COLIN DIXON, THOMAS ANDERSON, AND ARVIND KRISHNAMURTHY

# withstanding multimillion-node botnets

Colin Dixon is a graduate student at the University of Washington. While an undergraduate at the University of Maryland he worked on approximation algorithms and anonymous communication. His current research interests include computer security, network architecture, and distributed systems with a focus on deployable solutions for real-world problems.

*ckd@cs.washington.edu*

Tom Anderson is a Professor in the Department of Computer Science and Engineering at the University of Washington. He is an ACM Fellow and a winner of the ACM SIGOPS Mark Weiser Award, but he is perhaps best known as the author of the Nachos operating system.

*tom@cs.washington.edu*

Arvind Krishnamurthy is an Assistant Research Professor at the University of Washington, Seattle. His research interests are primarily at the boundary between the theory and practice of distributed systems. He has worked on automated mechanisms for managing overlay networks and distributed hash tables, network measurements, parallel computing, techniques to make low-latency RAID devices, and distributed storage systems that integrate the numerous ad hoc devices around the home.

*arvind@cs.washington.edu*

LARGE-SCALE DISTRIBUTED DENIAL OF service (DoS) attacks are an unfortunate everyday reality on the Internet. They are simple to execute and, with the growing size of botnets, more effective than ever. Although much progress has been made in developing techniques to address DoS attacks, no existing solution handles non-cacheable content, is unilaterally deployable, works with the Internet model of open access and dynamic routes, and copes with the large numbers of attackers typical of today's botnets. We believe we have created a practical solution.

## Setting the Stage

The current Internet is often compared to the Wild West and not without merit. A combination of the lack of accountability, the complexities of multiple legal jurisdictions, and an ever-changing technological battlefield has created a situation where cyber-criminals can operate lucrative businesses with little risk of being caught or punished.

The most brazen example of this is the growth of botnets. Attackers write viruses that compromise end hosts and tie them into a command and control system that enables the attacker to issue commands, install software, and otherwise control compromised machines. These networks are the basis for a whole underground economy in stolen financial information, stolen identities, spam email, and DoS attacks.

The size of these botnets is large and growing. A variety of recent estimates put the total number of bots on the Internet well into the millions and some estimates go upward of hundreds of millions [3, 5]. Recent examples including the Storm and Kraken botnets have made headlines in mainstream media. To make matters worse, the number of critical operating system vulnerabilities discovered is increasing steadily every year [2], giving botnets an ample supply of new recruits, so the problem is unlikely to get better on its own.

DoS attacks launched from botnets number in the hundreds each day and threaten large swaths of the Internet. If compromised nodes are typical of end hosts participating in other large peer-to-peer systems [10], a multimillion-node botnet would be able to generate terabits of attack traffic per second, sourced from virtually every routable IP prefix on

the planet. This scale of attack could, at least temporarily, overwhelm any current core link or router. It is also capable of indefinitely disrupting service to all but the best provisioned services on the Internet today.

In May 2006, a sustained attack against the anti-spam company Blue Security forced the company to close down its services [12]. The LiveJournal community was even knocked offline when it was caught in the crossfire. Further, attacks are not just limited to security companies. In April 2007, a sustained attack on government and business Web sites in Estonia effectively knocked the country off the Web [9]. Even nations are not safe. More disturbing is that, despite the attacks going on for weeks, no effective countermeasures were deployed and service was restored only after the attacks petered out.

## DoS Attacks

Although DoS attacks come in many flavors, we focus on resource exhaustion attacks. These attacks flood some bottleneck resource with more requests than can be handled, ensuring that only a small fraction of the legitimate requests are serviced.

In the past, syn floods and other techniques aimed to exhaust end-host resources such as memory and process table space, but as these vulnerabilities have been fixed, increasingly the trend is simply to exhaust the target's bandwidth. Attack packets can emulate normal user behavior, making them difficult to detect and drop. Attack traffic often crowds out legitimate traffic upstream from where the victim has power to filter traffic even if it could distinguish good packets from bad. Further, with the increasing size of botnets attackers can simply send normal traffic and make the attack indistinguishable from a flash crowd, forcing defenses to drop packets at random.

