Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks

Bryan Parno
Carnegie Mellon University
parno@cmu.edu

Dan Wendlandt
Carnegie Mellon University
dwendlan@cs.cmu.edu

Elaine Shi
Carnegie Mellon University
rshi@cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

Bruce Maggs
Carnegie Mellon University
Akamai Technologies
bmm@cs.cmu.edu

Yih-Chun Hu
University of Illinois at Urbana-Champaign
yihchun@crhc.uiuc.edu

ABSTRACT

Systems using capabilities to provide preferential service to selected flows have been proposed as a defense against large-scale network denial-of-service attacks. While these systems offer strong protection for established network flows, the Denial-of-Capability (DoC) attack, which prevents new capability-setup packets from reaching the destination, limits the value of these systems.

Portcullis mitigates DoC attacks by allocating scarce link bandwidth for connection establishment packets based on *per-computation* fairness. We prove that a legitimate sender can establish a capability with high probability regardless of an attacker's resources or strategy and that no system can improve on our guarantee. We simulate full and partial deployments of Portcullis on an Internet-scale topology to confirm our theoretical results and demonstrate the substantial benefits of using per-computation fairness.

Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: Security and protection
General Terms: Security, Design
Keywords: Network Capability, Per-Computation Fairness

1. INTRODUCTION

In a Distributed Denial-of-Service (DDoS) attack, an adversary, sometimes controlling tens of thousands of hosts, sends traffic to a victim to exhaust a limited resource, e.g., network capacity or computation. The victim of a network DDoS attack can often identify legitimate traffic flows but lacks the ability to give these flows prioritized access to the bottleneck link; in contrast, routers have the power to prioritize traffic, but cannot effectively identify legitimate packets without input from the receiver.

Network capabilities enable a receiver to inform routers of its desire to prioritize particular flows, offering a promising DDoS defense [3, 22, 25, 31, 32]. To set up a network capability, the source sends a capability request packet to the destination, and routers on the path add cryptographic markings to the packet header. When the request packet arrives at the receiver, the accumulated markings represent the capability. The receiver permits a flow by returning the capability to the sender, who includes the capability in subsequent packets to receive prioritized service from the routers.

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

The Denial-of-Capability Attack and Defenses. Current proposals for capability-based systems treat prioritized traffic (i.e., packets with a valid capability) preferentially over non-prioritized traffic. However, capability-based systems still suffer from a critical weakness: they cannot protect the initial capability request, because that request is sent unprotected as non-prioritized traffic. An attacker can flood the capability-setup channel, thus preventing a legitimate sender from establishing a new capability-protected channel. This attack, referred to as Denial-of-Capability (DoC) by Argyraki and Cheriton [4], is the Achilles heel of current capability proposals. Agryraki and Cheriton show that several thousand attackers can easily saturate the request channel of a typical network service, preventing legitimate senders from acquiring capabilities.

When describing the DoC vulnerability, Argyraki and Cheriton argue that the same mechanism that protects the request channel could be used to protect *all* traffic [4]. We strongly disagree: since only a single capability request packet is needed to set up capability-protected communication, a simple and highly efficient network-based defense suffices. As long as the mechanism provides a predictable and non-negligible probability that the sender's request packet reaches the receiver, it can prevent DoC attacks. For example, if the capability request channel suffers a 50% packet-loss rate, a legitimate sender only needs to send about two packets to set up a capability-protected communication channel. Alas, a 50% loss rate would be far too high for efficient communication using TCP, and thus such a mechanism would not be appropriate for protecting subsequent packets.

Previously proposed capability-based systems offer few, if any, defenses against a DoC attack. Early systems simply treat capability request packets as best-effort packets [3, 22, 31]. The most recent capability architecture, TVA [32], attempts to achieve DoC robustness by tagging each packet with an identifier indicating the packet's ingress point to the autonomous system (AS) and then fair-queuing packets at each router based on this identifier. However, our evaluation in Section 6 indicates that this heuristic is insufficient to thwart DDoS attacks on Internet-scale topologies.

In this work, we present Portcullis,[1] a system that uses computational proofs of work (puzzles) to enforce fair sharing of the request channel. As a result, Portcullis strictly bounds the delay any adversary can impose on a legitimate sender's capability establishment.

Why Puzzles? While we explore the design space of DoC solutions in Section 2.3, we now provide a high-level explanation of why puzzles are particularly well-suited for solving the DoC problem. We argue that approaches like TVA that attempt to use a packet identifier to group and prioritize aggregates of traffic are inadequate for networks as large and diverse as the Internet. A major reason is that, short of trusting all routers on the Internet, network

[1] A portcullis is a grille or gate that restricts entry into a castle.

identifiers are likely to be either spoofable or very course-grained. Additionally, a single network identifier (e.g., IP address) can represent vastly different numbers of actual users (e.g., hosts behind a NAT), limiting achievable fairness.

Proof-of-work schemes offer a compelling alternative. Instead of trying to use identifiers in the packet header to provide fairness, a router simply provides fairness proportional to the amount of work performed by a sender. Such work can be verified and is thus difficult for an attacker to productively spoof. Because only a single packet must reach the destination in order to successfully set-up a capability, proof-of-work schemes are sufficient to prevent DoC despite requiring high-loss rates that might make normal data communication impractical.

Walfish et al. propose a system called *speak-up* that encourages legitimate hosts to significantly increase their sending rates during application-layer denial-of-service attacks [26], effectively using bandwidth as "work". However, the use of bandwidth as a "currency" is questionable, because the bandwidth available to typical users may vary by factors of more than 1,500 (dial-up modem vs. LAN connection), potentially placing legitimate users at a significant disadvantage. Moreover, their results focused on application layer DDoS attacks and assumed that the network itself was uncongested. In the context of DoC and network-level congestion, a speak-up style approach would inevitably create significant negative externalities for the network, because the increased traffic from legitimate users can create new bottlenecks for clients accessing destinations not under attack.

In contrast, puzzles provide a compelling solution because the "work" performed by the end host, hence avoiding additional network congestion. Also, computational disparities between users are orders of magnitude smaller than disparities in network bandwidth (Section 7.1 demonstrates a 38x difference in puzzle computation power between a well-provisioned workstation and a cell phone). Note that previous work [21] claiming that puzzles do not work contains a crucial arithmetic miscalculation, and only considers a simple, fixed-cost puzzle scheme that differs significantly from the novel, variable proof-of-work scheme used by Portcullis (see Section 8 for more details).

Contributions. In this paper we propose Portcullis, a system that enforces *per-computation* fairness for a capability system's request channel. Portcullis makes the following contributions:

- We theoretically prove strict bounds on the delay that an arbitrary number of cooperating attackers computing and sharing puzzles can inflict on a legitimate client's capability setup using Portcullis (Section 4). This guarantee holds even if the legitimate sender possesses no information about current network conditions or the adversary's resources.
- We theoretically prove that no system can improve on the bounds provided by Portcullis.
- With Internet-scale simulations, we confirm experimentally that even when tens of thousands of attackers cooperate to compute and share puzzles, a legitimate client can quickly overcome the numerical disparity and establish a capability (Section 6).
- Portcullis's novel proof-of-work mechanism avoids the pitfalls of previous puzzle schemes: it does not require routers or servers to individually provide puzzles to the sender [5,27, 28], does not rely on the sender's IP address [5,27,28] (avoiding problems with NATs and IP spoofing), does not require senders to solve a different puzzle for each router along the path to the destination [28], and does not allow puzzle reuse at multiple servers nor require extensive CPU and memory at clients, routers or servers [29].

2. PROBLEM DEFINITION AND GENERAL COUNTERMEASURES

2.1 Background and Terminology

Capability-based systems divide packets into *priority packets*, *request packets*, and *best-effort packets*.[2] Priority packets are packets that carry a valid capability. Senders use request packets to establish a capability. As the request packet traverses the routers between the sender and the receiver, it accumulates the router markings that will form the capability. Best-effort packets are sent by legacy hosts that are not capability-aware. Some capability-based systems also treat packets with invalid capabilities as best-effort traffic, while others drop them. Proposed capability systems [3, 22, 31, 32] typically dedicate a large fraction of router bandwidth to priority packets, a small fraction (5–10%) of total bandwidth to request packets (often referred to as the *request channel*), and the rest (5–10%) to best-effort packets.

2.2 Problem Definition

Capability systems attempt to thwart DDoS attacks by prioritizing legitimate traffic. However, an attacker can also launch a DDoS attack on the request channel of the capability system. If the request packets of legitimate users do not reach the capability granter, then the capability system provides little protection against the effects of the traditional DDoS attack. Thus, providing a secure request channel is essential to the effectiveness of a capability system.

An effective request channel should guarantee that a sender successfully transmits a request packet with only a small number of retries, even in the presence of a large DDoS attack on the request channel itself. We consider the case in which the adversary controls n_m hosts each sending traffic at a rate r_m. We also assume the presence of n_g legitimate senders that each send request packets at a rate r_g (typically very low), but we make no assumptions about the relative size of n_g versus n_m.

