FastPass: Providing First-Packet Delivery

Dan Wendlandt, David G. Andersen, Adrian Perrig

March 29, 2006
CMU-CyLab-06-005

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

FastPass: Providing First-Packet Delivery

Dan Wendlandt, David G. Andersen, Adrian Perrig
Carnegie Mellon University

Abstract

This paper introduces FastPass, an architecture that thwarts flooding attacks by providing destinations with total control over their upstream network capacity. FastPass explores an extreme design point, providing complete resistance to directed flooding attacks. FastPass builds upon prior work on network capabilities and addresses the oft-noted problem that in such schemes, a sender must first get one packet through with no protection against DoS. FastPass provides cryptographic *availability tokens* to senders that routers verify before expiditing their delivery. We present two variants of the tokens. The first uses light-weight public key cryptography and is practical in high-speed routers with modest hardware additions. The second uses a symmetric hash-chaining scheme and is easily implemented in software. In sharp contrast to prior systems, our evaluation shows that hosts using FastPass can quickly communicate regardless of the size of the attack directed against the nodes.

1 Introduction

An ounce of prevention is worth a pound of cure.

– Benjamin Franklin

A packet saved is a connection earned.

– Anonymous

Over the past ten years, the Internet has expanded at a frantic pace, changing the way people communicate, shop, conduct business, and find information. While it seems likely that this growth will continue in many areas, we believe that a lack of guaranteed availability presents a significant barrier to the adoption of the Internet in many mission-critical settings, particularly those that involve elements of real-time control and response.

Denial-of-Service (DoS) attacks that consume network capacity are one of the persistent threats to availability in today's Internet architecture, and a subject of much recent study [16, 25, 26, 27, 34, 35]. Many of the techniques investigated in prior work make significant inroads against DoS, greatly increasing the resources an attacker must possess in order to mount an effective attack. Network capabilities schemes, such as SIFF and TVA, are a promising building block for a DoS-resistant network architecture. By prioritizing packets explicitly authorized by the destination, capability schemes elevate legitimate flows over floods of attack traffic.

In this paper, we study an extreme design point not offered by any DoS protection mechanism to date: **Regardless of how many nodes an attacker controls, an attack directed at a destination cannot prevent or delay an authorized source from communicating with the destination.** FastPass builds upon a existing network capability systems [34, 35]; its contribution is a mechanism that lets senders inform the network that a packet has been authorized by the receiver *with no active input from the destination*. This mitigates a fundamental criticism of capability systems: that they are vulnerable to attacks on the capability set-up channel [3].

Stateless router-based capability schemes require an initial request packet to reach a destination without the protection of a capability in order to "bootstrap" the process. In SIFF, an early stateless capability scheme, request packets were transmitted as best effort traffic. A host establishing a connection was treated identically to the larger volume of attack traffic. As a result, the host had to retry until its packets probabilistically got through. Improving on SIFF, TVA separated legitimate traffic from attackers by fair queueing traffic based upon the ingress interface to the current ISP. Though this provided a significant improvement over the prior best-effort design, in the best case, this scheme provides per-host fairness. Hosts still require $O(N)$ time to establish a capability.

This paper explores FastPass, which delivers the request in $O(1)$ time. FastPass routers verify cryptographic tokens in request packets to determine if a destination has authorized the traffic. Unlike conventional capabilities, these tokens are *not* path specific. As a result, they can be provided to the clients in advance or out of band. Once the capability has been established, subsequent packets are protected by the capability mechanism. On top of FastPass, we envision a system in which clients can obtain tokens in many ways. Examples include:

- A user obtains a token from a large, distributed token provider, similar to today's content delivery networks. The service may grant the token based upon a small financial deposit, proof of computational work, or by a history of being a good customer. The service is resistant to DoS because of its sheer size and geographic disparity.

- A bank customer obtains a cryptographic authentication dongle from the bank in order to securely access her account. The user's computer securely obtains a token from this dongle.

We first examine the properties that an ideal protected connection setup mechanism would provide (Section 2). We then examine two realizations of our system, FastPass-PK and FastPass-Hash (Section 3). The first is a design for a future Internet architecture that requires modest additional hardware in routers to verify tokens. The second provides fewer security guarantees but is less computationally intensive making it reasonable for deployment within today's Internet. Our evaluation (Section 4) finds that FastPass successfully provides O(1) setup time, and our implementation study (Section 4) finds that both schemes are practical for use even in high-speed routers.

2 Assumptions

We make assumptions in three main areas: router hardware, network architecture, and identification of malicious traffic.

In this section we discuss the foundations upon which our work builds; in Section 7, we discuss the trade-off between the benefits our system provides and the additional costs it imposes. While it may turn out that the costs are too high—or not—we believe that it is an aspect of the design space well worth understanding.

2.1 Router Hardware Assumptions

We consider future architectures that may have reasonable hardware support for cryptographic operations. The mechanisms we present require the addition of a shim header between the IP and transport headers, and we assume that routers and end-hosts could be upgraded to support this protocol in addition to whatever changes are needed for the core capability scheme.

2.2 Network Architecture Assumptions

FastPass extends a generic router-based *network capability* scheme similar to SIFF [34] or TVA [35]. A network capability is additional information carried by packets that encodes an authorization to send traffic to a destination. Capabilities can be validated by intermediate routers to determine their authenticity. Receivers provide these capabilities to the hosts that send packets to them to grant the senders permission to transmit.

A common realization of capabilities is for intermediate routers to cryptographically mark packets as they go by. The receiver can then echo the content of these markings (i.e., the capability) back to the sender, which the sender can then attach to subsequent packets to obtain an elevated level of service.

FastPass is agnostic to which capabilities it uses, but the capability scheme must (1) be **unforgeable**. Attackers must not be able to generate or steal capabilities to communicate with a particular destination. In addition, the capability scheme *should*: (2) require **constant router state**, independent of the number of flows or packets that traverse a particular router. (3) Be **efficient**, requiring a modest amount of computation at routers and a modest amount of space in packets.

FastPass requires a constant-memory traffic monitoring algorithm that can police flows to throttle those that exceed their rate. One such technique is a multi-stage filter, which can accurately track the largest flows through a router using constant space [12]. This technique is similar to that used in TVA to police elephants in constant space.

Finally, FastPass and its underlying capability-based architecture only affect traffic during periods of congestion. When the network is not congested, best-effort "unwanted" traffic will still reach the destination.

2.3 Identification of Malicious Traffic

DoS attacks on resources, such as CPU or memory, that are located directly on the end-host require only that the end-host identify and drop malicious requests. Bandwidth exhaustion attacks are significantly different, in that the resource exhaustion and damage to legitimate use occurs *before* the end-host can decide about which requests to accept or deny. FastPass and its underlying capabilities provide a mechanism for end-hosts to efficiently inform the network infrastructure about what traffic is legitimate and should be prioritized. As a policy-independent component of the network, FastPass itself does not detect attacks or decide what traffic to prioritize. Instead, it provides a building block flexible enough to handle a wide range of prioritization policies. For the remainder of this paper, we write under the assumption that end-hosts make these decisions, they could also be made by firewalls or other servers associated with the end host. Like all capability schemes, our evaluation of FastPass assumes that end-hosts can with some reliably distinguish legitimate traffic from attack traffic *once it has reached the host*, an assumption we believe can hold for most, but perhaps not all, Internet services.

We believe this assumption is realistic for a number of reasons. Internet services fall into several categories with respect to their ability to differentiate traffic. *Private* services (e.g., an organization's IMAP email server) have a pre-established, often small set of users. These services can easily distinguish traffic. *Authorized* services may serve large, open communities, but have the ability to distinguish users. A large online retailer would be an example of such a service. Finally, *public* services are those who have no means of differentiating users, and benefit from serving all users equally. Examples include sites such as Wikipedia.