Without information about which requests are legitimate and with limited buffer space, the only strategy for a victim is to serve requests at random. If there are G legitimate requests and B spurious requests, on average, O([G/(B+G)]) of the available resources go to legitimate requests. But B is often much larger than G, since, with a massive botnet, attackers can pick their target and focus their fire. Addressing this asymmetry is a main goal of our work.

## State of the Art

There are two sets of deployed solutions today. The first involves heuristic-based filters deployed in special-purpose boxes in the network with the goal of finding and dropping the bad traffic. The second set makes use of large-scale content distribution networks (CDNs), which aim to solve the problem by sheer over-provisioning.

Heuristic-based filtering relies on an arms race between attackers and defenders—one that we are unlikely to win. Attackers are faster on their feet and, in the end, have an easier goal. They only have to make their traffic indistinguishable from the legitimate clients, whereas the filters have to detect—in real time and at high data rates—the ever-shrinking differences between the attack traffic and legitimate traffic.

Additionally, because heuristics can cause collateral damage, they are only activated in response to an attack. This requires both detecting the attack and communicating that fact to the upstream filters. This communication itself can be interrupted by the attacker.

Large-scale content distribution networks, however, work remarkably well for read-only Web sites, by replicating data everywhere. Massive replication not only increases performance and resilience to flash crowds but also provides extra capacity to deal with a DoS attack. This approach is even available as a commercial anti-DoS service [1]. However, CDNs do not work for nonreplicable services, such as read/write back-end databases for e-commerce, modern AJAX applications, e-government, and multiplayer games, or for point-to-point communication services such as VoIP or IM—in other words, much of the Internet as we know it today. How can we ensure communication with a fixed endpoint when that endpoint is being flooded?

We address this problem with the key insight that we need a system as powerful as a botnet to defend against a botnet.

## Phalanx Architecture

Intuitively, Phalanx aims to use a large swarm of nodes (like those of a large-scale CDN) as points of presence for a protected server. Provided that the nodes' resources exceed those of attackers, legitimate clients will have some functioning channels for communication despite a widespread attack. In practice, the implementation of this, which allows for a reasonable deployment path, good performance, and an open model of communication, is somewhat more complicated than simply using nodes as proxies.

There are two key problems with simply using CDN nodes as proxies. First, these nodes cannot simply forward all traffic onto the server; instead, they have to do some kind of filtering at the behest of the server. Second, it is only in aggregate that the CDN nodes are resistant to attack, so any given connection must leverage a large set of these nodes to be resilient.

To solve the first problem we use the nodes as packet mailboxes rather than simple forwarding proxies. A mailbox in Phalanx is a best-effort packet store and pick-up point. The protected server must explicitly request each packet it wishes to receive and therefore is fail-safe: If a server doesn't request a packet, the packet is not delivered. These mailboxes are further explained below.

To solve the second problem, we send each packet through a different randomly chosen mailbox, thus drawing in a large number of mailboxes to protect each connection. If any given mailbox fails or is attacked, only a small fraction (often only one packet) of the connection will be lost. Because the mailbox used by a given packet is chosen randomly according to a seed known only to the two endpoints, the attacker must attack widely to have any impact. The exact mechanisms for this can be found below, in the section "Swarms and Iterated Hash Sequences."

The observant reader will note two remaining problems. First, there is nothing to stop an attacker from ignoring the mailboxes and attacking the server directly. To deal with this, we capitalize upon the request-response framework and install filters at the edge of the server's upstream ISP. These filters (whose functionality is described later as the filtering ring) simply drop all unrequested packets.

Second, now that we have blocked all unrequested packets, how can we initiate connections? For this, we extend the request-response framework and additionally send requests for new connections rather than explicit packets. These requests are a valuable scarce resource and so we protect access to them via authentication and fair queuing (as laid out under "Connection Establishment").

