Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles

(Extended Abstract)

XiaoFeng Wang* Michael K. Reiter†

ABSTRACT

We present *congestion puzzles* (CP), a new countermeasure to bandwidth-exhaustion attacks. Like other defenses based on client puzzles, CP attempts to force attackers to invest vast resources in order to effectively perform denial-of-service attacks. Unlike previous puzzle-based approaches, however, ours is the first designed for the bandwidth-exhaustion attacks that are common at the network (IP) layer. At the core of CP is an elegant distributed puzzle mechanism that permits routers to cooperatively impose and check puzzles. We demonstrate through analysis and simulation that CP can effectively defend networks from flooding attacks without relying on the formulation of attack signatures to filter traffic. Moreover, as many such attacks are conducted by "zombie" computers that have been silently commandeered without the knowledge of their owners, the overheads that CP imposes on heavily engaged zombies can increase the likelihood that the computer's owner detects the compromise and takes action to remedy it.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.6 [**Computer-Communication Networks**]: Internetworking—*routers*

General Terms

Security

Keywords

client puzzle, denial of service

1. INTRODUCTION

Current Internet sites continue to suffer from a range of *distributed denial-of-service* (DDoS) attacks, especially *bandwidth-exhaustion attacks*. In a bandwidth-exhaustion attack, adversaries employ DDoS tools to capture a fleet of "zombie" computers, from which they collectively generate a huge volume of traffic to overwhelm the bandwidth of the target network. As in many other types of denial-of-service attacks, bandwidth exhaustion attacks can be mounted with little cost to each zombie and its adjacent network, while in aggregate imposing significant burden on the target.

In this paper, we present *congestion puzzles* (CP), a new countermeasure to bandwidth-exhaustion attacks. A typical puzzle is composed of a moderately-hard function; solving the puzzle requires a brute-force search in the solution space. Once a link adjacent to a router implementing the CP mechanism (a *puzzle router*) is congested, the router requires the traffic flow to be accompanied by a corresponding *computation flow*, i.e., a continuous flow of puzzle solutions, thereby imposing a computational burden on clients who transmit via this router. The rate of the computation flow (average number of searching steps per second) is tied to the bandwidth consumed (bytes per second) by a *puzzle-based rate limiter* (PRL) implemented in the router. As a result, this coarsely requires from clients a computation flow commensurate with their bandwidth usage on the congested link, thereby impairing their ability to sustain a flooding attack. The consumption of CPU cycles in zombie computers may additionally alert the unwitting owners of those computers to their contribution to the attack, and motivate them to repair their computers.

While the CP mechanism can be somewhat effective when implemented by each router in isolation, our approach additionally extends to a *distributed puzzle mechanism* (DPM) through which a router can ask its upstream[1] routers to help control the attack flows before converging to the congested link. DPM enables multiple routers to efficiently coordinate with each other to generate and distribute puzzles and to check puzzle solutions. On the other hand, DPM also has routers work independently, and so is robust to attacks from corrupted routers.

CP offers many other advantages among approaches for defending against flooding attacks. First, unlike many proposals for deploying defenses in the network (e.g., [25, 21, 39]), CP does not require the formulation of accurate attack signatures by which routers detect or filter attack traffic. Second, congestion puzzles support incremental deployment; our simulation results suggest that the bandwidth-exhaustion attacks can be greatly mitigated with only a small fraction of routers implementing CP. Third, we demonstrate that CP permits lightweight implementation within

*School of Informatics and Computer Science Department, Indiana University at Bloomington, Bloomington, IN, USA; xw7@indiana.edu

†Department of Electrical and Computer Engineering, Department of Computer Science, and CyLab, Carnegie Mellon University, Pittsburgh, PA, USA; reiter@cmu.edu

CCS'04, October 25-29, 2004, Washington, DC, USA.

[1]Throughout the paper, we call the direction of attack flows (from zombies to the victim) the "downstream" direction and the reverse direction the "upstream" direction.

routers. Fourth, since we apply puzzles at the network (IP) layer, CP might assist in defending against higher-level denial-of-service attacks, as well.

2. RELATED WORK

2.1 Countermeasures to bandwidth-exhaustion attacks

Mechanisms to counter bandwidth-exhaustion attacks include aggregate-based congestion control [25, 21, 40], traceback [11, 8, 31, 34, 13, 4, 33] and filtering [17, 24, 36, 32, 22, 39].

Aggregate-based congestion control (ACC) has been proposed by Mahajan et al. [25] and implemented by Ioannidis and Bellovin [21]. This mechanism extends traditional flow-based congestion controls [15, 35, 18, 26] so as to manage packet flows at a finer granularity. An aggregate is defined as a collection of packets that share some property (signature). ACC provides mechanisms for detecting and controlling aggregates at a router using an attack signature, and a *pushback* mechanism to propagate aggregate control requests (and the attack signature) to upstream routers. ACC critically depends on the mechanism by which attacks are detected and an attack signature is formulated, and this can be a source of difficulty against an intelligent adversary that varies its traffic characteristics over time. A goal of CP is to avoid the need to formulate attack signatures.

A related congestion control mechanism is *level-k max-min fair router throttles* [40]. The mechanism differs from ACC in that a congested server is responsible for issuing congestion-control requests to routers k hops away (denote the set of these routers by $R(k)$) to help maximize the bandwidth allocated to those receiving the smallest allocation (*max-min fairness*). This approach does not depend on formulating attack signatures, but offers fairness only to the extent that the routers in $R(k)$ can provide it. With a low deployment depth (small k), it is possible that legitimate clients' flows may aggregate to a relatively high bandwidth flow before reaching a router in $R(k)$, thus being subjected to rate limiting. Another limitation of the mechanism is the assumption that all routers are trusted, which makes it vulnerable to attacks from compromised routers.

Several methods focus primarily on filtering or tracing spoofed traffic, such as ingress filtering [17], SAVE [24], Centertrack [36], hop-count filtering [22], Pi [39] and numerous works on *traceback* (e.g., [11, 8, 31, 34, 13, 4, 33]). These approaches are of less utility against non-spoofed traffic, and thus permit DDoS attacks from zombies using their real source addresses. In addition, many of these schemes rely upon some way of distinguishing attack packets from legitimate ones, thereby again raising the difficulties of generating attack signatures. Finally, some filtering schemes consider coordination among routers. For example, Shnackenberg et al. present an approach to express the interactions between routers for blocking malicious traffic [32]. Our approach also supports such coordination within the context of the CP mechanism.

Recently, Morein et al. propose an approach that uses overlay network to protect web servers from congestion-based DDoS attacks [27]. An overlay network is composed of a set of nodes across the Internet. The routers around the protected web server admit HTTP traffic from only trusted locations known to overlay nodes. A client who wants to connect to the web server has to first pass a reverse Turing test posed by an overlay node, which then tunnels the client's connection to an approved location so as to reach the web server. This approach, however, does not solve the general bandwidth-exhaustion problem: First, adversaries might still be able to use other protocols (e.g., UDP) or the traffic addressed to a less sensitive server to congest routers on paths to the web server. Second, this solution is tailored to protocols driven by human users, who can be called upon to pass a reverse Turing test. Third, once adversaries have implanted zombies at overlay nodes or routers, they might circumvent the defense mechanism.

