Mitigating Distributed Service Flooding Attacks with Guided Tour Puzzles

Mehmud Abliz*, Taieb Znati*†, and Adam J. Lee*
*Department of Computer Science
†Telecommunication Program
University of Pittsburgh
Pittsburgh, Pennsylvania 15260 USA
{mehmud, znati, adamlee}@cs.pitt.edu

***Abstract*—Various cryptographic puzzle schemes have been proposed as defenses against Denial of Service (DoS) attacks. However, these schemes have two common shortcomings that diminish their effectiveness as a DoS mitigation solution. First, the DoS-resilience that these schemes provide is minimized when there is a large disparity between the computational power of malicious and legitimate clients. Second, the legitimate clients also have to perform the same heavy puzzle computations that do not contribute to any useful work from the clients' perspective. In this article, we introduce *guided tour puzzles* (GTP), a novel puzzle scheme that addresses these shortcomings. GTP uses latency — as opposed to computational delay — as a way of forcing a sustainable request arrival rate on clients. Measurement results from a large-scale network test-bed shows that the variation in the puzzle solving times is significantly smaller than the puzzle solving time variation of computation-based puzzles. As attackers have much less control over the round-trip delays than they do over the computational power, a latency-based puzzle scheme, such as GTP, provides significantly better protection against strong attackers. Meanwhile, we show that GTP minimizes useless computations required for the client computers. We evaluate the effectiveness of guided tour puzzles in a realistic simulation environment using a large-scale Internet topology, and show that GTP provides a strong mitigation of DoS request flooding attacks and puzzle solving attacks.**

***Keywords*-denial of service; availability; tour puzzles; proof of work; client puzzles; cryptography.**

I. INTRODUCTION

A Denial of Service (DoS) attack is an attempt by malicious parties to prevent legitimate users from accessing a service, usually by depleting the resources of the server, which hosts that service. DoS attacks may target resources such as server bandwidth, CPU, memory, storage, or any combination thereof. These attacks are particularly easy to carry out if a significant amount of server resource is required to process a client request that can be generated trivially. Cryptographic puzzles have been proposed to defend against DoS attacks with the aim of balancing the computational load of the server relative to the computational load of the clients [1] [2] [3] [4] [5] [6].

In a cryptographic puzzle scheme, a client is required to solve a moderately hard computational problem, referred to as *puzzle*, and submit the solution as a proof of work before the server spends any significant amount of resource on its request. Solving a puzzle typically requires performing significant number of cryptographic operations, such as hashing, modular multiplication, etc. Consequently, the more a client requests service from the server, the more puzzles it has to solve, further expending its own computational resources. Puzzles are designed so that their construction and verification can be achieved with minimum server computational load in order to avoid DoS attacks on the puzzle scheme itself. Attacks aimed at the puzzle scheme itself are thereafter referred to as *puzzle solving attacks*.

Originally, cryptographic puzzles were proposed to combat spams [7]. They have then been extended to defend against other attacks, including DoS [2] [3] [6] [8] [9] and Sybil attacks [10] [11]. Furthermore, different ways of constructing and distributing puzzles have been explored [6] [12] [13] [14] [15]. Unfortunately, existing puzzle schemes have shortcomings that limit their effectiveness in defending against DoS attacks.

First, the effectiveness of computation-based puzzles decreases, as the variation in the computational power of clients increases. To illustrate this limitation, consider a system composed of a server whose capacity is R requests per second, N_l legitimate clients whose clock frequency is f, and N_m malicious clients whose clock frequency is $a \cdot f$, where a is a *disparity factor* that represents the degree of disparity between the CPU powers of malicious and legitimate clients. Furthermore, assume that legitimate clients can tolerate a maximum puzzle difficulty of D_{max}, expressed in terms of the number of instructions. The maximum protection the server can achieve against a DoS attack is by setting the puzzle difficulty to D_{max}. During an attack, the total load on the server is the sum of the loads generated by the legitimate and malicious clients, which can be expressed as $N_l \frac{f}{D_{max}} + N_m \frac{af}{D_{max}}$ (without loss of generality, we assume that when solving puzzles clients use their full CPU capacity). Therefore, to carry out a DoS attack against the server, an attacker must at least induce a load on the server that exceeds the server's full capacity, i.e., $N_l \frac{f}{D_{max}} + N_m \frac{af}{D_{max}} \geq R$. Using simple deductions, it is clear that the minimum number of malicious clients required to cause denial of service should satisfy the inequality $N_m \geq \frac{RD_{max} - N_l f}{af}$. Consequently, the minimum number of

malicious clients required to stage a successful DoS attack against the server becomes smaller as the disparity factor a increases, decreasing the effectiveness of a puzzle-based defense in mitigating the DoS attacks.