We only examine the case in which the request channel is congested, i.e., $n_m \cdot r_m + n_g \cdot r_g > \gamma$, where $\gamma = B \cdot \alpha$, B is the capacity of the bottleneck link, and α is the percentage of bandwidth reserved for the request channel. Since request packets contain no input from the capability-granting destination to allow distinctions between desired and undesired requests, the best the network can do is provide an equal level of service to all requesters. In other words, each requester should receive a $\frac{1}{n_m+n_g}$ share of the available request channel γ, regardless of whether that node is an attacker with a high request rate or a legitimate node with a low request rate. However, even with any reasonable fairness guarantee, the time required to establish a setup packet is still necessarily dependent on the total number of users $(n_m + n_g)$ and the amount of network capacity available.

2.3 Space of Countermeasures Against DoC

In this section, we divide the design space of potential countermeasures against DoC attacks into two classes based on identity and proof-of-work.

2.3.1 Identity-Based Fairness

Identity-based fairness schemes attempt to provide fairness based on some packet identifier (e.g., an IP address). These schemes are often susceptible to malicious spoofing of the identifier space that can greatly magnify attacker power. Identity-based fairness

[2]Both Machiraju et al. [22] and Yaar et al. [31] treat request packets as best-effort packets.

schemes can also experience problems when significant disparities exist with respect to the number of users sharing a single identifier.

Per-Source Fairness. A DDoS-defense system could attempt to share bandwidth equally over all sources of traffic. In other words, in a system with $n_g + n_m$ senders, a legitimate host would achieve an outbound sending rate of $r'_g = \min(r_g, \frac{\gamma}{n_g+m})$. Note that r'_g is independent of the aggregate attacking rate $n_m \cdot r_m$.

Unfortunately, at the network level, an adversary can easily spoof its IP address, and sources behind large NATs may be subject to grossly unfair treatment. Egress filtering can lessen the severity of this attack [13], but without ubiquitous deployment, we must assume that many adversaries can spoof IP addresses with impunity.

Per-Path Fairness. For per-path fairness, if $P = \{p_1, p_2, \ldots, p_k\}$ represents the set of paths leading to the bottleneck router and N_{p_i} represents the number of senders using path p_i, then a legitimate sender using path p_i should achieve an outbound sending rate of $r'_g = \min\left(r_g, \frac{\gamma}{|P|}\frac{1}{N_{p_i}}\right)$. To encode a path, Yaar, Perrig and Song propose Pi [30], a system in which routers insert path-dependent cryptographic markings into the packet header. However, router queuing based on such path markings breaks when malicious senders insert bogus initial markings in the path ID field, making it appear that such packets have traversed many distinct paths before arriving at a particular router. This increases $|P|$, hurting legitimate senders, and creates small values of N_{p_i} for the spoofed paths, helping the attacker. TVA bases its notion of fairness on path-dependent markings [32]. However, to avoid spoofing problems, their markings depend only on the interface from which a packet entered the current AS, and hence operates at a very coarse granularity.

Per-Destination Fairness. Alternately, a router could apportion request channel bandwidth based on a packet's destination address. While destination addresses cannot be spoofed, an attacker can "game" this approach by flooding packets to all destinations that share the victim's bottleneck link. Because legitimate users send packets only to a single host, per-destination queuing can actually amplify the power of an attacker.

2.3.2 Proof-of-Work Schemes

Proof-of-work schemes require senders to demonstrate the use of a limited resource to the network infrastructure, with fairness allotted proportionally to the "cost" of that resource. This solves the spoofing/gaming issue (as long as work indicates a real cost), but the resources needed for this work may have negative externalities.

Per-Bandwidth Fairness. With per-bandwidth fairness, a sender with bandwidth capacity κ should achieve an outbound sending rate of $r'_g = \min\left(r_g, \gamma\frac{\kappa}{\mathcal{K}}\right)$, where $\mathcal{K}$ represents the aggregate bandwidth of all senders. To attain per-bandwidth fairness, Walfish et al. propose a system called speak-up [26]. When a host experiences an increase in incoming traffic, it uses the speak-up system to encourage legitimate senders to significantly increase their sending rates. While their results demonstrate that each endhost will then receive service proportional to its bandwidth, the analysis is focused entirely on protecting end-host resources, not network links, and assumes the network is uncongested. Other fundamental problems with using bandwidth as a currency exist. First, requiring hosts to compete on the basis of bandwidth necessarily imposes substantial negative side-effects on the network as a whole, since hosts sending to destinations other than the victim may experience congestion because of the increase in traffic from legitimate senders. Second, large disparities can exist in the amount of bandwidth available to legitimate users. A user with a 100Mbps connection has over 1500 times more bandwidth than a user connecting at modem rates. This leads to significant inequalities between legitimate users.

Per-Computation Fairness. As an alternative to per-bandwidth fairness, we base our notion of fairness on computational effort. With per-computation fairness, the probability of request packet delivery is directly proportional to the amount of computational effort expended by a sender. Thus, a legitimate sender should achieve an outbound rate of $r'_g = \min\left(r_g, \gamma\frac{c_g}{C}\right)$, where c_g represents the sender's computational effort, and C represents the computational effort expended by other senders using the same link. If every sender has equal computational power, then per-computation fairness is equivalent to per-source fairness, but without the problems of shared or spoofed identifiers mentioned above. In the real world, computational disparities do exist, but they are not nearly as pronounced as the disparities in available bandwidth. As detailed in Section 7.1, a well-provisioned PC and a smartphone have only a 38x disparity for computational puzzles. Additionally, researchers have proposed the use of memory bound functions that can decrease computational disparities below 10x [1, 11, 12]. Finally, by shifting the playing field from bandwidth to computation, fewer externalities exist because the impact of the work is limited to a single machine, making per-computation a significantly more network-friendly approach.

3. PORTCULLIS ARCHITECTURE

Portcullis aims to provide a strong defense against large-scale DDoS attacks: even when under attack, a legitimate sender can successfully initiate a connection with the receiver and communicate with low packet loss. Portcullis augments a standard capability mechanism [3, 31, 32] with our new puzzle-based protection for capability request packets. Hence, the goal of the remainder of this paper is to design a DoC-resistant request channel for a capability system. This design is based on computational puzzles, which we prove can provide optimal fairness for the request channel (see Section 4). As a result, Portcullis strictly bounds the delay a collection of attackers can impose on legitimate clients.

To achieve per-computation fairness we leverage a novel puzzle-based mechanism, which enables all routers to easily verify puzzle solutions and uses the existing DNS infrastructure to disseminate trustworthy and verifiably fresh puzzle challenges (Section 3.4). By enforcing per-computation fairness in the request channel, Portcullis severely limits the attacker's flooding rate.

In order to provide per-computation fairness, the Portcullis puzzle system needs the following properties:

- **Authenticity:** Any host or router can verify the authenticity of a puzzle challenge and the correctness of the solution.
- **Availability:** The puzzle distribution service must be distributed and highly robust.
- **Freshness:** A solution to a puzzle must indicate *recent* computational effort.
- **Efficiency:** Routers must be able to quickly verify the correctness, authenticity, and freshness of a puzzle solution.
- **Granularity:** The puzzles should allow clients to demonstrate various levels of computational effort.

3.1 Assumptions and Threat Model

We assume space in the request packet header to encode capabilities, puzzles, and puzzle solutions. Because request packets represent a tiny fraction of data traffic, puzzle data represents a negligible amount of overhead.

In our threat model, we assume endhosts may be compromised and collude with each other. We also assume that malicious routers may assist the DoC attack, though we note that a malicious router on the path between a legitimate sender and receiver can always

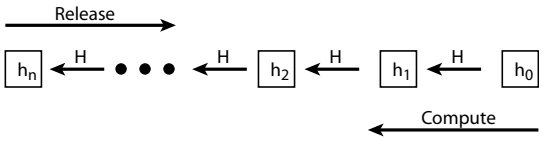


Figure 1: Hash Chain. ***The seed generator repeatedly hashes a random value h_0 to create a series of seed values. The hash-chain anchor, h_n, is signed and released. As time advances, additional seeds are released in reverse order. An authentic h_n can be used to authenticate later seeds h_i by repeatedly hashing the h_i value. If this process produces h_n, then the h_i value is authentic.***

simply drop packets. We do not assume trust relationships between routers and receivers, nor among the routers themselves. Thus, each router makes decisions independent of other routers.

3.2 Design Overview

The *seed generator* is a trusted entity that periodically releases *seeds* that senders can use to create puzzles. Senders obtain seeds from a *seed distribution service*, which need not be trusted. The *puzzle generation algorithm* is a public function for generating a puzzle based on the most recent puzzle seed and flow-specific information. Each puzzle solution is associated with a *puzzle level*. The puzzle level represents the expected amount of computation required to find a solution to the puzzle.

