Phalanx: Withstanding Multimillion-Node Botnets

Colin Dixon Thomas Anderson Arvind Krishnamurthy
University of Washington

Abstract

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 prevalence and size of botnets more effective than ever. Although much progress has been made in developing techniques to address DoS attacks, no existing solution 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.

In this paper, we present a novel DoS prevention scheme to address these issues. Our goal is to define a system that could be deployed in the next few years to address the danger from present-day massive botnets. The system, called *Phalanx*, leverages the power of swarms to combat DoS. Phalanx makes only the modest assumption that the aggregate capacity of the swarm exceeds that of the botnet. A client communicating with a destination bounces its packets through a random sequence of end-host mailboxes; because an attacker doesn't know the sequence, they can disrupt at most only a fraction of the traffic, even for end-hosts with low bandwidth access links. We use PlanetLab to show that this approach can be both efficient and capable of withstanding attack. We further explore scalability with a simulator running experiments on top of measured Internet topologies.

1 Introduction

Botnets are very, very large. A recent estimate put the number of compromised machines participating in botnets at 150 million [6], while more modest estimates put the number around 6 million [3]. Single botnets regularly contain tens of thousands of end-hosts and have been seen as large as 1.5 million hosts [31]. These networks are the basis of a lucrative and difficult to detect underground economy on the Internet today. They steal identities and financial information, send most of the world's spam, and are used in DoS protection rackets [25]. These uses and less obvious ones have been extensively covered in previous work [32, 17]. 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.

Our focus in this paper is the impact of massive botnets on proposed solutions to denial of service (DoS) attacks. If compromised nodes are typical of end hosts participating in other large peer-to-peer systems [18], a multimillion-node botnet would be able to generate over a terabit per second of traffic, 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 semi-permanently 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 [20]. The damage is not just limited to security companies. Starting in April a sustained attack on government and business websites in Estonia effectively knocked the country off the web [16]. Even nations are not safe.

While the scale of the attacks may be increasing, the threat of DoS has been well understood for the past few years and many solutions have been proposed [23, 7, 9, 40, 19, 5, 39, 38, 33, 37, 30, 13, 35, 24, 26]. We believe that these solutions have not met with significant success in the real world because while often elegant, the burden of effective deployment is too high. The primary focus of our efforts is to provide a solution that can be effective when deployed unilaterally by as few parties as possible even if this means sacrificing some elegance.

It is worth noting that for read-only web sites, massive replication has proven an effective solution, and is even available as a commercial service [1]. However, the concern of this paper is non-replicable services, such as read/write back-end databases for e-commerce, modern AJAX applications, e-government, and multiplayer games, or point-to-point communication services such as VoIP or IM. For this case, we argue that no existing commercial or research solution is adequate to the scope of the problem posed by multimillion-node botnets.

In this paper, we propose a DoS prevention system, called Phalanx, that is capable of withstanding a multimillion-node botnet and yet can be reasonably deployed by a single large ISP today. Phalanx combines ideas from several ideas from prior work; co-designing the client, server, router support, and overlay software, to yield an effective yet remarkably simple system. Rather than being directly delivered, traffic is sent through a massive swarm of packet forwarders which act as mailboxes. Hosts use these mailboxes in a random order, so that even an attacker with a multimillion-node botnet can cause only a fraction of a given flow to be lost. Lightweight capabilities are then used to ensure that only

requested packets are delivered from the mailboxes to the destination.

Of course, to be practical, the system must not impose an undue burden in the absence of attack: we need to design the system to be cheap and unobtrusive. Ideally, the mechanisms will never be used—once attacks become ineffective, they will simply stop.

We expand on these ideas in more detail in the rest of the paper. Section 2 provides some relevant background and an overview of our approach. We present the complete architecture in Section 3, while Section 4 evaluates Phalanx's performance and effectiveness. We discuss related work in Section 5, and conclude with Section 6.

2 Phalanx Overview

2.1 Background

While 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. The target resource is typically the weakest link on the path between the source and the destination, in terms of bandwidth, storage or computation.

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, $O(\frac{G}{B+G})$ of the available resources go to legitimate requests. 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.

Of course, damage can be mitigated if traffic can be classified into legitimate and attack. We believe such classification is becoming increasingly difficult as botnets grow because attacks no longer need to spoof addresses or send abnormally large amounts of traffic per host in order to be successful. Clever attackers can simply emulate normal user behavior on each bot. While our approach is compatible with traffic classification, we explicitly do not rely upon it.

2.2 Assumptions

In approaching the general DoS problem, we make some assumptions about the network in which we operate.

- **Swarm:** We assume access to a large pool of well-provisioned machines which are geographically and topologically distributed. In essence, we assume access to a botnet of our own to absorb attacks. Our prototype is built and tested PlanetLab; as future work, we are exploring modifying a popular BitTorrent client to convert the millions of BitTorrent users into a community-based botnet defense.
- **Strong Adversary:** Botnets will continue to increase in size. We assume nodes in the botnet need not send anomalous amounts or kinds of traffic for attacks to be effective.
- **Minimal Network Support:** We assume that simple modifications to routers are feasible if they can be implemented in hardware at data rate at the edge of an ISP.
- **Predictable Paths:** Recent work [22] has shown that Internet paths can often be predicted without direct measurement of each possible path. This knowledge helps our system establish efficient yet resilient paths for good traffic.

2.3 Goals

A complete solution to the DoS problem must satisfy a large number of requirements including resistance to attack, deployability, performance, and backward compatibility. The following list of goals for our system helps to make these requirements concrete:

- **Eventual Communication:** Regardless of the number of attackers, it should be possible for a connection to eventually become established. In particular this means that a host needs to be able to perform a name lookup and acquire some form of capability to communicate even in the face of massive attack. Further, this must all be possible while keeping the current Internet's open user model.
- **Protect Established Connections:** Once established, connections should be protected from collateral damage. Since any single path can easily be overwhelmed by a million-node botnet, each connection must leverage multiple paths to any destination. A connection should see degradation of at most $O(\frac{B}{B+G+M})$ where M is the set of mailboxes in Phalanx. In other words, performance should be proportional to the total number of good vs. bad nodes in the entire system, not the number of good vs. bad nodes focusing on a specific target.
- **Unilateral Deployment:** A single large ISP should be able deploy an effective DoS solution for its customers, even from a massive attack, without needing to first reach a global agreement with all or most other ISPs.
- **Endpoint Policy:** The destination, and not the network, should control which connections are to be established and which packets are to be delivered.
- **Resistance to Compromise:** The system must tolerate compromised nodes and routers; its correct behavior should rely on only a few, simple (and thus possible to secure) functions.