2.2 Client puzzles

Client puzzles have been proposed to defend against denial-of-service attacks in the context of TCP (e.g., [23, 38]), authentication protocols (e.g., [5]), and TLS (e.g., [14]), to name a few. To our knowledge, no puzzle protocol has been proposed to defend against DDoS attacks on the IP layer. Feng has argued the importance of implementing puzzles at the IP layer, because otherwise, any upper-level puzzle protection is still vulnerable to the DDoS attacks at this layer [16]. Feng further discussed desirable properties for IP puzzles, including efficiency; resistance to misuse and circumvention; fairness (in the sense that misbehavior should be punished); fine-grained control; and a simple and incentive-compatible path for deployment. We believe our proposal is satisfies many of these desiderata. Furthermore, our mechanism is compatible with existing network protocols and can operate in a decentralized way, so that multiple upstream routers can cooperate to defend against a bandwidth-exhaustion attack.

Whereas most puzzle proposals impose a number of computational steps to generate a solution, there exist other types of puzzles. Abadi et al. propose a "memory bound" puzzle that imposes memory accesses upon clients in an effort to impose similar puzzle solving delay even on different hardware [3]. Gligor presents an attractive approach that utilizes reverse Turing tests as puzzles to prevent automated flooding in network protocols that should be driven by humans [20]. A similar approach also appears in [27]. Gligor also offers insightful comments on the weaknesses of computation-based puzzles in providing guaranteed access for end-to-end services during DDoS attacks. At the IP layer, however, service is characterized by "best effort" delivery, with the goal of max-min fairness in bandwidth allocation [9]. Computation-based puzzles do have the potential to achieve this goal coarsely, and offer various pragmatic benefits: such puzzles are easier to generate and require less state in comparison to other types of puzzles.

3. ATTACK MODEL

We assume that adversaries can modify at most a small fraction of the legitimate packets destined for the target server or network. Attackers capable of tampering with these packets on a large scale do not need to flood the target's bandwidth. Instead, they can launch a DDoS attack by simply destroying these packets. However, our mechanism still works well when attackers have a limited capability to interfere the communication between the legitimate clients and the target server or network.

We also assume that adversaries cannot eavesdrop on most legitimate clients' flows. In practice, monitoring a large frac-

tion of legitimate clients' flows is difficult in wide area networks. This assumption allows us to employ very lightweight authentication schemes using sequence numbers or authentication "cookies".

We allow adversaries to forge any information in the packets they send, to coordinate their zombies perfectly, and to compromise some routers. Adversaries capable of spoofing packet information can simulate legitimate clients' traffic. With perfect coordination, adversaries might manage to reuse puzzle solutions through different routing paths to the target, in the hopes of circumventing the puzzle checking mechanisms distributed over multiple upstream routers.

4. CONGESTION PUZZLES (CP)

4.1 Overview of the CP mechanism

Before presenting the mechanism, we begin by adopting a particular puzzle type. Here we employ a puzzle similar to that of [5], consisting of a *server nonce* N_s created by the congested router and a *client nonce* N_c created by the client. A solution to this puzzle is a string X such that the first d bits of $h(N_s, N_c, X)$ are zeros, where h is a public one-way hash function. We call d the *puzzle difficulty*. We presume that generating candidate values for X is of negligible computational cost, and so treat the verification of a candidate X (i.e., an application of h) as the cost of a trial. This puzzle construction has the property that a congested router needs to generate a server nonce only once for clients to solve multiple puzzles. On the other hand, to avoid keeping too many client nonces for filtering duplicate puzzle solutions, the router will have to update its server nonce periodically. We call such a period the *nonce period*.

In order to transmit packets on a congested route, a client should install a *puzzle client* program. This is an application program that interacts with the operating system only through the standard application programming interface (API). This greatly enhances its ease of deployment: e.g., it could be automatically installed from trusted web sites. A client would have incentives to install this program because it increases the client's likelihood to get her packets through during network congestion. In the rest of this paper, we refer to a client with the puzzle client software installed as a "puzzle client".

The CP mechanism is mainly implemented in routers. A puzzle router will trigger the CP mechanism when an outbound link experiences sustained severe congestion, which can be detected by standard methods (e.g., [25]). For instance, a router may monitor the loss rate on the link: If the loss rate exceeds a threshold for several seconds, the router activates the puzzle mechanism.

Once activated, the CP mechanism distributes puzzle parameters (such as a server nonce and difficulty level) to clients, requiring computation flows (puzzle solutions) for traffic traversing the congested link. The manner in which these parameters are sent to the appropriate clients is detailed in Section 4.2. At a puzzle router's interface, a *puzzle-based rate limiter* (PRL) controls the rate of the inbound bit flows on the basis of the computation flows. We describe this mechanism in Section 4.3.

During a bandwidth-exhaustion attack, a single router usually cannot protect its bandwidth alone. Our solution lets the router push congestion control requests to its upstream routers, which can help prevent the attack flows from converging to the congested router. This is achieved using a *distributed puzzle mechanism* that allows puzzle routers to generate and distribute puzzles and to validate puzzle solutions in a distributed way. We present this mechanism in Section 4.4.

4.2 Puzzle distribution algorithms

A congested router needs to propagate a congestion notification and puzzle parameters to the sources (puzzle clients) of the responsible traffic. Moreover, it needs to periodically update its server nonce at these puzzle clients. Here, we present an algorithm that achieves these goals efficiently.

Our algorithm is based on ICMP messages [28]. ICMP is a set of control protocols that provide feedback about problems in Internet communication. An example is PING in which a client sends an echo request to a server to test whether it is reachable; upon receiving the echo, the server replies with the request message. The ICMP header starts with an 8-bit *type* field that determines the rest of the header; so far, 41 of the 255 available type values have been used by various protocols [6]. The PING echo request (ICMP type 8) also has a 16-bit identifier field and a 16-bit sequence number field to aid in matching echos and replies.

Our approach defines two new types of ICMP messages, a *probe packet* and a *puzzle-solution packet*, by which a puzzle client communicates with a congested router. These messages are constructed similar to PING messages, except that they are identified through new type values. A puzzle client uses probe packets to solicit a congestion notification and initial puzzle parameters from a congested router. A puzzle client uses a puzzle solution packet to deliver puzzle solutions to the router. A puzzle client further takes advantage of puzzle solution packets to solicit updated puzzle parameters. So as to permit seamless transition between puzzle parameter updates, routers permit overlapping nonce periods so that both old and new puzzle parameters are allowed for use during a transition period. We denote this transition period by $\mathcal{T}$.

Upon issuing one of these message types, the puzzle client generates and includes a random string called an *authentication cookie* in the message payload. Using this cookie, any router receiving the message can include this cookie in any response to the client, to authenticate itself to the client.[2] In addition to an authentication cookie, a probe message contains a payload of blank space, of length equal to that needed to store puzzle parameters (the difficulty level and server nonce). A puzzle-solution packet contains puzzle parameters, a puzzle solution and blank space for updating puzzle parameters. How routers process these messages is described below.

Puzzle distribution

Monitoring

Each puzzle client monitors network activity of its local system. Whenever the client system visits an IP address, the puzzle client sends probe messages to that address periodically. If there is no congestion, these messages are silently dropped by either the destination host or the router directly connected to that host.

[2]Recall that adversaries are assumed to have only limited capability to eavesdrop and intercept legitimate clients' packets. Other authentication mechanisms, once deployed, also can be used in our approach.