Second, existing puzzle schemes may exact heavy computational penalty on legitimate clients, when the server load becomes heavy and the need to increase the computational complexity of the puzzle becomes necessary to prevent overloading the server. The negative impact of such a penalty is further compounded by the fact that the puzzle-induced computation does not usually contribute to the execution of any task that is useful to the client, thereby further wasting client resources and limiting the client's ability to carry out other computational activities.

In this article, we propose a novel, latency-based puzzle scheme, referred to as *Guided Tour Puzzle (GTP)*, to address the shortcomings of current cryptographic puzzle schemes in dealing with DoS attacks. The guided tour puzzle scheme is the first to use the concept of latency, as opposed to computational delay, to control the rate of client requests and prevent potential DoS attacks on the server. The main contributions of the proposed work include:

- A comprehensive study of the cryptographic puzzles to derive a list of basic requirements;
- The introduction of two novel puzzle properties, namely *puzzle fairness* and *minimum interference*, that are essential to the effectiveness of puzzle-based defense against DoS attacks;
- Design and analysis of the GTP scheme, and showing that the proposed scheme meets the list of basic requirements while achieving puzzle fairness and minimum interference;
- A thorough evaluation of the guided tour puzzle effectiveness against distributed DoS attacks, using a realistic simulation framework.

The rest of the article is organized as follows. Section II provides a survey of cryptographic puzzle based DoS prevention schemes. Section III describes the system model and the threat model used. Section IV discusses the design goals of GTP scheme. Section V introduces the GTP scheme. In Section VI, we use analysis and measurement to show that guided tour puzzles meet the design goals of GTP scheme. The effectiveness of the GTP in mitigating DDoS attacks is evaluated in Section VII. A conclusion and future plans for extending the puzzle framework are presented in Section VIII.

II. Related Work

Currently, there are many different type of DoS and DDoS defense mechanisms such as filtering based [16] [17], traceback and pushback based [18] [19], capability based [20] [21] and cryptographic puzzle based defense mechanisms. One approach that is similar to GTP scheme is a system called *speak-up* that encourages legitimate hosts to increase their request sending rate during application layer DoS attacks [22]. This method uses a bandwidth based puzzle and is different from the latency based puzzle we proposed. WebSOS [23] is similar to GTP in that both builds a strong distributed network of protection points in front of a DoS vulnerable server. However, key differences exist. The overlay network in WebSOS is used as a tool to hide the IP addresses of the secret nodes that are permitted to send traffic to the protected server, whereas the set of tour guides act as a "delay box" to let the client wait between requests. The WebSOS is designed to protect private services whose users are known a priori, whereas the GTP scheme can be used by both private and public services.

Due to the enormity of various DoS defense solutions, here we limit our survey only to the cryptographic puzzle based mechanisms.

A. Client Puzzles

Dwork and Noar [7] were the first to introduce the concept of requiring a client to compute a moderately hard but not intractable function, in order to gain access to a shared resource. However this scheme is not suitable for defending against the common form of DoS attack due to its vulnerability to puzzle solution pre-computations.

Juels and Brainard [2] introduced a hash function based puzzle scheme, called *client puzzles*, to defend against connection depletion attack. Client puzzles addresses the problem of puzzle pre-computation. Aura et al. [4] extended the client puzzles to defend DoS attacks against authentication protocols, and Dean and Stubblefield [5] implemented a DoS resistant TLS protocol with the client puzzle extension. Wang and Reiter [6] further extended the client puzzles to prevention of TCP SYN flooding, by introducing the concept of *puzzle auction*. Price [24] explored a weakness of the client puzzles and its above mentioned extensions, and provided a fix for the problem by including contribution from the client during puzzle generation.

