Taming IP Packet Flooding Attacks*

Karthik Lakshminarayanan — UC Berkeley
Daniel Adkins† — UC Berkeley
Adrian Perrig — CMU
Ion Stoica — UC Berkeley

1 Introduction

One of the major problems faced by Internet hosts is denial-of-service (DoS) caused by IP packet floods. Hosts in the Internet are unable to stop packets addressed to them. Once a host's network link becomes congested, IP routers respond to the overload by dropping packets arbitrarily. This is contrary to the goals of the host, which could respond more effectively to overload if it had control over which packets were dropped. For example, a host may reject new connections rather than accept excess load. A host running multiple services may give higher priority to some services than others (service differentiation). Also, a host may provide lower quality service rather than reject requests (service degradation).

The main thesis of this paper is that hosts – not the network – should be given control to respond to packet floods and overload. Ideally, hosts should have fine-grained control over how routers process the packets addressed to them. For instance, hosts should be able to decide which packets to receive, which packets are dropped, and which packets are redirected. To illustrate this point, we show how hosts can implement the following useful defenses against packet flooding:

- Avoid receiving packets at arbitrary ports.
- Contain the traffic of an application (service) under a flooding attack to protect the traffic of other applications.
- Protect the traffic of established connections.
- Throttle the rate at which new connections are opened.

We present two possible realizations of the above defenses. One is based on the Internet Indirection Infrastructure [22], and the other is an IP-based solution in which hosts insert filters at the last hop IP router.

The rest of the paper is organized as follows. In Section 2, we argue why hosts should be given fine-grained control to respond to flooding attacks, and in the process specify the problem and goals. In Section 3, we demonstrate how hosts can defend themselves against flooding attacks using the ability to control how routers process packets in the network. We present two realizations of these responses in Section 4. We present related work in Section 5 and conclude in Section 6.

2 A case for host control

The term "denial-of-service" was originally coined by Gligor [11] in the context of operating systems. It has since been associated with many types of network attacks. It refers to attacks that exploit protocol vulnerabilities (e.g., SYN flooding [19]), attacks that exploit implementation vulnerabilities (e.g., Teardrop [7]), and packet flooding (e.g., DDoS). Shields [20] gives a definition of network denial-of-service which captures all of these different attacks.

In this paper, we focus on IP flooding attacks. Other DoS attacks can, in theory, be addressed at the hosts once the vulnerabilities are known. In contrast, hosts can do little against IP flooding attacks. IP flooding attacks are possible because hosts in the Internet have no control over which packets they receive. This problem is not unique to IP. Any entity in an open network with a public point of contact is vulnerable to flooding attacks.

We specify our goals in addressing the IP flooding problem in terms of how the aggregate throughput of applications running on hosts varies with the incoming traffic rate. Figure 1 shows how the application throughput[1] varies with the amount of incoming traffic for a typical Internet host. As the incoming traffic increases, the application throughput increases. However, as the incoming traffic exceeds the capacity of the network link, packet loss will ultimately cause congestion collapse of the TCP traffic. In contrast, the optimal throughput curve would stay constant at the link capacity.

Our goal is to make the throughput curve as close to optimal as possible. In this work, we do not focus on the mechanisms that hosts use to detect an attack. Instead, we focus on the mechanisms that hosts use to react once they determine that they are overloaded or under attack. Our central idea is to provide control to hosts over the packets they receive. Ide-

*This research was sponsored by NSF under grant numbers Career Award ANI-0133811, and ITR Award ANI-0085879. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, or the U.S. government.

†Daniel Adkins is supported by a fellowship from the Fannie and John Hertz Foundation.

[1] The definition of throughput is application-specific. For example, it could be the number of connections satisfied per second.

