DoS: Fighting Fire with Fire

Michael Walfish*, Hari Balakrishnan*, David Karger*, and Scott Shenker†

Abstract

We consider DoS attacks on servers in which attackers' requests are indistinguishable from legitimate requests. Most current defenses against this class of attack rely on legitimate users in aggregate having more of some resource (CPU cycles, memory cycles, human attention, etc.) than attackers. A server so defended asks prospective clients to prove their legitimacy by spending some of this resource. We adopt this general approach but use *bandwidth* as the constrained resource. Specifically, we argue that when a server is attacked, it should: (1) prevent overloading by limiting the incoming rate of requests (and dropping all others) and (2) encourage its legitimate clients to fight back with aggressive retransmission. This approach forces all clients to spend bandwidth to receive service, and the legitimate clients, with their greater aggregate bandwidth, will receive the bulk of the service.

1 Introduction

Our goal is to defend servers from Distributed Denial of Service (DDoS) attacks in which the attack traffic—usually generated by compromised and commandeered machines (*i.e.*, bots), often at the behest of extortionist [23, 26] criminal elements—mimics requests from legitimate clients, forcing the victimized server to spend much of its time on bogus requests. Examples of such attacks include using bots to: request large files from Web servers [24, 25]; make queries of search engines on Web sites [8]; and issue requests that cause computationally expensive operations (*e.g.*, database transactions) on Web sites [14]. Like the work in [11, 14], our proposed solution would be useful for servers whose scarce computational resources can easily be depleted by attackers without the server's access link being flooded. Such scarce resources include expensive database or application server licenses, CPUs, memory, and disks.

Because the attack traffic *looks* legitimate and is sent by hosts all over the Internet, network-layer and transport-layer DoS prevention tools—installing a box to filter out anomalies [3, 20], blacklisting particular IP addresses, using TCP SYN cookies [7], pushback [12, 18], etc.—do not work. Increasing the server's access bandwidth also doesn't help and in fact makes it easier for attackers to overload the server.

The general solution researchers have proposed is to ask all prospective clients to spend some scarce resource—a resource that the legitimate clients are presumed to have much more of than the attacking hosts. Conceptually, the server (or an external agent) sets a price in some currency, which slows the attack, allowing the legitimates to outspend the attackers and get service. Proposed currencies include: money [19] (clients give micropayment for server's attention); computational work[1] [1, 4, 5, 9, 10, 13, 19] (server won't serve until client completes a puzzle); and human attention [11, 14, 22] (Web server won't give service unless client solves a CAPTCHA [27]).

We adopt this general approach, but rather than invent another artificial currency, we propose to use *bandwidth* as the scarce resource. This approach, which we call *proof of net-work*, has two components. First, as is often done today, a middlebox in front of the server limits the rate at which requests reach the server, so that the server's resources do not become overloaded. If too many requests arrive, the middlebox drops some fraction of them on purpose, and the server continues to give good service to the requests that reach it.

Second, to avoid long timeouts, the middlebox immediately notifies clients when it has dropped their requests, and the clients immediately and automatically retransmit these requests. Each client is thus pushed into a high repeat-rate pattern typical of DDoS attackers. If the legitimate clients have more aggregate bandwidth than the attackers, then this approach of *fighting fire with fire* allows them to capture a large fraction of service. Note that rather than trying to reduce the rate at which attackers *can* send requests, this approach increases the rate at which legitimate users *do* send requests. In essence, the server asks the legitimate clients to fight a war of attrition with the bots; the legitimates can afford to keep retrying whereas the bots have a more limited bandwidth budget.

We will argue that using bandwidth as a constraint is, in some ways, more natural than other DDoS defenses (see §2.1,§6). Of course, it raises important issues, such as the apparent abuse of a communal resource (the network) and how to control congestion. In later sections, we discuss these questions and show how to integrate net-work into today's applications. For now, we defer these issues to outline and analyze an idealized version of net-work.