When a sender wishes to set up a prioritized flow, it obtains the latest seed from the seed distribution service and generates a puzzle using the puzzle generation algorithm. The sender then computes the solution to the puzzle. It includes the puzzle and solution in the header of the request packet. The routers verify the authenticity of the puzzle and the solution, and give priority to requests containing higher-level puzzles.

3.3 Seed Generation

The seed generator periodically releases a new seed for senders to use in creating puzzles. The seeds are released through the seed distribution service described in Section 3.4. The seeds must be unpredictable (i.e., it is computationally infeasible to guess future seeds based on previous seeds), and efficiently verifiable (i.e., one can easily confirm that a seed is from the seed generator).

Unpredictable and efficiently verifiable seeds can be implemented as follows. The seed generator randomly picks a number h_0, and uses a public hash function H to compute a hash chain of length n starting at h_0, i.e., $h_{k+1} = H(h_k||k)$ (see Figure 1). To prevent attacks against the hash chain, H should be a cryptographic hash function providing pre-image resistance and second pre-image collision resistance. The seed generator digitally signs the hash-chain anchor (the last value on the hash chain) h_n and releases the signature, $\text{SIGN}(h_n)$. Since hash chains can be of arbitrary length and yet stored efficiently, hash-chain anchors are released infrequently, e.g., once a year.

Every t minutes, the seed generator makes a new seed available in the form of a value from the hash chain in reverse order (that is, value h_{i+1} is released before h_i). Senders obtain the current seed from the seed distribution service and include it in their capability requests. The authenticity of the newly released seed can be verified by hashing it and comparing the result with the seed released in the previous time slot. For example, during the first time slot a sender would include seed h_{n-1} in a packet. Any router can verify the authenticity of h_{n-1} by checking that $H(h_{n-1}||n-1)$ equals the hash-chain anchor h_n.

3.4 Seed Distribution Service

The seed generator provides puzzle seeds to the seed distribution service, which makes them available to clients. A client contacts the seed distribution service to obtain the latest seed h_i. This seed is used to create puzzles (using the algorithm described in Section 3.5) for connections made during the next t minutes.

The seed distribution service also allows routers and senders to obtain the hash-chain anchor h_n needed to verify subsequent seeds. This yearly operation is the only time that routers need to contact the seed distribution service. To simplify routers, an ISP could have one or more non-router hosts contact the seed distribution service once a year and participate in an intradomain routing protocol. These hosts verify the authenticity of the signature on h_n, and then use the routing protocol to disseminate h_n to all of the ISP's routers. Because the anchor is small (approximately 80 bits), it could easily fit within a special field of a routing update.

In general, puzzle seed distribution could be handled by any distributed and well-provisioned set of servers. While using a privately operated content distribution network (CDN) is one viable approach, the simple nature of the puzzle seed and hash root data makes the existing DNS infrastructure an attractive choice.

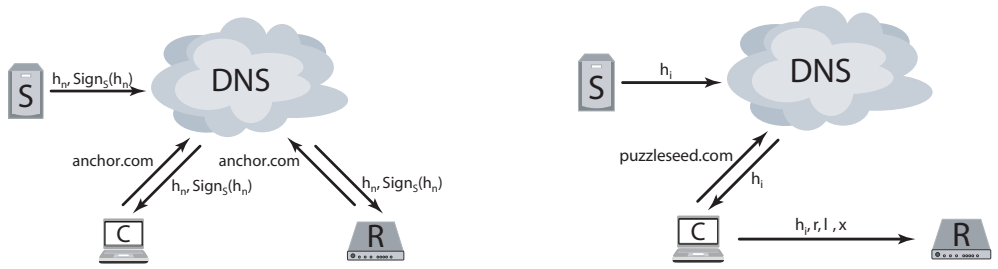
Seed Distribution Via DNS. In our DNS-based puzzle distribution design, one or more sets of global top-level domain (gTLD) servers store a DNS record for both the most recent puzzle seed as well as the signed root hash value. gTLD servers (e.g., the resolvers for the .com domain) are already highly provisioned and widely replicated because a successful DoS attack against these servers would make many network services unavailable.

Taking the example of the .com gTLD servers, in addition to storing all NS records for domains within .com, each server could have records for the special domains puzzleseed.com and anchor.com. These records would be of type TXT and would contain text-encoded values of the latest puzzle seed and hash-chain anchor (with signature). Both values are small enough to fit into a single UDP datagram. Use of the text record means local DNS servers require no modifications to query for or cache this data.

Figure 2 illustrates a sample implementation. Once a year, the seed generator run by a trusted party (e.g., ICANN) computes a hash chain and publishes the hash-chain anchor, as well as a signature on the hash-chain anchor, as a DNS record (see Figure 2(a)). Hosts and routers can perform a standard DNS query to retrieve this record, verify the signature, and store the hash-chain anchor value for the following year.

Once every t minutes, the seed generator inserts a new puzzle seed into DNS. To obtain the latest seed h_i, a client performs a standard DNS query, shown in Figure 2(b). Based on the seed, the client computes a puzzle (as discussed in Section 3.5), solves the puzzle, and includes the puzzle seed and solution in its capability setup packet. Note that a single seed h_i can be used to create puzzles for connections to multiple servers (e.g., downloading web content from multiple hosts would only require a single DNS query for the latest puzzle seed), though for each server under DDoS attack, the client must generate and solve a different puzzle. Routers receiving the setup packet can verify the authenticity of h_i using h_n (or the most recently authenticated seed value, e.g., h_j for $i < j \leq n$), and verify the puzzle solution using Equation 1.

If a body like ICANN is in charge of seed generation, it could easily include the task of puzzle distribution as part of the contracts it already establishes to run gTLD servers for domains like .com. Since providers of large and distributed DNS infrastructures such as Akamai often contain records for popular sites with TTLs of only a few minutes, updating this infrastructure to release a fresh puzzle seed on the order of 2–10 minutes would be quite feasible.



(a) **Yearly Setup** (b) **Connection Establishment**

Figure 2: Puzzle Distribution Via DNS. ***(a) Once a year, a trusted seed generator (S) publishes the anchor value h_n of a hash chain, along with a signature on h_n, as a DNS record. By performing a DNS lookup on a well-known name, clients and routers can obtain this record. (b) To establish a capability, the client performs another DNS request. The resulting DNS record contains the current puzzle seed, h_i. The client creates a puzzle based on h_i and includes the puzzle solution (x) it its setup packet. The router first verifies the puzzle seed h_i by repeatedly hashing it to get h_n. In most cases, the router will have already seen h_i and hashing will be unnecessary. Finally, the router uses Equation 1 to verify the puzzle solution.***

DNS TTLs allow ISPs to correctly cache the seeds if possible and answer client requests with no additional complexity. While recent work suggests that approximately 14% of local DNS servers violate the DNS standard by ignoring TTL values [23], adding an expiration time to the puzzle seed record allows clients to detect stale data and query the gTLD server directly for a fresh record.

Portcullis should not significantly increase the load on the DNS infrastructure for two reasons. First, we expect legitimate senders to request puzzle seeds only when contacting a destination under DoS attack. Second, studies of the behavior of local DNS resolvers from a moderately-sized academic institution [18] show that such local servers contact root and gTLD servers over $600,000$ times per week. In contrast, even if puzzle seeds changes every 5 minutes and sources in a domain experience constant DDoS attacks for the entire duration of the week, the puzzle queries would increase the number of gTLD DNS queries from that domain by only $2,000$ (e.g., less than 0.34%).

Additionally, a DoS attack on DNS does not affect Portcullis unless a simultaneous DoS attack is launched against a particular destination. An adversary able to deny DNS access to clients can already paralyze communication, and systems such as ConfiDNS [24] can allow users to quickly and securely circumvent attacks on local DNS resolvers. Note that a DNS-based implementation does not require secure DNS, nor does it require DNS servers to perform any cryptographic operations or in any other way deviate from normal operation. The hash-chain anchor is authenticated by the signature accompanying it, and subsequent puzzle seeds are authenticated based on the hash-chain anchor.

3.5 Puzzle Generation Algorithm

When an endhost decides to establish a connection, it acquires the latest random seed h_i from the seed distribution service. The sender then chooses a random 64-bit nonce r and computes a flow-specific puzzle as follows:

$$p = H(x||r||h_i||\text{dest IP}||\ell) \quad (1)$$

To solve the puzzle at difficulty level ℓ, the sender finds a 64-bit value x such that the last ℓ bits of p are all zero. The sender includes r, h_i, ℓ, and a puzzle solution x in each request packet. It need not include p, since the router will regenerate it during puzzle verification. Assuming the publicly-known hash function H is preimage resistant and has a good distribution over its range, a sender must resort to a brute-force approach by trying random values of x to find a solution for the chosen level.