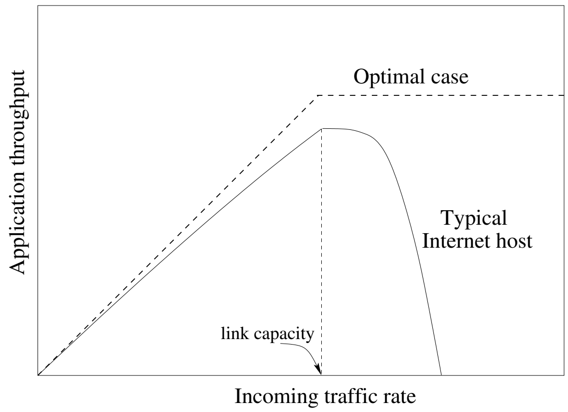

Figure 1: Application throughput as a function of incoming traffic rate for (a) typical Internet host and (b) optimal case.

ally, a host should be able to invoke responses that would have been possible had its network link not been congested.

A natural question is why not implement more sophisticated drop policies at routers instead. We believe that such an approach, while useful, is too restrictive. Hosts inherently have more information about the type and the importance of the traffic they receive than the network does. This places hosts in the best position to respond to IP flooding attacks. Consider a host that runs two services A and B, and assume that the traffic to service B surges abruptly causing congestion on the incoming link. The best possible response to this event may depend on knowledge available only to the host. If service B has higher priority than A, and if the host believes that the surge is because of a flash crowd (e.g., B is a web server that has just announced a new popular product), the host may decide to stop the traffic of the less important service A. In contrast, if A is the more important service and if the host believes that the surge is due to a DDoS attack, the host may choose to stop the traffic of B. Because the traffic in the two cases appears to be the same to the network, it would be impossible for the network to have an optimal response to the congestion without input from the host.

3 Responses to packet flooding

In this section, we present some responses to packet floods which hosts can use to improve their application throughput. In particular, the hosts specify the action to be performed on different *classes* of packets, where each class is determined by information in the packet header. How a class is defined by an application depends on the implementation of these responses. For example, a class could be defined as the set of SYN packets that arrive on a particular port. While our general philosophy is to provide arbitrary control, in this section we seek to illustrate how useful properties can be achieved by simple responses.

A. Avoid receiving packets at arbitrary ports

Internet hosts can receive unsolicited packets at ports where no service runs. Though these packets are dropped by the kernel, they consume network bandwidth and may affect other services. Thus, a host should receive packets only at ports on which it is listening or as part of an established connection. This response prevents arbitrary scanning of networks and also illegitimate packets sent to random ports.

B. Contain the traffic of individual applications

With the ability to decide which packets are dropped, hosts can contain the traffic of individual applications that might be under a flooding attack, thus protecting the other applications that run on the same host.[2] Let us consider the example of a host with two applications expecting requests, a web server and a transaction server. If either application is attacked, both are affected because they share a common network link. When incoming traffic exceeds the link capacity, the host should be able to decide which packets to drop depending on which service has higher priority. In our example, the host should be able to keep the transaction server running while possibly stopping the traffic to the web server.

C. Protect the traffic of established connections

To maximize the application throughput, hosts need to protect the traffic of established connections against arbitrary traffic. This makes it more difficult for an attacker to perform IP flooding attacks because it is harder to establish a connection and sustain the traffic on that connection rather than send arbitrary packets. This is because establishing a connection requires the attacker to handle data and signaling packets (e.g., ACK, SYN_ACK packets) from the victim.

D. Throttle rate of connection setup

While the previous defense protects the established connections, it is still difficult for legitimate clients to open new connections in the presence of DoS attacks. To alleviate this problem, a host under attack should be able to reduce the fraction of connection attempts made by the attacker. One way to achieve this goal is to make it harder for a client to open a new connection by using cryptographic puzzles [17] or captchas [23]. While this approach will throttle the rate of connection setup of all clients, it will a have a much greater effect on the attacker than on legitimate clients. This is because a legitimate client opens, in general, far fewer connections than an attacking host does.

[2] We are not referring to bandwidth sharing between conformant flows which systems such as Congestion Manager [5] achieve.

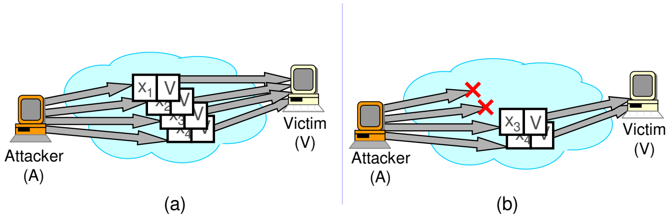

Figure 2: (a) Flooding attack via victim's public triggers. (b) Dilute the attack by dropping two of the four public triggers.