Private and authorized services can continue to operate while serving only authorized users, perhaps with some reduction in traffic volume. Public services present a greater challenge. It is here that FastPass's ability for third-party

token granting can help. One can imagine a third-party service that collects a small, refundable deposit on behalf of one or more public services. In return, it grants these users tokens to ensure their ability to use the service, revoking the deposit only if the user (identified by their token) sends malicious traffic. While this is certainly not the only way to protect a large service, it hints at the ways that even a completely open service could be protected in a FastPass-enabled environment.

3 Design

FastPass aims to provide guaranteed connectivity in the face of attacks that flood a destination with traffic from remote machines. This section presents the design of FastPass by enumerating the goals it seeks to meet, outlining its basic mechanism of operation, and then examining two variants, one which provides better security and the other which is much less computationally expensive.

3.1 Definitions and Goals

We define *Time to Communication* (TTC) as the time required to successfully establish an "uninterruptable" connection protected by capabilities or filters. We measure TTC from the initiation of communication (e.g., sending a TCP SYN packet). In a capability-based scheme, the TTC would end when the sender receives a reply to its capability request. In a reactive filtering-based scheme like Pushback or AITF, we instead measure TTC as the time required to install enough filters to allow normal communication.

Both capability and filter based schemes protect against *direct bandwidth flooding attacks* in which hosts anywhere on the Internet direct traffic to the destination in order to congest a bottleneck link. We consider TTC within this framework and defer until Section 6 a discussion of *indirect* attacks that require an attacker to be located near the victim behind the bottleneck link.

FastPass provides a bootstrapping mechanism for a general capability-based *traffic authorization* mechanism, a process by which routers determine the relative priority for forwarding packets when capacity is limited. Many mechanisms could improve availability by protecting the initial set-up packet. To understand why we designed FastPass as we did, we first enumerate several desirable properties for whatever mechanism determines packet priority.

1. Provide a constant, small TTC: Mission critical applications and services should be able to guarantee connection establishment within a small, bounded amount of time. To do so, they must have complete control over the use of incoming capacity provisioned by their provider. This property should hold in the face of large traffic floods and be independent of the network capacity, the number of attackers, and the network topology between the senders and receivers. In effect, guaranteed availability should depend *only on the amount of incoming traffic already authorized by the destination*. This property is not achieved by current DoS-prevention mechanisms.

2. Be robust to malicious participants & non-participants: Any Internet-scale traffic authorization system should be robust to negligent or malicious components in the infrastructure. For the foreseeable future, routers will continue to have bugs, software will be subverted, and operators will make mistakes or even act maliciously. An ideal traffic authorization scheme is robust to misbehavior by entities not on the direct path between sender and receiver. If attacks remain possible, an attacker's influence should diminish as the distance from the victim increases [5].

3. Support a wide range of admission policies: Receivers may apply vastly different policies for allocating network resources. Private sites may allow access only to expressly authorized users. Public sites may require proof of computation or a Captcha [28] (reverse Turing test) before granting access. Each of these policies has merit in different circumstances; an infrastructure should not mandate a particular one. A Captcha, for example, would be inappropriate for a service accessed primarily by other programs, and computational puzzles can be biased against small clients such as PDAs. An ideal scheme could be adapted to support arbitrary authorization schemes.

4. Permit fine-grained control of incoming requests: An ideal scheme would provide control over the rate of traffic for both the data connection and the capability request channel. A receiver should be able to precisely specify which senders are allowed to send packets and how many packets they may send.

5. Support a flexible notion of identity: The identifier used by a destination to inform the infrastructure about its preference for or against a flow should be flexible and topology independent. For example, an authorization mechanism should not require that the user connect from a given IP address. The mechanism for identity should also have sufficient granularity as to identify different hosts on the same network, or even two different hosts behind a single NAT IP address.

By focusing on these design principles FastPass presents a novel solution to the Denial-of-Capability vulnerability and therefore represents a significant step forward in developing robust network availability solutions.

3.2 FastPass Public Key Design

The section presents FastPass-PK, a traffic authorization scheme based on the use of public key signatures to allow routers to verify traffic authorization tokens.

3.2.1 Overview

In FastPass, hosts obtain a *traffic authorization token* that allows them to connect to a destination even in the case of extreme network congestion. A host may obtain these to-

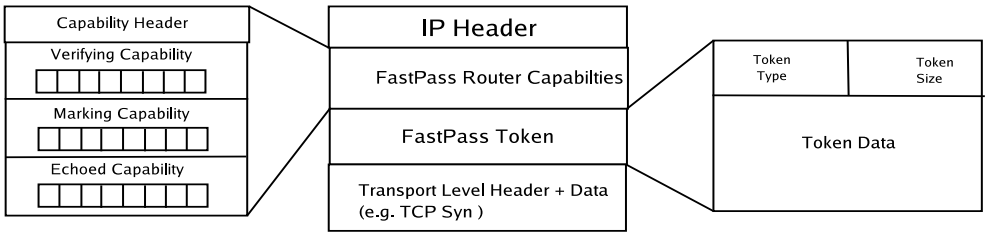

Figure 1: The FastPass connection setup packet adds two headers after the IP header. The capability header contains fields for three capabilities: one to be verified for priority forwarding, one to be marked by the router to build a new capability, and one to be echoed back to the remote host. The token header contains proof of authorization by the destination specified in the IP header.

kens in a variety of ways, as noted in the introduction: directly from the destination at an earlier time, from a third party service, or an entirely out of band mechanism.

The basis of FastPass-PK's tokens is a destination-specific public key signature, which routers can verify using the destination's public key to confirm that a destination has authorized the token-holder to send traffic to it.

At a high level, FastPass operates as follows:

1. A destination domain D generates a public/private key pair, K_D and ${K_D}^{-1}$.

2. D distributes its public key K_D along with its reachability information in a BGP-like inter-domain routing protocol. At each AS, this key is injected into the IGP. Each router thus has the public key of all destinations in its routing table.

3. D uses its private key ${K_D}^{-1}$ to run a token granting service, or provides pre-made tokens to a trusted third party to distribute tokens on its behalf.

4. When a source S wishes to have protected connection set-up to D at some point in the future, it requests a token from D's token granting service.

5. D's token granting service may require some proof of identification or work in order to grant S a token. To provide a token, it uses the private key to create a digital signature proving that the token-owner has the right to contact D at some point in the future. The token consists primarily of the signature itself, as well as some meta-data for expiration time and verifying uniqueness, as discussed below.

6. When S decides to contact D, it includes the token within a capability setup request packet (see Figure 1) and sends the packet to D.

7. Routers along the path mark the capability header as they forward it in order to create a router-based capability that will protect future packets sent to D. The exact markings are specific to the underlying capability scheme, such as TVA.

8. When forwarding a capability request, the router gives priority to packets by checking to see if they have a valid token. The router uses D's public key (acquired in step #2) and meta-data in the packet to verify that the token contains a signature generated using D's private key and is therefore authorized.

9. Request packets with valid tokens are placed in a high-priority traffic queue for forwarding, while packets with failed tokens are forwarded as best-effort traffic.

3.2.2 FastPass-PK Keys and Trust Model

FastPass-PK performs keying on a per-domain basis; we borrow from the idea of a "Failure Atomic Unit" in routing [29], and assume that the indivisible units of routing on the core Internet are directly tied to a particular organization and to a group of machines that are "linked by failure" (e.g., they share a common bottleneck, access link, etc.). While this granularity is not strictly necessary—FastPass could operate on a per-prefix basis—per-domain is a better match in the face of route aggregation and limited routing table sizes.