- **Autoconfiguration:** Since Internet paths do change, both because of malicious activity and normal functioning, these changes should not interrupt protection.
- **Efficiency:** Communication performance should be close to that of the current Internet, even under attack. Otherwise an attack can be at least partially successful simply by evoking a response.

Many existing solutions achieve some of these goals, but as we explain later in Section 5, none achieves even the first three goals, much less all of them. We do leverage several key ideas from prior work, but we defer a detailed comparison until after we have described Phalanx.

3 Phalanx Architecture

In this section we describe the Phalanx architecture in detail. At a high level the architecture calls for sending all traffic through a set of mailboxes (Section 3.2) rather than directly to the destination. This approach requires some mechanism to prevent traffic from bypassing these mailboxes (Section 3.3) and also a system for handling connection setup (Section 3.4). Table 2.3 provides a summary of the mechanisms used in Phalanx along with their function, the goals they help achieve and where they are discussed in this paper. Additionally, the notation that we use to simplify discussion can be found in Table 3.2.1.

3.1 Components

Phalanx consists of three main components, which when combined, meet the goals laid out in Section 2.3. First, since it is easy for a large botnet to overwhelm any specific Internet path, we rely on a *swarm* which can match an attacking botnet in strength. This swarm puts legitimate users on the same footing as attackers by artificially inflating the number of "good" hosts for any given destination. The swarm appears to clients and servers as simple *packet mailboxes* using the API sketched in Table 3.2.1, allowing for a best effort packet store and pick-up point. These mailboxes are further explained in Section 3.2.

Second, a destination must explicitly request a packet from the mailboxes for it to be delivered; we use a set of Bloom filters at all of the border routers of the destination's ISP to drop any unrequested packet (with high probability). This *filtering ring* implements an implicit per-packet network capability, rather than the per-connection capability [39, 40] or per-source address connectivity [10] in other proposals. This filtering ring is described in detail in Section 3.3.

Third, we use *resource proofs* and *authentication tokens* to facilitate connection setup. In an open Internet, there will often be no way to distinguish an initial connection request as being either good or from an attacker. Resource proofs approximate fair queueing for these initial connection requests. We opt for computation-based proofs as in Portcullis [26] and OverDoSe [30] over bandwidth-based proofs [37]. The form of Phalanx resource proofs can be found in Section 3.4.3. Authentication tokens provide a way for clients with a preexisting relationship to a specific server to bypass its resource proof. Section 3.4.2 is a short description of Phalanx authentication tokens.

Together these components create a carefully chosen battleground where good users are on equal footing with attackers and can leverage the swarm's resources to fight back against an attacking massive botnet.

3.2 Mailboxes

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 [7]. Rather than packets being delivered directly to the specified destination as in previous anti-DoS overlays [33, 19, 5], traffic is first delivered to a mailbox where it can either be "picked up" or ignored by the destination. Traffic which is ignored is eventually dropped from the buffers at packet mailboxes.

The interface which each mailbox exports is sketched in Table 3.2.1. The two basic operations are to `put` and `get` packets. The semantics are somewhat different from the traditional notions of `put` and `get`. A `put` inserts a packet into the mailbox's buffer, possibly bumping an old entry, and returns. A `get` behaves somewhat less intuitively. Rather than behaving like a polling request, a `get` instead 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. This is conceptually similar to i3 [34], except triggers are valid for only one packet.

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.

3.2.1 Swarms & Iterated Hash Sequences

Mailboxes act as proxies, receiving and temporarily buffering traffic on behalf of end-hosts. If only a single such proxy existed, then we would just be moving the DoS problem from the end-host to the mailbox. In-

Mechanism	Function	Goals	Section
Mailboxes	lightweight network indirection primitive	Unilateral Deployment, Autoconfig	3.2
Iterated Hash Sequences	pseudo-random mailbox sequences	Protect Established Connections, Resistance to Compromise	3.2.1
Scalable Mailbox Sets	provide both efficiency and resilience according to current conditions	Efficiency	3.2.2
Filtering Ring	drop unrequested traffic before it can cause damage	Unilateral Deployment, Resistance to Compromise	3.3
General Purpose Nonces	allow small numbers of unrequested packets through the filtering ring	Eventual Communication	3.4.1
Cryptographic Puzzles	approximate fair queueing for connection establishment	Eventual Communication	3.4.3
Authentication Tokens	allow for pre-authentication of trusted flows	Eventual Communication	3.4.2
Congestion Control	adapt to access link heterogeneity	Autoconfig	3.5

Table 1: A summary of mechanisms used in Phalanx.

Symbol	Meaning
h	cryptographic hash function
x, y	shared secret to generate mailbox sequences
C	client, endpoint instigating the connection
S	server, endpoint receiving the connection
x_i	ith element of the sequence based on h and x
M	the set of mailboxes $\{M_i, \ldots, M_{\|M\|}\}$
$M[x_i]$	mailbox corresponding to x_i ($M_{x_i \bmod \|M\|}$)
K_Y	the public key belonging to Y
k_Y	the private key belonging to Y
$(z)_{K_Y}$	z encrypted using Y's public key
$(z)_{k_Y}$	z signed using Y's private key
w	the window size for requesting packets

Table 2: Notation

stead, we rely upon swarms of mailboxes to provide an end-host with many points of presence throughout the network. Assuming that the mailboxes' resources exceed that of attackers, legitimate clients will have some functioning channels for communication despite a widespread attack.

Individual flows are multiplexed over many mailboxes. Each packet in a flow is sent to a cryptographically random mailbox. Any given mailbox failure will only slightly affect a flow by causing a small fraction of the packets to be lost (often only a single packet). 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. The set of mailboxes M to use for this connection is determined by the destination, as described in the next section. The *sequence* of mailboxes is built by iterating a cryptographic hash function such as SHA-1 on a shared secret. We discuss how this secret is established in Section 3.4. Equipped with this shared sequence, both endpoints know in advance the precise mailbox to use for each packet in the connection.