## AN EXAMPLE

Before launching into each mechanism in detail we will narrate an example of how Phalanx would be used to protect a standard Web server with dynamic content, such as an e-commerce site. The example can also be followed in Figure 1, where the numbered steps will be mentioned.

First, the client looks up the address of the server and subsequently requests the static, cacheable content of the page via any current CDN-style system with high availability, such as Akamai, CoDeeN, or Coral (1). As part of fetching this content, the client receives a static and cacheable Java applet, which then serves as a zero-installation client to allow for interaction with Phalanx mailboxes. At this point, the Java applet is responsible for rendering the dynamic, noncacheable portions of the page and speaking the Phalanx protocols.

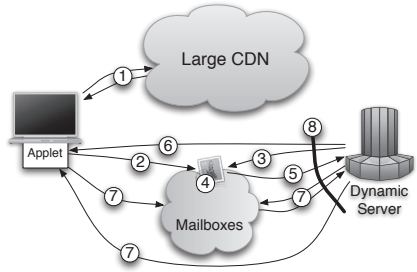


**FIGURE 1: A DIAGRAM ILLUSTRATING A SIMPLE HTTP-STYLE REQUEST DONE WITH PHALANX. THE NUMBERS CORRESPOND TO THE ACCOMPANYING DESCRIPTION.**

The applet begins by making a name request for the dynamic content server to the distributed name service (1). Again, because the naming information is static and cacheable, this service can be provided by any highly available name service, such as CoDoNs or Akamai's DNS service. The name service returns a list of "first-contact" mailboxes. These first-contact mailboxes hold the first packet requests that the server has issued to allow new connections to be made.

The applet requests a challenge from one of these first-contact mailboxes and replies with either a puzzle solution or an authentication token (2). In either case, the applet will resend the request, possibly with a more complex puzzle solution and/or a different mailbox if the connection is not established in a reasonable period of time.

At the mailbox, a steady stream of first packet requests has been arriving from the dynamic content server (3). One of these first packet requests is eventually assigned to the client's connection request (4), at which point the applet's request is forwarded to the server (5) to cross back through the filtering rings (8) without being dropped. This ensures that the rate of connection requests reaching the server is under the server's control.

Eventually, a response will come back from the server (6) containing a list of mailboxes to use for the remainder of the connection along with a shared secret allowing standard Phalanx communication to commence. At the same time, the server will send packet fetch requests to the first several mailboxes to be used in preparation for receiving further packets from the client.

The client uses the shared secret to determine the sequence of mailboxes to use and begins to send packets to these mailboxes. These data packets are paired with their corresponding requests and forwarded onto the server passing through the filtering ring (8) by virtue of the holes opened by the requests. This constitutes the normal behavior of the Phalanx connection (7).

If at any point in time the server decides that the connection is no longer desirable or it simply starts running low on resources, it can either decrease the rate at which it requests new data packets or simply stop requesting packets altogether.

## MAILBOXES

We now proceed to describe each of these components in a bit more detail. The basic architecture of an established Phalanx connection is shown in Figure 2. A more complete description is available on the USENIX Web site in the proceedings of NSDI '08 [8].

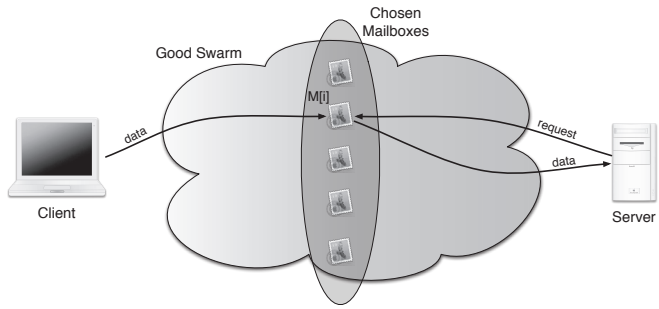


**FIGURE 2: THE BASIC ARCHITECTURE OF AN ESTABLISHED PHALANX CONNECTION. EACH PACKET IS SENT THROUGH A ONE-HOP DETOUR VIA A RANDOMLY CHOSEN MAILBOX.**

The basic building block in Phalanx is the packet mailbox. Mailboxes provide a simple abstraction that gives control to the destination instead of the source. Rather than packets being delivered directly to the specified destination as in previous anti-DoS overlays [4, 11, 17], traffic is first delivered to a mailbox, where it can either be "picked up" or ignored by the destination. Traffic that is ignored is eventually dropped from the buffers at packet mailboxes.