Distributing congestion notifications

Once a puzzle router detects congestion on one of its outbound links, it generates a server nonce, activates a puzzle-based rate limiter (Section 4.3) and admits a constant flow of probe messages to the congested link from each of its inbound interfaces. For every probe message received, the router inserts the server nonce and puzzle difficulty into its payload (in addition to the authentication cookie), and changes the type of the message to PING echo request. This message will therefore elicit a PING reply to the client containing these parameters.

Updating puzzle parameters

1. Upon receiving a PING reply, the puzzle client first checks the reply with the authentication cookie it contains. If correct, the client stops probing and starts sending puzzle solution packets to the IP address it is visiting (Section 4.3).
2. Upon receiving a puzzle solution packet, the congested router utilizes the puzzle in a rate limiting algorithm; see Section 4.3. If the router has updated its server nonce and/or requested difficulty level, it places these new values into the packet payload (along with the authentication cookie), sets the packet type to PING echo request, and forwards it. This message will thus elicit a PING reply from the destination host to inform the client of the new puzzle parameters.
3. If a puzzle client does not receive any PING replies within a period $\mathcal{T}$, it stops sending puzzle solution packets and starts sending probe messages.

The cost of puzzle distribution is modest. The extra traffic caused by probe messages takes only a small portion of bandwidth because a probe packet will typically be much smaller than the packets in a communication flow. To add or update puzzle parameters in a packet, a router only needs to overwrite existing payload fields.

4.3 Puzzle-based rate limiter (PRL)

During a bandwidth-exhaustion attack, every puzzle client sending packets through a congested link is supposed to generate a virtual "computation flow". The average rate of this computation flow r_c (average number of hash operations per second) is tied to the rate of the client's bit flow r_b (bytes per second) through a public *control function* F:

$$r_b \leq F(r_c, d) \tag{1}$$

where d is the difficulty level of puzzles. F is an increasing function of r_c and a decreasing function of d.

We assume the hash function of our puzzle is a random function (i.e., *random oracle* [7]). That is, for each input, the hash function independently and randomly (with uniform distribution) maps it to an output in its range. The only restriction is that the same input always yields the same output. In practice, a good candidate for random oracle is MD5 with its output truncated [7]. The random oracle model gives us a geometric random variable for the steps used to solve a puzzle. Specifically, to solve a puzzle with initial d (or more) zero bits, a hash step can be viewed as a Bernoulli experiment with a probability of 2^{-d} to succeed. Therefore, the average number of hash operations for finding a solution is 2^d. With this model, a simple construction of the control function is as follows:

$$F(r_c, d) = \alpha 2^{-d} r_c \tag{2}$$

where α is a parameter called *control ratio* measured by *bytes per hash operation*. The control ratio describes the relation between bit flow and computation flow. For example, $\alpha = 10,000$ means that to sustain a bit flow of a rate $r_b = 10,000$ bytes/second, the client is expected to perform 2^d hash operations/second, equivalent to solving at least one puzzle no easier than d per second on the average.

A puzzle router uses the control function to limit the rate of *congestion flows* (flows heading toward the congested link) at its network interfaces. This mechanism is called *puzzle-based rate limiting* (PRL). Without direct observation of computation flow, PRL estimates r_c as $r_p 2^d$, where r_p is the rate of puzzle solutions no easier than d. Specifically, PRL implements a *token bucket* and a *virtual waiting queue* at each network interface. For every inbound puzzle-solution packet carrying a correct puzzle solution, PRL adds α tokens to the token bucket at its inbound interface. An inbound packet will be forwarded toward the congested link by removing number of tokens equal to the packet size from the token bucket. When the tokens are depleted, PRL decides on the fate of the packet according to the virtual waiting queue. If there is sufficient room for queuing the packet, PRL forwards it. Otherwise, PRL discards it. We illustrate the mechanism in Figure 1.

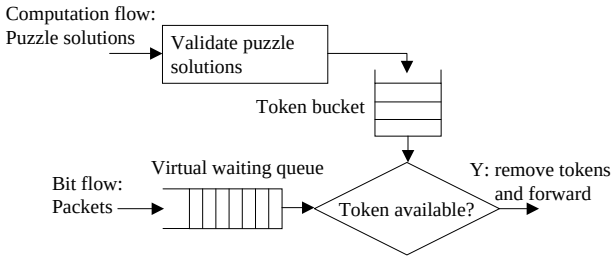

Figure 1: *Puzzle-based Rate Limiter*

If a puzzle router needs to forward puzzle-solution packets to the next hop (Section 4.4), these packets also need to be rate limited by the token bucket because they also belong to the congestion flows, even though the puzzle solutions they carry are part of computation flows. This prevents adversaries from using puzzle packets to aggravate the congestion.

The CP mechanism may control the high-bandwidth flows by tuning puzzle difficulty d. PRL has two thresholds: $th_2 > th_1$. If the loss rate of r_l bits/s at the congested link exceeds th_2, PRL raises d until the loss rate drops just below th_2 but above th_1. If $r_l < th_1$, PRL starts to reduce d. PRL may raise d quickly to suppress attack flows, while lowering d slowly and carefully to prevent intermittent attacks. A problem here is that the puzzle difficulty d only gives a very coarse control of the congestion flows, suppressing r_b exponentially. This can be complemented by fine-tuning the control rate α to maximize the bandwidth utilization.

The idea of PRL is to constrain the upper bound of r_b with r_c and d. For this purpose, *it is not important for a puzzle router to determine whether a particular puzzle solution is correct or not, as long as the router can make a good estimate of* r_c. Therefore, the router only needs to randomly sample some of the puzzle-solution packets to estimate the ratio of wrong solutions. We will elaborate on this in Section 5.

The basic PRL mechanism does not differentiate between congestion flows according to their source IPs: As long as they arrive on the same network interface and are destined to the same congestion IP (the destination IP or IP prefix to which a significant fraction of traffic is destined[3]), they are all controlled by the same token bucket. This gives adversaries opportunities to "free ride" on legitimate clients' puzzle solutions, i.e., if their attack traffic arrives on the same interface as the legitimate clients'. This problem would be mitigated with the wide deployment of puzzle routers, since they can better separate the bit flows from different sources based on inbound interfaces. However, when only a few routers have implemented puzzles, free riding could be significant.

Here, we design a simple algorithm to mitigate this problem, called *IP caching*. For each interface, a puzzle router randomly caches a small set of source IPs or IP prefixes from the incoming puzzle-solution packets. For each IP or IP prefix cached, PRL employs a separate token bucket (called *IP bucket*) to control its bit flow. The rest of the congestion flows are handled by a *main bucket* in the same manner as the basic PRL. PRL updates its IP cache with a *least frequently used* (LFU) algorithm: When the cache is full, the IP whose bucket has the fewest tokens added within some period will be merged into the main bucket, to make room for another IP bucket. A more detailed description of the algorithm will be presented in the full version of this paper.

Although adversaries may use spoofed source addresses, without the ability to deploy zombies arbitrarily or to eavesdrop globally (Section 3), they will be unable to free ride on the vast majority of legitimate clients' flows. The adversaries may also try to use randomly generated source IPs to fill a puzzle router's cache. This attempt can be discouraged with the LFU algorithm: If the adversaries cannot solve a sufficient number of puzzles for these IPs, they will be quickly removed from the cache. In our experiments (Section 7.2), we found that the effectiveness of PRL improved greatly with only a small set of IPs cached in the puzzle routers.