FastPass-PK uses the Rabin-Williams cryptosystem [23], known for its extremely fast signature verification, which only requires a single modular exponentiation. Each destination domain can generate its own public/private key pair (K_D, K_D^{-1}) independent of any other entity, where K_D denotes the public key, and K_D^{-1} denotes the private key. Although the public key could be signed by a Public-Key Infrastructure (PKI) and distributed as public-key certificates, we propose to distribute the public keys directly through the inter-domain routing protocol along with the reachability information. In general, overloading a protocol with additional functionality and requirements is not advisable, but in this case, the public key information is used to authenticate tokens for reaching the destination. Since the token verification process is intimately tied to the forwarding process for reaching the destination, distribution through the routing protocol is the preferred distribution mechanism to avoid inconsistencies between routing state and public-key information. The reason why we can avoid a PKI is due to the observation that traffic follows the reverse path of route

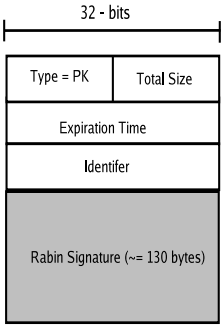

Figure 2: The public key token layout. At the top is the standard token header including expiration and id fields. The majority of the packet consists of the Rabin signature.

updates in BGP—thus, a malicious router that could replace the public key would be on the path towards that destination anyways. Hence, a PKI does not introduce a security advantage that would offset the overhead of establishing a PKI in this specific case.

Tokens consist of a short message with a digital signature of that message, to guarantee the authenticity of the token when it is verified by a router. The message consists of two values: I, an identifier unique to all currently valid tokens distributed by a given destination; and T, the global time at which the token expires and is no longer considered valid. The signature S is a Rabin-Williams signature of these two values under the private key of D: $\{I \mid\mid T\}_{K_D^{-1}}$. The signature ensures integrity and authenticity of the token information, preventing alteration of the identifier or the expiration timestamp.

Source address information is omitted from the token by design. Tokens can thus be given to a sender without knowing anything about how, or even with what device, that individual will eventually contact the destination. This provides the *flexible identity* identified above that handles issues such as mobility or NATs and also greatly expands the possible mechanisms for token distribution.

3.2.3 Token Granting and Management

FastPass intentionally does not specify a specific mechanism for token dissemination or for the access control policy used by the destination to grant these tokens. This generality is a major strength of the design.

Tokens can be generated by standard PCs without specialized hardware. In fact, in the simple case tokens may be generated by the same machines that provide the service being protected. In this way, a host already using the service can assure reachability at a future time. A different service might use a trusted third party running a large Akamai-like CDN to distribute tokens. A destination can pre-generate tokens and provide them to the third-party with instructions on the admission policy for distributing the tokens. More generally, any out-of-band mechanism, ranging from text-messaging to carrier pigeon, can be used to communicate token information. Finally, mechanisms such as smart cards could allow trusted users to generate tokens themselves without ever contacting the destination.

In addition to standard capability functionality, end-hosts will also need to have software for managing their tokens. This consists primarily of storing acquired tokens and potentially refreshing tokens that are about to expire; no cryptographic processing must be performed. Token management software might also assist in the transaction between the host and the token granting service, for example, prompting a user for a username and password or performing computation as a proof-of-work.

3.2.4 FastPass-PK Protocol Details

FastPass tokens protect capability request packets by giving them priority over un-authorized packets as they proceed to the destination and acquire their initial router-based capability.

Conceptually, capability systems have two output queues per interface: authorized traffic and best effort traffic. The former is strictly prioritized over the latter.

When a router receives a capability request packet, it first marks the packet's capability header to add its contribution to the future router-based capability. It then checks to see if a token has been included in the packet. If no token exists, a next-hop lookup is performed and the request is placed in the best-effort queue for forwarding. If there is a token, the router first checks the expiration time T of the token against its internal clock to see if the token is still valid. This function requires coarse time synchronization on the order of minutes. Next, the router uses a Bloom Filter-based [13] duplicate detection scheme (Section 3.2.5) to test if it has seen the token ID I before. If the token is expired or a duplicate, a route lookup is performed and the packet is placed in the best-effort queue with the token removed.

If I and T are valid, the router looks up the next hop and the public key associated with the destination of the packet. The router then checks the Rabin-Williams signature of I and T and places verified tokens in the authorized traffic queue. Those that fail are stripped of their token and transmitted best effort.

As in TVA, we assume that capability request traffic is limited to a small static fraction, such as 5%, of link capacity, with the remainder of the link available for capability protected and best-effort traffic. For each outgoing interface, a router can only transmit packets with the valid bit set at 5% of the link rate. If more valid requests exist for an outgoing queue then capacity to transmit them, as many are checked as possible and the remainder are forwarded best-effort.

Once the packet arrives at the destination, this host then determines whether or not to echo a capability to the requester. Possessing a valid token does not imply that the

destination will grant the requesting host the right to send it capability-protected data traffic. A token simply grants the sender the ability to prioritize a request as it traverses the path to the destination. Note that if desired, the token protected request packet can also include a return-path token to protect the server's capability reply packet.

3.2.5 Minimizing Token Reuse

An attacker with a single token must not be able to reuse this token to mount a successful attack. FastPass ensures two properties governing token reuse: (1) A token can only be used at a very infrequent rate on any given path; (2) As a result, only a small number of duplicated tokens will reach the links near the victim.

FastPass provides these properties by hashing the token's ID and the destination AS into a Bloom filter and rejecting ID/destinations that have appeared previously. This approach limits token reuse to once per s seconds, an effective limit on the power of a replay attack. Bloom filters provide very compact lookups and have no false negatives, but they do have a small false positive probability, which we can drive to arbitrarily low rates. To illustrate this, consider a router with a gigabit link on which 5% of the capacity is allocated to token requests. The router will process up to 40,000 requests/sec. A Bloom filter of 160 KB can prevent duplicates for one second with a false positive rate under $\frac{1}{1{,}000{,}000}$. A circular buffer configuration of s Bloom filters can therefore filter traffic for s seconds with a $\frac{s}{1{,}000{,}000}$ false positive rate. The false positive rate can be reduced with more memory, and its effects can be driven arbitrarily small by giving valid users multiple independent tokens to use on subsequent request attempts.

Importantly, this scheme also ensures that a particular token can only be used once to attack any given link while the router duplicate state is still valid. DDoS attacks depend on a flood of traffic converging on the victim; the Bloom filters ensure that only a small trickle of replayed tokens are able to reach the victim's bottleneck.

3.2.6 Token Verification Speed

When we began designing a traffic authorization scheme, we intended for the public key scheme to be a strawman, because we "knew" that it was too computationally expensive to allow near the data plane. To our surprise, as we discuss in Section 5, a user-level software router implementation of our system using Rabin-Williams signatures is capable of processing 12,400 requests/sec, and a hardware implementation should be capable of well over 1 million.

3.3 Fastpass-Hash

While FastPass-PK is in the context of a future Internet architecture, it is computationally expensive enough that a change in the trend of computational power vs. network capacity could render it quite expensive. As an alternative, we present FastPass-Hash, which requires little additional computation to process verification packets, at the cost of some resistance to compromised or malicious routers.

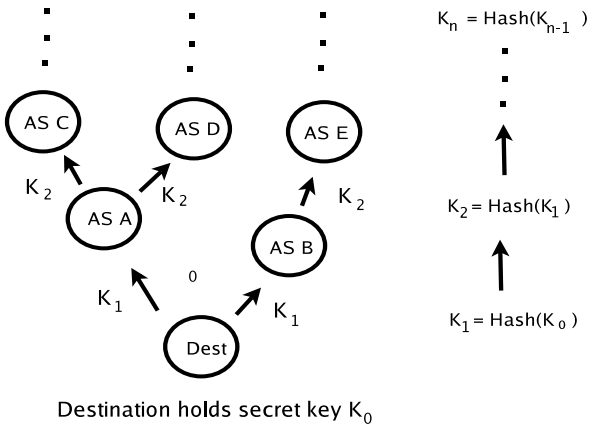

Figure 3: The hash-chain tree rooted at a destination with secret key $K0$.