*MIT CSAIL, {mwalfish,hari,karger}@csail.mit.edu
†UC Berkeley and ICSI, shenker@icsi.berkeley.edu

[1] Laurie and Clayton [17] argue against proof-of-work as a spam defense; in the appendix, we say how their work relates to our proposal.

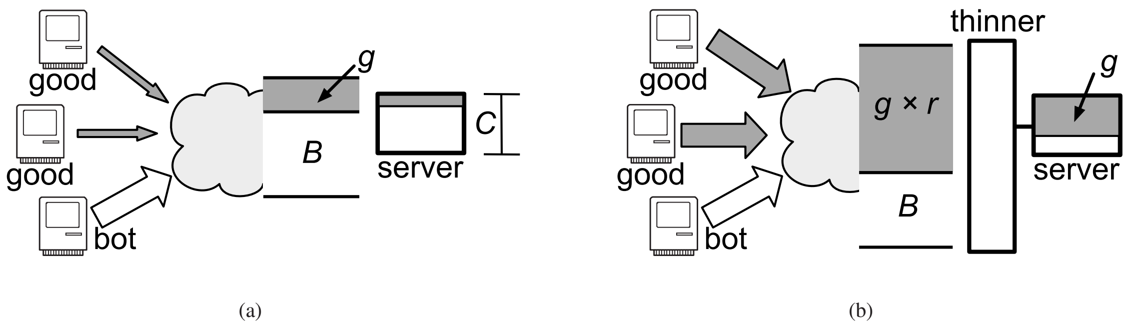

Fig. 1: An attacked server, $B + g > C$, (a) without net-work (b) with net-work. Net-work requires every client to spend bandwidth; good clients can afford this price. The traffic from good clients is shaded, as is the portion of the server spent on their requests.

2 Overview and Analysis of Net-work

A particularly problematic aspect of current DDoS attacks is that a small number of bad clients can deny service to a far larger number of legitimate clients. This denial happens because each bad client may make requests at rates far outstripping normal clients. At the same time, we observe that *some* limiting factor must prevent the bad clients from sending even more requests—otherwise, they could deny service more effectively with fewer machines. We posit that in many cases, this limiting factor is bandwidth and that bad clients exhaust all their available bandwidth generating spurious requests for service. In contrast, good clients who spend substantial time quiescent are likely using only a small portion of their available bandwidth.

We aim to change this balance by increasing the bandwidth required for a request to reach a server under attack. Such a change ought to decrease the number of requests sent from a (currently maxed-out) bad client that reach the server, while good clients with spare bandwidth will hopefully be able to absorb the cost.

2.1 Idealized Net-work

We first discuss an idealized approach to net-work and later describe extensions to make it more realistic. While net-work applies to a variety of servers, the idea is best illustrated with simple request-response servers where each request is cheap for clients to issue but expensive to serve. Real-world examples include single-packet Web requests, load-balancing DNS servers used by content distribution networks, and NFS servers.

Suppose the server has the capacity to handle requests at some known rate C, which is larger than the rate g at which "good" or legitimate clients submit requests. At the same time, a set of "bad" or attacking clients submits requests at rate b. Good clients may be denied service when $g + b > C$; often, $b \gg C$. Let G and B denote the total bandwidths (measured in requests per time unit) available to the good and bad clients. (G and B are not physical link capacities but rather "disposable bandwidth", reflecting constraints like the need for bots to avoid detection.) We pessimistically assume that the bad clients spend everything they can, *i.e.*, that $b = B$. On the other hand, we posit that $G \gg g$, *i.e.*, that the good clients actually have substantially more bandwidth to spend on the given service than they are using to submit requests.

Figure 1(b) shows an idealized implementation of net-work. First, a *thinner* (which for now we assume is a single machine but will generalize later) is a front-end to the server. It winnows traffic from the inbound rate to C by randomly dropping excess requests.[2] Second, the thinner does not drop requests silently—rather, on each drop, it sends a synchronous "*please retry*" message to the client, causing the client to retry the request, *i.e.*, to do more net-work.