4.4 Distributed puzzle mechanism (DPM)

During a bandwidth-exhaustion attack, a router usually cannot protect itself alone. A cooperative solution, which involves upstream routers to help throttle the attack flows, could offer better defense [19]. At a high level, a congested puzzle router may pass a *congestion notification* including congestion IPs and its puzzle parameters to upstream routers, requesting that they activate PRL to prevent attack flows from converging. This, however, may not work well if adversaries manage to send duplicate puzzle solutions through different paths to the victim. Since individual routers do not have a global view, they cannot determine whether a puzzle solution has already been used on another routing path, and thus are unable to prevent the attack flows from reaching the congested router. In this section, we present a *distributed puzzle mechanism* (DPM) to counter this attack.

Our distributed puzzle mechanism requests individual puzzle routers on the puzzle distribution paths to generate their own *path nonces* and attach them to the congestion notification during the puzzle distribution phase. On the path from the congested router to a client, we denote the path nonce of the ith router (starting from the first router upstream of the congested router) by N_i. We call the sequence $N_s|N_1|N_2\cdots|N_{i-1}|N_i$ router i receives from its downstream routers[4] (including itself) the *nonce sequence*.

For two nonce sequences L_1 and L_2, we denote by $L_2 \in L_1$ if L_2 is a prefix of L_1; we also denote the part remaining after deleting the contiguous sequence L_2 from L_1 by $L_1 - L_2$.

Distributed Puzzle Mechanism (for puzzle router i)

Upon receiving a congestion notification M on interface I

1. Randomly generate a path nonce N_i, append it to M and save the nonce sequence $L_i = N_s|N_1|\cdots|N_i$ and the congestion IPs.
2. Forward M to the upstream neighbors from which packets with congestion IPs come.
3. Activate PRL on all inbound interfaces except I to control the flow with congestion IPs.

Upon receiving a probe packet

Process as a normal probe packet, using L_i as the server nonce.

Upon receiving a puzzle-solution packet with a nonce sequence $L = N_s|N_1|\cdots|N_k|N_c$

1. if ($L_i \notin L$) then drop the packet and return.
2. if ($L - L_i$ appeared before) then drop the packet and return.
3. Validate the puzzle solution (Section 5.1) and then save $L - L_i$ for checking repeated puzzles.
4. Forward the puzzle-solution packet to the next hop.

For solving puzzles or validating solutions, nonce sequences are treated as server nonces. Each router i also takes the sequence of path nonces starting from its upstream neighbor to the puzzle client (i.e., $L - L_i$) as the client nonce, which we call the *client nonce sequence*. Figure 2 illustrates the mechanism.

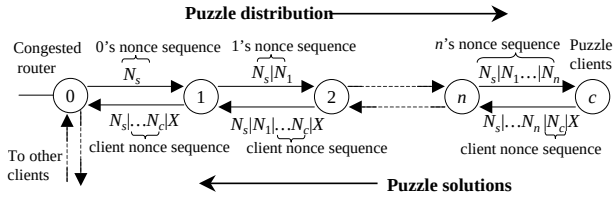

Figure 2: *Distributed Puzzle Mechanism. X represents the puzzle solution.*

By using path nonces, DPM gives different responders different puzzles (nonce sequences), thus preventing the adversary from replaying the solutions via different paths.

DPM can also be used to mitigate multiple congested links occurring simultaneously on a routing path. In this case, individual congested routers need to specify their own puzzle difficulty levels, while sharing the same nonce sequence. A puzzle client then has to solve the puzzle with the highest difficult level according to these routers' specifications.

[3]This IP adress or IP prefix can be obtained using an approach proposed in [25].

[4]Recall N_s is the nonce of the congested router.

5. IMPLEMENTATION COSTS

In this section, we show that with a proper management, the overheads of the CP mechanism (in terms of both computation and memory) can be easily afforded by a modern router.

5.1 Probabilistic validation of puzzle solutions

Essentially, our puzzle-based rate limiting controls the rate of bit flow r_b according to the rate of computation flow r_c. This implies that a puzzle router does not need to know whether a particular puzzle solution is correct, as long as it can reasonably estimate r_c. Probabilistic validation (PV) is based on this idea. Specifically, a puzzle router employs a *sampling probability* p_c to determine whether to validate an inbound puzzle-solution packet; if the packet goes without being validated, the router tosses another coin biased to a *false probability* p_e (see below) to decide whether to add tokens into the token bucket or not. This process is illustrated in Figure 3.

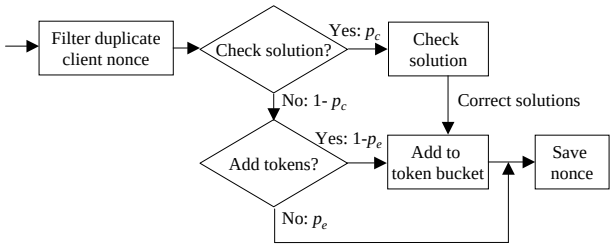

Figure 3: *Probabilistic Validation. "A:p" refers to an event A (Yes or No) that happens with a probability p.*

The false probability p_e represents the ratio of false puzzle solutions contained in the current computation flow, which is estimated from puzzle solutions sampled in the recent past. This raises two research questions, however: (1) how to choose the sampling probability p_c and (2) how to estimate the false probability p_e.

Intuitively, one can use a constant p_c, that is, sample every puzzle-solution packet with the same probability. This treatment, however, does not work well due to the variation in the arrival rate of these packets that the router must accommodate. In particular, an adversary may produce a large volume of such packets in an effort to depleting a puzzle router's CPU resources. Therefore, we employ a *dynamic sampling probability* such that when the arrival rate is within a puzzle router's processing capability, most puzzle solutions will be validated. When the arrival rate grows, the router reduces the number of samples to protect its CPU resources.

We design a very simple dynamic sampling method. At time t, a puzzle router first estimates the packet arrival rate of puzzle-solution packets r_a^t with a typical exponential-averaging rate estimator [35]. Then, the router compares r_a^t with a *sampling index* η, which roughly indicates the average number of hash (e.g., MD5) computations the router is willing to perform in one second for every interface, to compute the sample probability at time t as $p_c^t = \min\{\frac{\eta}{r_a^t}, 1\}$. This sampling probability changes dynamically with the packet arrival rate of puzzle-solution packets.

A follow-up question is how to estimate the false probability p_e. Since every sample has been chosen with a different probability, a simple averaging over all the validation results gives a biased estimate of the ratio of false puzzle solutions. Here, we present two simple estimators which works well with dynamic sampling: *weighted averaging* (WA) and *exponential averaging* (EA).

At time t, WA averages the validation outcomes of the sampled puzzle solutions, weighted by the inverse of the sample distribution over all puzzle-solution packets received before t. In other words, it gives the samples drawn with small p_c heavy weights and these with large p_c light weights.[5] Specifically, WA works as follows. The router keeps the total number of puzzle-solution packets received before t: Θ_t and the sum of all the sampling probabilities before t: $W_t = \sum_{t' \le t} p_c^{t'}$. On validating a puzzle solution at time t, the router increases the total number of samples: $n \leftarrow n+1$ and updates a value V. If the puzzle solution is correct, $V \leftarrow (1 - \frac{1}{n})V$; otherwise, $V \leftarrow (1 - \frac{1}{n})V + \frac{1}{np_c^t}$. Then the estimate of the false probability p_e^t can be computed as: $p_e^t = \min\{\frac{W_t V}{\Theta_t}, 1\}$. The router can reset all the parameters whenever the congested router changes the puzzle difficulty.