Waters et al. [9] proposed outsourcing of puzzle distribution to an external service called *bastion*, in order to secure puzzle distribution from DoS attacks. However, the central puzzle distribution can be the single point of failure, and the outsourcing scheme is also vulnerable to the attack introduced by Price [24].

Wang and Reiter [8] used a hash-based puzzle scheme to prevent bandwidth-exhaustion attacks at the network layer. Feng [25] argued that a puzzle scheme should be placed at the network layer in order to prevent attacks against a wide range of applications and protocols. And Feng and Kaiser et al. [3] implemented a hint-based hash reversal puzzle at the IP layer to prevent attackers from thwarting application or transport layer puzzle defense mechanisms.

Portcullis [26] by Parno et al. used a puzzle scheme similar to the puzzle auction by Wang [6] to prevent denial-of-capability attacks that prevent clients from setting up

capabilities to send prioritized packets in the network. In Portcullis, clients that are willing to solve harder puzzles that require more computation are given higher priority, thus potentially giving unfair advantage to powerful attackers.

In all of proposals above, finding the puzzle solution is parallelizable. Thus an attacker can obtain the puzzle solution faster by computing it in parallel using multiple machines. Morever, they all suffer from the resource disparity problem, and interferes with the concurrently running user applications. In comparison, guided tour puzzles are non-parallelizable, and addresses the problems of resource disparity and interference with user applications.

B. Non-Parallelizable Puzzles

Non-parallelizable puzzles prevents a DDoS attacker that uses parallel computing with large number of compromised clients to solve puzzles significantly faster than average clients. Rivest et al. [27] designed a *time-lock puzzle*, which achieved non-parallelizability due to the lack of known method of parallelizing repeated modular squaring to a large degree [27]. However, time-lock puzzles are not very suitable for DoS defense because of the high cost of puzzle generation and verification at the server.

Ma [14] proposed using *hash-chain-reversal puzzles* in the network layer to prevent against DDoS attacks. Hash-chain-reversal puzzles have the property of non-parallelizability, because inverting the digest i in the chain cannot be started until the inversion of the digest $i+1$ is completed. However, construction and verification of puzzle solution at the server is expensive. Furthermore, using a hash function with shorter digest length does not guarantee the intended computational effort at the client, whereas using a longer hash length makes the puzzle impossible to be solved within a reasonable time.

Another hash chain puzzle is proposed by Groza and Petrica [15]. Although this hash-chain puzzle provides non-parallelizability, it has several drawbacks. The puzzle construction and verification at the server is relatively expensive, and the transmission of a puzzle to client requires high-bandwidth consumption.

More recently, Tritilanunt et al. [28] proposed a puzzle construction based on the subset sum problem, and suggested using an improved version [29] of *LLL lattice reduction* algorithm by Lenstra et al. [30] to compute the solution. However, the subset sum puzzles has problems such as high memory requirements and the failure of LLL in dealing with large instance and high density problems.

Although the non-parallelizable puzzles addresses one of the weaknesses of client puzzles discussed in Section II-A, they still suffer from the resource disparity problem and interferes with the concurrently running user applications on client machines. Guided tour puzzles, on the other hand, address these two weaknesses of non-parallelizable puzzles.

C. Memory-Bound Puzzles

Abadi et al. [12] argued that memory access speed is more uniform than the CPU speed across different computer systems, and suggested using memory-bound function in puzzles to improve the uniformity of puzzle cost across different systems. Dwork et al. [13] further investigated Abadi's proposal and provided an abstract memory-bound function with an amortized lower bound on the number of memory accesses required for the puzzle solution. Although these techniques seem promising, there are several issues that need to be resolved regarding memory-bound puzzles.

First, memory-bound puzzles assume a upper-bound on the attacker machine's cache size, which might not hold as technology improves. Increasing this upper-bound based on the maximum cache size available makes the memory-bound puzzles too expensive to compute by average clients. Secondly, deployment of proposed memory-bound puzzle schemes require fine-tuning of various parameters based on a system's cache and memory configurations. Furthermore, puzzle construction in both schemes is expensive, and bandwidth consumption per puzzle transmission is high. Last, but not least, clients without enough memory resources, such as PDAs and cell phones, cannot utilize both puzzle schemes, hence require another service that performs the puzzle computation on their behalf.