4 Two realizations

In this section, we first demonstrate how the defenses described in the previous section can be realized using the Internet Indirection Infrastructure ($i3$ [22]). Using the ideas that embody this solution, we then propose a mechanism that does not require an indirection layer. It only requires that ISPs allow hosts to add filters at their last hop router.

4.1 $i3$-based approach

The first solution is based on $i3$, an indirection layer that gives hosts control over which packets they receive by using a rendezvous primitive. The use of $i3$ provides a method of identifying hosts without using IP. Our choice of $i3$ for the indirection layer was influenced by our familiarity with it. For ease of exposition, we use the notation in [22]. We only need to know a subset of the capabilities of $i3$, which we summarize here.

1. Sources send packets to a logical *identifier* and receivers express interest in packets by inserting a *trigger* into the network.
2. Packets are of the form $(id, data)$ and triggers are of the form $(id, addr)$, where $addr$ is either an identifier or an IP address. Given a packet $(id, data)$, $i3$ will search for a trigger $(id, addr)$ and forward $data$ to $addr$.
3. If a host wants to stop receiving packets from a particular trigger, it can simply remove that trigger.
4. *Client-server communication:* Servers that expect connections from arbitrary clients must have triggers whose identifiers are well-known. These triggers are called *public* triggers Once a client contacts a server through its public trigger, they exchange a pair of identifiers which they use for the remainder of the communication. Triggers corresponding to these identifiers are referred to as *private* triggers.

We now discuss in detail how hosts can use $i3$ to achieve the desired defenses.

Avoid receiving packets at arbitrary ports. Clients in $i3$ can hide their IP addresses and publish the identifiers of only their public triggers. We assume here that it is very hard for the attacker to find the IP address by other means.

Contain the traffic of individual applications. The traffic of different applications can be distinguished from one another by the triggers used for communication. For example, each application could have a different public trigger, or each client contacting a server could do so with a different private trigger. To contain the traffic of an application, we could associate a drop probability with each trigger. If an application is attacked or becomes overloaded, we could raise the drop probability of its triggers to reduce its traffic. If necessary, we could disconnect the application entirely by setting the drop probability to zero or by removing its triggers.

Even though $i3$ does not associate drop probabilities with triggers, we can emulate the effect with multiple triggers (Figure 2). A server has a total of n public triggers, and each client is expected to randomly choose one of these triggers when it attempts to connect. The server, however, at any time maintains only a subset m of the triggers, thus effecting a drop rate of m/n. The set of triggers that the server maintains is changed rapidly over time so that the attacker does not have time to find out which triggers are active and respond accordingly. This would imply that a fraction m/n of the traffic (both attacker and legitimate) corresponding to that particular server is dropped. Note that it is not necessary that a client know the active triggers. If a client fails to connect it can simply retry with another trigger. The client will eventually connect after approximately n/m tries. For further details, we refer the interested reader to our technical report [1].

Protect the traffic of established connections. In $i3$, a client can send packets to a host only using the host's public triggers or the private triggers corresponding to the client's connections. The host can protect the traffic of its established connections by dropping some of the packets destined to the public triggers. Using the notation in the above example, a host while maintaining m of its n public triggers can choose an appropriate value of m such that the traffic of its established connections is not affected.

Throttle rate of connection setup. Consider a server S that is under a flooding attack. S can use indirection to redirect traffic to a powerful third party server A, called a *gatekeeper*, which shields the server S from the attack (Figure 3). The gatekeeper gives *cryptographic puzzles* to the client which have to be solved in order to contact the server. This will considerably slow down attacking hosts that attempts to open a large number of connections. In contrast, the impact this has on a typical client which opens very few connections will be small.