This scheme builds on the well-known principle of shedding requests to cope with overload, but with a crucial difference. Today, when a good client fails to get a response, it retries, often with human involvement and always after a lengthy timeout. With net-work, however, a client retries automatically as soon as it gets the "please retry". Since this call to retry usually occurs within one round-trip, the legitimates can dramatically increase their retry rates and drown out the attackers.[3]

Below we derive the expected number of retries, r, that good clients will need to do, and we show that r automatically changes with the attack size. For now, observe that r can be viewed as a price, with bandwidth as the currency, and that the thinner does not communicate r to the clients—instead, good clients keep resending until they get through. The "natural" emergence of this price, we argue, contrasts with other currency-based schemes. With other currencies, the server must check clients' payment (*e.g.*, by verifying a puzzle solution) and also must report an explicit price (*e.g.*, by sending a puzzle with a certain degree of hardness) or have the clients guess the current price. See §6.1 for more comparison.

[2] C need not be known; however the details of this case are beyond this paper's scope, so we assume throughout that C is known.

[3] One reason for long timeouts and exponential backoff in current retry methods is congestion control; we show in §3.1 how clients perform proper congestion control by using these explicit "please retries" to distinguish between server overload and congestion.

2.2 Analysis and Requirements

We now consider the price, or expected number of retries, r, for legitimate clients. Observe that the thinner, which needs to know only C, admits incoming requests with probability p to make the total load reaching the server C. Legitimate clients thus have to retry $r = 1/p$ times on average before getting through. The "bad load" that actually reaches the server reduces from B, the attackers' full budget, to Bp. Thus, the thinner's dropping policy, combined with the fact that good clients retry their "good load" of g until getting through, results in the equation $g + Bp = C$, which implies $r = 1/p = B/(C - g)$. Note that r changes with the attack size, B.

For net-work to be most useful, legitimate clients must be able to afford the inflation of their requests, *i.e.*, gr must be smaller than G, the total bandwidth available to the good clients. Plugging r into the inequality $gr < G$ yields a *server provisioning* requirement: $C > g(1 + B/G)$. This requirement can be explained as follows: to use net-work, the server must have enough capacity to serve both the good clients, g, *and* the diminished bots, $B(g/G)$. The bots originally sent at rate B, and good clients originally used a fraction g/G of their bandwidth. Thus, from the server's perspective, each bot has become only as threatening as a good client; a bad client can no longer amplify its impact on the server. In particular, if each individual bad client has the same amount of bandwidth as each individual good client, then, from the *server's* perspective, net-work makes every bot look like a good client. From the *network's* perspective, net-work makes every good client look like a bot in terms of the number of packets sent.

2.3 When is Net-work Useful?

Net-work is only needed if $g + B > C$ and can only prevent overload when $C > g(1 + B/G)$. If $G \gg B$ then the required over-provisioning (relative to the good load g) is minor. If $G \approx B$ then the server has to be over-provisioned by some small multiple of g, which may not be feasible in some situations. If $G \ll B$ then C must be roughly gB/G, which is far larger than what is needed for the legitimate load (by a factor of B/G) but is far smaller than the capacity ($g + B$) that would have been needed to absorb the attack without net-work (by roughly a factor of g/G). However, even if the server is under-provisioned, the fraction of service spent on good requests goes from $g/(g+B)$ without net-work to $G/(G+B)$ with net-work. Because we expect that $G \gg g$ always holds, this change represents a significant improvement even in the case of enormous attacks.

The requirement that the server be large for net-work to *guarantee* service to all legitimate clients may seem unreasonable, but it applies to all currency-based schemes: a server unprepared to treat every bot as a single legitimate client can always be overloaded (see §6.1). Indeed, one can do better only by explicitly distinguishing legitimate users from bots, which may not always be feasible (see §6.2).