We intentionally do not make hash puzzles depend on the source IP address. Including the source address causes problems for hosts behind NATs or proxies, yet does little to prevent attackers from sharing puzzles because an attacker can simply spoof its IP address. To limit puzzle sharing, routers drop duplicate puzzles (we discuss how Portcullis is effective despite attackers sharing puzzles in Section 5.1). Since routers drop duplicate puzzles, senders are motivated to choose r at random. However, the input to the hash does include the destination IP address, which prevents an attacker from reusing the same puzzle to attack multiple destinations (unlike source information, the destination address cannot be spoofed). We also include the difficulty level of the puzzle in the hash computation to prevent an adversary from reusing computation for a hard puzzle as a solution for an easy puzzle. In other words, if the adversary attempts to solve a level 7 puzzle, she may discover viable solutions for puzzles at level 1-6. If we did not include ℓ, an attacker could expend the computational effort to find a solution to a level 7 puzzle, and receive solutions to lower-level puzzles "for free". Committing to ℓ by including it in the hash prevents this.

3.6 Puzzle Verification by the Router

Because puzzle seeds are included in each packet and can be verified with the hash-chain, routers only need to update their hash verification state when a new hash-chain begins (e.g., yearly). When a router receives a packet that includes a puzzle solution, it first verifies the authenticity of the seed h_i for the puzzle. The authenticity of h_i can be verified by computing $H(h_i||i)$ and comparing it with the seed released in the last time slot (h_{i+1}). If traffic arrives sporadically at a router, the router may need to hash h_i several times and compare each value with a previous seed seen by the router to verify its authenticity. However, since each seed is valid for t minutes, a new seed need only be verified a single time, and verification can consist of a simple equality check for the remainder of the period.

To verify the puzzle solution, the router computes the same hash shown in Equation 1, using the nonce r, the seed h_i, the sender-supplied solution x, and the destination IP in the request packet. The router accepts the puzzle solution if the last ℓ bits of p are zero. With Portcullis, the router only needs to compute a single hash to verify the solution to a puzzle.

3.7 Router Scheduling Algorithm

The router's request channel scheduling algorithm should: 1) limit reuse of puzzle solutions, and 2) give preference to senders who have solved higher-level puzzles.

When requests arrive at a router, the associated puzzle is first verified for correctness. To prevent exhaustion attacks, only correct puzzle solutions and their input parameters are entered into a Bloom filter [6] configured to detect the reuse of puzzle solutions seen in the past period t. Each puzzle is uniquely identified by the tuple $(r, h_i, \ell, \text{dest IP})$. Since r is chosen randomly from 2^{64} possible values, the space of potential puzzles for a given destination address is quite large, and the probability of multiple legitimate clients using the same puzzle for capability setup within the window of eligibility t for h_i is negligible.

Bloom filters support a tradeoff between router state and the probability of incorrectly dropping a unique solution. They provide compact lookups, have no false negatives, and allow false positive probabilities to be driven to arbitrarily low rates with the use of additional memory. To illustrate this, consider a router with a 2 Gbps link on which 5% of the capacity is allocated to capability requests and a circular buffer of F Bloom filters, where each filter contains all puzzles seen over a one second period. The router may receive 80,000 requests/sec, with each of the F filters being checked to see if a puzzle is duplicated. If k different hash functions are used, inserting n puzzles into a single table of size m bits gives a false positive probability of approximately $(1-e^{\frac{kn}{m}})^k$. With an optimal k value, this can be estimated as $(0.6185)^{\frac{m}{n}}$. Thus, a filter of 300 KB can prevent duplicates for one second with a false positive probability of under $\frac{1}{10^6}$ per packet. A circular buffer configuration of F Bloom filters can therefore filter traffic for F seconds with less than a $(1-(1-\frac{1}{10^6})^F) \approx \frac{F}{10^6}$ false positive probability per packet.

If a request clears the Bloom filter check, the router places it in a priority queue based on its puzzle level.

3.8 Legitimate Sender Strategy

We now briefly outline the puzzle-solving strategy used by a legitimate sender. Later in Section 4 we precisely define this strategy and prove that it will, with high probability, allow the sender to establish a capability, regardless of the attacker's power or strategy. We assume that the legitimate sender has neither knowledge of the amount of congestion on the path to the DDoS victim, nor knowledge of the attacker's power or strategy (though a sender with some or all of this information can further optimize her strategy). Essentially, a legitimate sender will compute a solution to the lowest-level puzzle and transmit a request. If the request fails, the sender solves a puzzle that requires twice the computation of the previous puzzle and sends a new request packet. The sender continues to double her computational effort until she succeeds in establishing a capability.

3.9 Overhead Analysis

Packet Overhead. We use 64 bits to represent the puzzle solution x. Hence the highest level puzzle would require approximately 2^{63} hash computations. Contemporary PCs can compute approximately 2^{20} cryptographic hashes per second. Hence we expect that these puzzles will remain computationally difficult for years to come. We use 6 bits to denote the puzzle level ℓ. The 64-bit nonce r is sufficient to distinguish between different sources connecting to a particular destination within a time interval t. We use $h_i = 80$ bits to represent the puzzle seed. Hence we need approximately 27 bytes to encode the puzzle and puzzle solution in the packet header. This small overhead can be piggybacked on any other data that might be included in the request packet, such as for TCP connection establishment. Because this overhead need only be incurred on request packets and request packets constitute a small fraction of the total amount of traffic, an extra 27 bytes should be acceptable.

Puzzle Verification Overhead. In analyzing the impact of puzzle verification on routers, it is important to note that only a fraction of a router's capacity is devoted to capability setup traffic, suggesting that puzzle verification need not necessarily operate at full line speed. Nonetheless, minor hardware improvements would easily allow routers to verify puzzles at line speed. The additional hardware could be incorporated into new generations of routers or developed as modules to extend older routers. Within an AS, only the border routers need to verify puzzles, setting or clearing a bit in the header that internal routers can use to determine if a puzzle solution is valid. As we show in Section 6, even if the victim's ISP is the only entity to upgrade its routers, the victim still receives substantial benefits.

Commercially available ASIC [15] and FPGA [16] cores for SHA-1 are capable of performing these hash functions at well over 1Gbps in a small amount of space. For example, the ASIC implementation of SHA-1 only requires 23,000 gates, whereas a typical ASIC has millions. Similarly, the FPGA implementation takes 577 slices, where a typical FPGA has tens of thousands of slices. Because multiple puzzles can be verified in parallel, the use of several SHA-1 cores on a single ASIC or low-cost FPGA could handle line-speed puzzle verification, even for several OC-192 links. In fact, the greatest limitation to using a single chip for puzzle verification is the available bandwidth for bringing data on and off the chip. In addition, the latency introduced by each verification will be low, since verifying each puzzle involves computing a function over less than 50 bytes. Hence when the hash function operates at 1Gbps, verifying a puzzle will introduce well under 1 μs of latency.

Routers could also perform puzzle verifications in software. Since a modern PC can perform 2^{20} SHA-1 computations per second (see Section 7.1), a software implementation could support approximately one million request packets per second.

Router Scheduling Overhead. The router scheduling algorithm used by Portcullis requires several hash computations for the Bloom filter. These can be computed in parallel, even for a single packet, and as discussed above, hash computations can be implemented very efficiently in hardware. Again, we only apply this algorithm to request packets, which constitute a small fraction of a router's total bandwidth. If every router dedicates 5% of its bandwidth for the request channel, a software implementation is sufficient to support a gigabit link, and a hardware implementation can easily handle faster line-rates.

4. THEORETICAL FAIRNESS ANALYSIS

In this section, we prove two main results. First, allocating service based on per-computation fairness provably guarantees that a legitimate sender can establish a capability with high probability, regardless of an attacker's resources or strategy. This guarantee holds even if the legitimate sender has no information about current network conditions or the adversary's resources. Second, assuming that routers cannot independently distinguish legitimate clients from malicious ones, we prove a lower bound indicating that no system can improve on this guarantee.

4.1 Assumptions and Problem Definition

Assumptions. We assume that routers cannot independently distinguish packets from legitimate and malicious senders. We allow all attackers to collude, jointly compute puzzles, and synchronize their floods, but we assume they have bounded resources, though the bound on their resources need not be known to the legitimate

senders. To simplify the analysis, we assume that all endhosts have the same hardware configuration and hence, equal computational resources. Finally, we consider network latency negligible relative to the sender's puzzle computation time.

Problem Definition. We consider a scenario with a single bottleneck router. Request packets from different sources arrive at the bottleneck router, but the remainder of the network has infinite capacity. Thus, a packet can only be queued at the bottleneck router.

For the purposes of the DoC attack, the adversary controls n_m compromised endhosts. We discretize time into small time slots and assume that a single legitimate sender starts a connection setup process at an arbitrary point in time between $(0,\infty)$.