To implement this defense, server S stores a private trigger (t, S) where t is known only to the gatekeeper A. In turn, A inserts a public trigger (x, A) and advertises x as being the public ID of server S. When a client C wants to contact S, C sends a message to ID x. This message is in turn delivered to A (step 1 in Figure 3). Upon receiving this message, A sends

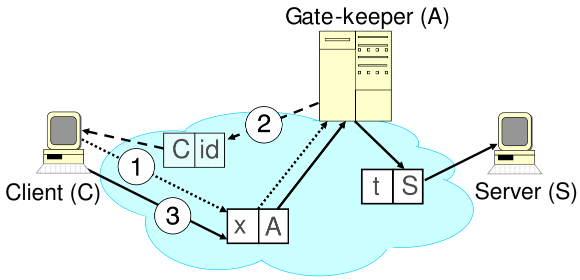

Figure 3: Slowing down a DoS attack on public triggers. When a client C wants to contact a server S, it must first solve a puzzle before the gatekeeper A will forward its packet to S.

a cryptographic puzzle back to client C via the private trigger (id, C) that was inserted by the client C (step 2 in Figure 3). Client C then solves the puzzle and sends the answer back to ID x. Upon receiving this message, A verifies that C has solved the puzzle and forwards the packet to ID t (step 2 in Figure 3). Finally, the packet is delivered to server S which allocates a private trigger for client C as before.

To avoid replay attacks, A will respond to each message with a unique puzzle. Once it sends the puzzle, A stores it and waits for a reply. On receiving the reply (*i.e.*, the solution to the puzzle), A removes the puzzle. Of course, if A does not receive a reply within a specific period of time, A will remove the puzzle from its list.

We also note that these schemes would be adopted by servers *only when under attack*. Hence, under normal operation, clients will not have the burden of either solving cryptographic puzzles or trying multiple times to reach a server.

In this section, we have shown how hosts can protect themselves by controlling which packets they receive. However, this is only half the solution. It is a challenge to design an indirection layer which is itself robust to DoS attacks, *i.e.*, one should not be able to use the indirection primitives that the infrastructure exports to attack either the infrastructure or the hosts. A solution to address this challenge is presented in [1].

4.2 IP-based approach

In this section, we present how the indirection-based filtering techniques can be extended to the Internet with minimal changes to edge routers directly connected to the end-hosts. The practical viability of these techniques and whether ISPs would be willing to install them are discussed at the end of this section.

We make the following assumptions: (i) The edge ISP is better provisioned than the host so that it may sustain attack traffic, (ii) The ISP is willing to install filters on the host's behalf, (iii) ISP filters must be modified to enable the port that the server runs on to allow incoming traffic, (iv) Unmodified clients would be able to connect to the servers in the normal case, but may need to do special work (like extra computation of cryptographic puzzles) when the server they contact is experiencing a flooding attack.

The basic idea of the solution is to provide a configurable white list of allowed ports at the edge-router directly connected to the hosts. Configuration settings include which ports to open, the rate at which bandwidth needs to be shared across different ports, etc. Also, to allow unmodified applications to use the mechanisms, the edge-routers perform minimal NAT-like functionality to setup white lists of connections that are legitimately setup using the public ports where services are run at the hosts.

Edge-routers that are directly connected to hosts need to maintain per-flow state on behalf of the hosts, i.e., if R is the edge-router through which all packets destined for S must pass, then R maintains per-flow state for S. We now explain the mechanisms in the context of the responses that it allows for hosts.

Avoid receiving packets at arbitrary ports. S instructs R to allow traffic on certain public ports only. This functionality is similar to *port forwarding* that many NAT-based firewalls allow. Once a client C establishes a connection to S (by using a port that is white listed by S in R), R will maintain state to allow C's packets through to S. R can do this transparently in the same manner as a NAT does. When the connection is terminated, R removes the associated state. S also has the power to stop malicious clients by terminating their connections.

Contain the traffic of individual applications. S can specify how exactly to split the bandwidth among its various applications. This functionality is similar to traffic shaping that many routers already implement.

Protect the traffic of established connections. S can ask R to reserve a fraction of S's bandwidth for established connections. Under congestion, R will shape traffic according to the rules that S has specified. R will limit the rate of packets to S's public ports in order to ensure that S's ongoing connections will receive their reserved bandwidth.

Throttle rate of connection setup. For redirecting traffic to gatekeepers, one can use DNS (like Akamai does) to send the traffic to the gatekeeper. In fact, the edge routers, if modified further, can themselves act as gatekeepers.