Mailboxes export two basic operations: `put` and `get`. A `put` inserts a packet into the mailbox's buffer, possibly evicting an old entry, and returns. A `get` installs a best-effort interrupt at the mailbox. If a matching packet is found before the request is bumped from the buffer, the packet is returned.

The mailbox abstraction puts the destination in complete control of which packets it receives. Flow policies can remain at the destination where the most information is available to make such decisions. These policies are implemented in the network via requests and the lack thereof. If no requests are sent, then no packets will come through. This behavior ensures that most errors are recoverable locally, rather than requiring cooperation and communication with the network control plane. This is in contrast to accidentally installing an overly permissive filter in the network and then being unable to correct the problem because the control channel can now be flooded.

## SWARMS AND ITERATED HASH SEQUENCES

Individual flows are multiplexed over many mailboxes. Each packet in a flow is sent to a cryptographically random mailbox. Since each mailbox is secretly selected by the endpoints, an attacker cannot "follow" a flow by attacking each mailbox just before it is used.

We construct a pseudo-random sequence of mailboxes during connection setup by exchanging the set of mailboxes $M$ and a shared secret $x$. The se-

quence of mailboxes is built by iterating a cryptographic hash function, such as SHA-1 or MD5, on the shared secret. Equipped with this shared sequence, both endpoints know in advance the precise mailbox to use for each packet in the connection.

To construct a sequence of mailboxes, we first define a sequence of nonces $x_i$ based on the shared secret $x$ and the cryptographic hash function $h$, as follows:

$$x_0 = h(x||x)$$

$$x_i = h(x_{i-1}||x)$$

Including $x$ in every iteration prevents an attacker who sniffs one nonce from being able to calculate all future nonces by iterating the hash function themselves. Our current implementation uses MD5 [15] as the implementation of $h$ and thus uses 16-byte nonces for simplicity. This sequence of nonces then determines a corresponding sequence of mailboxes $M[x_i]$ by modulo reducing the nonces, as follows:

$$M[x_i] = M_{x_i \bmod |M|}$$

Note that M need not be all mailboxes in the Phalanx deployment, as each flow can use a subset of the mailboxes. Indeed, a different set of mailboxes can be used for each half of the flow (client-to-server and server-to-client); both sets can be dynamically renegotiated within a flow.

Each nonce serves as a unique identifier for a packet and is included in the header to facilitate pairing each incoming packet with its corresponding request. Thus the receiver can know precisely which source sent which packet. Further, including a nonce in each packet simplifies the logic needed to drop unrequested packets. Lastly, nonces provide a limited form of authentication to requests; to subvert the system the attacker must snoop the nonce off the wire and then deliver a replacement packet to a mailbox before the correct packet arrives.

## FILTERING RING

With Phalanx, a protected destination only receives those packets it explicitly requests from a mailbox. To enforce this, we drop all other packets for the destination at the edge of its upstream ISP.

Each request packet carries a unique nonce that allows a single data packet to return. In the simple case of symmetric routes, the border router records the nonce on the outgoing packet and matches each incoming packet to a recently stored nonce, discarding the packet if there is no match. Each nonce is single use, so once an incoming packet has matched a nonce, we remove that nonce.

To be effective, the filtering ring must be comprehensive enough to examine every packet destined for a protected destination, regardless of the source of the traffic. To prevent this an attacker might try to flood the border router (or, more precisely, the link immediately upstream from the border router). As we observed earlier, a massive botnet may be able to flood any single link or router in the network. However, this would disconnect only those mailboxes that used that specific router to access the destination; other mailboxes would continue to deliver packets unaffected.