3.3.1 Hash-Chain Token Design

FastPass-Hash replaces FastPass-PK's use of public-key cryptography with significantly faster symmetric key operations based on a chain of hash values.[1] The disadvantage to this approach is that routers now contain a destination specific secret that if exposed increases the destination's vulnerability to bandwidth exhaustion attacks. We now explain the ways in which FastPass-Hash deviates from the design presented above.

Each destination domain D has a secret symmetric key K_0, used to sign capabilities. Instead of distributing its public key, the destination releases a hash of the secret key under a public one-way hash function H to create $K_1 = H(K_0)$. D releases K_1 to each of its directly connected upstream providers along with reachability information in BGP. Each provider uses H to hash the value they received and create $K_2 = H(K_1)$, which is then passed on to any domains to which that route to D is advertised.

The result is a tree with branches that are identical hash-chains K_0, K_1, ..., K_i rooted at the destination and following the same AS-level path traversed by its BGP reachability information, as shown in Figure 3. Each domain uses the K_i it received as the shared secret between a destination and all routers in that domain. Note that a domain's key is not unique, as all AS's i hops away from D will have K_i.

This shared secret allows the router to verify that tokens came from D, although with weaker guarantees than public key cryptography, as we discuss below.

[1] The first network capability design used a hash chain to time out capabilities, not to authorize request packets [2].

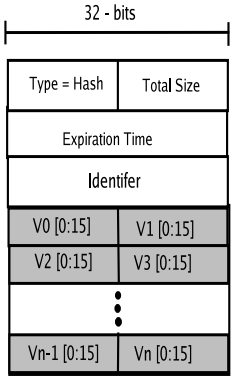

Figure 4: The FastPass-Hash token contains a standard token header as well as a series of 16-bit values derived from the hash V_i = Hash(T,I).

As in the public key case, each token contains T and I values representing the token expiration time and a token ID unique to the destination. These values are included in the token as plaintext values. A FastPass-Hash token consists of a list of values V_1, ... , V_n, where $V_i = H(K_i, I, T)$. V_i is the value needed to prove authorization to a router located in an AS that received the key K_i. When generating the token FastPass-Hash uses the AES block cipher in one-way function mode [8] with a publicly known 128-bit AES key k_{AES}. The output of this function is a 128-bit hash, $O = E_{k_{AES}}(I \mid\mid T) \oplus (I \mid\mid T)$. Only the first 16 bits of this value are used as V_i to conserve packet space. When generating the token, the granter decides on the number of V_i values to generate by estimating a bound on the AS-path length between the source and destination.

Router behavior upon receiving a token mirrors the description of FastPass-PK, with the exception of how the token itself is verified. A route lookup for the destination will yield the key K_i, which the router uses along with the plaintext I and T values to perform the hash operation. In parallel the route then compares this computed value against all of the values V_i included within the token. If one matches, the router verifies that the token is not a duplicate and has not expired and enqueues it with priority. If any of the tests fail, the token is stripped and the request is placed in the best-effort queue.

3.3.2 Security Analysis

The main drawback of FastPass-Hash is that the compromise or malicious operation of a router containing K_i means that arbitrary tokens can be forged for destination D in *all* domains that have a secret K_j where $j \geq i$. We refer to such AS's as "outer domains". This results from the fact that knowledge of the compromised hash value allows computation of all subsequent hash values. However, FastPass-Hash has the desirable property that the highest degree of trust goes to those few domains closet to the destination. Logically, the closest domains have D as a customer, as a peering partner, or as a customer of a customer, etc, and likely have incentives to protect the key. As a result, FastPass-Hash provides strong protection on par with FastPass-PK against a large number of directed attackers attempting to exhaust a bottleneck link near the destination. As the hash-chain lengthens there is a higher likelihood that someone in a chain has exposed a key value, but the ability to cause damage through the use of an exposed secret also declines with distance from D, offsetting much of the risk.

Standard security measures must be used to safe-guard routing information during communication and storage to keep the K_i values private. However, the possibility of an intentional compromise by an operator also merits consideration. We reject the use of "trusted

computing" mechanisms to hide keys from operators as too heavyweight and opaque for actual deployment. We briefly suggest two flexible approaches to reduce the risk of key exposure:

1. "Distancing" untrusted domains: It may be undesirable to trust a neighboring domain even though it is topologically close. To solve this problem, a domain could hash a value several times before passing it on to a less trusted upstream or peer, reducing the harm from key exposure by one of these parties.

2. Releasing multiple hash-chains: Another way to mitigate the risk of exposed keys is to release multiple key chains along different paths. If a key of one chain is exposed, paths covered by other chains remain protected. However, unless the client is capable of informing the token service which chain it is on, tokens must include values for all possible key chains.

Fastpass-Hash provides a light-weight counterpart to FastPass-PK which trades off security for a fast and cheap implementation that maintains the remainder of key FastPass properties, including bounded time to communication. Significantly, FastPass-Hash is fast enough that it could be used to protect *all* packets, not just connection setup requests, if a designer were so inclined.

4 Analysis & Evaluation

This section presents an analytical evaluation of FastPass, followed by an evaluation of our implementation of FastPass-PK within the Click modular router [20].

4.1 Attack Power Analysis

In today's Internet, attackers can easily overwhelm victims with traffic. Each attacker can generate an abnormally high volume of traffic, sending at a rate proportional to its upstream capacity C. A well-behaved sender will back off when its packets are lost, retrying at an interval of seconds, sending at a rate $R_{good} << C$. The *attack power* of an aggregate of N_{bad} attackers is thus $\frac{N_{bad}C}{N_{good}R_{good}}$. When the number

of attackers is large, as in a DDoS attack, the effect of the attack is to effectively eliminate the good traffic, as even a few percent loss causes good senders to time out and retry. Early capability-based systems such as SIFF used this model, relying on retries over time to eventually establish a protected channel.

To address this, a number of approaches have been proposed to even the playing field. In the absence of spoofing, a system that fair-queues based upon the sender's IP address reduces the effect of asymmetric sending rates. With perfect fair queueing, the attack power of a set of attackers if $\frac{N_{bad}}{N_{good}}$, a significant improvement. Similar parities can be achieved by requesting that good senders *increase* their sending rate when the receiver is under attack [30] or through the use of computational puzzles [9]. These designs effectively leave an attacker's power linear in the number of attack hosts instead of linear in the aggregate bandwidth of those hosts, a significant improvement.

TVA fair queues requests based upon their ingress interface into the current ISP. Because using either the source IP or a path marking such as Pi [33] is susceptible to spoofing either by end-hosts or by compromised routers outside of the ISP, the designers of TVA felt that this approximation is a better choice than a potentially "more fair" but less secure approach based on per-source fair queueing. Unfortunately, the necessity of relying on only information generated at the ISP gives attackers an advantage under some real-world topologies, which we examine further in Section 4.3.2.

In contrast to prior schemes, FastPass sets up a secure channel in constant time, regardless of the number of attackers, provided the attackers do not have valid tokens and that the source has not "oversold" its capacity (Section 6). FastPass avoids the topological dependence of earlier schemes by making use of an identifier that both precisely identifies the source *and* is unforgeable.