`putPacket(packet` p`)` places a packet in the local packet buffer
`getPacket(nonce` n`)` places a request in the local packet buffer; when/if a packet arrives or has arrived it is returned
`requestConnection(serverKey` K_S`)` asks for a challenge to earn access to establish a connection; returns a random nonce
`submitSolution(string` a`, integer` b`)` provides a cryptographic puzzle solution to the challenge as a resource proof
`submitToken(authentication token` t`)` provides proof of pre-authentication
`issueNonces(signed nonce list` N`)` registers a set of general purpose nonces to be used for initial packet contact with the signing destination

Table 3: The Mailbox API.

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 [29] 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 re-negotiated within a flow.

One such shared secret and iterated hash function is used for each direction of communication. For the sake of discussion, we assume that the shared secret x generates the sequence x_i used for the client to server direction while the shared secret y generates the sequence y_i used for the server to client direction.

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. Including a nonce in each packet simplifies the logic needed to drop unrequested packets as described in Section 3.3. Lastly, nonces provide a limited form of authentication to requests; the attacker must snoop the nonce off the wire, and then deliver a replacement packet to a mailbox before the correct packet arrives, in order to subvert the system.

Communication proceeds with each packet and corresponding request going to the next mailbox in the sequence. Note that the data request is asynchronous with the data arrival—it may precede it or follow it at the mailbox (unlike i3 [34]). In the sequence given below, the requests precede the data packets, but in practice they will be sent simultaneously and the ordering does not matter.

$M[x_i] \leftarrow$ S: request for x_i
C $\rightarrow M[x_i] \rightarrow$ S: x_i, data
$M[x_{i+1}] \leftarrow$ S: request for x_{i+1}
C $\rightarrow M[x_{i+1}] \rightarrow$ S: x_{i+1}, data

Here we only show one direction of communication, from the client to the server. The reverse direction proceeds in exactly the same way but selects mailboxes using the iterated hash sequence y_i based on the different shared secret y.

For flow control, each endpoint maintains a sliding window of w requested packets. This window is advanced each time a packet is received, or at the most recent packet rate if all packets in the window are lost. For optimal throughput, the window should be at least as large as the bandwidth-delay product, but there are several advantages to keeping w modest in size. First, w represents the number of outstanding requests which might be received all at once from a malicious sender. A wily attacker may delay delivering packets to mailboxes until as many gets have been registered as possible.

Second, keeping w small reduces the length of time packets are queued at mailboxes before being requested (or equivalently requests are queued before being matched). Recall that mailbox queueing is best effort and thus packets and requests can be dropped. We assume that each mailbox is well-provisioned in that it can queue at least a few seconds of packets and requests in DRAM at its network access link bandwidth.

3.2.2 Mailbox Sets

Picking the subset of mailboxes in Phalanx to use for a specific connection poses a tradeoff of performance and resilience. For best performance, we would like mailboxes that are only slight detours off the best path to the destination; for resilience, we would like mailboxes that exhibit the greatest path diversity to the filtering ring. We opt to make the mailbox sets dynamically negotiable within each flow to get the best of both worlds. Initially, we start each connection with a small number of mailboxes (ten in the prototype), chosen to achieve good resilience without sacrificing performance. In the prototype, we use iPlane's route and performance predictions [22] to guide this choice. If the flow sees significant loss, indicating a possible attack, additional mailboxes are added to increase path diversity.

For example, a connection from Los Angeles to Seattle might start off using mailboxes in different ISPs in San Francisco and Portland for their low additional latency. If loss rates cross a given threshold, then mailboxes in Denver and Salt Lake City can be added to increase path diversity. If the attack continues and becomes more severe, the connection might start redirecting packets through mailboxes across the US, Asia and Europe. If an attack is ongoing, new connections might be started off with a larger and more diverse set of initial mailboxes. As an optimization which we do not yet implement, mailbox sets can be passed by reference. The set of all mailboxes for a particular destination is relatively static and can be widely distributed; a particular connection need only agree on which random subset of this list to use, e.g., by hashing on the shared connection secret.

Because both endpoints need to be using the same set of mailboxes to ensure proper delivery, re-negotiating the set of mailboxes in the middle of a connection is not trivial. By default, Phalanx uses two mailbox sets (one for each direction) and each endpoint controls the set which it receives through. If an endpoint wishes to make any changes, it piggybacks (a pointer to) the new set in a normal packet, along with a first sequence number for which the changes will be valid.

The endpoint then waits for that sequence number and branches in both directions: one assuming the changes were received successfully, the other assuming that they were lost. Whenever one branch receives a packet, the endpoint knows which branch was correct and drops the other branch. If the change request arrives at the sender too late (i.e., after the packet with the sequence number has already been sent on the old path), the sender must

ignore it; the receiver is then free to try again. This provides support for dynamic re-negotiation and thus helps provide both performance and resiliency according to the prevailing conditions.

3.3 Filtering ring

With Phalanx, a protected destination only receives those packets which it explicitly requests from a mailbox. To enforce this, we drop all other packets for the destination at the edge of its upstream ISP (See Figure 1). This means that a protected destination cannot serve as a mailbox, and more importantly, this breaks legacy clients. We have not implemented legacy client support in our prototype yet, but we envision an applet that a web site would provide its clients to mediate their access through Phalanx. This does not create a "chicken and egg" problem, because the applet would be read-only data that can be widely replicated; as we observed earlier, existing commercial anti-DoS solutions are effective for distributing read-only data.

Implementing the filtering ring is straightforward, even at hundred gigabit data rates. Each request packet carries a unique nonce that allows a single reply 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 that once an incoming packet has matched a nonce, we remove that nonce.

Of course, an attacker might try to flood the border router (or more precisely, the links immediately upstream from the border router) to prevent returning packets from ever reaching the destination. 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 indirectly, as 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.

Our implementation of the filtering logic uses two lists of nonces, efficiently encoded using Bloom filters [12]. A whitelist contains a list of requested nonces while a blacklist contains a list of nonces which have already entered the filtering ring. The whitelist ensures that only requested packets get through, while 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 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.