III. System Model

In this section, we introduce our system model, including a system overview, a model of cryptographic puzzle protocol, and a threat model.

A. System Overview

We consider an Internet-scale distributed system of clients and servers. A *server* is a process that provides a specific service, for example a Web server or an FTP server. A *client* is a process that requests service from a server. The term *client* and *server* are also used to denote the machines that run the server process and the client process respectively. Clients are further classified as *legitimate clients* that do not contain any malicious logic and *malicious clients* that contain malicious logic. In the denial of service context, a malicious client attempts to prevent legitimate clients from receiving service by flooding the server with spurious requests. An *attacker* is a malicious entity who controls the malicious clients. We refer to a *user* as a person who uses a client machine.

B. Threat Model

The attacker attempts to disrupt service to the legitimate clients by sending apparently legitimate service requests to the server to consume its computational resources. We consider DoS attacks that flood the server with large amount of requests and attacks that attempt to thwart puzzle defense using massive computational resources. It is assumed that

network resources are large enough to handle all traffic, and the resource under attack is server computation.

Our threat model assumes a stronger attacker than previous schemes do. First, we assume that the attacker may possess hardware resources that are several orders of magnitude more powerful than that of the average legitimate clients. Meanwhile, the attacker can take maximum advantage of her resources by perfectly coordinating all of her available computation resources. Next, the attacker can eavesdrop on all messages sent between a server and any legitimate client. We assume that the attacker can modify only a limited number of client messages that are sent to the server. This assumption is reasonable since if an attacker can modify all client messages, then it can trivially launch a DoS attack by dropping all messages sent by all clients to the server. Finally, the attacker may launch attacks on the puzzle scheme itself, including puzzle construction, puzzle distribution, or puzzle verification.

IV. Design Goals

The design goal of GTP scheme is twofold. First, it aims to achieve the general puzzle properties that have been discussed in the existing literature. Second, it must meet the puzzle fairness and minimum interference requirements that are proposed by us to address the limitations of existing puzzle schemes. These requirements and puzzle properties are explained next.

A. General Properties

Computation guarantee. The computation guarantee (also referred to as "bounds on cheating" [31]) means a cryptographic puzzle guarantees a lower and upper bound on the number of cryptographic operations spent by a client to find the puzzle answer. In other words, a malicious client should not be able to solve a puzzle by expending significantly less operations than required. This is discussed in [7].

Efficiency. The construction, distribution, and verification of a puzzle by the server should be efficient in terms of CPU, memory, bandwidth, hard disk, etc. Specifically, puzzle construction, distribution, and verification should add minimal overhead to the server to prevent the puzzle scheme itself from becoming an avenue for denying service [7] [4] [3].

Adjustability of difficulty. This property is also referred to as *puzzle granularity* [28]. Adjustability of puzzle difficulty means the cost of solving the puzzle can be increased at a fine granularity from zero to impossible [4]. Adjustability of difficulty is important, because finer adjustability enables the server to achieve better trade-off between blocking attackers and the service degradation of legitimate clients.

Correlation-free. A puzzle is considered correlation-free if knowing the solutions to all previous puzzles seen by a client does not make solving a new puzzle any easier [4]. If a puzzle is not correlation-free, then it allows malicious clients to solve puzzles faster by correlating previous answers.

Stateless. A puzzle is said to be stateless if it requires constant memory at the server for storing client information or puzzle-related data. This property is discussed in [4].

Tamper-resistance. A puzzle scheme should limit replay attacks over time and space. Puzzle solutions should not be valid indefinitely and should not be usable by other clients [4] [3].

Non-parallelizability. Non-parallelizability means a puzzle solution cannot be computed in parallel using multiple machines [28]. Non-parallelizable puzzles can prevent attackers from distributing computation of a puzzle solution to a group of machines to obtain the solution quicker.