3 Toward Practical Net-work

The idealized solution in the previous section made several unrealistic assumptions:

A1 The network, excluding all access links, has infinite capacity and never becomes congested.

A2 The thinner is provisioned to have infinite capacity, *i.e.*, its access link and processing capacity allow it to respond to every arriving request.

A3 The latency from retrying requests is not problematic.

A4 The server does the same amount of work for each request.

In this section, we address these assumptions in turn.

3.1 Effects on the Network

Assumption A1 is obviously wrong, and we now discuss how to account for limited bandwidth. When a client retries aggressively, it should perform proper congestion control. Fortunately, the thinner's "please retry" message gives the necessary feedback to uphold the principle of packet conservation: the client retries only in response to this message, and the thinner sends the message only when it receives a request. If the thinner does not receive a request, or if the thinner's message is lost, then the client must back off because the loss could have been due to congestion. The details of the back-off depend on the transport protocol; UDP-based RPC (*e.g.*, DNS) would have a multi-second back-off, while a TCP connection between client and thinner would automatically ensure correct congestion response.

For this congestion-controlled approach to thwart DoS attacks, the thinner's incoming and outgoing links need to be uncongested. Otherwise, legitimate clients will back off, ceding the server to attackers. Below, in §3.2, we say how to provision the thinner accordingly.

By congesting intermediate links, attackers can certainly cause legitimate clients to back off in response to network congestion. However, net-work neither introduces nor facilitates such link attacks: they can and do happen today. (Of course if net-work is effective in preventing server attacks, it might cause attackers to turn to link attacks instead, but these will be no more harmful than they currently are.) One might object that net-work increases the overall traffic load significantly. However, only traffic involving a server under attack actually expands; if, as we expect, the fraction of such servers at any time is small, then the overall traffic increase is small.

3.2 Provisioning the Thinner

Because clients treat unanswered requests as in-network congestion, clients' requests must not be lost *at* the thinner. The thinner must therefore have enough bandwidth and processing power to send "please retry" signals in response to all arriving requests, both adversarial and good (and the good traffic is inflated from retries). However, the owners of most servers cannot afford a host with this much processing power and connectivity. Instead, a cost-effective solution is for groups of servers to *aggregate*, sharing a thinner. Data centers today co-locate Web servers, with traffic to these servers taking common links; thus, a thinner could be shared there. More generally, a server could buy "thinning" from a provider, which in turn can use an overlay. In this case, protected servers would need to filter traffic not vetted by the thinning service; servers with thin access links would need router support for this filtering. We leave the detailed design of such an overlay-based thinning service—which is similar in spirit to [2, 16, 22]—to future work.

3.3 Latency

In the idealized solution, a client's expected latency is r round-trips between it and the thinner. There are two problems: the latency is large, and it has high variance ($\approx r^2$ RTTs) because of the random process. To address both problems, the thinner can announce r to the client, which then sends $r-1$ retries over a reliable, congestion-controlled stream. After the client sends these $r-1$ retries, the thinner passes the client's request to the server.

With this approach, the thinner must compute the price r explicitly. One possible price computation is to divide time into epochs and in epoch t to charge $r_t = \frac{I_{t-1}}{C}$, where I_{t-1} is the inbound request rate. ($I_{t-1} = B_{t-1} + g r_{t-1}$, but the thinner does not know the contribution from each term.) In the simple case when B_t is constant (*e.g.*, attackers spend all disposable bandwidth), we can prove that r_t converges to the price from §2.2, $r = \frac{B}{C-g}$. If attackers modulate their traffic, the price can fluctuate. To prevent attackers from gaming the scheme, we propose to measure I_t over a random number of past time epochs, leaving a detailed investigation of this issue to future work. Observe that this deterministic price preserves the "naturalness" of bandwidth as currency: the price is estimated directly from the incoming packet rate.

If the thinner can signal only "please retry" (rather than an amount, $r-1$), it can still enforce r retries, which reduces the variance from the idealized solution.