Sometimes, adversaries may change their strategy during a DDoS attack. For example, they could honestly solve puzzles initially, and then suddenly produce large numbers of false solutions. In this case, an estimator that can quickly adapt to the adversary's behavior is desired. One such estimator that works well in practice is exponential averaging. EA is as simple as follows: If the router samples a correct puzzle solution at time t, then $p_e \leftarrow (1-\lambda)p_e$; otherwise, $p_e \leftarrow (1-\lambda)p_e + \lambda$, where $0 < \lambda < 1$ is a small constant. The idea of EA is to bias the false probability towards the most recent observations. Therefore, it reflects the adversaries' recent strategy. It does not even need to compensate for the dynamic sampling, given that an appropriate λ is chosen to give a weight to the new sample. In our experiments, we have observed that EA achieved a slightly better performance than WA.

Both EA and WA make a good estimate of r_c with a very small number of samples. Our experiments show that during a bandwidth-exhaustion attack, a router sampling no more than 80 puzzles per second (80 MD5 operations[6]/second) controlled congestion flows effectively. Such computing loads would effect a modern router negligibly. For example, a route-switch processor (RSP) of Cisco 7500 series router [2] has a MIPS 4600 CPU with a clock speed ranging from 100Mhz to 250Mhz. In his paper on MD5 performance [37], Touch shows the performance of optimized MD5 on a comparable CPU MIPS 4400 (with a clock speed 150Mhz) can achieve a rate of about 51.2Mbps. A puzzle-solution packet usually does not exceed 100 bytes. Therefore, performing 100 MD5 operations per second takes only about 0.16% of the router's CPU time. From router CPU usage graphs posted on the Web[7], we conclude that routers generally

[5]Essentially, WA and dynamic sampling are similar to the *importance sampling* in statistics, which concentrates sampling on the important part of a dataset. The difference is that in a DDoS attack, it is hard to tell which part of a computation flow is important: Adversaries with perfect coordination among their zombies can manipulate the flow. Here, the dynamic sampling just serves for protecting routers from exhausting its computing resources.

[6]MD5 operation here refers to the operation of computing an MD5 hash function with the puzzle parameters as input.

[7]For example, `http://supervisor.etsi.org/mrtg/routers/212.234.161.57.9.html`, `http://www.net.fiu.edu/mrtg/cpu/fiulrcpu.html`.

would have plenty of available compute cycles to handle this load.

5.2 Minimizing the memory for storing client nonces

In order to prevent adversaries from reusing puzzle solutions, a puzzle router is expected to keep all client nonce sequences (except those in invalid puzzle solutions) throughout a nonce period. This may constitute a considerable memory expense. In this section, we show how to use a space-efficient data structure called *Bloom filter* [10] to compress the required storage to a size acceptable to a modern router.

A Bloom filter is implemented using a large bit vector with m bits. The bit vector initialized to zeros. For every new puzzle-solution packet, the Bloom filter employs k independent uniform hash functions to map the client nonce sequence to k bits in the vector and then sets each of these bits to 1. Bits can be set multiple times.

A duplicate client nonce sequence can be easily detected by computing the k bits with k hash functions. If any one of these bits is zero, the client nonce sequence has not appeared before within the current nonce period. If all bits have been set, it is highly likely that the puzzle solution is duplicate. It is possible that some unused nonce happens to collide with these stored in the Bloom filter, thereby causing a *false positive*. However, the probability of the false positives can be controlled.

The hash functions implemented in the Bloom filter can be very light-weight, e.g., much more efficient than MD5, since no cryptographic strength is required for these hash functions. Specifically, it does not have to be difficult to find the preimage given a hash image. Previous research presents promising candidates, e.g., the salted CRC-32 [33], which can perform at link speed.

One prominent property of a Bloom filter is that there is an explicit tradeoff between the size of the filter and the probability of false positive. Let n be the maximal number of nonces a puzzle router plans to store. After the Bloom filter is full, the probability of a false positive is: $P = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{\frac{-kn}{m}})^k$. For example, with $m = 16n$ and $k = 8$, the false positive probability is about 0.00058. This gives legitimate clients 25 hops away a mistaken reject rate less than 0.015.

Modern routers could afford the memory for implementing a Bloom filter. Snoeren et al. even suggest to use this method to record the trace of every packet traversing a core router [33]. Their research further shows that mere software support is sufficient for slow-to-medium speed routers (up to OC-12). With proper hardware support, it works for fast routers (OC-48 and faster) [30]. Our approach only records the trace of nonces in a nonce period and thus requires smaller memory in general.

For example, a Cisco 7500 series router has a packet switch capability up to 2.2M packets/second.[8] Given a (m/n) ratio of 16, if all these packets are puzzle solutions, a puzzle router needs 88MB memory to keep all the client nonce sequences within a nonce period of 20 seconds. On the other hand, even the memory on a single RSP can be extended to 256MB or more [2]. Actually, this throughput of puzzle-solution packets is unreasonable because these packets are used to reserve the bandwidth. Let r_p be the rate of the puzzle-solution packets. Since the puzzle router estimates the rate of computation flow as $r_p 2^d$, this packet rate can reserve a bandwidth up to $r_p \alpha$ (see (2)). This suggests that r_p (packets/second) should not exceed $\frac{1}{\alpha}$ of the total bandwidth (bytes/second) too much. For example, given $\alpha = 10,000$, a puzzle-solution packet rate of 1M packets/second does not make sense on a 1Gbps link because this puzzle flow attempts to reserve a bandwidth up to 80Gbps, far beyond the link's capacity. Therefore, a puzzle router can use some standard rate limiter [25] to limit the arrival rate of puzzle-solution packets to an appropriate ratio of its switch/forward capability, before the puzzle flow is processed by PRL. Here we take a ratio of $\frac{\kappa}{\alpha}$, where $0 < \kappa \leq \alpha$ is a constant. With $\kappa = 1$ $\alpha = 10,000$, the size of the Bloom filter is reduced to 1.1MB, which can be easily built into modern routers.

[8]The length of the packet is usually set to 1000 bits.

6. SECURITY ANALYSIS

6.1 Fairness in bandwidth allocation

An important goal for the current best-effort Internet is to fairly allocate available bandwidth among competing users. The classic principle of fairness is *max-min fairness* [9]. Formally, let $\{1, \cdots, U\}$ be the set of sources competing for a link with a capacity C. Let $Z = (x_b^1, \cdots, x_b^U)$ be the vector of bit rates these sources demand. Let $(r_b^1, \cdots, r_b^U)$ be the rate allocation to these sources. An allocation is *feasible* if $\sum_{i \in Z} r_b^i \leq C$. A feasible allocation is *max-min fair* when it is impossible to increase a source i's allocation r_b^i given $r_b^i < x_b^i$, without losing feasibility or reducing the rate of another source i' with an allocation $r_b^{i'} \leq r_b^i$. Roughly speaking, this principle says that an allocation should give the largest possible share of the bandwidth to those sources with the lowest demands for bandwidth, while at the same time not wasting any bandwidth. In bandwidth-exhaustion attacks, adversaries strive to violate this principle, obtaining an unfair share of bandwidth.