We consider $\mathscr{R}$, a class of router scheduling policies with *finite output bandwidth*, i.e., a router outputs a maximum of γ requests in each time slot. Under router policy $R \in \mathscr{R}$, we define $\mathscr{G}^R$ to be a class of strategies for a legitimate uninformed sender. The sender is *uninformed* in the sense that it is not required to know the real-time condition of the network or the adversary's computational capacity or strategy.

Under router policy $R \in \mathscr{R}$, and legitimate sender strategy $G \in \mathscr{G}^R$, we define $\mathscr{A}^{R,G}(n_m)$ to be the class of adversary strategies using n_m compromised machines. Thus, the adversary is aware of the legitimate sender's strategy, though it does not know when the legitimate sender will begin. The adversary's goal is to maximize the connection setup time for the legitimate sender.

We define $t(R,G,A(n_m))$ as the expected connection setup time for a legitimate sender, assuming a router policy $R \in \mathscr{R}$, a legitimate sender's strategy $G \in \mathscr{G}^R$, and an adversary $A(m) \in \mathscr{A}^{R,G}(n_m)$ in control of n_m compromised machines. The setup time is the time that elapses from when the sender starts sending request packets, until the moment a request packet is successfully received at the destination.

Finally, we define the Portcullis router scheduling policy and the Portcullis legitimate sender's policy.

DEFINITION 1: PORTCULLIS ROUTER SCHEDULING POLICY, R_0
Each request carries an unforgeable proof of the amount of computation performed by the sender. In each time slot, if no more than γ requests arrive, the router outputs all of them; otherwise, the router preferentially outputs requests carrying larger amounts of computation and drops any remaining requests.

DEFINITION 2: PORTCULLIS LEGITIMATE SENDER POLICY, G_0
The legitimate sender continues to send request packets until one transmits successfully; on the i^{th} attempt, it attaches a proof of $\chi \cdot 2^{i-1}$ computation, where χ represents the amount of computational effort an endhost can exert in a single unit of time.

4.2 Main Results

The first result demonstrates that using Portcullis, a sender can always successfully transmit a packet in time bounded by the amount of attacker computation:

THEOREM 4.1. *Under the Portcullis router scheduling policy R_0, a legitimate sender utilizing the Portcullis sending policy $G_0 \in \mathscr{G}^{R_0}$ to traverse a bottleneck link under attack by n_m malicious hosts successfully transmits a request packet in $O(n_m)$ amount of time in expectation, regardless of the strategy employed by the adversary.*

Our second result states that for any scheduling policy, and any sending algorithm, a legitimate sender cannot perform better than the guarantee provided by Theorem 4.1:

THEOREM 4.2. *$\forall R \in \mathscr{R}$, $\forall G \in \mathscr{G}^R$, $\exists A(n_m) \in \mathscr{A}^{R,G}(n_m)$ such that the expected time for a legitimate sender to successfully transmit a request is $\Omega(n_m)$.*

4.3 Proofs

Given the definition of R_0, we first prove the following lemma, which we use in our proof of Theorem 4.1.

LEMMA 4.3. *Assume routers adopt the Portcullis scheduling policy R_0. Let ϕ denote the total amount of computational resources controlled by the adversary, ($\phi = n_m\chi$, where χ is the amount of computational effort a single endhost can exert in a single unit of time). If the legitimate sender attaches a proof of $\phi/\gamma+\delta$ computation to a request packet (where $\delta > 0$ and γ is the number of requests output by the bottleneck router in each time slot), then regardless of the adversary's strategy, the request packet successfully transmits with probability at least $\frac{\delta}{\phi/\gamma+\delta}$.*

COROLLARY 1. *If the legitimate sender attaches a proof of $2\phi/\gamma$ computation (i.e., $\delta = \frac{\phi}{\gamma}$) to a request packet, the request packet succeeds with probability at least 1/2.*

Before proving Lemma 4.3, we offer some insight into the result. Intuitively, to prevent the successful transmission of a legitimate request in a particular time slot, an adversary needs to send at least γ requests in the same time slot, each containing a larger proof of computation than the legitimate request. If the adversary wishes to sustain a flood rate of γ in the long run, she can afford to put no more than ϕ/γ computation into each request. Alternatively, the adversary can flood at rate γ in a fraction p of the time and attach $\phi/(\gamma \cdot p)$ amount of computation to each request. Lemma 4.3 states that if the legitimate sender is aware of the adversary's total computational resource ϕ and the bottleneck bandwidth γ, it benefits the sender to attach a proof of slightly more than ϕ/γ computation to its request. As a result of this strategy, the request successfully transmits with non-negligible probability, no matter what strategy the adversary uses. Corollary 1 is a special case of Lemma 4.3. If the legitimate sender performs $2\phi/\gamma$ computation on a request, the request gets through with probability at least 1/2.

Proof of Lemma 4.3: Assume the sender puts a request packet on the wire in the i^{th} time slot, and attaches a proof of $\phi/\gamma+\delta$ computation to the request packet. To prevent this request from getting through, the adversary needs to inject at least γ requests in the i^{th} time slot, and each request packet should contain at least $\phi/\gamma+\delta$ amount of computation. Since the adversary has a total amount of computational resources ϕ, if she wishes to flood with at least γ requests, each carrying a proof of at least $\phi/\gamma+\delta$ computation, then she can do so during no more than a fraction $p_f = \frac{\phi/\gamma}{\phi/\gamma+\delta}$ of the time. Because the legitimate sender puts a packet on the wire at a random point of time, its probability of success is $1-p_f = \frac{\delta}{\phi/\gamma+\delta}$. ■

With Lemma 4.3, it is straightforward to prove our two main results. Note that the Portcullis sending policy does not require the sender to know the adversary's strategy, nor the number of machines employed by the adversary.

Proof of Theorem 4.1: After $k = O(\log n_m)$ attempts (for some k), the sender will try a request packet carrying a proof of $2\phi/\gamma = 2\chi n_m/\gamma$ computation. Applying Lemma 4.3 with $\delta = \phi/\gamma$, this request has probability at least $1/2$ of arriving successfully. To compute the expected time until a request succeeds, we note that the time spent solving the puzzle for attempt $k+i$ is $(2\phi/\gamma)2^i$. Furthermore, the probability that attempt $k+j$ fails for any j (which is relevant only if attempts k through $k+j-1$ also fail) is at most $1/2^j$. Hence, the probability that attempts k through $k+i-1$ fail and $k+i$ succeeds is at most $1/2^{i(i-1)}$. Thus the series for the expected time converges to $O(\phi/\gamma) = O(\chi n_m/\gamma)$. ■

Proof of Theorem 4.2: Divide the compromised machines evenly into $\tau = n_m/2\gamma$ groups, each of size 2γ. Starting at time 0, the i^{th} group is activated during the i^{th} time slot. Each compromised machine follows the legitimate sender's algorithm for setting up a connection. Regardless of whether a compromised machine is able to set up a connection, it stops after τ time slots and restarts the legitimate sender's algorithm. Because the bottleneck router can only output γ requests per time unit, the expected time for each compromised machine to set up a connection is at least $m/2\gamma - \tau/2 = n_m/4\gamma$. Since a router cannot independently distinguish a legitimate request from a malicious request, and a compromised machine uses the exact same algorithm as a legitimate sender, by symmetry, a legitimate sender requires $\Omega(n_m)$ time slots to establish a new capability.

■

5. POTENTIAL ATTACKS

In this section, we analyze other potential attacks and explain how Portcullis defends against them.

5.1 Attacks by Malicious Endhosts

Sharing Puzzle Solutions. It is possible for a malicious endhost to compute a puzzle solution and share it with many other colluding nodes. Perhaps counter-intuitively, sharing puzzle solutions is not very effective at increasing attack power, because the attacker has no more power to congest any single link in the network than it did before.

Even if all colluding endhosts that share puzzle solutions target a bottleneck link, they cannot break our basic fairness guarantee (Section 4), because the Portcullis router on that link discards duplicate puzzle solutions. Hence, no matter how many times an adversary sends a puzzle solution on the same link, she will only receive prioritized bandwidth proportional to the amount of computational effort she performed.

However, the attacker can now use the same puzzle to attack *different links* simultaneously. Yet this has little effect on our calculated amount of work per-client, which was based solely on the combined CPU capacity of all malicious hosts and the capacity of the bottleneck link on the path between the client and the destination. Essentially, for any particular client the puzzle-sharing scenario is no different than if all attacking hosts had computed on behalf of a single host that was capable of flooding that client's bottleneck link to the destination. Since we already incorporate this case into our analysis, the guarantees provided by Portcullis still hold. However, an adversary with well-positioned hosts capable of saturating specific links within the core of the network may be able to reuse puzzle solutions over subdivisions of the puzzle-seed validity window t to attack different links along a path leading away from the victim, such that some portion of the path is consistently congested. With precise timing, this technique may amplify the attacker's power by a constant factor, but this additional power diminishes as the attacker targets more links.