While Bloom filters provide only probabilistic guarantees, this is sufficient for our purposes as they do not yield false negatives. Even in the unlikely event that an attacker's guessed nonce is a false positive for the whitelist it will then be added to the blacklist making it good for only one packet. It is not possible that a correct returning packet will miss in the whitelist because that would require a false negative. There is still a (small) possibility that a legitimate packet will be incorrectly dropped because of a collision in the blacklist, but Phalanx is designed to be robust to packet loss.

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 50MB 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 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 are small relative to those needed for core Internet routing tables and packet buffering.

To be effective, the filtering ring must be comprehensive – able to examine every packet destined for a protected destination, regardless of the source of the traffic. Bots are everywhere, even inside corporate networks. As a result, it seems likely that filtering rings would be deployed in depth, as shown in Figure 1. 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 (more border routers to upgrade), but the value would also increase as the system would be able to withstand larger-scale botnets.

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

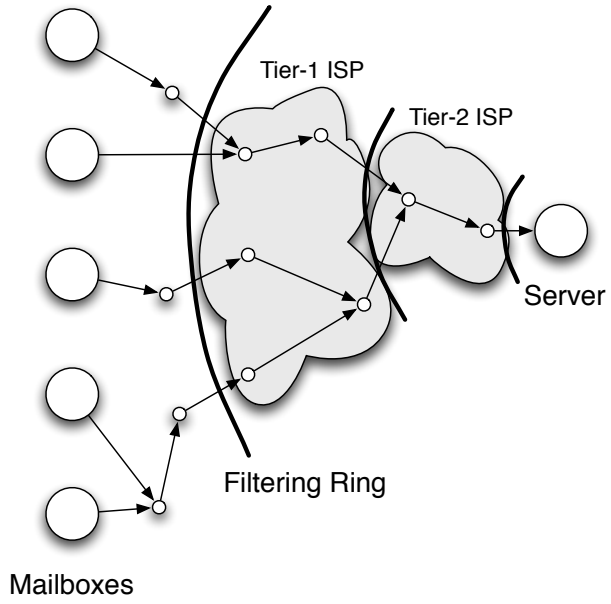

Figure 1: Filtering ring placement. Unlike in current Internet routing where all packets for a single connection take a single path across each ISP boundary, in Phalanx packets from a given connection may use many different paths across the set of filtering rings.

filter at the return point will drop the packet. This problem becomes more likely as the nesting level increases.

To address this problem, we allow destinations to loosely source route request packets via IP-in-IP tunneling to each mailbox. The source route is chosen to be the reverse of the predicted sequence of filtering ring nodes from the mailbox back to the destination. This guarantees that each filtering ring will be appropriately primed. We use iPlane to gather the data to make the route prediction [22].

3.4 Connection Establishment

Thus far, we have described a way for a destination to receive packets it is expecting. In order to establish a connection, a first (unexpected) packet must be delivered.

We allow for connection establishment by issuing periodic requests which ask for connection establishment packets rather than specific data packets. These *general purpose nonces* are described in Section 3.4.1. 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 [8]. 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 as explained in Section 3.4.2 or present a *cryptographic puzzle* solution as explained in Section 3.4.3.

3.4.1 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 connection requests.

In order for the general purpose nonce mechanism to be resilient to DoS attack, it is necessary to spread them 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, and 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. An ISP already manages the statistical multiplexing of resources We discuss in Section 3.6 how the client learns the initial set of mailboxes to use for a specific destination.

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.

3.4.2 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 preauthenticated 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. Cryptographic puzzles provide resource proofs to approximate fair queue-

ing of requests, when no prior relationship exists between source and destination.

Authentication tokens are simply a token 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. Authentication tokens take the form $(K_C, t)_{k_S}(N)_{k_C}$.

The first portion is the public key of the client and an expiration time signed by the server. This represents that the server has given the client the right to initiate connections until the listed expiration time. The second portion is a random fresh nonce issued by a mailbox and signed by the client. This proves that the client is in control of the private key to which the token was originally issued.

The authentication token connection establishment protocol then proceeds as follows:

C $\rightarrow M[*]$: request challenge
C $\leftarrow M[*]$: N
C $\rightarrow M[*] \rightarrow$ S: N, $(K_C, t)_{k_S}(N+1)_{k_C}$, $(K_C, x)_{K_S}$
C $\rightarrow M[x_0]$: request for x_0
C $\leftarrow M[x_0] \leftarrow$ S: x_0, $(K_S, y)_{K_C}$, data
$M[y_0] \leftarrow$ S: request for y_0
C $\rightarrow M[y_0] \rightarrow$ S: y_0, data
C $\rightarrow M[x_1]$: request for x_1
$\vdots$

First a challenge nonce is requested and received. The nonce is created to be self certifying as in SYN cookies [11] by making it a secret cryptographic hash of the mailbox and client IP addresses and ports as well as some local, slowly-increasing timer. This prevents a SYN flood style attack on memory at the mailbox.

Next the authentication token is presented along with the client's public key and a shared secret to be forwarded to the server. If the token is valid, the mailbox forwards the whole request on to the server which can then choose to accept the connection or not as it sees fit.

3.4.3 Crypto-puzzles

The crypto-puzzle is designed to be a resource proof allowing hosts which spend more time solving the puzzle to get higher priority to the limited number of general purpose nonces each mailbox possesses. While there are many kinds of resource proofs, we opt for a computational resource proof rather than a bandwidth resource proof [37] because computation tends to be much cheaper and less disruptive when heavily used.

We borrow the solution presented in Portcullis [26] and OverDoSe [30] where the crypto-puzzle used 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:

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

The random nonce is included in both strings to prevent attackers from building up tables of strings which 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 need only search 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 pre-computing 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.

Assuming that attackers have some fixed computational ability, we know that there is some number of bits b_a such that if attackers continuously solved cryptographic puzzles in b_a bits, they would not be able to consume all general purpose requests. Knowing this, it is easy to show that any client which solves a crypto puzzle in b_a bits will be guaranteed to get a general purpose nonce.

If the attackers solve puzzles in b_a or more bits, then by definition of b_a there will be left over general purpose nonces for the client to use. If the attackers solve puzzles in fewer than b_a bits, then the client will preferentially receive general purpose nonces based on the priority queuing.