Here, we discuss how congestion puzzles help achieve a "weighted max-min fairness", allocating the bandwidth fairly with regard to individual clients' computation efforts. For simplicity, we analyze the CP mechanism over a simplified model of deployment: If a puzzle router's shortest path to the congested router is composed of other puzzle routers, we say it belongs to a "core". On the boundary of the core, puzzle routers are linked to legacy routers not supporting puzzles through their network interfaces. We call these puzzle routers "boundary routers". The core can classify all the packets heading towards the congested link into multiple "flows" according to buckets (IP buckets or main buckets), inbound interfaces and the boundary routers from which they enter the core. In other words, each flow can be characterized by the attribute vector (bucket, interface, boundary router), which we refer to as a virtual "port"; denote these ports $1, \cdots, U$. Let $x_b^1, \cdots, x_b^U$ be the bit rates these ports demand, let $r_c^1, \cdots, r_c^U$ be the rates of computation flows on these ports, and let $r_b^1, \cdots, r_b^U$ denote the bit rates allocated to each port.

Upon tuning the control parameters (including the puzzle difficulty d and control ratio α) to the level such that the bandwidth of the congested link has been just allocated, the bandwidth allocation of the congested link $(r_b^1, \cdots, r_b^U)$

will become a "weighted max-min fair" allocation, in the following sense: for each port i, any increase in r_b^i given $r_b^i < x_b^i$ will cause a decrease in the rate $r_b^{i'}$ for some other port i' satisfying $r_b^{i'}/r_c^{i'} \leq r_b^i/r_c^i$. Intuitively, this holds because a port i with a low demand and a high computation flow rate such that $x_b^i/r_c^i \leq \alpha 2^{-d}$ will get all the bandwidth it asks for, i.e, $r_b^i = x_b^i$; otherwise, the port will get a fair share of bandwidth proportional to the rate of its computation flow $r_b^i = \alpha 2^{-d} r_c^i$. Therefore, to obtain a large share of bandwidth, an adversary must generate computation flows with sufficiently high rates to sustain their demand.

6.2 Robustness against thwarted routers and other misuses

An important security feature of the CP mechanism is that a malicious upstream router can only affect the clients sending packets through it, not any other clients, because its downstream neighbor will control its flow. Some other mechanisms, such as the level-k max-min throttle [40], do not have this feature.

Authentication cookies prevent the adversary from cheating clients into solving puzzles by using false replies to probe packets. Even in the case that adversaries have captured a router, they still cannot force puzzle clients without traffic going through the compromised router to solve puzzles.

Recently, Price has reported an attack on puzzle protocols [29] in general, in which (in our context) a thwarted router may claim a false congestion and pass the puzzles issued by a congested router to the puzzle clients whose bit flows go though the malicious router. In this way, the adversary may recruit some clients to unwittingly solve puzzles for him.

This problem can be addressed by routers requiring the IP address that a puzzle client is visiting to be a part of its client nonce, and discarding puzzle-solution packets that do not satisfy this constraint. Then, unless the puzzle client does have packets through the congested link (and thus should be solving puzzles anyway), the malicious router cannot utilize solutions generated by such puzzle clients.

This treatment, however, still cannot prevent adversaries from coaxing clients into solving puzzles by using other protocols. For example, they may host a music sharing website to ask each visitor to solve a puzzle before downloading songs [29]. Such a threat becomes credible only when the adversary is able to field a service so attractive that a large number of clients are willing to burn their CPU cycles in exchange. Even if the adversary can do so, the expense to maintain the service may also be considerable. Essentially, this is analogous to paying someone money for solving puzzles. Although the adversary may avoid computation costs, he has to pay for the attack in the other way.

7. EXPERIMENTS

In this section, we evaluate the performance of congestion puzzles under bandwidth-exhaustion attacks. Our experiment is based on NS-2 [1], the most widely used network simulator, and CAIDA's Skitter map [12], a traceroute map of real Internet topologies. Due to the limitations of NS-2, we had to keep the scale of our simulation within thousands of nodes. However, we also limited the bandwidth of congested link to only 20Mbps. We believe that a realistic network with higher bandwidth (eg., 1Gbps) could withstand larger scale attacks by using our techniques.

From the skitter map, we randomly selected 1,500 paths. Each path ends with an end host. We randomly chose 500 hosts as legitimate clients. The number of zombies was set to 100, 300, 500, 800 and 1,000. Their locations were also randomly drawn from the end hosts.

On the basis of the 1,500 paths, we constructed a network with NS-2. A congested link which was the adversaries' target connected a web server to the network. The bandwidth of the congested link was set to 20Mbps and every other link to 30Mbps. Each legitimate client simulated traffic for browsing web pages withthe NS-2 web traffic generator. Each adversary produced UDP packets at a constant rate of 300Kbps to target the congested link. The minimum rate of the attack traffic (with 100 adversaries) was 30Mbps and the maximum rate (with 1,000 adversaries) was 300Mbps.

The congested router has a nonce period of 20 seconds. Each end host installed a puzzle-client agent. On receiving congestion notification, each puzzle client started to continuously solve puzzles of the difficulty level d given by the congested router. We set the time to perform one MD5 operation to 10 microseconds. Each puzzle client determined the number of MD5 steps n for finding a puzzle solution according to a geometric random variable with a distribution $(1 - 2^{-d})^{n-1}2^{-d}$. This realistically simulated the puzzle-solving delay. After solving a puzzle, the puzzle client sent a puzzle-solution packet to the congested router.

7.1 Puzzle difficulty

We first evaluated the performance of congestion puzzles using different levels of puzzle difficulty. Figure 4 top depicts the impact of puzzle difficulty (x-axis) on the legitimate clients' packet acceptance rate (the number of packets sent vs. the number of packets received by the web server). Here, difficulty level 0 represents the case without congestion puzzles. We note that the sending rates of both attackers and legitimate clients are unaffected by the puzzle solving difficulty, as puzzle solving (by puzzle clients) is decoupled from application traffic, though obviously difficulty impacts this traffic reaching the target.

Without puzzles, legitimate clients stood little chance to connect to the web server. The situation improved with increase of the puzzle difficulty. When there were less than 300 zombies, the peak of the acceptance rate arrived with $d = 18$, more than 90%. In the presence of more zombies, more difficult puzzles were expected for choking the attack flows. Especially, in the case that the number of zombies exceeded that of the legitimate clients, we needed $d = 20$ to secure an acceptance rate above 85%. Higher difficulty levels were unnecessary and adversely affected legitimate clients' packet acceptance rates.

Adversaries' traffic was substantially controlled with the increase of puzzle difficulty. This is presented in Figure 4 bottom. This experiment suggests that by tuning puzzle difficulty appropriately, congestion puzzles can effectively contain a bandwidth-exhaustion attack.