4.3 $i3$-based approach vs. IP-based approach

Generality. The indirection primitive that $i3$ exports provides a clean mechanism for hosts to exercise control. Furthermore, the indirection primitive gives an elegant way of redirecting traffic seamlessly to a third party which would require DNS hacks for implementing in IP. In contrast, in the IP-based solution, the use of IP addresses combined with port numbers to identify services running on a host is not general enough. In particular, the filters are not pushed arbitrarily into the network. [3]

Deployability. The indirection-based approach assumes the existence of an infrastructure such as $i3$. As we show in an earlier paper [22], such an infrastructure would greatly simplify deployment of a wide range of IP services, as well as enable the deployment of the flooding attack prevention techniques we outline in this paper.

The IP-based solution, on the other hand, requires changing the edge router of the ISP that provides service to the host. It also assumes that the ISP is willing to cooperate with the hosts by allowing them to install filters into the ISP network. This technique is, however, *incrementally deployable* in the Internet. Whether ISPs would allow this is a separate issue which we do not address in this paper.

5 Related work

The proposed solutions to prevent DoS attacks can be roughly divided into two classes: IP-level and overlay-based.

IP-level techniques are based upon *traceback* [6, 8, 18, 21] and *pushback* [16], both of which require router support. The goal of IP traceback is to detect the source of DoS attacks even when the attacker spoofs source addresses. It is complementary to our work as it provides an additional level of support for filtering illegitimate packets at the IP layer. With the pushback technique, routers identify the offending traffic aggregate and then push filters to upstream routers to limit the aggregate near its origin. In contrast to our mechanisms, pushback operates at a coarse level and treats all packets of the aggregate identically. Another IP-level approach is Path Identification (Pi), where routers mark packets that enables the victim to detect spoofed source addresses and to filter out malicious traffic [25].

Jamjoom and Shin [13] noted that during flash crowds, traditional drop policies hurt TCP throughput. Their solution, persistent dropping, uses filters in IP routers to isolate TCP SYN packets from TCP packets of ongoing connections. Their solution is effective against flash crowds, but not against DDoS where the traffic is not necessarily TCP.

Anderson *et al.* [3] prevent packet flooding by requiring clients to get tokens before contacting a server. The use of tokens allows for a coarse-grained bandwidth reservation policy. In contrast, we advocate for giving hosts far more control over the policy. For example, in our solution a host can easily differentiate between the traffic of established connections and the traffic of new connections, or between the traffic of two applications running at the same host.

Secure Overlay Services (SOS) [15] was the first solution to explore the idea of using overlay networks for proactively defending against DoS attacks. SOS protects hosts from flooding attacks by (i) installing filters at the ISP providing connectivity to the host and (ii) using an overlay network to authenticate the users. Mayday [2] generalizes this architecture and analyzes the implications of choosing different filtering techniques and overlay routing mechanisms. However, these solutions assume that the set of authorized users is known in advance, and that the set changes infrequently so that updating the authentication rules in the overlay nodes occurs rarely. Their solutions would not extend to a general IP setting (e.g., for a web server) where it is not always meaningful to speak of authorized users.

The first use of cryptographic puzzles is due to Merkle [17], who used puzzles for the first instantiation of a public key protocol. Dwork *et al.* propose puzzles to discourage spammers from sending junk email [10]. Juels *et al.* used puzzles to prevent SYN flooding [14]. Aura *et al.* [4] and Dean and Stubblefield [9] and Wang and Reiter [24] propose puzzles to defend against DoS on the initial authentication. Gligor [12] analyzes the waiting time guarantees that different puzzle and client challenge techniques provide. He argues that application-level mechanisms are necessary for preventing service flooding, and proposes a scheme that provides per-request, maximum-waiting-time guarantees for clients under the assumption that lower-layer, anti-flooding countermeasures exist.

6 Conclusion

In the current Internet, hosts are practically defenseless against IP flooding attacks, as a sufficient amount of malicious traffic exhausts the last-hop link capacity and renders the link unusable. The reason flooding attacks can occur is that a host is unable to control which packets it receives.