Knowing b_a in advance is not necessary because solving puzzles in all numbers of bits from 1 to b_a only takes twice as long as solving a puzzle in b_a bits. Thus, at a cost of a factor of 2 in latency, we can try all possible numbers of bits until we find the "correct" number.

In addition to reducing overhead, aggregating general purpose nonces across multiple IP addresses has one further benefit. Without aggregation, a botnet can amass its resources to drive up the cost of acquiring all of the nonces for a specific destination. With aggregation, the botnet must compete with a larger number of good clients, and a faster refresh rate of general purpose nonces, in order to target any specific destination.

Connection establishment using crypto-puzzles proceeds as follows:

C $\rightarrow M[*]$: request challenge
C $\leftarrow M[*]$: N
C $\rightarrow M[*] \rightarrow$ S: N, a, $h(a||N)$, b, $(K_C, x)_{K_S}$
C $\rightarrow M[x_0]$: request for x_0
C $\leftarrow M[x_0] \leftarrow$ S: x_0, $(K_S, y)_{K_C}$, data
$M[y_0] \leftarrow$ S: request for y_0
C $\rightarrow M[y_0] \rightarrow$ S: y_0, data
C $\rightarrow M[x_1]$: request for x_1
$\vdots$

With the exception of the substitution of a crypto-puzzle solution for an authentication token, this is identical to the solution presented in Section 3.4.2.

3.5 Congestion Control

Forwarding traffic through mailboxes with significant path diversity makes Phalanx interact somewhat poorly with normal TCP. Packets will be frequently reordered, and losses from one mailbox should not necessarily cause a connection to reduce its rate. Although this scenario bears resemblance to multipath congestion control, the issue in Phalanx is further complicated by the fact that attackers can exploit congestion response as an avenue for DoS [21, 27].

Instead, we build a simple congestion control protocol based on the assumption that congestion only occurs at the access links of the sender, receiver and/or mailbox. Receivers advertise a maximum packet rate they are willing to receive from a particular sender, based on local policies and available resources. A sender uses the receiver's advertised rate along with current observed packet receipt rate to adjust its sending rate.

Essentially, the receiver picks a rate it is willing to receive, say 50 packets per second. To begin with the sender sends 50 packets a second through its set of mailboxes. If losses bring the rate below 50 packets a second, indicating either congestion or a DoS attack, the sender will ramp *up* its packet rate to compensate, using either forward error correction or retransmissions to recover lost packets. While this is not TCP friendly, it is impossible to be both TCP friendly and resistant to flooding attacks.

It is also possible for mailboxes to become overloaded, e.g., due to true congestion. A mailbox experiencing congestion is free to simply drop packets, but it can also set the IP ECN bit in packets passing through it. This signals to the destination that it should reduce the rate through this particular mailbox. The destination can achieve this by simply removing that mailbox from future flows, or reconfiguring the mailbox sets of existing flows to exclude the congested mailbox. In our prototype, we have implemented a finer-grained approach. Each mailbox in the mailbox set is assigned a weight, corresponding to the estimate of how much traffic that mailbox can successfully handle. This weight is used to bias the random selection of mailboxes to favor those that can handle more traffic; the weights can be dynamically adjusted, in the same manner as in Section 3.2.2, by agreement between the source and destination.

3.6 Name Service Lookup

In order to provide a complete DoS solution, all components of a connection must be protected, from looking up the server, to logging in, to closing the connection. As we attempt to solve the more general problem of providing DoS protection for typical public server on the current Internet, protecting lookup is as important as protecting the actual connection.

Fortunately, lookup services typically serve small amounts of static content and can thus be highly replicated to provide DoS resilience. Other highly replicated name services based on DHT like CoDoNS [28] provide such solutions. One approach would be to run the DHT-based lookup service on top of all mailboxes.

Rather than returning the address of a server, the name service instead returns a list of mailboxes willing to handle connections for the server. As before, this list of mailboxes can be chosen using iPlane in the absence of an attack, or taken randomly from a topologically diverse list to provide better resilience. These points of contact can differ from the mailboxes which will eventually be used for normal communication; those mailboxes are negotiated during connection setup.

The server must be able to update these records, but we imagine the set of first contact mailboxes will not change especially quickly and it is not necessary for the change to be atomic. This enables almost any off-the-shelf DHT semantics to serve our purposes.

3.7 A Working Example

To illustrate the complete system in action, we consider the example of fetching a web page from a Phalanx-protected server. The example can also be followed in Figure 2 where the numbered steps will be mentioned.

First, the client looks up the address of the server for 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 (as in steps 1 and 2). 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, non-cacheable portions of the page and speaking the Phalanx protocols.

The applet begins by making a name request for the dynamic content server to the distributed name service (3). Again, because the naming information is static

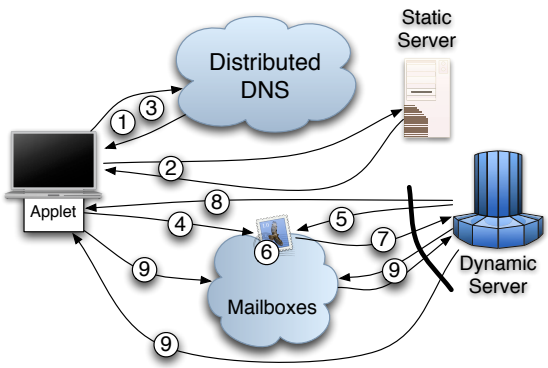

Figure 2: A diagram illustrating a simple HTTP-style request done with Phalanx. The numbers correspond to the description in Section 3.7.

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 general purpose nonces that the server has issued to allow new connections to be made.

The applet requests a challenge nonce from one of these mailboxes and replies with either a puzzle solution or an authentication token (4). In either case, the applet waits some period of time for a response before retrying the request possibly with a more complex puzzle solution and/or trying a different mailbox.

At the mailbox, a steady stream of general purpose nonces has been arriving from the dynamic content server (5). One of these general purpose nonces is eventually assigned to the client's connection request (6) at which point the applet's request is forwarded to the server along with the general purpose nonce (7) to cross back through the filtering rings without being dropped.