Timing Amplification. Sections 4 and 6.2 describe an optimal attacker strategy, assuming the attacker wishes to delay all senders equally. However, an attacker can also spend more time (than strictly optimal) computing, and hence send request packets with higher-level puzzles during short periods of time. Nonetheless, these bursts of packets do not affect the average time for a user to establish a capability, since the extra computation time leaves a window in which the adversary is not sending packets, allowing some legitimate senders to quickly establish capabilities with very low-level puzzles.

5.2 Attacks by Malicious Routers

Clearly no DoC-prevention scheme can prevent a malicious router from dropping capability request packets forwarded through that router. As a result, we only consider attacks where a malicious router seeks to flood or help malicious endhosts flood the request channel of a remote network link.[3] For instance, the malicious router can fail to enforce rate regulation in the request channel, or it can use its own packets to attack the request channel. With a partial deployment, the malicious router can potentially congest the request channel of a downstream legacy link. However, as soon as the request packets traverse a legitimate Portcullis-enabled router downstream, the attack traffic is subject to regulation based on per-computation fairness. Hence Portcullis achieves graceful performance degradation in the face of such a partial deployment attack.

6. EVALUATION

In this section, we describe the details of our simulations. We evaluate both simple flooding DoC attacks and Portcullis-aware DoC attacks. We also compare Portcullis with previous architectures, in both full and partial deployments.

6.1 Internet Scale Simulation

We simulate the benefits of the per-computation fairness provided by Portcullis using an Internet-scale simulation. The topology for this simulation is derived from CAIDA Skitter probe results [7], which record router-level topology. The Skitter map forms a tree rooted at the trace source (a root DNS server) and spans out to over 174,000 networks scattered largely uniformly across the Internet. We use the identical topology, but reverse the direction of packet flow such that packets from clients (both legitimate and attackers) flow up the branches of the tree to the root, which in our scenario is the victim. We make the conservative assumption that a single link connects the victim to the rest of the network. Multiple links would increase the difficulty of a DDoS attack, since an attacker would have to flood all of the links to deny service to legitimate clients. This realistic topology is essential to evaluate the performance of TVA [32], which depends on topology to help it differentiate legitimate traffic from attack traffic.

Since the Skitter map does not include bandwidth measurements, our simulations employ a simple bandwidth model in which the senders' uplinks have one tenth the capacity of the victim's network connection, while the rest of the network links have 10 times that of the victim's network connection. Thus, each host has a small link connecting it to a well provisioned core that narrows down to reach the victim. Experiments using a uniform bandwidth model produced similar results, though Portcullis performed even better; space constraints prevent us from including these results.

To make these values concrete, sender's uplinks have a total capacity of 20 Mbps, the victim's link to the rest of the network has a total capacity of 200 Mbps, and the core links are 2 Gbps. Assuming each request packet is approximately 1000 bits, and each link reserves 5% of its capacity for request traffic, an attacker can flood its uplink's request capacity by sending requests at 1 Mbps.

In our experiments, we measure the time each of 1,000 legitimate clients requires to establish a capability. We vary the number of attackers from 1,000 up to 20,000 (thus allowing the attackers to significantly outnumber the legitimate senders). For the Random, TVA, speak-up, and Portcullis-Flooder scenarios, attackers send requests at the full request capacity of their uplink. Both legitimate clients and attackers are placed randomly in destination networks.

[3]Flooding a link controlled by the router itself is essentially the same as dropping packets; hence it is out of the scope of this paper.

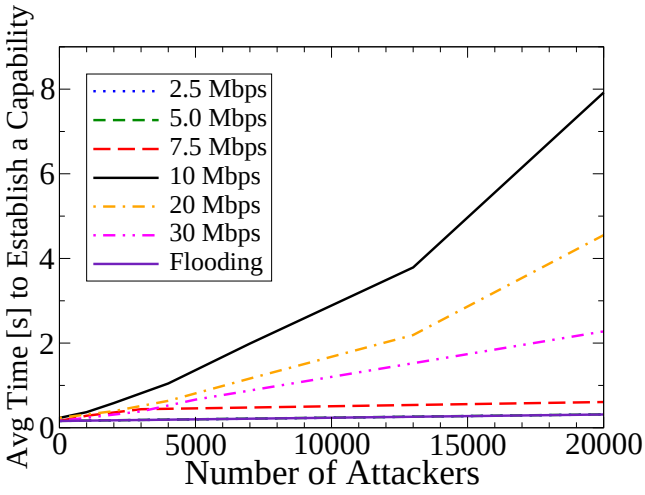


Figure 3: Portcullis Attacker Strategies. ***The ideal strategy is indicated by the top line, representing an attacker who spends all of her CPU resources to create just enough packets to saturate the victim's 10 Mbps link to the network. The Flooding attacker represents a traditional attacker who simply floods the network with legacy packets. Both the Flooding attacker and the attacks that fail to fill the victim's link (i.e., collectively sending requests at 2.5, 5.0 or 7.5 Mbps), have virtually no effect on capability establishment time, even for large numbers of attackers.***

The exact strategies used by both attackers and clients are varied in the course of the experiments, and are explained in detail below. For experiments involving puzzle computation, we assume all client machines have equal computational resources. The puzzle difficulty levels are adjusted such that solving a puzzle at level ℓ requires the sender to spend $10 \cdot 2^{\ell-1}$ milliseconds computing. When testing Portcullis, legitimate senders employ the Portcullis sending policy from Section 4.1. In other words, a legitimate sender will compute for 10 ms, and send a request at puzzle level 1. If that request fails, the sender will compute for 20 ms and send a request at level 2, etc., until she receives a capability. In all experiments, we delay the time at which legitimate senders begin sending requests until after the traffic from the attackers reaches a steady-state. Thus, legitimate senders face the full brunt of the DoC attack.

6.2 Portcullis Attacker Strategies

The optimal attacker strategy in network DDoS attacks today is simply to target bottleneck links near the victim with as many packets as possible in order to decrease the probability of a legitimate packet finding space in a router's queue. However, with Portcullis the choice of attacker strategies is more subtle, as the attacker must decide whether it is better to send many low priority packets, or fewer packets each with higher priority.

We assume that attackers can pool their CPU resources to collectively solve puzzles in order to maximize the power of their attack. As we discussed in Section 5, sharing puzzle solutions does not significantly impact legitimate senders, so our simulation assumes that all puzzle solutions are unique. As our analytical results demonstrate, the ideal attacker strategy is to send the highest priority puzzles possible while still saturating the victim's bottleneck link(s). Figure 3 illustrates this, where the ideal strategy (top line) is for the attackers to collectively send requests at 10 Mbps (the request capacity of the victim's network link) and devote their pooled CPU resources to computing the hardest puzzles possible for those requests. To send more than 10 Mbps, an attacker must devote less CPU power to each puzzle, lowering the computational threshold for legitimate senders. Sending requests with higher puzzle levels means that the attacker does not have the CPU resources to saturate the link. Thus, legitimate packets reach the victim even when they are of lower priority than attack traffic.

This graph powerfully demonstrates results presented analytically earlier in the paper: even when attackers cooperate to compute puzzles, a legitimate client can quickly increase its level of puzzle difficultly until the collective CPU power of the adversary is insufficient to keep the link saturated with equally difficult puzzles. Wait times are approximately 8 seconds, even with $20,000$ attackers using an optimal strategy.

6.3 Comparative Simulations

Our second set of simulations compare Portcullis, TVA [32], speak-up [26], and a simple random-drop "legacy" forwarding scheme on the same Internet-scale topology. For the Portcullis simulations, we show both an attacker who employs the optimal puzzle-solving strategy discussed above, as well as an attacker that simply floods packets without solving puzzles.

With TVA, each router performs queuing based on the ingress point of the packet into the current AS. Because the Skitter maps do not include AS information, we use the Team Cymru "IP to ASN" service [9], which creates mappings based on a diverse set of BGP feeds. For the less than 2% of router IPs that did not successfully map to an AS, we consider that router to be a member of the most recent known AS in the path. These mappings result in an average AS-path length of approximately 4.1, which is only slightly less than the average length of 4.5 determined by previous measurement work [2]. Since TVA does not specify a value for source retransmission rates of request packets, we use a highly aggressive retransmission rate of one packet/10ms for TVA clients. In practice, such a high rate for legitimate senders may cause congestion for traffic to alternate destinations, but in this simulation the higher transmission rate is strictly better for TVA.

For speak-up, both legitimate and malicious senders saturate their uplinks with request packets. In the randomized dropping (legacy router) scheme, each router simply chooses packets randomly from its incoming queue until its outgoing queue reaches capacity, dropping all remaining packets.