We find that end-hosts are in the best position to detect and react to DoS attacks. Thus, we present two proposals that allow hosts to take control of its incoming traffic. One mechanism relies on an indirection infrastructure that provides a general and architecturally clean solution. Our second solution, which is incrementally deployable in the Internet, relies on host and ISP collaboration, where the host can control fine-grained network filters on the last hop router.

Our solutions have the immediate benefit that victims of DoS attacks can start defending themselves. Another benefit is that hosts without server functionalities cannot be attacked by arbitrary attackers any more, as a host would enable packets only those connections that it has established. This would rule out many flavors of worms that attack hosts which are not servers. Only servers that can be contacted by arbitrary hosts need a rendezvous mechanism. Our solutions constrain attackers to attack through that narrow interface, thus protecting the ongoing connections.

However, there are still some questions that remain open. While we advocate full control over packets, neither of the

[3] Indeed, in the Internet white lists have to be pushed completely into the network to be effective, whereas black lists can be pushed only where required.

two realizations we propose achieve it. They approximate that notion by classifying packets based upon information in headers such as port numbers or IDs. The natural question that arises is how much control is necessary for hosts and at what cost will this control come. Another point is that our methods help hosts cope with packet floods directed at them, but do not protect the network itself. Ultimately, we need to identify the source of DDoS attacks and stop them at the entry points in the network.

We hope that our insights will influence the design of future networks to be robust to DoS attacks.

7 Acknowledgments

We thank Mark Handley, Kevin Lai, Sridhar Machiraju and Michael Walfish for useful discussions and comments about the paper. We also thank the anonymous reviewers for their insightful comments that helped improve the paper.

References

[1] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Towards a more functional and secure network infrastructure. Technical Report CSD-03-1242, UC Berkeley, 2003.

[2] D. G. Andersen. Mayday: Distributed filtering for Internet services. In *USITS*, Seattle, WA, 2003.

[3] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *Proc. of Hotnets-II*, Cambridge, MA, Nov. 2003.

[4] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Security Protocols—8th International Workshop*, Lecture Notes in Computer Science, Cambridge, United Kingdom, Apr. 2000. Springer-Verlag.

[5] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proc. of ACM SIGCOMM '99*, Cambridge, MA, Sept. 1999.

[6] S. Bellovin, M. Leech, and T. Taylor. ICMP traceback messages, Internet draft, Work in progress, February 2003.

[7] CERT Advisory CA-97.28. IP denial-of-service attacks, Dec. 1997.

[8] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *Information and System Security*, 5(2):119–137, 2002.

[9] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proc USENIX Security Symposium*, 2001.

[10] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. International Association for Cryptologic Research, Springer-Verlag, 1993.

[11] V. D. Gligor. A note on the denial of service problem. In *Proc. of 1983 Symposium on Security and Privacy*, pages 139–149. IEEE, 1983.

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

[13] H. Jamjoom and K. G. Shin. Persistent dropping: An efficient control of traffic aggregates. In *Proc. of ACM SIGCOMM '03*, pages 287–297, Karlsruhe, Germany, Aug. 2003.

[14] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. of the Symposium on NDSS*, 1999.

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

[16] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwith aggregates in the network. *ACM Computer Communication Review*, 32(3):62–73, July 2002.

[17] R. Merkle. Secure communication over insecure channels. *Commun. ACM*, 21(4):294–299, Apr. 1978.

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proc. of ACM SIGCOMM 2000*, pages 295–306, Stockholm, Sept. 2000.

[19] C. L. Schuba, I. V. Krsul, M. G. Kuhn, and E. H. Spafford. Analysis of a denial of service attack on TCP. In *Proc. of IEEE Symposium on Security and Privacy*, May 1997.

[20] C. Shields. What do we mean by network denial-of-service? In *Proc. of the 2002 IEEE Workshop on Information Assurance and Security*, West Point, NY, June 2002.

[21] D. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proc of IEEE INFOCOM*, 2001.

[22] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proc. of ACM SIGCOMM 2002*, pages 10–20, Pittsburgh, PA, Aug. 2002.

[23] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard AI problems for security. In *Eurocrypt*, 2003.

[24] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

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