Even a multimillion-node botnet would be unable to sustain enough traffic to completely disconnect a tier-1 ISP from the Internet. To have an effective defense against such a large-scale attack, a destination must either be a direct customer of a tier-1 that provides a filtering ring or be protected in-

directly, as a customer of an ISP that is a customer of that tier-1. Since each connection can spread its packets across a diverse set of mailboxes, connections might experience a higher packet loss rate during an attack, but otherwise would continue to make progress.

Deploying the filtering ring at a tier-1 has risks, though. Bots are everywhere—even inside corporate networks—and, as a result, it seems likely that filtering rings would be deployed in depth. Inner layers would provide protection against the limited number of potential attackers close to a server, while outer layers would provide the powerful filtering to deal with the brunt of larger attacks. Initially, small-scale ISPs close to the destination could offer a limited DoS protection service, capable of withstanding moderate-sized botnets. Moving outward, the cost of deploying the filtering ring would increase (as more border routers would need to be upgraded), but the value would also increase as the system would be able to withstand larger-scale botnets.

Our implementation of the filtering logic uses two lists of nonces, efficiently encoded using Bloom filters [7]. A whitelist contains a list of requested nonces, whereas a blacklist contains a list of nonces that have already entered the filtering ring. The whitelist ensures that only requested packets get through, and the blacklist ensures that at most one packet gets through per request. As request packets leave the ring, the router adds their nonces to the local whitelist. When data packets enter the ring, their nonces are verified by checking the whitelist and then are added to a blacklist. Bloom filters must be periodically flushed to work properly; to minimize the impact of these flushes, two copies of each list are maintained and they are alternately flushed.

We believe that the Phalanx filtering ring is efficient enough to be implemented even for high-speed links inside the core of the Internet, provided there is an incentive for ISPs to deploy the hardware, that is, provided that ISPs can charge their customers for DoS protection. (Note that ISPs that provide transit need to modify only their ingress routers and not all routers.) A 100-gigabit router line card would need about 50 MB of hash table space. For each delivered packet, six Bloom filter operations are required: The request packet places a nonce in the current copy of the whitelist, then when the actual packet is received it is checked against all four tables (the current and previous whitelist and the current and previous blacklist) and then added to the current blacklist. Both the storage and computation demands are small relative to those needed for core Internet routing tables and packet buffering.

Although the filtering ring will require either deploying new hardware in the network or upgrading the software running on existing routers, it does not require pervasive deployment. Upgrades need only be made at the border of ISPs looking to offer DoS protection. At first, filtering could be done by pairing a commodity server with each border router in an ISP and later moving the functionality into the routers if higher performance was needed and when appropriate software updates have been released.

Our discussion to this point has assumed routing symmetry. Of course, the real Internet has a substantial amount of routing asymmetry. A request packet sent to a mailbox may not leave the filtering ring at the same point as the corresponding data packet returns; if so, the Bloom filter at the return point will drop the packet. This problem becomes more likely as the nesting level increases.

To address this problem, we modify filtering ring nodes to stamp request packets as they pass through and allow mailboxes to loosely source route

data packets via IP-in-IP tunneling back through the filter ring nodes that are known to have been primed. This solves the problem of route asymmetry while only requiring cooperation from the nodes that are already being changed to do filtering.

### CONNECTION ESTABLISHMENT

Thus far, we have described how to protect established connections but have yet to properly describe the details of connection establishment.

We allow for connection establishment by issuing periodic requests that ask for connection establishment packets rather than specific data packets. These general-purpose nonces are described above. Simply allowing for such first packets doesn't solve the problem, as they immediately become a scarce resource and this capability acquisition channel can be attacked [6]. To solve this problem, we require clients to meet some burden before giving them access to a general-purpose nonce. Clients can either present an authentication token signed by the server or present a cryptographic puzzle solution.

