, we propose a new approach to preventing and constraining denial-of-service (DoS) attacks. Instead of being able to send anything to anyone at any time, in our architecture, nodes must first obtain "permission to send" from the destination; a receiver provides tokens, or capabilities, to those senders whose traffic it agrees to accept. The senders then include these tokens in packets. This enables verification points distributed around the network to check that traffic has been certified as legitimate by both endpoints and the path in between, and to cleanly discard unauthorized traffic. We show that our approach addresses many of the limitations of the currently popular approaches to DoS based on anomaly detection, traceback, and pushback. Further, we argue that our approach can be readily implemented in today's technology, is suitable for incremental deployment, and requires no more of a security infrastructure than that already needed to fix BGP's security weaknesses. Finally, our proposal facilitates innovation in application and networking protocols, something increasingly curtailed by existing DoS measures.

1 Introduction

> *"Generality in the network increases the chance that a new application can be added without having to change the core of the network"*
> – Blumenthal & Clark [4]

The Internet owes much of its historic success and growth to its openness to new applications. A key design feature of the Internet is that any application can send anything to anyone at any time, without needing to obtain advance permission from network administrators. New applications can be designed, implemented and come into widespread use much more quickly, if they do not need to wait for key features to be added to the underlying network.

Quietly, however, the Internet has become much less open to new applications over the past few years. Perversely, this has happened as a rational response of network and system administrators needing to cope with the consequences of the Internet's openness. The Internet architecture is vulnerable to denial-of-service (DoS) attacks, where any collection of hosts with enough bandwidth (e.g., using machines taken over by a virus attack) can disrupt legitimate communication between any pair of other parties, simply by flooding one end or the other with unwanted traffic. These attacks are widespread, increasing, and have proven resistant to all attempts to stop them [15].

Operationally, to deal with persistent and repeated DoS and virus attacks, network and system administrators have begun deploying automated response systems to look for anomalous behavior that might be an attack. When alarms are triggered, often by legitimate traffic, the operational response is typically to "stop everything and ask questions later." Unfortunately, any new application is likely to appear to be anomalous! Our experience with this comes from operating the PlanetLab testbed over the past year. PlanetLab is designed to make it easy to develop new, geographically distributed, Internet applications [17]. On several occasions, we have observed innocuous, low-rate traffic from a single application trigger alarms that have caused entire universities to be completely disconnected from the Internet. Since alarm rules are by nature secret, the only way to guarantee that a new application does not trigger an alarm (and the resulting disproportionate response) is to make its traffic look identical to some existing application. In other words, the only safe thing to do is to precisely mimic an old protocol.

Trading openness for security might be reasonable if it was effective, but unfortunately none of the proposals for addressing DoS are sufficient for completely eliminating DoS attacks over the long run. Attackers are winning the arms race with anomaly detection by making their traffic look increasingly like normal traffic. The CodeRed and follow-on viruses have demonstrated repeatedly that it is possible to recruit millions of machines to the task of sending normal HTTP requests to a single destination [13, 12]. This problem is fundamental to the Internet architecture: no matter how over-provisioned you are, if everyone in the world sends you a single packet, legitimate traffic will not get through.

We argue for taking a step back, to ask how, at an architectural level, we can completely address the DoS problem while still allowing new applications to be deployed. Our goal, in essence, is to let any two nodes exchange whatever traffic they like (subject to bandwidth constraints of intermediate links), such that no set of third parties can disrupt that traffic exchange. Our approach is based on tokens, representing one-time temporary leases or capabilities to send, that authenticate that the packet is desired by the destination. Capabilities have been used in many systems, with mixed success; we argue that in the Internet context their use can lead to a more secure *and* more open system than the current approach of ad hoc anomaly detection.

The rest of this paper presents our argument in more detail. Section 2 puts our approach in the context of related efforts to develop DoS solutions. Section 3 outlines the goals of our approach, and Section 4 gives a detailed design of our proposed solution. Section 5 summarizes our results and discusses future work.

2 Background and Related Work

Internet denial-of-service (DoS) attacks work by flooding some limited resource on the Internet, thereby preventing legitimate users from accessing that resource. Targets include the bandwidth of access links and other network bottlenecks, and also the computing and memory resources on servers, clients, routers, and firewalls. For example, some low-end routers will crash if they are sent pings at too fast a rate, because their CPU becomes overwhelmed. While this might seem to be easily fixed by building more robust software, as a practical matter, almost every device connected to the Internet has some vulnerability to a flooding attack, just as almost every host on the Internet is vulnerable to a virus attack.