7.2 Partial deployment

In this experiment, we investigated the performance of congestion puzzles when puzzle routers were only partially deployed. In these experiments, we randomly chose some percentage of routers out of the network as legacy (non-puzzle) routers. However, we fixed the routers close to the congested router (within five hops) to be puzzle routers; de-

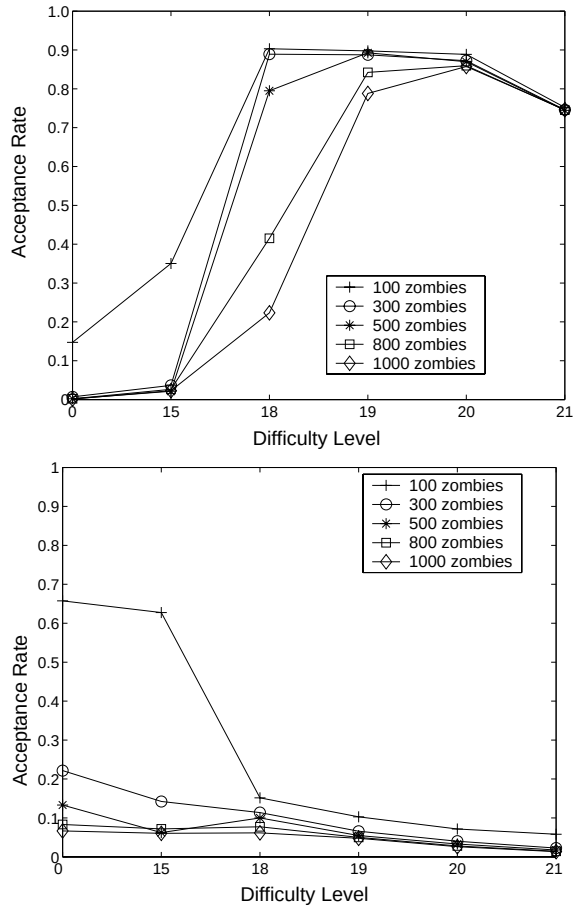

Figure 4: *Impact of puzzle difficulty on the packet acceptance rate. Up: Legitimate clients, Bottom: Adversaries*

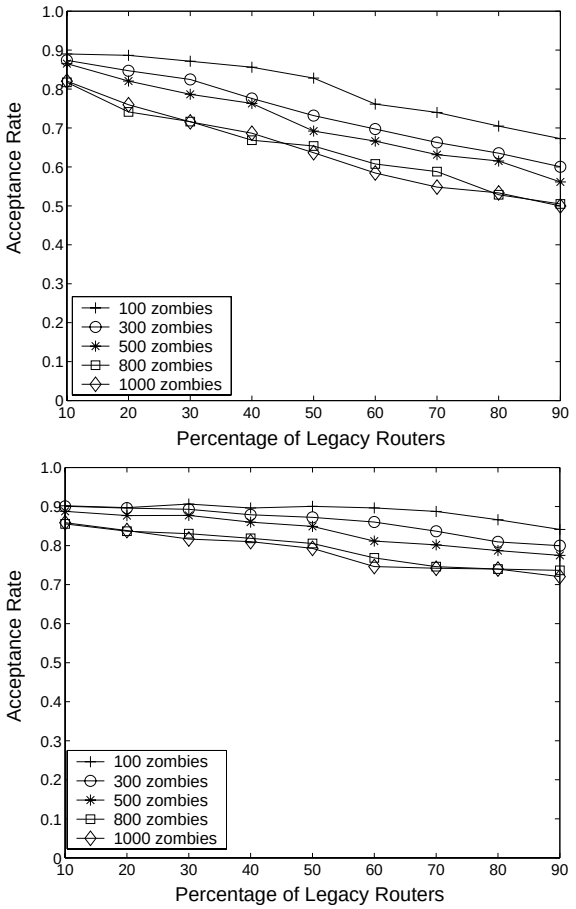

Figure 5: *Legitimate client acceptance rate for partial distribution. Top: Without IP caching, Bottom: With IP caching*

ployment was "partial" only further away. We believe this setup will reflect a situation that might occur in practice were congestion puzzles adopted, in which puzzle routers are deployed in clusters to defend important stub networks. Note that these routers are included in the calculation of the percentage of legacy routers. In this context, we tested congestion puzzles with and without IP caching, which helps achieve a fine-grained control of the inbound flows when adversaries attempt to free ride on legitimate clients' computation flows.

We present the experimental results in Figure 5, in which the x-axis represents the percentage of legacy routers out of all the routers in the network, and the y-axis is the acceptance rate of legitimate clients' packets. In the cases that the number of zombies did not exceed that of legitimate clients, we set the puzzle difficulty $d = 19$, otherwise, we set $d = 20$.

The figure on the top describes the experiment without IP caching. Legitimate clients' acceptance rate decreased with the increase of the percentage of legacy routers. Until the legacy routers took 50% of the whole network, the acceptance rate kept above 60% even with 1,000 zombies. The mechanism also performed well with a small number of zombies. For example, with 90% legacy routers and 100 zombies, near 70% acceptance rate was achieved. However, a minimal deployment (90% legacy routers) plus a large number of zombies (1000) intensified the free riding problem, thereby reducing the acceptance rate to about half.

The free-riding problem could be effectively suppressed with IP caching. In the figure on the bottom, we show the results of the experiment in which each puzzle router randomly cached 10 IPs/port. This treatment made the mechanism perform well even when only a very small fraction of routers supported puzzles: With 1000 zombies and 90% legacy routers, more than 70% of legitimate clients' packets were still able to reach the web server in spite of the bandwidth-exhaustion attack.

7.3 Performance of probabilistic validation

In this experiment, we empirically studied the idea of probabilistic validation when zombies generated false puzzle "solutions".

We considered the adversaries with two strategies. With a stationary strategy, each zombie decided on whether to generate a false puzzle solution according to a fixed probability p drawn uniformly at random. With a dynamic strategy, a zombie randomly chose two probabilities $p_1 < p_2$ and a switching time $0 < t < 20$, and generated false puzzle solutions with p_1 before t and then switched to p_2.

We evaluated PV with dynamic sampling and either the weighted averaging (WA) or exponential averaging (EA). To protect routers from spending computation on a flow containing hardly any correct puzzle solutions, we set a policy that once a router made more than 300 samples from a puzzle solution flow and found its false probability always above

0.95, the router would stop checking the flow and drop all the packets.

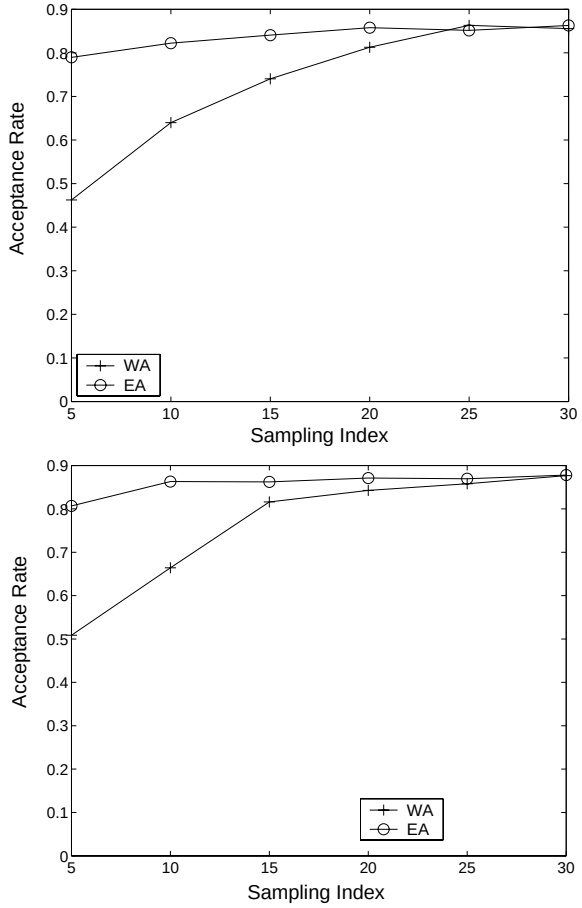

Figure 6: *Legitimate client acceptance rate for probabilistic validation. Top: Stationary strategy, Bottom: Dynamic strategy*

In Figure 6, we present the experimental results. In the experiment, we fixed the number of zombies to 1,000 and set the puzzle difficulty to 20. The x-axis in the figures gives the sampling index, a rough indication of the maximal number of MD5 operations each router was willing to perform per network interface every second. The top figure shows the case with stationary adversaries. WA was pretty sensitive to the sampling index. It performed well after the index exceeded 20. In contrast, EA behaved in a more stable fashion, only varying a little (about 5%) while the index increasing from 5 to 30. Both estimators helped the CP mechanism achieve more than 85% acceptance rate with large index.