3.4 Servers with Unequal Tasks

Assumption A4 is not true in general. For servers whose work is not constant for each request, net-work, as described so far, will rightly crowd out adversaries, but the few adversarial requests getting through may consume much of the server's resources. This problem is not specific to net-work; Web services—either unprotected or protected with other DDoS prevention schemes—already have this problem.[4]

```
<head>
<meta http-equiv="refresh" content=0>
</head><body>Please wait.</body>
```

Fig. 2: HTML refresh can implement the "please retry" signal.

To protect such servers with net-work, the server's owner must choose: either pessimistically "charge" for every request as if it were the "hardest", or price requests differentially. With the former, the server must be provisioned as though every request were hard. This choice results in over-provisioning in the common case when the server's load is a proper mix of requests.

With differential pricing, if the thinner can discern the difficulty of requests, and if the load profile (*i.e.*, what fraction of requests have a given difficulty) when the server is not under attack is known, the thinner can extract payment "up-front". The thinner simply drops incoming requests at rates to preserve the profile. If the thinner cannot discern the difficulty of requests, then the server should demand ongoing retries while serving the request, being prepared to abort if it does not receive retries. To have a price emerge naturally, the thinner can again divide time into epochs and demand the "natural" price in each epoch (as in §3.3). However, there are many details to work out on how to implement such a scheme, and we will be investigating them in future work.

4 Deployment and Applications

We now show how net-work fits into several network applications, requiring few or no client or server changes.

4.1 Protecting a Web Server

The thinner here is a front-end proxy to the server that replies to requests with a simple "HTML refresh" page, as in Figure 2. This page is the "please retry" signal since the `meta` line causes the HTML client to send another HTTP request immediately. The request-retry process is repeated until the proxy probabilistically admits the client or extracts an explicit price, at which point the proxy relays the client's original request to the server.

This approach requires the client to wait several round-trips. That latency may be a few seconds but may still be much better than an overloaded server under attack. If a server operator wants to further reduce latency for clients, the thinner can send an HTML page with multiple embedded objects, the number of objects reflecting the current price. The client would have to retrieve

[4]Even with CAPTCHAs, one could hire thugs or cheap labor to solve them and to then submit very expensive queries to a Web site.

each of those objects before getting service. Or, the thinner can cause the client to run JavaScript that submits a lengthy POST, the size reflecting the desired payment. Observe that these approaches implement the idea from §3.3 of communicating an explicit price, and they also respect congestion control.

4.2 Protecting UDP-based RPC Systems

A UDP-based RPC system can use net-work by adding the "please retry" message to the published interface. A richer modification is to ask the client to "retry $r - 1$ times". Now, the client would have to implement congestion control while sending those retries "at once"; in practice, the client could use a congestion manager [6] or DCCP [15].

The legacy DNS protocol can yield the desired behavior: the thinner issues a DNS referral to itself for a name with many components, the number of components reflecting the desired price. After the referral has been resolved, the thinner sends the request to the protected DNS server. DNS servers that might need such protection are those doing non-trivial work for each client, *e.g.*, load-balancing mappers used by content distribution networks. We note that with this approach, any distributed system with "request referrals" can use net-work—possibly with no client changes.

5 Critique and Objections

We posited that attackers' bottleneck resource is *upload* bandwidth, and our premise is that the scarce server resource is computational (*e.g.*, CPU or database license). If we are wrong—which we plan to check with further study—the analysis in §2.2 still applies (B and G can refer to download capacity, C to the server's outbound bandwidth), though the "please retry" must instead be a request for the client to *download* something, *e.g.*, a large dummy object.

We now state what we view as the two most serious objections to net-work. First, clients with low outgoing bandwidth will be penalized with longer latencies. This bias against low-bandwidth clients arises in part because the range of bandwidths is significant (and larger than the range of memory speeds, used in computational work schemes). Second, net-work will cost clients money if their ISPs charge per byte. We do not yet have satisfactory responses to these objections.