As DoS attacks have become more frequent and persistent, the DoS problem has inspired an avalanche of research into solutions. Our work generalizes and builds on several of these efforts, particularly the pushback [10] and secure overlay proposals [8, 1].

Most of the evaluation of DoS solutions has focused on their feasibility and/or effectiveness in preventing DoS attacks. While this is clearly important, we also draw attention to a different kind of issue with existing work in this field: the future *consequences* for the Internet and its applications if these approaches are adopted. Our experience with new applications on PlanetLab indicates that a good DoS solution must not only be effective, it must also permit the seamless introduction of new network services.

Source Address Filtering

One of the earliest proposals to mitigate DoS attacks was to deploy source address filtering at all network ingress and egress points [5]. This would prevent attackers from placing arbitrary source addresses in their packets and would therefore be useful in reducing the kinds of attacks that could be launched. Source filtering can be generalized to allow for filtering any packet that cannot have legitimately arrived at any point in the middle of the network [16].

Ingress filtering becomes effective only with a high degree of deployment; a source address only provides proof of authorship if every node in the network is part of the trusted computing base. Despite being recommended as a "best practice" for over five years, there are still many gaps in the enforcement of ingress filtering. Even with complete deployment, advances in attack methods have largely rendered source filtering irrelevant. Source addresses can be spoofed among all addresses sharing the same network prefix behind the filter; this can be thousands of addresses. Worse, automated virus tools have made it easy to enlist very large numbers of hosts in a given attack; attacks today often comprise lots of legitimate, unspoofed packets – if a million machines send your DSL modem a single TCP SYN packet, it doesn't matter that they are all using their real source address – your link will be unusable.

Traceback and Pushback

Several sophisticated methods have been proposed for tracing attacks back through the network toward their source [3, 18, 19]. Traceback concentrates on identifying the hosts responsible for an attack and, like source filtering, does little to prevent sources from sending traffic. DoS attacks are generally launched from a (possibly large) number of compromised host machines. Traceback mechanisms can be invaluable in identifying such compromised hosts, but this is too late to prevent the attack and of limited use in determining the ultimate perpetrators of the attack.

To address this limitation, some researchers have proposed adding network support for dynamic traffic filters, called pushback [10, 7]. Although the pushback proposals to date have focused on controlling link bandwidth floods, in theory any host or resource on the Internet could use dynamic pushback to prevent resource exhaustion. With pushback, a node or link characterizes the types of packets causing the flood, and sends requests upstream to rate limit them closer to the source.

Unfortunately, it can be difficult to design filters that perfectly discriminate between good and bad traffic, especially at line rates in the middle of the network. Discrimination based on packet headers is vulnerable to spoofing; discrimination based on packet contents is foiled by the increasing use of end to end encryption. Sophisticated attacks can also use probes to reverse engineer a filter as a prelude to evading it [1]. For example, the initial pushback implementation simply rate limits all traffic to the same destination. This approach will not only throw out good traffic with the bad, it can be foiled by attacks that rotate destination addresses through a bottleneck link.

As we will describe later, our approach can be seen as a particular instance of dynamic filtering that uses end-to-end cooperation to make filtering more effective and easier to implement.

Overlay Filtering

Overlays have been proposed as a way of incrementally deploying DoS filtering, given the lag time to add support for sophisticated filters in router hardware. For example, CenterTrack [20] re-routes all traffic aimed at a destination under attack, through a special-purpose intermediate node; because it is out of the normal path, the intermediate node can do sophisticated analysis and filtering.

More generally, the Secure Overlay Service (SOS) [8] and Mayday [1] pass all traffic to a protected destination through a large overlay. After authenticating an incoming packet, the overlay adds a secret into the packet header before forwarding it to the destination. Downstream routers can then be configured to discard all packets for the destination that do not contain the secret. This is an ingenious approach that is vulnerable to an attacker discovering the secret. The secret is a target because it is shared among all traffic through the overlay to the same destination.

Our approach has a similar flavor to SOS and Mayday, in that we also include a nonce token in every packet as a lightweight authenticator. However, in our system, tokens are transient and limited in scope to a single source-destination path. This limits the damage that can occur when one of the tokens is discovered by an attacker. Unlike overlays, our system uses regular Internet routes; ISPs have policy control over routing using BGP in the same manner that they depend on today. We also target dynamic communication patterns, instead of just static pre-approved ones.