4.2 Evaluation Implementation

We have implemented FastPass-PK in the Click modular router [20], running on top of a generic capability system based on TVA [35]. Our prototype runs in userland because of the absence of public-key functions in the Linux kernel.[2] The implementation uses the Rabin-Williams signature implementation from SFS [21], which in turn uses the GNU MP library. Our implementation forwards 19,200 152 byte unsigned packets per second and 12,400 public-key signed packets per second. The unsigned packets are limited by running in user space, and the signed packets (intended to be 5% of the link capacity) are limited by cryptographic processing.

We ran experiments using the Emulab [31] pc3000 hosts equipped with 3GHz 64-bit Intel Xeon processors and 2GB RAM. To maximize the data-rate available for comparing TTC for different schemes, unless otherwise noted the entire capacity of the router dedicated to the request channel. To assure that cryptographic processing or userlevel forwarding do not create bottlenecks in our testbed, we artificially constrain link capacities in the network. In the scenarios described below, the request channel for a backbone link is 5 Mbps and the request channel the bottleneck link is 1 Mbps. With five percent of a link dedicated to requests, this setup represents 100Mbps and 20Mbps links, respectively. We limit each attackers to 240 Kbps, or a quarter of the victim's capacity.

Each Emulab host emulates a single large Internet domain, similar to a transit AS. Domains represent different regions of trust in the network, and so requests are prioritized as they enter the domain. With TVA, this means that packets are fair-queued based on their entrance point into the trust domain. FastPass examines tokens when packets reach a new domain. To provide a more realistic topology, each domain connects to a number of "neighbor" networks that are co-located on the same physical machine in our experiments. These neighbors represent customers or peers of the main network, and are connected to the domain with 2Mbps links. These additional networks are important because TVA handles requests by attempting to isolate traffic from different networks from those that contain attackers. Links between large domains have a static latency of 10 ms, but no latency is imposed on the communication between a domain and its neighbor networks.

For each experiment run, each domain in the topology has the same number of attacking hosts, and sends the same number of capability set-up requests. Attackers and senders are randomly assigned to a neighbor network, with attackers constantly flooding capability setup requests to the victim. Legitimate senders are randomly placed into a neighbor network and send their initial request at a random time. Our experiments set the capability request timeout to 500ms, an aggressive but reasonable value that favors schemes such as TVA or SIFF which retransmit during congestion.

We limit the number of neighbor networks with attackers to at most one-third of the neighbors of each domain. This favors TVA, which assumes that good request traffic can be isolated from bad traffic on a per-domain basis.

4.3 Time to Communication

We calculate Time to Communication (TTC) as the time from when a request is first sent to the time the capability reply is received, *minus* the minimum propagation time between the nodes. TTC thus includes the time the packet is queued and time spent waiting for a timeout as a result of packet loss, but is independent of the propagation delay. A perfect TTC is zero.

The following experiments compare FastPass, TVA's AS-input marking, and the simple best-effort requests used in SIFF in order to understand the reasons for their differing performance.

[2] We are currently porting our implementation to kernel-Click and believe this implementation will be complete by the camera-ready deadline, if applicable.

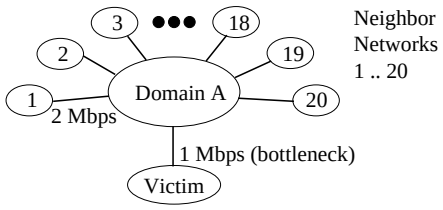

Figure 5: Barbell Topology, with one domain and 20 neighbor networks. The attack victim is connected to the domain by the 1 Mbps bottleneck link.

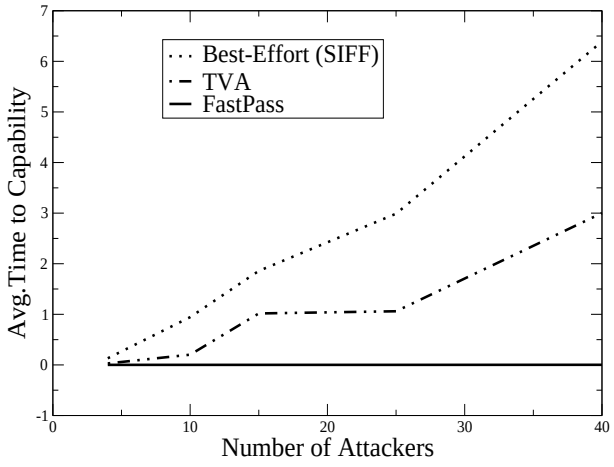

Figure 6: Average TTC for FastPass, TVA, and best-effort as the number of attackers increases. TVA and best-effort grow linearly, while FastPass is constant.

4.3.1 Simple Barbell Topology

We first consider the simple barbell topology in Figure 5. There is one domain with 20 neighbors that is connected to the destination by a 1Mbps bottleneck link. Figure 6 shows the average TTC each scheme provides for a given number of attackers. TVA clearly outperforms best-effort, but the TTC of both grows linearly with additional attackers. In contrast, FastPass's TTC is constant (and very small), since packets with valid tokens have priority over attack traffic, regardless of the number of attackers.

Interestingly, the barbell is an ideal topology for TVA, because the bottleneck router can perfectly distinguish good and bad traffic on a per-domain basis. Figure 7 shows that when $\frac{2}{3}$ of the domains are "good", those good domains have a perfect TTC, while nodes in the remaining domains must compete with the attackers near them. This causes the long tail observed in the figure. FastPass achieves a near-zero TTC for all hosts.

4.3.2 Dual Domain Topology

The "dual domain" topology shown in Figure 8 contains more than one trust domain, so TVA cannot perfectly isolate all legitimate users. Good and bad traffic from domain B has mixed by the time it reaches the bottleneck, so all good nodes in domain B are at a disadvantage. Figure 9 shows the CDF of TTC using this topology. TVA clients in domain "A" have a high success rate, but TVA clients in domain "B" experience significant delays and failures. Note that the "x" axis is longer than in the previous figure. FastPass again satisfies all requests immediately, demonstrating the value of an authorization mechanism that does not depend on topology.

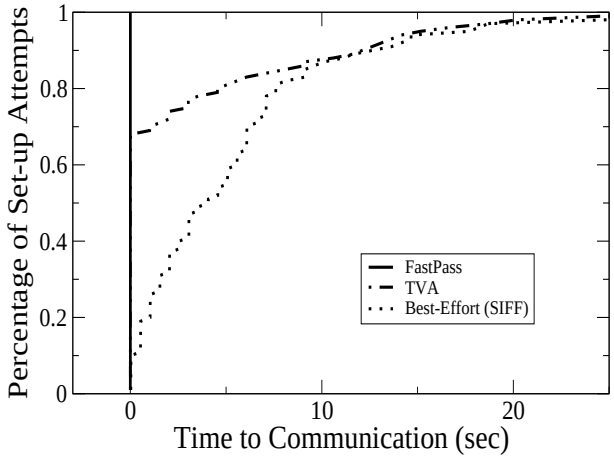

Figure 7: The CDF showing completion times for the single run of each mechanism with 40 total attackers on the Barbell topology.

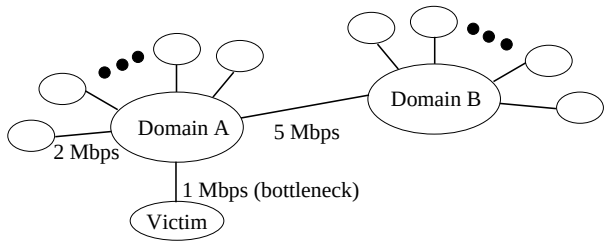

Figure 8: The dual domain topology, each with 20 neighbor networks. Domains A and B are connected via a 5 Mbps link, while Domain A is the sole provider of access for the victim over a link of of 1 Mbps.