Eventually, a response will come back from the server (8) containing a list of mailboxes to use for the remainder of the connection along with a shared secret allowing standard Phalanx communication to commence. Along with the response, the server will send packet fetch requests to the first several mailboxes to be used in preparation to receive further packets from the client.

The client uses the shared secret to determine the sequence of mailboxes the server expects the client 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 by virtue of the holes opened by the requests. This constitutes the normal behavior of the Phalanx connection (9).

If at any point in time the server decides that the connection is no longer desirable or simply starts running low on resources, it can either decrease the rate at which

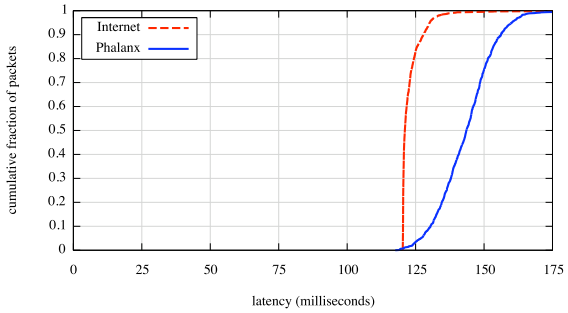

Figure 3: The cumulative distribution of round trip latencies over 1000 packets sent between Berlin `edi.tkn.tu-berlin.de` and North Carolina `planet02.csc.ncsu.edu` using both standard UDP (Internet) and Phalanx.

it requests new data packets or simply stops requesting packets altogether.

4 Evaluation

To evaluate Phalanx, we implemented a prototype server, client, filtering ring, and mailbox. Our prototype is approximately 1750 lines of C code between the four functions. We have not yet integrated our code with a scalable name system; several good candidates exist such as CoDoNS [28], DDNS [14], and Overlook [36], and integrating our system with one of these options is part of future work. We also augment the experimental results from our prototype with simulation results on a large-scale Internet topology.

4.1 Micro-benchmarks

To get a high-level idea of the performance of Phalanx in normal operating conditions with no ongoing attacks, we ran a series of experiments on PlanetLab. Each run sent packets between two randomly chosen PlanetLab nodes at different sites in late September 2007. (We eliminated PlanetLab sites where no node responded to a ping, and we avoided measuring during the time immediately preceding the conference submission deadline.) For each pair of nodes, we used iPlane to select an additional ten PlanetLab nodes, each at a physically different location, to serve as mailboxes. The nodes were selected as those that iPlane predicted would offer the lowest latency for a one hop path between the pair of nodes.

For each pair, we sent constant bitrate UDP traffic at approximately 25 kilobytes per second (25 packets per second) from the source to the destination first using Phalanx and then using plain UDP. In both cases we sent approximately 1000 packets per connection.

Triangle routing packets through mailboxes will usually incur some extra latency, particularly on PlanetLab where scheduling delays can dominate. However, iPlane's predictions can help counter this by selecting mailboxes which offer the least latency. Figure 3 shows a

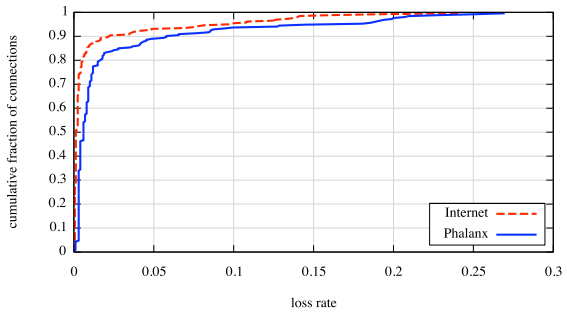

Figure 4: Cumulative distribution of loss rate, across all pairs of PlanetLab nodes, for standard UDP packets sent over the Internet and packets sent via Phalanx.

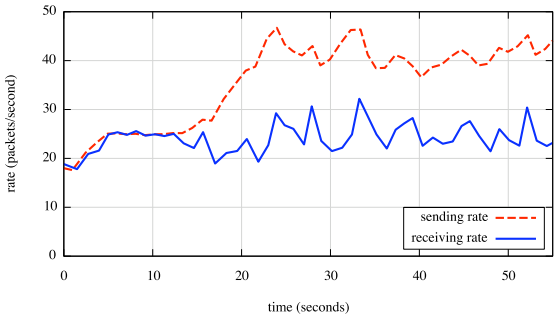

Figure 5: The rate at which packets are sent and received in the face of a modeled attack. Starting at 12 seconds, 5 of the 10 Phalanx mailboxes are "attacked", cause 75% loss of arriving data packets. Noting the decrease in throughput, the receiver asks the sender to send faster and is able to maintain an average throughput of 25 packets per second.

sample run for traffic between a node in Berlin and and a node in North Carolina, measured for standard UDP and for Phalanx. As the figure shows, despite crossing the Atlantic, Phalanx has only a modest effect on per-packet latency. iPlane is able to find ten suitable mailboxes which do not significantly affect end to end latency. On average the latency goes up by a little less than 25 milliseconds or 20%.

A denser deployment of mailboxes, for instance using BitTorrent's client network or Akamai's worldwide network of servers, should improve these numbers significantly.

Because of its indirection architecture, Phalanx requires multiple packets to be successfully sent and received in order to successfully deliver what would be a single packet in the underlying Internet. As a consequence, we would expect that Phalanx's packet loss rates would be worse than that of normal UDP.

Figure 4 illustrates this effect, showing the cumulative distribution of the measured loss rate across all pairs of PlanetLab nodes, for the experiment described above. While Phalanx does see more loss than the standard UDP, the effect is close to a constant factor increase in loss rate.

4.2 Attack Resilience

We next study Phalanx's ability to counter attacks. For this, we repeat the previous experiment between the PlanetLab nodes in Berlin and North Carolina, but configured so that half of the mailboxes simulate an attack where they drop 75% of the arriving packets. This might occur if the mailboxes themselves were being flooded, or equivalently, if a portion of the filtering ring was being attacked. The sender and receiver respond to this loss by increasing the sending rate through the unaffected mailboxes to compensate; they could also expand the number of mailboxes in use, but this was not necessary to compensate for the simulated attack.

Figure 5 shows the result. After twelve seconds, the simulated attack begins. By increasing the sending rate, the receiver is able to maintain an average of 25 packets per second despite half of the mailboxes dropping most of their packets.