B. Puzzle Fairness and Minimum Interference

Puzzle Fairness. Puzzle fairness means that a puzzle should take approximately the same amount of time to compute by any client, regardless of the CPU power, memory size, and bandwidth available to that client. If a puzzle scheme achieves fairness, then a malicious client with very strong computational resources will effectively be reduced to a legitimate client. Without puzzle fairness, few powerful malicious clients can solve puzzles at a higher rate than many computationally weaker legitimate clients, and may lead to the occupation of most of the server's capacity by few malicious clients.

Minimum Interference. This property requires that puzzle computation at the client should not interfere with the normal usage of the client computer by its users. If a puzzle scheme consumes too many resources, then it interferes with users' normal computing activity. For example, if computing a hash reversal puzzle expends most of the CPU cycles, then a user may feel a very slow response in using other applications that are running concurrently on the client machine. Consequently, users may avoid using any service that deploys such a puzzle scheme.

V. Guided Tour Puzzle

This section presents the GTP scheme. We start out with the main idea behind the GTP scheme, and describe a very basic puzzle protocol. Then, the limitations of the basic protocol is discussed and a solution is given to address each limitation.

A. The Basic Protocol

When a server suspects that it is under attack or its load is above a certain threshold, it asks all clients to solve a puzzle prior to receiving service. In the GTP protocol, the puzzle is simply a tour that needs to be completed by the client via taking round-trips to a set of special nodes, called *tour guides*, in a sequential order. We call this tour a *guided tour*, because the client should not know the order of the tour a priori, and each tour guide must direct the client towards

Table I: A summary of notations.

N	Number of tour guides in the system
G_j	j-th tour guide ($1 \leq j \leq N$)
k_S	Secret key only known to the server
k_{Sj}	Shared key between the server and G_j
$k_{i,j}$	Shared key between G_i and G_j ($i \neq j$)
L	Length of a guided tour
A_x	Address of client x
i_s	Index of the s-th stop tour guide ($1 \leq i_s \leq N$)
t_s	Coarse timestamp at the s-th stop of the tour
R_s	Client puzzle solving request at s-th stop
B	Size of the $hash$ digest in bits

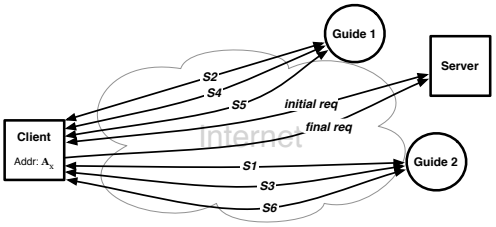


Figure 1: Example of a guided tour; the tour length is 6, and the order of visit is: $G_2 \rightarrow G_1 \rightarrow G_2 \rightarrow G_1 \rightarrow G_1 \rightarrow G_2$.

the next tour guide. Each tour guide may appear zero or more times in a tour, and the term *stop* is used to represent a single appearance of a tour guide in a given tour.

Figure 1 shows an example of a guided tour with two tour guides and 6 stops. The tour guide at the first stop of a tour is randomly selected by the server, and will also be the last stop tour guide, i.e., a guided tour is a closed-loop tour. The tour guide at each stop randomly selects the next stop tour guide. Starting from the first stop, the client contacts the tour guide at each stop and receives a reply. Each reply contains a token that proves to the next stop and the last stop that the client has visited this stop. Prior to sending its reply, the tour guide at each stop verifies that the client visited the previous stop tour guide, so that the client cannot contact multiple tour guides in parallel. After completing $L-1$ stops in a L-stop tour, the client submits the set of tokens it collected from all previous stops to the last stop tour guide (which is also the first stop tour guide), which will issue the client a proof of tour completion. The client then sends this proof to the server, along with its service request, and the server grants the client service if the proof is valid.

There are several issues concerning the basic protocol. First of all, a security mechanism must be in place to enforce the sequentiality of a single tour. Second, as a guided tour does not create a computational or bandwidth bottleneck at the client machine, an attacker may take many tours simultaneously, thereby qualifying itself for more resources of the server. Third, an attacker may cause DoS on the server indirectly by attacking the tour guides and the puzzle scheme itself. In the following subsections, we address each of these challenges individually.