5 Implementation Considerations

When proposals such as secure BGP have met with concerns about using public key cryptography on the *routing* plane for performance reasons [18], why do we believe it may be feasible to use similar techniques on the much more performance-critical *forwarding* plane? This section examines the performance of optimized public key cryptographic operations on modern hardware to explore the hardware costs of deploying a FastPass-like architecture. Perhaps surprisingly, we find that a high-end PC could support a giga-

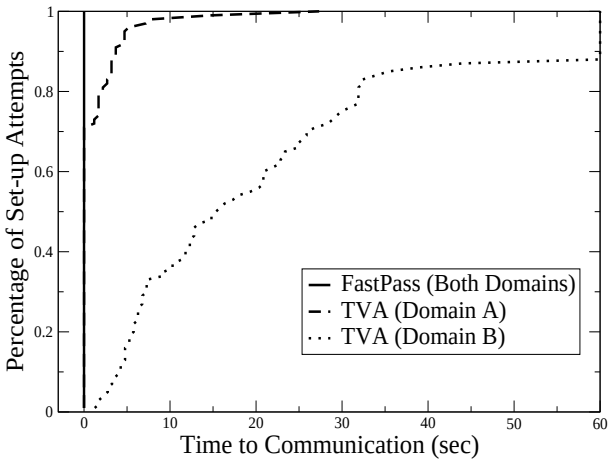

Figure 9: CDF from the dual domain topology showing FastPass and TVA performance with 40 attackers. The FastPass lines for both domains overlap, but the TTC for TVA clients in domain B is degraded.

bit per second of normal FastPass-protected traffic with no hardware cryptographic assistance.

Our implementation of the FastPass public key tokens uses the extremely fast Rabin-Williams (RW) signature scheme, which has the exceptional property that it requires only a single modular multiplication for signature verification. This overhead is in stark contrast, for example, with RSA verification, which requires roughly 1,500 modular multiplications. It is the extremely low cost of RW verification that makes FastPass-PK practical.

The second reason FastPass is practical is because the tokens need be applied to only a fraction of the traffic—the capability request packets. Schemes such as TVA then automatically refresh capabilities by piggybacking them on top of existing, protected traffic. In the resulting system, the token-protected request channel can be a small fraction—perhaps 5%—of the capacity of the network links.

This section first examines the cost of verifying RW signatures in software on a modern microprocessor, and then extrapolates from recent FPGA implementation results to estimate the performance of implementing FastPass with customized hardware support. Following this exploration, we examine the (much lower) costs of the hash-chain scheme.

5.1 Software FastPass Verification

While hardware implementations are the likely choice for high-speed routers, lower-end routers may also need to verify tokens. In addition, tokens will often be generated by servers running commodity hardware. Table 1 shows the speed of signature verification and generation on a 3.2Ghz Pentium-IV based PC with 1GB of RAM running Linux 2.6.11. The Rabin-Williams implementation is from SFS [21].

Operation	single op	rate
Rabin verify - 1024	61 μsec	16,459 / sec
Rabin verify - 2048	182 μsec	5,490 / sec
Rabin sign - 1024	3.4 msec	297 / sec
Rabin sign - 2048	20 msec	48 / sec

Table 1: Rabin-Williams software benchmark results.

Signature verification, at roughly 12,400 operations per second, is the clear bottleneck for a software router. Click is capable of forwarding over 400kpps on older hardware [20], and recent measurements indicate that it can exceed 1Mpps. We therefore conservatively assume that a kernel-Click router performing FastPass-RW verification could support 12kpps of token traffic, which then permits 240kpps of data traffic. At an average packet size of 512 bytes, such a router could support 983Mbps.

Recent hardware trends are also very favorable for a software FastPass-PK implementation. First, newer 64-bit architectures speed many of the large multiplication operations needed for cryptographic operations. While our Emulab evaluation used 64-bit machines, the Rabin-Williams implementation was not optimized to make use of them. Second, each packet's RW signatures can be verified independently. This type of "embarrassingly easy" parallelism is ideally suited to the multi-core processor architectures that are emerging.

5.2 Hardware FastPass Verification

How fast could a dedicated hardware implementation of FastPass perform? This section presents a *rough*, conservative estimate of how fast hardware implementations in FPGA and ASIC packages might perform.

Rabin-Williams verification is based upon modular multiplication of very large numbers. Most hardware implementations of RW use optimized Montgomery multiplication [24] to achieve very fast ASIC and FPGA results. As we explain below, a contemporary implementation on a high-end FPGA should be able to achieve over $\frac{1}{2}$ million ops/sec in an FPGA, and nearly 2.5 million ops/sec in a custom ASIC implementation. This rate is 10% of the forwarding rate of the fastest OC-192 line card available for Cisco routers [10]. While an ASIC implementation may be costly, the line cards that would require it already cost in excess of $100,000.

An FPGA implementation in 2003 by McIvor et al. achieved 69,306 1024-bit modular multiplications/sec on a Xilinx XC2V3000 FPGA [22]. This implementation used a 71Mhz clock rate and consumed 10,332 FPGA slices. We conservatively assume that a modern FPGA implementation can operate at 140Mhz [32][3]. A large FPGA provides roughly 100,000 slices. As a result, a parallelized FPGA

[3]The 2V3000 was a mid-range FPGA when the original study was performed. Here, we compare to the capabilities of the most recent Xilinx virtex 4 4VLX200 FPGA, with a maximum system clock speed of 500Mhz.

implementation on a single core should be able to achieve $\frac{140}{71} * \frac{100{,}000}{10{,}332} \approx 20$ times the performance of a non-parallel implementation in 2003, or roughly 1.3 million ops/sec. I/O limitations reduce the off-chip I/O to about 9Gbit/sec, or about 3 million ops/sec. Applying a reasonable fudge factor to account for the other, much less expensive operations in verifying a RW signature, we conclude that an FPGA implementation can support 500,000 packets per second of capability request traffic. We assume that an ASIC could run at twice the clock rate (280Mhz) with at least 3x as many computation units as the much less dense FPGA implementation. An ASIC could therefore support well in excess of 2.5Mpps.

5.3 Cryptographic Optimizations

Unlike conventional cryptographic applications, FastPass has additional flexibility that may lead to even more efficient solutions. For example, rare incorrect token verification is tolerable. A false positive (permitting a forged signature) is preferable to a false negative (ignoring a valid signature), since the former is likely to be caught at a subsequent router. Even false negatives may be tolerable if their probability is sufficiently low, and they do not deterministically deny service to a particular client. As an example, Bernstein has devised an approximate Rabin-Williams verification with a low (2^{-100}) false positive rate that provides significant performance benefits [7] that shows promise in being adapted to provide significant performance improvements in FastPass-PK.

5.4 FastPass-Hash Implementation

The Hash-chaining scheme used in FastPass-Hash (Section 3.3) requires routers to implement a keyed hash (a MAC). Our design uses the AES block cipher in one-way function mode [8], which requires one block cipher operation for input sizes smaller than the hash size (128 or 256 bits). Our design deliberately keeps the token size under 256 bits to eliminate the otherwise expensive key-setup step for hash generation.

The OpenSSL AES-128 CBC implementation on our Emulab nodes performs 4.1 million ops/sec, and the AES-256 cbc performs 3.3 million ops/sec. As a result, a software implementation of FastPass-Hash is would be suitable even for some of the fastest links available today. For extremely cutting-edge interfaces (e.g., OC-768 line cards), the hardware support for the FastPass-Hash scheme is very simple. Numerous AES hardware implementations exist, for FPGAs or ASICs, that can process many gigabits of data per second. The fastest implementations are, in fact, faster than any commonly available wide-area network link [15].