6 Net-work Compared to Other Defenses

6.1 Relation to Currency Schemes

In some currency-based approaches, the price r for each request is stated explicitly; in others [28], service is auctioned off so that the price is discovered dynamically. In our context, the price, r retries, also emerges dynamically, as showed in §2.2. The analysis presented there applies to any other currency; it shows that if good clients can afford to pay the currency at rate G while bad clients can afford B, then one can limit requests into the server to rate $g(1 + B/G)$ while still giving service to legitimate clients. Furthermore, *all* currency schemes have the properties stated in §2.3, namely that the scheme is most effective when the server is provisioned for the above rate and when $G \gg B$—but that the scheme is still useful when these requirements are not met.

In this paper, we proposed bandwidth as a resource with $G \gg B$. We do not claim that it is strictly better than all other proposals, but we do think that it is a particularly natural currency, as argued in §2.1. Extending the arguments there, we note that if the price in another currency is wrongly set, causing the server to drop excess demand, then the client must adjust to pay a higher price (*e.g.*, by solving a harder puzzle [28]) and *must then send a retry*. In other words, bandwidth always plays a role in currency-based schemes. Also, under net-work, clients' work is observable; in contrast, puzzles might be broken, and the server cannot tell when such breakage happens.

We do not know of another proposal to use bandwidth as a currency. Gligor [11] does observe, however, that client retries and timeouts require less overhead while still providing the same qualitative performance bounds as proof-of-work schemes. Because the general approach does not meet his more exacting performance requirements, he neither suggests asking clients to retry aggressively, nor considers the mechanisms and extensions in §3–4, nor explores the ramifications of actually deploying the approach in today's Internet.

6.2 Human Attention and Other Defenses

Whereas currency-based defenses grant *some* access to bots, schemes [11, 14, 22] that use CAPTCHAs [27] to tell apart bot and human grant *no* access to bots when the server is heavily loaded. These solutions are not right for our purposes because they assume an express preference for human clientele. Also, giving humans a CAPTCHA requires a behavioral change, yet many humans (26%, in [14]) may not want to make this change. Kill-Bots [14] strives to address these limitations by first trying to identify bots—defining a bot as a source IP address that retries too many times without answering a CAPTCHA—and then filtering them out, at which point the server will no longer be loaded and any legitimate client can gain access. Kill-Bots can redress the biases above under the assumptions that the server is properly provisioned, that bots can be identified by their retry pattern, and that each bot can receive traffic at no more than a few IP addresses.

For lower-level DDoS defenses not based on currency or human attention, see the survey by Mirkovic and Reiher [21] and the bibliographies in [14, 22, 29].

7 Conclusion

Attackers are now consuming expensive server resources in ways that mimic legitimate users. The frustration of server owners is palpable, yet their options are paltry: overprovision vastly, adopt a bias toward humans answering CAPTCHAs, yield to extortion [23, 26], or demand that clients somehow pay for access. We have taken the last approach, and we proposed bandwidth as the currency. Bandwidth's advantage is simplicity and deployability. Its disadvantage (besides those in §5) is that it is not a fully local resource, unlike the CPU in proof-of-work. Nevertheless, viewing backbone bandwidth as free may be a reasonable first approximation (see §3.1).

This view—coupled with defending servers by asking good clients to crowd out attackers—may look like Internet vigilantism. However, net-work upholds important principles (*e.g.*, respect for congestion control and limiting clients' claims on servers). Thus, we believe it merits consideration by the community and investigation by us. Building on our initial backward-compatible design for the Web and DNS, we plan to conduct measurements to check our premises, estimate typical values of B and G, further develop a control scheme to estimate the retry price dynamically, experiment with a working prototype, and incorporate net-work into other legacy systems.