Many of the differences between our approach and that of SOS and Mayday are due to a difference in goals; they attempt to filter unwanted packets using existing network router hardware, and we explore the case where we are free to redesign the Internet to be architecturally more resistant to DoS attacks.

Anomaly Detection

Perhaps the most active area of DoS prevention work is anomaly detection [2, 6, 11]. Rule-based or statistical techniques are used to classify traffic patterns as as friendly or malicious. Malicious traffic causes a number of actions to be performed, such as raising alarms, installing network filters, and sending automatically generated emails to the administrators at sites suspected of generating the traffic. The argument has been made that only an automated response to a DoS attack or security intrusion can be fast enough to prevent damage or loss of service [14].

We have grave misgivings about the consequences of deploying such systems.

One of a large collection of PlanetLab "incidents" will serve to illustrate this. A network tomography project running on PlanetLab sent a small number of packets that were almost but not completely like traceroute to a few hosts in every BGP-visible prefix; only one organization (out of 10,000) complained, with an automated email message generated by an anomaly detection system. This caused 10% of PlanetLab sites to immediately disconnect their machines.

Ultimately, anomaly detection is not a sufficient response to the problem—the decision as to whether a particular flow is an attack or not needs to be made end-to-end at the application level. Worse, in the limit anomaly detection leads to closed systems as ISPs and sysadmins lock down everything that isn't completely standard in the arms race with attackers. Since the filter policies are typically secret and complaints sent out-of-band, a legitimate application developer may never know exactly why the traffic triggered an alert. How is an application to know, for example, the maximum rate that a low-end router can accept ping packets before crashing, if the router has no way of informing the sender of its resource limits? Innovation in the network in effect becomes a practical impossibility: a new application must be extraordinarily conservative in what it sends to avoid triggering a disproportionate response from system and network administrators.

3 Towards a Capability-Based Internet

If we could start over, how would we re-design the Internet to be resistant to DoS attacks? Source authentication is insufficient – given the widespread success of virus attacks, knowing the source of a packet is potentially helpful but by no means a complete solution. Source authentication also provides no way for a fragile host to signal its resource limits. Rather, we argue that any complete solution to DoS attacks must give control over resource usage to the owner of the limited resource – the destination host and the intermediate links along the path. Further, any solution we consider should be open to new applications; the network should not be able to prevent two consenting hosts from exchanging packets, subject to resource constraints inside the network. Finally, the solution should be secure; no host distant from the path between source and destination should be able to disrupt their communication.

We argue that tokens, each representing a temporary single-use capability, issued by destinations, included in every packet, and enforced inside the network, provide a complete, open and secure solution to DoS attacks. Putting practical considerations aside, a conceptually simple scheme might be as follows. Each destination would generate certificates as tokens representing permission-to-send, each with a timestamp to limit hoarding of certificates, signed by the destination's private key. If every packet included a certificate, routers along the path could verify that the packet was recently requested by the destination, discarding those that were not and allowing the remainder to compete for bandwidth as usual. Certificates would be requested by a source, and granted by a destination, using a protected setup channel. Again ignoring practical considerations, we could extend this to allow administrative domains to protect their links from persistent overloading, by including the path in the certificate and requiring that every domain in the path countersign the certificate before it is considered valid. In this way, only legitimate traffic will be able to transit the network.

Our challenge, of course, is to design a solution which is not only complete, open and secure, but also feasible: scalable, able to implemented in today's technology, and incrementally deployable. The remainder of this paper considers this question of feasibility. While we are willing to trade silicon for increased robustness, the use of public key signature and verification is not likely to be viable on a per packet basis in the foreseeable future. Even assuming short signatures that cannot be broken for short periods of time (say 128 bit signatures using elliptic curve cryptography) such a scheme would involve considerably more computation than other kinds of security designs such as IPSEC and S-BGP. To be viable, any realistic scheme must involve far less computation and use as little packet space and router state as possible. Since we are dealing with denial-of-service, the mechanisms must also not be vulnerable to attack themselves, e.g., by overloading.

4 Strawman Design

The strawman design presented here is the basis of our argument that the capability approach is feasible. There are clearly more details to be considered to reduce the design to practice, along with much room for improvement. For example, we have not elaborated how to handle node failures other than an expectation of applying soft-state techniques. Nonetheless, we argue that the core mechanisms presented here achieve our goals: they do not impose unreasonable implementation requirements in terms of PKIs, packet overhead, etc. The strawman is unlike a "public key signature per packet" scheme in these respects.