5.5 Ingress Verification

If verifying tokens for even a small fraction of traffic proves too expensive, an ISP may choose to verify signatures only at their trust boundaries, using an internal "verified" flag to permit subsequent routers to act on the request packets without extra computation. This scenario is similar to the use of MPLS or other technologies to avoid large routing tables and routing lookups in the core of a meshed network [11], and is already familiar to large ISPs. In a similar vein, for either performance or deployment, routers could offload token verification to either a directly connected processor or—because tokens occupy only a small fraction of the network capacity—to a remote server via a tunnel. Such an approach may be easier to deploy in the short term, and could help pool hardware resources more efficiently.

6 Oversubscription & Provisioning

FastPass provides an extremely reliable way for receivers to inform the network whether traffic is desired or not. This provides strong guarantees in the face of a *directed attack* in which malicious hosts anywhere on the Internet generate floods of traffic addressed to the destination. Resilience to directed attacks alone, however, does not guarantee that a sender and receiver can communicate when they desire. Any link along the path between the two hosts could still be congested by traffic flowing between other hosts. We term the deliberate creation of such congestion by colluding attackers an *indirect attack* This section examines how such attacks can be performed and considers a number of potential defenses against them. It concludes by explaining why the damage caused to a particular victim by an indirect attack is strictly less than the damage that can be caused by a direct attack.

In a collusion attack, a host C shares a bottleneck link with a victim destination D. C authorizes hosts anywhere on the Internet to send it large amounts of traffic, filling the bottleneck. Such an attack can be effective against both the data channel and the token validation channel in FastPass, and against the capability request channel in other systems. In general, any system that determines priority purely based on whether it is authorized by the destination is vulnerable to this type of attack. For example, filtering schemes such as Pushback and AITF permit victims to filter traffic that is destined to the victim; they cannot block traffic destined to other hosts, even if that traffic congests a link shared by the victim. The same problem was noted as a challenge to TVA [35].

Collusion is an instance of oversubscription, in which the collective amount of traffic authorized by destinations is greater than the bottleneck capacity of the links it traverses. This general problem of quality of service has been a subject of considerable research. While solving this general problem is important, we restrict our attention to over-

subscription caused by malice, because it is not as easily addressed through over-provisioning.

In general, an indirect attack is strictly weaker than a direct attack because of topological dependencies. First, an attacker must have nodes located strategically near the victim, as opposed to a direct attack in which the attack nodes can be virtually anywhere. (We note, however, that a domain that was sloppy with its key management could serve as such a resource; an attacker need not always have a compromised host in that location.) Second, by definition, an indirect attack spreads its resources more widely than a direct attack, unless the routes to the colluder take identical paths through the minimum capacity links to the victim.

An indirect attack can be launched against either a sender or a receiver, but attacks against receivers are more serious: Sending nodes located near a sending victim can attempt to clog the victim's upstream capacity, but the attacking nodes are limited by their own access link capacity. In contrast, a receiving node near a receiving victim can authorize an *unlimited* amount of inbound traffic.

Fortunately, destination-based fairness or per-destination capacity limits offer a good—though not perfect—solution to the problem of indirect attacks. ISPs can use a number of simple, effective tactics to reduce the power of many collusion attacks. For example, the ISP can push a set of limits to its border routers. Examples of effective limits include:

Per-destination-AS fair queueing: Ensure that traffic is shared between destination autonomous systems, perhaps with appropriate weights to account for large and small customers, or those who pay different amounts.

Customer capacity limit propagation: If a customer has an access link capacity of x, then no border router should pass more than x traffic to the customer.[4]

Elephant squashing: Like the monitoring scheme used in SIFF and TVA, monitor the largest F flows and limit them so that no destination AS consumes more than $\frac{1}{F}$ of the total link capacity.

While such limits are not perfect, they greatly reduce the effects of collusion attacks. Interestingly, these tactics are completely *ineffective* against directed attacks that come from widely dispersed sources: under such an attack, they merely push the packet loss to the edge, but good flows still experience vastly increased loss. Only in conjunction with a scheme to prevent directed attacks do they show their value. With each of these schemes, the fairness decisions can be made on a *per-AS* basis instead of a per-flow basis, requiring far fewer resources to implement. The limits could likely be tied in with a customer's service level agreement (SLA) that may already exist to govern latency and throughput guarantees.

[4]An intriguing question for future study is whether such limits on the data plane could be enforced *automatically* by the border router attached to the customer by preventing the customer from authorizing more than a small multiple of its actual bottleneck capacity. Unfortunately, such a restriction is not possible for the token channel, since they may be granted and used at arbitrary times.

Per-destination filters of this sort limit destination-based attacks to a multiple of a colluder's capacity, the same limit that is already faced for sender-directed collusion attacks. Protection for senders can be provided more easily by fair queueing on a per-sender basis in the links near the sender. At this point, the level of aggregation is relatively low, and ISPs already have per-sender configuration information.

Finally, indirect attacks are more susceptible to correction through policy or legal means. In a directed attack, the attacking nodes can be located well away from the victim, both physically and in jurisdiction. In contrast, colluding attacks require nodes located near the victim. Such nodes would be more related to the victim—sharing the same upstream ISP, for instance—which would facilitate identifying the colluders and enforcing AUPs and applicable laws.

7 Discussion

FastPass attempts to satisfy an ambitious goal: providing 100% resilience to directed attacks. At the cost of surprisingly modest hardware additions, we believe that it does so. In Section 3.1, we enumerated several goals that an ideal traffic authorization scheme should meet. How well does FastPass meet those goals, and where does it fall short?

1. Constant, Small TTC. FastPass succeeds in this regard; to our knowledge, FastPass is the only DDoS-resilience scheme that provides O(1) time to capability, regardless of an attacker's strength. As noted in Section 6, however, FastPass and its kin require additional support to deal with indirect attacks, and do not do so perfectly.

2. Robust to Malice. The robustness of FastPass depends on three subsystems: The cryptographic strength of its keys, the security of the token distribution mechanism, and the routing system that distributes keys. FastPass-PK and FastPass-Hash's keys are as secure as their underlying public and symmetric cryptosystems, respectively. We assume for now their strength, but recognize that a compromise in these schemes could require extensive changes to Internet routers.

Preserving token secrecy is important to ensure that tokens are only used by their intended recipient. While FastPass allows arbitrary token distribution mechanisms, such mechanisms should be designed with care to avoid subverting the effectiveness of tokens. Secrecy during use is of lesser import: an attacker could steal the token to send its own requests by sniffing the token and then sending it on its own *ahead* of the original token, but such an attack is quite difficult unless the attacker already controls the path to the destination.

Finally, key distribution in FastPass is accomplished by routing protocols. As explained in Section 3.2.2, FastPass tightly binds the availability of a *route* with the availability and correctness of a *key*. As a result, FastPass key distribution for a destination is vulnerable only to attackers who can already interfere with routing to that destination. Ide-

ally, FastPass would be used in conjunction with a secure routing protocol such as S-BGP [18] to prevent alteration of either routes or keys.

By disseminating keys via routing, remote stub domains may not be able to verify requests until they reach a router in the "default-free" core. Fortunately, this lack of filtering does not harm the destination, but it does let a stub domain waste its own bandwidth. [5]

3. Support Many Admission Policies. FastPass succeeds very well in this regard. Unlike systems that use a fixed policy such as fair-queueing, FastPass allows each domain to allocate tokens in a manner it chooses.

4. Fine-grained Control of Requests. As with admission policies, FastPass permits senders to allocate tokens as they wish. Tokens are *not* single-use, as we would prefer, because of the rotating bloom filter that performs duplicate suppression. Instead, tokens grant permission to send a low rate of requests for a period of time (at least minutes, possibly hours or days). We believe this approximation is sufficient for most applications.

A perfect system would perform token verification on a per-host basis instead of a per-domain basis. FastPass compromises this goal to avoid the routing table explosion that would result from per-host keys. As a result, FastPass requires a mechanism within large domains to permit hosts to request capabilities that they can give to their communicants. We leave the design and implementation of such an architecture for future work.