#### PASSING THROUGH THE FILTERING RING

Rather than invent new mechanisms to deal with allowing first packets through the filtering ring, we reuse the existing request packet framework to punch nonspecific holes in the filtering ring. Destinations send each mailbox a certain rate of general-purpose requests. Each request contains a nonce to be placed in such first packets. When a mailbox wishes to send a first packet, it places one of these general-purpose nonces into the packet, allowing it to pass through the filtering ring.

These general-purpose requests implement a form of admission control. Each general-purpose nonce announces the destination's willingness to admit another flow. This further increases the destination's control over the traffic it receives, allowing it to directly control the rate of new connections.

For the general-purpose nonce mechanism to be resilient to DoS attack, it is necessary to spread the nonces across a wide set of well-provisioned mailboxes; a particular client only needs to access one. Refreshing these general-purpose nonces can pose an unreasonable overhead for destinations that receive few connection requests; as a result, our prototype supports nonces issued for aggregates of IP addresses. Thus, an ISP can manage general-purpose nonces on behalf of an aggregate of users, at some loss in control over the rate of new connections being made to each address. Of course, the ISP must carefully assign aggregates based on their capacity to handle new connection requests; for example, Google should not be placed in the same aggregate as a small Web site, or else the attacker could use general-purpose nonces to flood the small site.

When a client wishes to contact some server, it first contacts a mailbox and asks that mailbox to insert a general-purpose nonce into its first packet and forward it to the destination. Because general-purpose nonces are a scarce resource, the mailbox needs rules governing which connections to give these nonces and in what order. The next two sections deal with those mechanisms.

#### AUTHENTICATION TOKENS

Each packet requesting to initiate a connection must either carry an authentication token or a solution to a cryptographic puzzle. These provide the

burden of proof necessary for a mailbox to allow access to general-purpose nonces. Authentication tokens provide support for pre-authenticated connections, allowing them to begin with no delay. For example, a popular e-commerce site such as Amazon might provide a cookie to allow quicker access to its Web site to its registered users or even just to users who had spent more than $1000. Cryptographic puzzles provide resource proofs to approximate fair queueing of requests, when no prior relationship exists between source and destination.

Authentication tokens are simply tokens signed by the server stating that the given client is allowed to contact that server. An additional message exchange is required to prove that the client is in fact the valid token holder.

#### CRYPTO-PUZZLES

The crypto-puzzle is designed to be a resource proof allowing hosts that spend more time solving the puzzle to get higher priority for the limited number of general-purpose nonces each mailbox possesses. Although there are many kinds of resource proofs, we opt for a computational resource proof rather than a bandwidth resource proof [18] because computation tends to be much cheaper and less disruptive when heavily used.

We borrow a solution from prior work [14, 16] where the crypto-puzzle is to find a partial second pre-image of a given random challenge string such that, when hashed, both strings match in the lower b bits. The goal for each client is then to find some string a given a challenge nonce N such that:

$$h(a||N) \circ h(N) \bmod 2^b$$

The random nonce is included in both strings to prevent attackers from building up tables of strings that cover all $2^b$ possible values of the lower b bits in advance. In effect, they need to cover $2^{b + |N|}$ possible values to find matches for all values of the lower b bits and for all possible nonces, whereas solving the puzzle online only requires searching $2^{b-1}$ strings on average. Because the length of the nonces is under the control of the mailboxes, it is possible to make the precomputing attack arbitrarily harder than waiting and solving puzzles online.

First packets are granted general-purpose nonces, with priority given first to those with valid authentication tokens and then in decreasing order of matching bits in the crypto-puzzle solution. This allows any source to get a first packet through against an attacker using only finite resources per first packet, albeit at an increase in latency.

## Evaluation

Evaluating systems such as Phalanx at scale has always posed a problem because they are fundamentally intended to operate at scales well beyond what can be evaluated on a testbed. To address this issue, we built a simulator that captures the large-scale dynamics of Phalanx and allows us to simulate millions of hosts simultaneously.