Our strawman augments the existing Internet infrastructure with incrementally deployable Request-To-Send (RTS) servers coupled to "verification points" (VPs) which sit on the data path of Internet links. RTS servers are the means by which sources obtain tokens to send packets. They are co-located with BGP speakers at network boundaries and communicate in a hierarchy per destination. VPs perform access control by verifying the existence of a valid token in all non-RTS traffic. VPs can be "bump on the wire" boxes or implemented as part of router line cards (with which they share much functionality). They are deployed near RTS servers at network choke points, such as customer access links and BGP peering points. We describe the scheme in terms of the sequence of its operation from the point of view of a single source talking to a single destination.

Obtaining Permission to Send

A source must obtain tokens before it can send a series of packets to a destination that participates in our scheme. RTS servers assist destinations in granting these tokens and prepare VPs for subsequent token checking.

Autonomous systems whose clients wish to have their inbound traffic filtered advertise the fact by annotating their BGP advertisements with a community attribute giving the (IP) address of their RTS server. As the BGP advertisement propagates across network boundaries, any AS in the path that wishes to mediate the communication adds its RTS server to the BGP advertisement, forming a chain of RTS servers from source to destination. RTS servers can thus be gradually deployed wherever they are found to be useful. Further, the security of RTS chains leverages the security of BGP advertisements; if an attacker can convince an upstream domain to accept a bogus BGP advertisement, connectivity is easily disrupted, making DoS trivial.

Sources can now discover a series of RTS servers through which to send their request; there is a hierarchy of RTS servers along the paths to each participating destination. A source obtains tokens by sending an RTS packet to the first RTS server on the path to the destination. This is relayed along the chain of RTS servers to the destination's RTS server, leaving soft-state in each server to allow the response to traverse the reverse path back to the client in a manner similar to RSVP, PIM and other protocols [21]. Note that, because they are coupled, the sequence of RTS servers corresponds to a sequence of VPs that will be traversed by regular packets sent by the source towards the destination.

RTS packets are sent via RTS servers rather than directly to the destination to protect the channel used to obtain tokens from flooding: RTS servers limit the RTS packets passed towards a destination prefix to a rate which is a small fraction of the destination's access bandwidth. The destination prefix advertises its token channel bandwidth via another BGP attribute. Combined with queue management at the RTS servers, this prevents any single network location from hogging the RTS channel. Note that unlike the current Internet, a distributed attack flooding the RTS channel has no effect on already established connections with valid tokens. Mission critical communication could, for example, always maintain valid tokens. Although we do not discuss it further here, the scheme could be extended to allow destinations programmable control over how their token channel bandwidth is allocated.

Eventually, an RTS packet will reach the destination. It must decide whether to allow the source to send further packets. We envision that simple policies of the form used today (which constrain communication patterns but do not prevent DoS) will be effective. For example, a site may allow incoming traffic from well-known remote locations or in response to outgoing traffic, or a public server may allow traffic from registered users or all sources with the same likelihood subject to an aggregate limit. All of these policies will provide some form of protection from DoS. How best to set these policies is a topic for future research.

If the destination decides to allow the source to send it packets, it mints capabilities by calculating a chain of K 64-bit one-way hash values $h_1, \ldots, h_K$. Many hash chains could be inexpensively pre-computed even for $K >> 1000$, but in practice $K < 100$ would be sufficient for most flows. The values h_i are capabilities; possession of each allows the client to send a limited number n of packets, e.g. 50, in the next t seconds, e.g. 10. The destination then sends the *last* hash value h_K, along with a random 32-bit initial sequence value s_0, to the source via the chain of RTS servers. We explain the reason for using a hash chain shortly. Each RTS server remembers the values and associates them with the flow in the VP coupled to the RTS server. At this stage, the source and VPs along the path from source to destination have all received an initial capability.

Sending with Capabilities

The source now has an initial capability h_K and is authorized to send n packets along the network path towards the destination within the next t seconds. Each packet is labeled with the capability itself and the associated sequence value. This labelling could be done in several ways: new header fields, IP options, a shim layer on top of IP, etc. Discussion of their relative merits is beyond the scope of this paper.