5. Flexible Identity. FastPass does not encode identity verification into the network; it requires only that a host possess a token with the proper signature. Identity is negotiated end-to-end between senders and token granters. Tokens can be used per-flow, per-host-pair, or (at high cost) even per-packet. As a result, FastPass can be used to differentiate hosts behind NATs, different processes on a machine, and so on, as an application desires.

7.1 Open Issues

Deployment. In this paper, we have studiously avoided the issue of incremental deployment and other pesky practical problems, preferring to design FastPass unencumbered by the reality of the Internet's hundred million or so current users. Our design of FastPass-Hash and the FastPass shim header is compatible with today's protocols, but we reserve for the future an analysis of the effectiveness of a partial deployment.

Token granting services. The mechanism by which hosts obtain tokens is clearly an important one, and a complete release of FastPass must include at least one option.

Routing Changes. Current capability systems are bound to a specific path. If the path changes and the bottleneck is congested, senders must acquire a new capability. FastPass makes this acquisition easy, but senders will still experience transient packet loss before they realize they must acquire a new capability. This aspect of capability systems could interfere with some load balancing schemes, and is an important aspect of future work.

[5] An interesting issue for future work is a keying scheme in which the token validation keys could be combined as a part of address aggregation. Such a scheme would have the attractive property that authorization policies become more specific as traffic nears the victim, as suggested by Ballani et al. [5].

8 Related Work

Approaches to dealing with flooding attacks fall broadly into two categories, detection and prevention. Detection-based approaches generally attempt to provide a means to trace packets back to the host or network from which they originated [6, 26, 27], or to force attackers to use their real IP addresses during attacks [14]. While these approaches provide a means to deter attacks via legal or social consequences, unlike FastPass, they do not *prevent* a DoS attack from harming the victim.

FastPass, on the other hand, attempts to *prevent* a destination from being overwhelmed with unwanted packets. FastPass shares this goal with the capability systems [2] that it builds upon, such as TVA [35] and SIFF [34].

An alternative to capability systems are those that permit senders to shut out unwanted packets. Pushback allows a victim to push relatively coarse-grained filters out into the network that filter the largest sources of traffic to the victim [16]. AITF refines this approach to support much more fine-grained filtering [4]. Additionally, AITF relies on a strong distinction between the "network edge" and a largely trusted and well-connected "network core".

In stark contrast to all filtering schemes, FastPass adheres to a *default deny* policy for preferentially treating packets. Default deny is a long-standing principle in the design of secure systems.

Overlay-based approaches such as Mayday [1] and SOS [19] aim to protect servers by deploying networks of nodes that filter traffic on their behalf. While this approach has the advantage of being compatible with today's Internet architecture, its requirement for a large network of proxies means that it is primarily suited to protecting well-funded services that can afford such an overlay network, and less suited to protecting smaller services or individual clients.

While FastPass and related schemes address packet floods, other attacks target a server by flooding it with expensive application-layer requests. Approaches such as Kill-Bots use Captchas to rapidly distinguish human- and machine-generated requests at Web servers [17]. This work complements FastPass: A successful approach to DoS mitigation must address attacks at all layers.

9 Conclusion

The time-to-capability (TTC, i.e., the duration to set up a capability-protected connection) is a critical metric for capability-based architectures. Early capability systems,

such as SIFF, did not protect the capability request channel, but relied on statistical measures to acquire a capability [34]. Consequently, the waiting time to getting a capability was linear in the attacker's network capacity. .More recent systems, such as TVA [35] fair queue based on the ingress link to an ISP. While this approach is promising, but in the worst case, the TTC can exponentially increase with distance to the destination.

FastPass considers an extreme design point: providing guaranteed first packet delivery in the face of directed attacks, in the context of a future network infrastructure providing dedicated cryptographic support. Our analysis and evaluation show that this approach drastically outperforms prior systems, and that our design could be implemented with modest hardware requirements. While much work remains to be done, and many attacks remain to be addressed, we believe that FastPass could be a valuable component of a future network architecture.

References

[1] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2003.

[2] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial of service with capabilities. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, Nov. 2003.

[3] K. Argyraki and D. R. Cheriton. Network capabilities: The good, the bad, and the ugly. In *Proc. ACM Workshop on Hot Topics in Networks (Hotnets-IV)*, Nov. 2005.

[4] K. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *Proc. USENIX Annual Technical Conference*, Anaheim, CA, Apr. 2005.

[5] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! In *Proc. ACM Workshop on Hot Topics in Networks (Hotnets-IV)*, Nov. 2005.

[6] S. Bellovin. *ICMP Traceback Messages, Internet-Draft, draft-bellovin-itrace-00.txt, Work in Progress*, Mar. 2000.

[7] D. J. Bernstein. A secure public-key signature system with extremely fast verification. 2000.

[8] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-functions constructions from PGV. In *Advances in Cryptology (CRYPTO 2002)*, 2002.

[9] W. chang Feng, E. Kaiser, W. chi Feng, and A. Luu. The design and implementation of network puzzles. In *Proc. IEEE INFOCOM*, Mar. 2005.

[10] Cisco Systems. Cisco 12000 series one-port oc192c/stm-64c dpt line card, Feb. 2006.

[11] B. Davie and Y. Rekhter. *MPLS: Technology and Applications*. Academic Press, San Diego, CA, 2000.

[12] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM*, 2003.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. ACM SIGCOMM*, pages 254–265, Sept. 1998.

[14] P. Ferguson and D. Senie. *Network Ingress Filtering*. Internet Engineering Task Force, May 2000. Best Current Practice 38, RFC 2827.

[15] A. Hodjat and I. Verbauwhede. Minimum area cost for a 30 to 70 gbits/s aes processor. In *2004 IEEE Annual Symposium On VSLI (ISVSLI)*.

[16] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, Feb. 2002.

[17] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *Proc. NSDI*.

[18] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure border gateway protocol (S-BGP) - real world performance and deployment issues. In *Proc. NDSS*, 2000.

[19] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proc. ACM SIGCOMM*, pages 61–72, Aug. 2002.

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

[21] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *SOSP*, pages 124–139, Dec. 1999.

[22] C. McIvor, M. McLoone, J. McCanny, A. Daly, and W. Marnane. Fast montgomery modular multiplication and rsa cryptographic processor architectures. In *Proc. Asilomar Conference on Signals, Systems and Computers*, pages 379–384, Nov. 2003.

[23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[24] P. L. Montgomery. Modular multiplication without trial division. *Math. Computation*, (44):519–521, 1985.

[25] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. In *Proc. ACM SIGCOMM*, Aug. 2001.

[26] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking*, 9(3), June 2001.

[27] A. C. Snoeren, C. Partridge, et al. Single-packet IP traceback. *IEEE/ACM Transactions on Networking*, 10(6), Dec. 2002.

[28] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology – EuroCrypt*, 2003.

[29] M. Vutukuru, N. Feamster, M. Walfish, H. Balakrishnan, and S. Shenker. Revisiting internet address: Back to the future! Technical Report pending assignment, MIT Computer Science and Artificial Intelligence Laboratory, Feb. 2005.

[30] M. Walfish, H. Balakrishnan, D. Karger, and S. Shenker. DoS: Fighting fire with fire. In *Proc. ACM Workshop on Hot Topics in Networks (Hotnets-IV)*, Nov. 2005.

[31] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. USENIX OSDI*, pages 255–270, Dec. 2002.

[32] Xilinx. Virtex-4 overview. `http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/overview/index.htm`, Feb. 2006.

[33] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *Proc. IEEE Symposium on Security and Privacy*, 2003.

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

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