The simulator uses a router-level topology gathered by having iPlane [13] probe a list of approximately 7200 known Akamai nodes from PlanetLab nodes. These Akamai nodes serve as stand-ins for appropriately located mailboxes. Each PlanetLab node serves as a stand-in for a server that is under attack.

We assume that attackers target the mailboxes, the server, and the links near the server. Traffic is assumed to flow from clients to mailboxes unmolested. We assign link capacities by assuming mailbox access links are 10 Mbps, the server access link is 200 Mbps, and link capacity increases to the next category of {10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 40 Gbps} as the links move from the edge to the core.

We assign attackers with attack rates according to end-host upload capacity information gathered in our previous work [10, 13] and assume that good clients communicate at a fixed rate of 160 kbps.

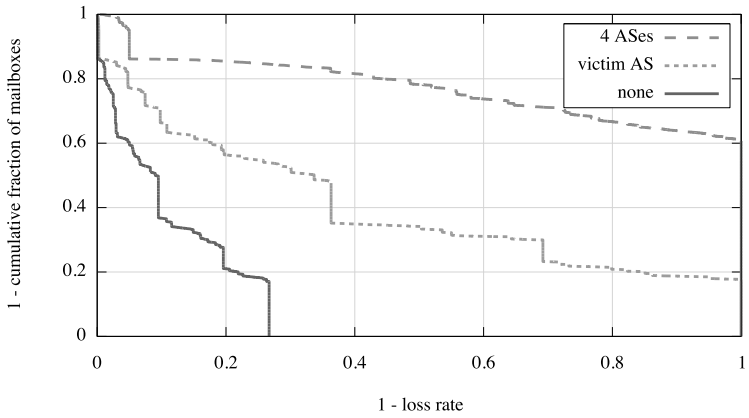


**FIGURE 3: THE CUMULATIVE FRACTION OF MAILBOXES SEEING AT MOST A GIVEN FRACTION OF GOODPUT WHEN COMMUNICATING WITH THE SERVER**

By using IP to AS mappings, we are able to simulate the behavior of the system under varying levels of deployment of the Phalanx filtering rings. Figure 3 shows the effect of increasing deployment of filtering rings for a server located at planetlab-01.kyushu.jgn2.jp. (The results are similar when we use other PlanetLab nodes as servers.) In this simulation, there are 100,000 attacking nodes and 1000 good clients all trying to reach the victim server. We simulate varying degrees of deployment by iteratively adding the largest adjacent AS to the current area of deployment.

As one might expect, even a little deployment helps quite a bit. Only deploying filters at the victim AS provides significant relief and allows some mailboxes to see lossless communication. Deploying in just four ASes (including the tier-1 AS NTT) results in the vast majority of mailboxes seeing lossless communication, effectively stopping the attack in its tracks if we assume that connections use any degree of redundancy to handle losses.

We next look at the scalability of Phalanx in handling attacks involving millions of bots. For this experiment we consider a somewhat stronger deployment: upgrading the mailboxes to 100 Mbps access links. Figure 4 examines the effect on mailbox loss rate as we increase the number of attackers. Most connections easily withstand the brunt of an attack involving one million nodes, and Phalanx still allows some (though severely degraded) communication through when facing 4 million nodes.

However, as the graph shows, increasing the capacity of the mailboxes by a factor of 5 to 500 Mbps is able to once again bring the attack into check. Thus, while any given deployment will have a breaking point, an increased deployment can bring increased protection to deal with even larger attacks.