When each VP along the path receives a packet, it checks

the capability, sequence number, and flow identifiers (e.g. source and destination addresses) in its store of currently valid capabilities. If the capability is found and the parameters match, then the packet is forwarded and the count of times the capability has been used is increased; if the count has reached n or t seconds has passed then the capability is flushed from the VP. If a packet's token does not match a stored value, then the token is deemed invalid, meaning that the packet was never authorized or is no longer authorized, and it is discarded. Downstream bandwidth is thus reserved for packets with valid tokens.

All this assumes that attackers cannot reliably guess the capability values (and other parameters) sent to sources, and cannot snoop links along the path, since it is the simple possession of these values that provides authorization to use a network path. The choice of 64 bit hash values provides a space large enough that brute-force guessing attacks, where attackers send packets with random capabilities, are infeasible over the short period that the value is significant. In the second case, when attackers can snoop links, they by definition have access to the network path between source and destination and can disrupt communications even if the capability value is not known. Because tokens are specific to a given connection, the mechanism prevents snooping attackers from being able to use their knowledge to disrupt traffic on other, unrelated paths.

Acquiring new capabilities in-band

The above procedure provides the basic mechanism by which a destination can authorise the transmission of n packets from a particular source. We assume that the same procedure is independently used by the destination to authorize packets that are sent from it to the source, e.g., to carry the other half of a TCP connection. To send more packets, the source must acquire a new capability from the destination. This could be achieved by repeating the procedure, but doing so has drawbacks. It can potentially impose considerable load on both the destination RTS server and intermediate RTS/VP pairs. Furthermore, the connection setup procedure follows a network path which is symmetric with respect to the RTS server chain, and heavy usage of this path may be undesirable to carriers.

We avoid these problems as follows: after nearly n packets have been received by the destination, it can send the next capability h_{K-1} back to the source via the normal IP return path from destination to source (as opposed to the reverse path via RTS servers). This path will have already been authorized, e.g., to return ACKs in response to TCP packets received by the destination. After sending n packets using h_K, the source switches to using h_{K-1}, increments the sequence number, and continues to send to the destination.

It only remains for the VPs along the path to switch from h_K to h_{K-1}. The reason for using a hash chain should now be apparent: it provides auto-keying at VPs. When a VP receives a packet from a known flow but with an incremented sequence number and new token, it computes the hash of the new token to see if it matches the old capability. If so, the VP updates both its sequence number and capability for the flow. Note that to handle floods of bogus packets, the VP must be provisioned to do this for potentially every packet at line rate, even though the sequence number makes this worst case scenario unlikely. We don't see this as a problem: the gate count to perform a 64-bit one-way hash in hardware is, these days, insignificant compared to current operations performed by medium- to high-end routers on packets, and the operation can be overlapped with route lookup. Packet reordering can also be handled by having VPs retain the previous capability until it has been exhausted.

With this combined scheme, the destination can authorize communications from a source once per epoch, yet retain the ability to selectively shut off any flow or client address within a window of n packets by simply not revealing the previous hash value in the chain.

Protecting RTS servers

How do we protect RTS servers from DoS attacks on the channel reserved for RTS packets? Note that communications between RTS servers are tightly constrained: RTS servers should only receive requests from local clients (known through configuration) or adjacent RTS servers (known through BGP advertisements), and so network filtering can discard all other traffic to RTS servers. This can be conveniently implemented by VPs. One could also use quasi-static filtering rules installed at routers, in the same way that communication between BGP routers is protected today. This prevents attackers from blocking the RTS channel except in their immediate vicinity and provides "defence in depth": even if attackers compromise hosts, routers, VPs, or RTS servers themselves, the next filtering point along the forwarding path will limit the damage.

Incremental Deployment

A key motivation for our work is to enable organizations to cooperate in addressing DoS attacks, without relying on any new services to be provided by their ISPs. For example, suppose Intel and the University of Washington wanted to work together to ensure that no hosts in either organization participated in DoS attacks against the other. By setting up an RTS server and a VP at the ingress/egress router to each organization, and by advertising their participation through BGP, the two domains could guarantee that all traffic between them was validated. The edge VPs in this case would perform a network translation function, converting normal TCP connection requests into a token request to the remote RTS server. As an edge device, we believe a Gb/s VP can be assembled from commodity hardware combined with fast packet processing software [9].