Acknowledgments

We thank J.D. Zamfirescu for his contributions in discussions of this work. Excellent comments by David Andersen, Russ Cox, Nick Feamster, Jaeyeon Jung, Brad Karp, Maxwell Krohn, Karthik Lakshminarayanan, Rodrigo Rodrigues, Sara Su, and Mythili Vutukuru improved this paper. This work was supported by the NSF under grants ANI-0225660 and CNS-0520241, by an NDSEG Graduate Fellowship, and by British Telecom.

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *NDSS*, 2003.

[2] D. G. Andersen. Mayday: Distributed filtering for Internet Services. In *USITS*, Mar. 2003.

[3] Arbor Networks, Inc. http://www.arbornetworks.com.

[4] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Intl. Wkshp. on Security Prots.*, 2000.

[5] A. Back. Hashcash. http://www.cypherspace.org/adam/hashcash/.

[6] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *SIGCOMM*, Aug. 1999.

[7] D. J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html.

[8] Criminal Complaint: USA v. Ashley, Hall, Schictel, Roby, and Walker, Aug. 2004. http://www.reverse.net/operationcyberslam.pdf.

[9] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *CRYPTO*, 2003.

[10] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.

[11] V. D. Gligor. Guaranteeing access in spite of distributed service-flooding attacks. In *Intl. Wkshp. on Security Prots.*, 2003.

[12] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *NDSS*, 2002.

[13] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[14] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *USENIX NSDI*, May 2005.

[15] E. Kohler, M. Handley, and S. Floyd. Datagram congestion control protocol (DCCP). draft-ietf-dccp-spec-11.txt, IETF draft (Work in Progress), Mar. 2005.

[16] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. In *HotNets*, Nov. 2003.

[17] B. Laurie and R. Clayton. "Proof-of-Work" proves not to work; version 0.2, Sept. 2004. http://www.cl.cam.ac.uk/users/rnc1/proofwork2.pdf.

[18] R. Mahajan et al. Controlling high bandwidth aggregates in the network. *CCR*, 32(3), July 2002.

[19] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *Proc. IEEE ACSAC*, Dec. 2001.

[20] Mazu Networks, Inc. http://mazunetworks.com.

[21] J. Mirkovic and P. Reiher. A taxonomy of DDoS attacks and DDoS defense mechanisms. *CCR*, 34(2), Apr. 2004.

[22] W. Morein et al. Using graphic turing tests to counter automated DDoS attacks against web servers. In *ACM CCS*, Oct. 2003.

[23] *Network World*. Extortion via DDoS on the rise. May 2005. http://www.networkworld.com/news/2005/051605-ddos-extortion.html.

[24] E. Ratliff. The zombie hunters. The New Yorker, Oct. 10 2005.

[25] SecurityFocus. FBI busts alleged DDoS mafia. Aug. 2004. http://www.securityfocus.com/news/9411.

[26] *The Register*. East European gangs in online protection racket. Nov. 2003. http://www.theregister.co.uk/2003/11/12/east_european_gangs_in_online/.

[27] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *CACM*, 47(2), Feb. 2004.

[28] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *IEEE Symp. on Security and Privacy*, May 2003.

[29] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM*, Aug. 2005.

Appendix: Proof-of-Work for Spam Control

Laurie and Clayton [17] argue that proof-of-work for spam control—wherein the sender attaches to each e-mail a proof that it spent cycles on that e-mail and the receiver checks this proof—is "not a viable solution to the spam problem" [17]. Given their analysis, we must ask why bandwidth is a viable DDoS defense. A comprehensive answer is outside our scope; two general observations follow. First, Laurie and Clayton's security analysis assumes a goal of reducing spam to 1% of legitimate mail. For DDoS, our goal need not be as stringent: we believe server operators would be happy if their servers spent less than 10% of their resources on bots. Given this order-of-magnitude difference, we conjecture that there is headroom between what legitimate clients are limited to and what they actually want to spend. Second, we further conjecture that most legitimate clients' use of upload bandwidth on DDoS-prone services (versus on peer-to-peer applications) is relatively rare and could thus expand without significant collateral damage.