Figure 4 compares the speed with which $1,000$ legitimate clients acquire a capability when using various defense mechanisms. The graphs represent different numbers of attackers ($1,000$ and $20,000$), which are representative of our results for different numbers of attackers in between. Note that the x-axis uses a logarithmic scale.

The two lowest lines represent TVA and the randomized-drop router strategy. With both strategies, many clients fail to acquire a capability within the simulation period of 100 seconds when faced with $20,000$ attackers. A full Internet topology greatly reduces the benefits of TVA, because with each AS hop, legitimate traffic "mixes" and becomes indistinguishable from attack traffic with respect to TVA's priority mechanism. In fact, if each AS has i ingress points, and there are λ AS hops, the likelihood of a packet successfully reaching the destination scales with the inverse of $i^{\lambda-1}$ when the number of attackers is large. That is, loss rates with TVA are heavily topology-dependent because they are exponential in the number of AS-hops contained in the network path. On realistic topologies, this mixing of traffic results in performance that is similar to the randomized best-effort transmission of request packets. The original analysis of TVA [32] did not show this effect because their simple topology contained only a single hop before the bottleneck link, meaning that no mixing of good and bad traffic occurred.

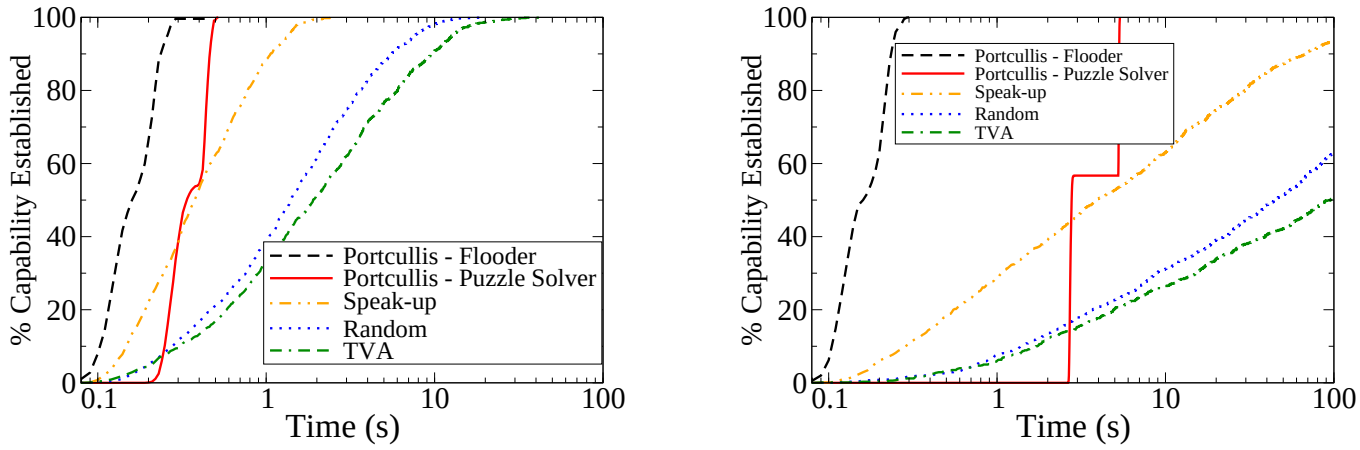


Figure 4: Capability Setup Time. ***Cumulative distribution functions of the time required for a legitimate sender to acquire a capability when faced with*** $1,000$ ***attackers (left) and*** $20,000$ ***attackers (right). The Portcullis Puzzle Solver attacker uses the optimal strategy discussed in Section 6.2. Note that the x-axis uses a logarithmic scale.***

Speak-up hosts gradually establish capabilities, but a significant portion (20%) take half a minute or more to succeed. Speak-up's performance declines as the number of attackers increases, since the attackers have more bandwidth relative to the legitimate senders.

The Portcullis-Flooder line in Figure 4 demonstrates that Portcullis provides clear benefits if the attacker naively uses the same flooding strategy used against TVA. But what happens when the attacker is smart and harnesses all of its computational power to compute puzzles using an optimal strategy?

As we see in Figure 4, Portcullis guarantees legitimate clients the ability to achieve fairness regardless of topology, even if the attacker uses the ideal puzzle computation strategy. In contrast, TVA cannot offer a legitimate client real fairness once its traffic mixes with the higher-rate attack traffic. Portcullis's performance illustrates the benefits of a scheme that is orthogonal to topology.

The threshold-style shape of the line for the puzzle-solving attacker scenario illustrates the puzzle scheme's operation. Legitimate senders start with low-level puzzles that cannot compete with the attacker's high-level puzzles. However, legitimate senders continue to increase their puzzle levels until they receive a capability. When legitimate senders reach the puzzle level employed by the attacker, some portion of their packets are randomly selected and reach the victim, creating the first jump in the percentage of capabilities established. If a legitimate sender's packet does not make it through, the sender must spend time computing a new puzzle at a higher puzzle level. The higher puzzle level of this next packet guarantees that it receives priority over the attacker's packets, and hence the rest of the legitimate senders can establish capabilities. Thus, the distance between the two "surges" represents the time spent computing the higher-level puzzle.

6.4 Partial Deployment

While the previous experiments assume a complete deployment scenario, we also run simulations to evaluate the effectiveness of Portcullis in partial deployment. We focus on the performance for an early adopter, so in our simulations, only the victim's ISP upgrades its routers. We define the victim's ISP to encompass the victim's link to the network, plus the next three hops on paths leading out from the victim. The remaining routers simply randomly choose among incoming packets.

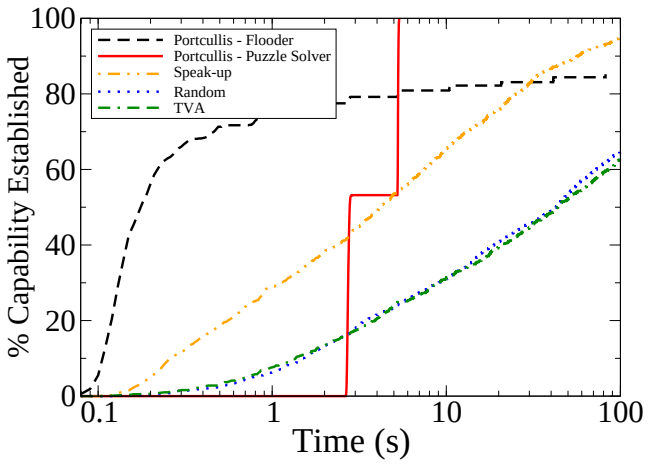


Figure 5: Partial Deployment. ***Time to establish a capability versus 20,000 attackers when only the victim's ISP upgrades its routers. Again, the x-axis employs a logarithmic scale.***

Figure 5 summarizes our results for 20,000 attackers (the results look similar for 1,000 attackers). The speak-up and Random results remain the same as in Figure 4, since neither one affects the forwarding algorithm. TVA performs slightly worse, since fewer attack packets are filtered early in the network; however, even with full deployment, TVA has difficulty distinguishing attack packets, so partial deployment has a relatively small effect. Portcullis's results versus the puzzle-solving attacker remains unaffected, since the puzzle-solving attacker does not generate enough packets to congest the core of the network (where the legacy routers reside); congestion only occurs near the victim, where the legitimate senders' increasing puzzle levels quickly break through.

Against the flooding attacker, Portcullis performs somewhat worse than before, since about 15% of legitimate senders do not receive a capability. However, the vast majority of legitimate senders that do receive a capability do so extremely quickly (note the logarithmic x-axis). Senders fail to receive a capability when their traffic is swamped by attack traffic early in the core of the network

Platform	SHA-1 hashes/minute	Normalized to Nokia 6620
Nokia 6620	25 K	1.00x
Nokia N70	36 K	1.44x
Sharp Zaurus PDA	56 K	2.24x
Xeon 3.20GHz	956 K	38.24x

Table 1: Computational Capabilities. This table summarizes the rate at which various platforms can perform the SHA-1 hashes needed to solve and verify puzzles (averaged over 10 trials, with negligible deviations).

at a legacy router, before reaching the victim's ISP. Nonetheless, this experiment demonstrates that even if a single ISP upgrades to use Portcullis, more than 85% of legitimate senders will be able to quickly establish a capability in the face of a DDoS attack by 20,000 attackers.

7. DISCUSSION

7.1 Asymmetric Computational Power

Computational puzzles give an advantage to endhosts with faster CPUs. Because the typical life-time of a PC is 3 to 5 years, and according to Moore's Law, computing power available for a fixed cost doubles every 18 months, the oldest endhosts would be expected to be at most 4 to 10 times slower than the newest endhosts. To take an extreme case, our experiments show that a desktop PC with a Hyper-Threaded Intel Xeon 3.20GHz processor and 3GB of RAM has an approximately 38x computational advantage over a Nokia 6620 cellphone. On the cellphone, we used an unoptimized C++ implementation of SHA-1 based directly on the FIPS specification [19]. We also employed the same code on a slightly newer phone, the Nokia N70, as well as on the Sharp Zaurus, a PDA that uses an Intel PXA255 CPU operating at 400MHz. On the PC, we used the OpenSSL implementation of SHA-1.