Surprisingly, adversaries gained nothing from the dynamic strategy. Actually, both WA and EA performed better there. Two factors might have contributed to this result. First, both estimators (especially EA) might be quite capable of catching up to the adversaries' strategy. Second, since we measured the acceptance rate over the packets transmitted in the whole nonce period, the adversaries' relatively honest behavior (before switching time t) might help to improve the final result.

In both experiments, routers made few samples. The maximal MD5 rate was lower than 80 per second for the most heavily-loaded router; the average rate was lower than 10 per second. Such computation load is very affordable for a modern router.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we presented congestion puzzles (CP), a new countermeasure to bandwidth-exhaustion attacks. Our approach is a first attempt to integrate client puzzle defense mechanisms at the network (IP) layer. The CP mechanism can effectively suppress attack flows in a bandwidth-exhaustion attack. It further encourages the owners of zombies to stop attacks and thus mitigates DDoS attacks involving large numbers of zombies. The CP mechanism can achieve a weighted max-min fairness in allocating bandwidth, without depending on the detection of attack signatures, which may be difficult to obtain in the presence of intelligent adversaries. That said, if attack signatures are available, this would support an extension of our approach in which routers give different puzzle difficulties to different flows; we hope to explore this in future work.

In this work we have employed puzzles based on computation, which have the advantages of simplicity and implementation ease. However, they can cause unfairness in puzzle-solving time over different hardware platforms. In future work, we intend to consider the use of memory-bound puzzles [3] at the IP layer. We also intend to explore the effectiveness of this technique for managing *flash crowds*, in which a large number of legitimate clients visit an Internet site at the same time, causing network congestion.

9. REFERENCES

[1] The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[2] Overview of cisco 7500 series router. `http://www.cisco.com/en/US/products/hw/routers/ps359/prod_brochure09186a008009200c.html`.

[3] M. Abadi, M. Burrow, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, 2003.

[4] M. Adler. Tradeoffs in probabilistic packet marking for ip traceback. In *Proceedings of 34th ACM Symposium on Theory of Computing (STOC-02)*, 2002.

[5] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. In *Proceedings of the Cambridge Security Protocols Worshop 2000.* LNCS, Springer-Verlag, 2000.

[6] I. A. N. Authority. ICMP type numbers. November, 2003. `http://www.iana.org/assignments/icmp-parameters`.

[7] M. Bellare and P. Rogaway. Random oracle are practical: A paradigm for designing efficient protocols. In *Proceedings of First ACM Annual Conference on Computer and Communication Security*, 1993.

[8] S. Bellovin, M. Leech, and T. Taylor. The ICMP traceback messages. In *Internet-Draft, draft-ietf-itrace-01.txt*, December 1999. `ftp://ftp.ietf.org/internet-drafts/draft-ietf-itrace-01.txt`.

[9] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, Englewood-Cliffs, New Jersey, USA, 1992.

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

[11] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Unpublished paper*, December 1999.

[12] Caida. Skitter. 2003. `http://www.caida.org/tools/measurement/skitter`.

[13] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to ip traceback. In *Proceedings of Network and Distributed System Security Symposium (NDSS-01)*, February 2001.

[14] D. Dean and A. Stubblefield. Using client puzzles to protect tls. In *Proceedings of 10th Annual USENIX Security Symposium*, 2001.

[15] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queue algorithm. In *Proceedings of ACM SIGCOMM*, 1989.

[16] W. Feng. The case for tcp/ip puzzles. In *Proceedings of ACM SIGCOMM Future Directions in Network Architecture (FDNA-03)*, 2003.

[17] P. Ferguson and D. Senie. RFC 2267: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, Jan. 1998. `ftp://ftp.internic.net/rfc/rfc2267.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2267.txt`.

[18] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, August 1999.

[19] X. Geng and A. Whinston. Defeating distributed denial of service attacks. *IEEE IT Professional*, 2(4):36–41, July—August 2000.

[20] V. Gligor. Guaranteeing access in spite of service-flooding attack. In R. Hirschfeld, editor, *Proceedings of the Security Protocols Workshop*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[21] J. Ioannidis and S. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS-02)*, 2002.

[22] C. Jin, H. Wang, and K. Shin. Hop-count filtering: An effective defense against spoofed traffic. In *Proceedings of ACM CCS*, 2003.

[23] A. Juels and J. Brainard. Client puzzle: A cryptographic defense against connection depletion attacks. In S.Kent, editor, *Proceedings of NDSS'99*, pages 151–165, 1999.

[24] J. Li, J. Mirkovic, and M. Wang. Save: Source address validity enforcement protocol. In *Proceedings of IEEE INFOCOM*, 2002.

[25] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *CCR*, 32(3):62–73, July 2002.

[26] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of ICNP*, November 2001.

[27] W. Morein, A. Stavrou, D. Cook, A. Keromytis, V. Misra, and D. Rubenstein. Using graphic turing tests to counter automated ddos attacks against web servers. In *Proceedings of ACM CCS*, 2003.

[28] J. Postel. RFC 792: Internet Control Message Protocol, Sept. 1981. `ftp://ftp.internic.net/rfc/rfc792.txt`.

[29] G. Price. A general attack model on hash-based client puzzles. In *Proceedings of the 9th International Conference on Cryptography and Coding*, 2003.

[30] L. Sanchez, W. Milliken, A. Snoeren, F. Tchakountio, C. Jones, S. Kent, C. Partridge, and W. Strayer. Hardware support for a hash-based ip traceback. In *Proceedings of the DARPA Information Survivability Conference and Exposition II, 2001. DISCEX'01.*

[31] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for ip traceback. In *Proceedings of ACM SIGCOMM*, August 2000.

[32] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for intrusion detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition 2000*, March 2000.

[33] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-based ip traceback. In *Proceedings of the ACM SIGCOMM*, August 2001.

[34] D. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *Proceedings of IEEE INFOCOMM*, April 2001.

[35] I. Stoca, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM*, 1998.

[36] R. Stone. An ip overlay network for tracking dos floods. In *Proceedings of USENIX Security Symposium*, 2000.

[37] J. Touch. RFC 1810: Report on MD5 performance, June 1995. `ftp://ftp.internic.net/rfc/rfc1810.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1810.txt`.

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

[39] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, May 2003. `http://www.ece.cmu.edu/~adrian/projects/pi.ps`.

[40] D. Yau, C. Liu, and F. Liang. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *Proceedings of IEEE International Workshop on Quality of Service (IWQoS-02)*, 2002.