4.3 Simulation

Evaluating systems like 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 [22] 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 which 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 [22, 18] and assume that good clients communicate at a fixed rate of 160 Kbps.

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 6 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

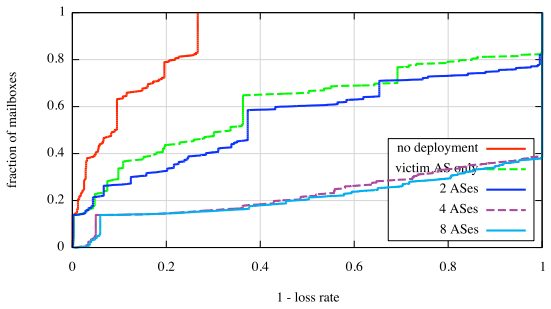

Figure 6: The cumulative fraction of mailboxes seeing at most a given fraction of goodput when communicating with the server.

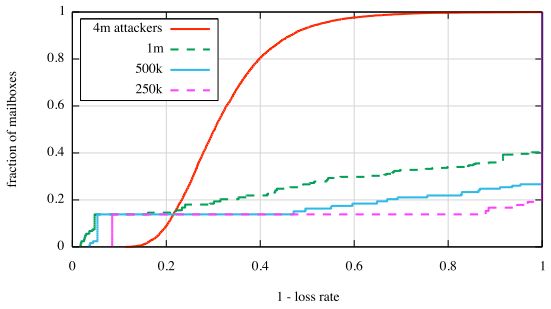

Figure 7: The cumulative fraction of mailboxes seeing at most a given loss rate for a varying number of attackers.

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 4 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 7 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. In practice, Akamai advertises more than 15,000 nodes and many of them are connected via 1 Gbps links, which would easily give another order of magnitude in protection.

We are also able to quantitatively show the benefit of the multipathing approach in Phalanx over previous capability-based schemes like TVA [40]. Figure 8

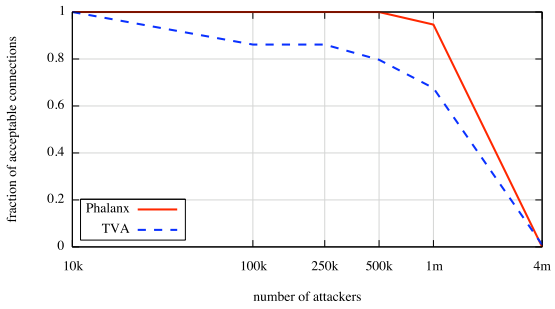

Figure 8: The fraction of connections which see less than 3% loss when sending 3 copies of each packet.

shows the fraction of connections which suffer acceptable loss when using a single-path system, like TVA, as compared with Phalanx. We use the same setup as the scalability experiment (mailboxes with 100 Mbps access links), with filtering rings being equivalent to TVA-enabled routers. Further we assume that all packets are sent three times to attempt to overcome loss. We define acceptable loss to be less than 3% after the redundancy.

The benefit of multipathing is immediately clear as Phalanx is able to provide almost all connections with acceptable loss rates even when defending against one million attackers. TVA on the other hand is forced to use a single path per connection meaning that some unfortunate connections will always have to endure high loss. Further, a calculating adversary can inflict the maximum possible damage with his available resources by targeting them on specific sets of links rather than being forced to spread his attack out.

5 Related Work

Our work on Phalanx leverages many elements from prior attempts to design a solution to mitigate DoS attacks. As we have argued, the primary differences come down to goals: ours is to design a system that could be easily deployed today, does not change the open model of Internet access, yet is powerful enough to deal with massive botnets.

5.1 Network Filtering

CenterTrack [35] and its later cousin dFence [24] propose new functionality for individual ASes to ensure that all traffic for a specific destination (e.g., one under attack) flows through certain filtering boxes. While effective for many attacks found in the wild today, this centralized approach is unlikely to scale to be able to handle the terabit attacks possible with multimillion-node botnets.

Pushback schemes [8, 9, 23] attempt to scale the power of network filtering by pushing filters to routers immediately downstream from where the traffic is entering the network. These filters are specially constructed to block attack traffic while allowing good traffic through. For example, in Active Internet Traffic Filtering [9], victims

alert their upstream routers of undesired flows. These routers temporarily block the flows while they in turn negotiate with routers further upstream to block the flows.

In the case of a multimillion-node botnet, where bots are found in almost every corner of the Internet, containing a botnet would require support for installing and managing filters at routers throughout the public Internet.

5.2 Capabilities

Capability schemes [7, 40, 10, 38, 39] have also received quite a lot of attention from the research community. Capability approaches drop (or relegate to lower priority) all traffic which does not carry a certificate proving that the packet was explicitly allowed by the recipient.

For example, in SIFF [39], capabilities are created during connection setup by each router on the path from source to destination. These routers stamp the initial connection request packet with time-limited cryptographic information. If the connection is desired by the destination, it returns the stamps to the sender, enabling it to prove to each router in the path that the connection is to be permitted. Because the capabilities are based on cryptographic hashes of flow identifiers, a secret key and local timestamps at routers, they cannot easily be forged or reused spatially or temporally.

The solutions for acquiring capabilities are noticeably less fleshed out and are usually some form of approximate fair queueing on the flow and/or path identifier. Furthermore, the capabilities are path-specific and any change to the route forces all flows using that path to reacquire their capabilities.

In Phalanx, we use a per-packet capability to identify which specific packets are to be allowed through the filtering ring at the request of a destination. Flows whose packets are not requested, do not have permission to send and are thus dropped. Additionally we make use of a more robust resource proof-based fair queuing scheme for the connection setup channel. Lastly, Phalanx uses loose source routing and rapidly refreshed capabilities to ensure that routing changes do not prevent the system from providing service even under attack.

5.3 Overlays

Overlay schemes [19, 5, 30, 33] leverage a set of trusted or semi-trusted nodes to act as proxies for end-hosts which may become the subject of attack. The common idea is to use more fully functional, and easier to deploy, machines to perform complex operations on packets, ones that might not be feasible at line rate inside of routers.