Table 1 summarizes our results. The Nokia 6620 performs approximately 25K hashes/second on average, while the PC performs approximately 956K hashes/second, indicating a disparity of only 38x (with even smaller disparities for the newer N70 and the PDA), as opposed to the 1,500x disparity for per-bandwidth fairness.

To help mask differences in CPU speed, researchers have studied memory-bound functions [1, 11, 12]. Because memory access latencies exhibit smaller variations across classes of devices (on the order of 5-10x), using memory bound puzzles is an interesting topic for our future research.

Alternately, providers of mobile Internet services may offer their clients access to a proxy that computes a rate-limited number of puzzles on behalf of each client. Such an arrangement may also address power concerns for mobile devices. However, since clients only employ Portcullis when the site they wish to contact is heavily congested, we expect puzzle solving to be sufficiently infrequent that it should not significantly impact battery life.

7.2 Puzzle Inflation

When senders (legitimate or malicious) send high-level puzzles to a destination under attack, their packets will share links with "innocent bystander" packets intended for other destinations. We show that these high-level puzzle solutions will not "inflate" the puzzle level required of the bystander packets.

We can analyze the situation by considering three possible conditions for the link in question. First, if the link's request capacity is not exhausted, then the bystander packets will be completely unaffected. Second, if the link's request capacity is entirely consumed by packets with high-level puzzles, then bystander senders must send high-level puzzles as well, since the link is effectively under a DDoS attack, even though it has not necessarily been specifically targeted. Finally, the link's request capacity may be exhausted by a mixture of high-level and bystander packets. As a result, the bystander packets essentially compete for the capacity not consumed by the high-level packets. The bystander packets can solve puzzles to improve their odds against other bystander packets, but the puzzle-level need not be the same as the high-level puzzles. While the bystander packets are competing for less than the link's full request capacity, the senders of the high-level puzzles actually use less bandwidth than they otherwise would, since the computational time required to solve high-level puzzles forces them to send at a much lower rate than they could at lower puzzle levels. Thus, Portcullis only cause limited, "local" increases in puzzle levels which will not cascade across the network.

8. RELATED WORK

Below, we review related work not already discussed, focusing particularly on the areas of capability-based systems and computation-based systems for DoS defense.

Capability-Based Systems. Early capability systems require significant state within the network, as well as trust relationships (i.e., secure keys) between infrastructure nodes and endhosts [3, 20]. Later schemes provide improved efficiency but do not defend against request channel flooding. For example, Machiraju et al. propose a secure Quality-of-Service (QoS) architecture [22]. They use lightweight certificates to enable routers to designate bandwidth reservations, and they propose a stateless recursive monitoring algorithm for routers to throttle flows that attempt to exceed their allotted bandwidth. Yaar et al. propose SIFF, a capability-based system that allows a receiver to enforce flow-based admission control but makes no effort to defend against DoC attacks [31].

Computation-Based Systems. Several researchers have proposed computational puzzles for DDoS defense; however, none of these schemes defend against network flooding attacks. Dwork et al. propose puzzles to discourage spammers from sending junk email [12]. Juels et al. use puzzles to prevent SYN flooding [17]. Aura et al. [5], Dean and Stubblefield [10], and Wang and Reiter [27] propose puzzles to defend against DoS attacks on application-level client authentication mechanisms. These systems require the server under attack to provide and verify the puzzle and solution and are generally inappropriate for attacks that require in-network prioritization. Gligor [14] analyzes the wait-time guarantees that different puzzle and client-challenge techniques provide. He argues that application-level mechanisms are necessary to prevent service-level flooding and proposes a scheme that provides per-request, maximum-waiting-time guarantees for clients under the assumption that lower-layer, anti-flooding countermeasures exist.

The approach of Waters et al. [29] comes closest to the proof-of-work mechanism used by Portcullis. They utilize a distribution mechanism for puzzle challenges based on a trusted and centralized bastion host. Unfortunately, this approach allows attackers to reuse puzzle solutions for multiple destinations. In addition, to verify puzzle solutions, the verifier must generate a large lookup table by performing many public-key operations, which would impose an excessive burden on routers since puzzle seeds change frequently.

Adopting an economic approach in "'Proof of Work' Proves Not to Work", Laurie and Clayton analyze the effectiveness of using computational puzzles to fight spam [21]. However, Wobber discovered an arithmetic error in a profit margin calculation that undermines one of the key results [8]. Thus, the correct conclusion of

their argument is that computational puzzles are a viable solution at current spam response rates. Also, their arguments only consider a simple fixed-rate payment system that differs significantly from the proof-of-work scheme used by Portcullis.

9. CONCLUSION

The Denial-of-Capability (DoC) attack is a serious impediment for capability-based DDoS defense mechanisms. Portcullis strictly bounds the amount of delay a collection of attacking nodes can create for any client. With realistic Internet-scale simulations, we show the strong fairness Portcullis's computational puzzles provide. Portcullis introduces a powerful mechanism for providing DDoS resistance, but that benefit requires additional complexity. Only time will tell if the Internet will need the strict availability guarantees originally proposed by past capability schemes and now made robust against DoC by Portcullis. In the mean time, we believe Portcullis provides an important design point to inform the debate on highly available network architectures.

Acknowledgements

The authors would like to thank Jonathan McCune for his technical assistance, Diana Parno for her extensive editorial contributions, and David Maltz and our anonymous reviewers for their helpful comments and suggestions.

This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grants CNF–0433540 and CNF–0435382 from the National Science Foundation. Bryan Parno is supported in part by an NDSEG Fellowship, which is sponsored by the Department of Defense. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF, or the U.S. Government or any of its agencies.

10. REFERENCES

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

[2] Lisa Amini and Henning Schulzrinne. Issues with inferring Internet topological attributes. In *Proceedings of the Internet Statistics and Metrics Analysis Workshop*, October 2004.

[3] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing Internet denial-of-service with capabilities. In *Proceedings of Hotnets-II*, November 2003.

[4] Katerina Argyraki and David Cheriton. Network capabilities: The good, the bad and the ugly. In *Proceedings of Workshop on Hot Topics in Networks (HotNets-IV)*, November 2005.

[5] T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Proceedings of Security Protocols Workshop*, 2001.

[6] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Journal of Internet Mathematics*, 1(4), 2005.

[7] CAIDA. Skitter. `http://www.caida.org/tools/measurement/skitter/`.

[8] Richard Clayton. `http://www.cl.cam.ac.uk/~rnc1/` Accessed May, 2007.

[9] Team Cymru. The team cymru ip to asn lookup page. `http://www.cymru.com/BGP/asnlookup.html`.

[10] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *Proceedings of USENIX Security Symposium*, 2001.

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

[12] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of CRYPTO*, 1993.

[13] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, January 1998.

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

[15] Helion Technology Limited. Fast SHA-1 hash core for ASIC. Cambridge, England. Available at `http://www.heliontech.com/downloads/sha1_asic_fast_helioncore.pdf`. November 2005.

[16] Helion Technology Limited. Fast SHA-1 hash core for Xilinx FPGA. Cambridge, England. Available at `http://www.heliontech.com/downloads/sha1_xilinx_fast_helioncore.pdf`. November 2005.

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

[18] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. *Transactions on Networking*, 10(5), 2002.

[19] National Institute of Standards and Technology (NIST) Computer Systems Laboratory. Secure hash standard. 180-1, April 1995.

[20] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP Packet Flooding Attacks. In *ACM HotNets-II*, November 2003.

[21] Ben Laurie and Richard Clayton. "Proof-of-work" proves not to work. In *Proceedings of WEIS*, May 2004.

[22] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. In *International Workshop on QoS*, May 2002.

[23] Jeffrey Pang, James Hendricks, Aditya Akella, Bruce Maggs, Roberto De Prisco, and Srinivasan Seshan. Availability, usage, and deployment characteristics of the domain name system. In *Proceedings of the Internet Measurement Conference*, October 2004.

[24] Lindsey Poole and Vivek S. Pai. ConfiDNS: Leveraging scale and history to improve DNS security. In *Proceedings of USENIX WORLDS*, November 2006.

[25] Elaine Shi, Bryan Parno, Adrian Perrig, Yih-Chun Hu, and Bruce Maggs. FANFARE for the common flow. Technical Report CMU-CS-05-148, Carnegie Mellon, February 2005.

[26] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS defense by offense. In *Proceedings of ACM SIGCOMM*, September 2006.

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

[28] X. Wang and M. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *ACM CCS*, October 2004.

[29] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of ACM CCS*, November 2004.

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

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

[32] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *Proceedings of ACM SIGCOMM*, August 2005.