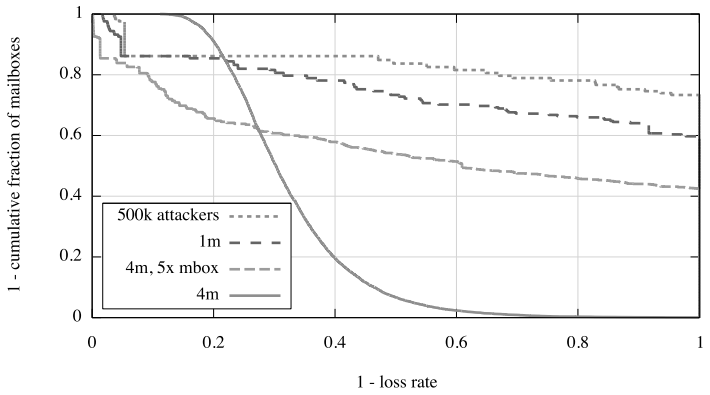


**FIGURE 4: THE CUMULATIVE FRACTION OF MAILBOXES SEEING AT MOST A GIVEN LOSS RATE FOR A VARYING NUMBER OF ATTACKERS**

## Conclusion

In this article, we presented Phalanx, a system for addressing the emerging denial-of-service threat posed by multimillion-node botnets. Phalanx asks only for two primitives from the network. The first is a network of overlay nodes, each implementing a simple, but carefully engineered, packet forwarding mechanism; this network must be as massive as the botnet that it is defending against. Second, we require a filtering ring at the border routers of the destination's upstream tier-1 ISP; this filtering ring is designed to be simple enough to operate at the very high data rates typical of tier-1 border routers. We have implemented an initial prototype of Phalanx on PlanetLab and have used it to demonstrate its performance. We have further demonstrated Phalanx's ability to scale to million-node botnets through simulation.

### ACKNOWLEDGMENTS

We would like to thank Arun Venkataramani for a set of conversations which helped us realize the need for more scalable DoS protection. We would also like to thank our NSDI shepherd, Sylvia Ratnasamy, as well as our anonymous reviewers, for help and valuable comments. This work was supported in part by National Science Foundation Grant No. CNS-0430304.

### REFERENCES

[1] Akamai: http://www.akamai.com/.

[2] Microsoft's unabated patch flow: http://www.avertlabs.com/research/blog/index.php/category/security-bulletins/ (May 9, 2007).

[3] "Surge" in Hijacked PC Networks: http://news.bbc.co.uk/2/hi/technology/6465833.stm (March 2007).

[4] D.G. Andersen, "Mayday: Distributed Filtering for Internet Services." In *USITS*, 2003: http://www.usenix.org/events/usits03/tech/andersen.html.

[5] N. Anderson and Vint Cerf: One Quarter of All Computers Part of a Botnet: http://arstechnica.com/news.ars/post/20070125-8707.html (January 25, 2007).

[6] K. Argyraki and D. Cheriton, "Network Capabilities: The Good, the Bad and the Ugly." In *HotNets IV*, 2005.

[7] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, 13(7): 422–426 (1970).

[8] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: Withstanding Multi-million Node Botnets." In *NSDI*, 2008: http://www.usenix.org/events/nsdi08/tech/dixon.html.

[9] P. Finn, "Cyber Assaults on Estonia Typify a New Battle Tactic," *Washington Post*, May 19, 2007: http://www.washingtonpost.com/wp-dyn/content/article/2007/05/18/AR2007051802122.html.

[10] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Leveraging Bittorrent for End Host Measurements." In *PAM*, 2007.

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

[12] B. Krebs, "Blue Security Kicked While It's Down," *Washington Post*, May, 2006: http://blog.washingtonpost.com/securityfix/2006/05/blue_security_surrenders_but_s.html.

[13] H.V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services." In *OSDI*, 2006: http://www.usenix.org/events/osdi06/tech/madhyastha.html.

[14] B. Parno, D. Wendlant, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks." In *SIGCOMM*, 2007.

[15] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321 (Informational), April 1992.

[16] E. Shi, I. Stoica, D. Andersen, and A. Perrig, OverDoSe: A Generic DDoS Protection Service Using an Overlay Network, Technical report, Carnegie Mellon University, 2006: http://www.cs.cmu.edu/~dga/papers/CMU-CS-06-114.pdf.

[17] A. Stavrou and A.D. Keromytis, "Countering DoS Attacks with Stateless Multipath Overlays." In *CCS*, 2005.

[18] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense." In *SIGCOMM*, 2006.