For example, one of the first overlay proposals for DoS protection was Secure Overlay Services (SOS) [19]. In SOS and later systems such as Mayday, only pre-authenticated users were allowed to route packets through the overlay. Hence these systems could not be used for protecting general Internet traffic. A valid source could send packets to any node in the overlay by including an authentication token, and these packets were then routed to a specific output node (generalized later to be a set of output nodes), which then forwarded the packets to the destination. If the destination IP address and the identity of the overlay output node were both kept secret, an attack would be difficult to mount.

Phalanx builds on the basic approach of using an overlay network; in our view, overlays are the only way we will be able to deploy a solution to multimillion-node botnets in the foreseeable future. However, we generalize on the prior overlay work, so that in Phalanx: (i) any Internet host can send packets through the overlay without pre-authentication, (ii) any overlay node can send packets to any destination, and (iii) there is a simple, scalable, and efficient protocol for installing network filters to prevent unauthorized packets from reaching the destination.

5.4 Resource Proofs

A relatively new technique borrows resource proofs from techniques to deal with Sybil attacks [15]. Resource proofs allow service to be given in proportion to the resources available to a given user. This has the effect of preventing attackers from flooding and thus drowning-out well behaved users.

Speak-up [37] proposes using bandwidth for resource proofs by having legitimate hosts send more requests during times of attack. This results in per-bandwidth fair queue. OverDoSe [30] and Portcullis [26] both instead have hosts solve cryptographic puzzles to provide per-computation fair queueing.

In Phalanx, as in Portcullis, we use cryptographic puzzles to provide per-computation fair queueing because computation is a local-only, reasonably cheap resource especially in comparison to bandwidth.

5.5 Architectures

In addition, several proposals have suggested completely new architectures that would make denial of service attacks much harder to mount. For example, Off By Default [10] proposes that the network be modified to establish routes only where explicitly allowed by the recipient; in other words, only legitimate sources would be allowed to send packets to a specific destination. Establishing and tearing down routes is a fairly heavyweight operation, however, while Phalanx achieves the same goal on a per-packet granularity.

Phalanx is perhaps closest in spirit to the Internet Indirection Infrastructure (i3) proposal, in that every packet is sent through a mailbox abstraction [34]. Secure-i3 [4] extends the original i3 approach for routing via DHTs

to use an almost-unlimited address space to help prevent certain attacks. Like Phalanx, Secure-i3 makes use of a level of indirection in between sender and receiver as well as a large swarm computers acting as intermediaries. Unlike Phalanx requests, i3 triggers last for long periods of time, usually minutes, instead of being installed on a packet granularity. Further, i3 assumes that all packets are carried over i3, so that it cannot be attacked from below. Phalanx is designed to withstand attacks carried over the underlying Internet.

6 Conclusion

In this paper, 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 used it to demonstrate its performance. We have further demonstrated Phalanx's ability to scale to million node botnets through simulation.

7 Acknowledgments

We would like to thank Arun Venkataramani for a set of conversations which helped us realize the need for more scalable DDoS protection. We would also like to thank our shepherd, Sylvia Ratnasamy, as well as our anonymous reviewers for their help and valuable comments. Additionally, this work was supported by National Science Foundation grant #CNS-0430304.

References

[1] Akamai technologies. 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. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Towards a more functional and secure network infrastructure. Technical report, UC Berkeley, 2003.

[5] D. G. Andersen. Mayday: Distributed filtering for internet services. In *USITS*, 2003.

[6] N. Anderson. Vint Cerf: one quarter of all computers part of a botnet. http://arstechnica.com/news.ars/post/20070125-8707.html, January 25 2007.

[7] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. In *HotNets-II*, 2003.

[8] K. Argyraki and D. Cheriton. Network capabilities: The good, the bad and the ugly. In *HotNets-IV*, 2005.

[9] K. Argyraki and D. R. Cheriton. Real-time response to denial-of-service attacks. In *USENIX*, 2005.

[10] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! In *HotNets-IV*, 2005.

[11] D. J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 1996.

[12] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[13] M. Casado, P. Cao, A. Akella, and N. Provos. Flow-cookies: Using bandwidth amplification to defend against DDoS flooding attacks. In *IWQoS*, 2006.

[14] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. In *IPTPS*, 2002.

[15] J. R. Douceur. The sybil attack. In *IPTPS*, 2001.

[16] P. Finn. Cyber assaults on Estonia typify a new battle tactic. http://www.washingtonpost.com/wp-dyn/content/article/2007/05/18/AR2007051802122.html, May 19 2007.

[17] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *CCS*, 2007.

[18] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging bittorrent for end host measurements. In *PAM*, 2007.

[19] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *SIGCOMM*, 2002.

[20] B. Krebs. Blue security kicked while it's down. http://blog.washingtonpost.com/securityfix/2006/05/blue_security_surrenders_but_s.html, May 2006.

[21] A. Kuzmanovic and E. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants. In *SIGCOMM*, 2003.

[22] 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.

[23] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *CCR*, 2002.

[24] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *NSDI*, 2007.

[25] A. McCue. Bookie reveals $100,000 cost of denial-of-service extortion attacks. http://software.silicon.com/security/0,39024655,39121278,00.htm, June 11 2004.

[26] B. Parno, D. Wendlant, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting connection setup from Denial-of-Capbility attacks. In *SIGCOMM*, 2007.

[27] B. Raghavan and A. Snoeren. Decongestion Control. In *Hotnets-V*, 2005.

[28] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM*, 2004.

[29] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), Apr. 1992.

[30] 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.

[31] E. Skoudis. Big honkin' botnet - 1.5 million! http://isc.sans.org/diary.html?storyid=778, October 2005.

[32] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the internet in your spare time. In *USENIX Security*, 2002.

[33] A. Stavrou and A. D. Keromytis. Countering DoS attacks with stateless multipath overlays. In *CCS*, 2005.

[34] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM*, 2002.

[35] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *USENIX Security*, 2000.

[36] M. Theimer and M. B. Jones. Overlook: Scalable name service on an overlay network. In *ICDCS*, 2002.

[37] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS defense by offense. In *SIGCOMM*, 2006.

[38] D. Wendlant, D. G. Andersen, and A. Perrig. Bypassing network flooding attacks using fastpass. Technical report, Carnegie Mellon University.

[39] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, 2004.

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