Additional organizations could join simply by advertising their RTS servers, enabling them to control any unwanted traffic sourced at the participating sites. Since uni-

versities are responsible for a significant fraction of the Internet's DoS traffic as well as much of its legitimate new application traffic, we expect that they would be motivated to join simply to be good citizens; they may also have an economic incentive to do so to reduce bandwidth costs. Essentially, our proposal provides a mechanism for an organization to program the reverse firewall at the organization sending it traffic.

This can all be accomplished without any explicit cooperation from ISPs or router hardware changes. Once a significant number of sources of DoS traffic have deployed RTS servers and VPs, we believe the network effect will engage, providing an incentive for commercial organizations and ISPs to join simply to reduce the volume of DoS traffic reaching their sites.

5 Discussion

We feel the scheme outlined above convincingly argues for the feasibility of deploying explicit authorization in the Internet to cleanly address DoS. While we cast our work in an Internet context, it is more broadly an attempt to define the structures needed for any large network to be DoS-resistant. Moreover, the use of capabilities for sending packets also up opens several interesting research areas.

First, there is much scope for policies that distinguish traffic from different sources. A destination can vary the granularity of authorization so that highly trusted clients can be efficiently granted large transmit windows, whereas strangers can be treated cautiously. This can be done by choosing the number of packets and time window per capability, and by sending the source r capabilities in a batch by releasing h_{K-r}; the source can compute the hash in the forward direction to recover the intermediate capabilities. When tokens are granted sparingly they control the size of the "permission window" and provide a coarse but interesting form of service differentiation. For example, an e-commerce site might allow minimal access to clients who have not registered. Or the initial capability could be accompanied by other parameters such as a fine-grained token bucket. This provides an easy, low-cost way to introduce QoS enforcement into the network on the back of a scheme with immediate tangible benefits, i.e. DoS prevention.

Second, our scheme can be extended to protect links inside the network from being flooded. Because RTS servers mediate token communication between sources and destinations, an ISP operating an RTS server can control resource usage of any bottleneck links in the ISP. As in the pushback proposal [10], we envision this as a coarse-grained mechanism to prevent nodes from persistently sending at too high a rate, not one appropriate for implementing fine-grained congestion control. If a connection refuses to obey congestion signals, the RTS server can disable future traffic by waiting for the token to expire and refusing to grant additional tokens. This, and many other details such route aggregation, route changes and failures, granularity issues, and service differentiation, remain to be worked out as we make our proposal more concrete.

References

[1] D. Andersen. Mayday: Distributed Filtering for Internet Services. In *Proc. of USITS 2003*.

[2] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffi c Anomalies. In *Proc. Internet Measurement Workshop 2002*.

[3] S. Bellovin. ICMP Traceback Messages. `http://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt`, Internet Draft), 2000.

[4] M. Blumenthal and D. Clark. Rethinking the design of the Internet: The end to end arguments vs. the brave new world. In B. Compaine and S. Greenstein, editors, *Communications Policy in Transition: The Internet and Beyond*. MIT Press, 2001.

[5] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks that Employ IP Source Address Spoofi ng. Internet RFC 2827, 2000.

[6] A. Hussain, J. Heidemann, and C. Papadopolous. A Framework for Classifying Denial of Service Attacks. In *Proc. ACM SIGCOMM 2003*.

[7] J. Ioannidis and S. Bellovin. Implementing Pushback: Router-Based Defense Against DoS Attacks. In *Network and Distributed System Security Symposium*, 2002.

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

[9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[10] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *Computer Communications Review*, 32(3), July 2002.

[11] Mazu Networks. Self-Optimizing Network Traffi c Security. `http://www.mazunetworks.com/nts.html`, 2003.

[12] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The Spread of the Sapphire/Slammer Worm. `http://www.cs.berkeley.edu/~nweaver/sapphire/`, Jan. 2003.

[13] D. Moore, C. Shannon, and J. Brown. Code Red: A Case Study on the Spread and Victims of an Internet Worm. In *Proc. Internet Measurement Workshop 2002*.

[14] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. IEEE Infocom 2003*.

[15] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proc. Usenix Security Symposium 2001*.

[16] K. Park and H. Lee. On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets. In *Proc. ACM SIGCOMM 2001*.

[17] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. HotNets-I*, 2002.

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

[19] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-Based IP Traceback. In *Proc. ACM SIGCOMM 2001*.

[20] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proc. Usenix Security Symposium 2000*.

[21] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Network*, Sept. 1993.