B. Ensuring Sequential Guided Tour

We set up N tour guides in the system, where $N \geq 1$. The server keeps a secret k_S that only it knows, and a set of keys $k_{S1}, k_{S2}, \ldots, k_{SN}$ are shared between the server and each tour guide. Each tour guide G_i maintains a pairwise shared key $k_{i,j}$ with every other tour guide G_j, where $i \neq j$ and $1 \leq i, j \leq N$. The total number of keys need to be maintained by each tour guide or the server is N, and this key management overhead is acceptable since N is usually a small number in the order of 10 or less. The tour length L is decided by the server to adjust the puzzle difficulty. Notations are summarized in Table I.

The four steps of the GTP protocol is as follows.

1) Service request: A client x sends a service request to the server. If the server load is normal, the client's request is serviced as usual; if the server is overloaded, then it proceeds to the next step.

2) Initial puzzle generation: The server replies to the client x with a message that informs the client to complete a guided tour. The reply message contains $\{L, i_1, t_0, h_0, m_0\}$, where i_1 is the uniform-randomly selected index of the first stop tour guide, t_0 is a coarse timestamp, h_0, m_0 are message authentication codes that provide message integrity. h_0 and m_0 are computed as follows:

$$h_0 = hmac(k_S,\ (A_x||L||i_1||t_0)) \quad (1)$$

$$m_0 = hmac(k_{Si_1},\ (A_x||L||i_1||t_0||h_0)) \quad (2)$$

where, $||$ denotes concatenation, A_x is the address (or any unique value) of the client x, and $hmac$ is a cryptographic hash-based message authentication code (HMAC) [32]. Since m_0 is computed using the key k_{Si_1} that is shared between the first stop tour guide G_{i_1} and the server, it enables G_{i_1} to do integrity checking later on.

3) Puzzle solving: After receiving the puzzle information, the client visits the tour guide G_{i_s} at each stop s, where $1 \leq s \leq L$, and receives a reply. Each reply message contains $\{h_s, m_s, i_{s+1}, t_s\}$, where i_{s+1} is the uniform-randomly selected index of the next stop tour guide, t_s is the timestamp at stop s, and h_s, m_s are computed as follows:

$$h_s = hmac(k_{i_s,i_1},\ (h_0||A_x||L||s||i_s||i_{s+1})) \quad (3)$$

$$m_s = hmac(k_{i_s,i_{s+1}},\ (m_{s-1}||A_x||L||s||i_s||i_{s+1}, t_s)) \quad (4)$$

At each stop s, the client sends a puzzle solving request message R_s that contains $\{h_0, L, s, t_{s-1}, m_{s-1},\ i_1, i_s\}$ to the tour guide G_{i_s}, and the tour guide G_{i_s} replies to the client only if m_{s-1} is valid. In other words, each stop enforces that the client correctly completed the previous stop of the tour.

At the $(L-1)$-th stop, the tour guide $G_{i_{L-1}}$ knows that the next stop is the last stop, and replaces i_{s+1} with i_1 (recall that the first stop i_1 is also the last stop) when computing

h_s and m_s. After completing the $(L-1)$-th stop, the client computes h_L as follows

$$h_L = h_1 \oplus h_2 \oplus \ldots \oplus h_{L-1} \quad (5)$$

where $\oplus$ means exclusive or, and submits $\{h_0, h_L, L, m_{L-1}, i_1, i_2, \ldots, i_L\}$ to the first stop tour guide G_{i_1}. Using these information, G_{i_1} can compute $h_1, h_2, \ldots, h_{L-1}$ using formula (3), and subsequently h_L using formula (5). Note that only G_{i_1} can compute values h_1 to h_{L-1}, since only it knows the keys k_{i_1,i_2} to $k_{i_1,i_{L-1}}$ that are used in the HMAC computations.

If the h_L submitted by the client matches the h_L computed by G_{i_1} itself, then G_{i_1} sends back the client a token h_{sol} that can prove to the server that the client did complete a tour of length L. The token h_{sol} is computed as follows:

$$h_{sol} = hmac(k_{Si_1}, (h_0||A_x||L||t_L)) \quad (6)$$

4) Puzzle verification: The client submits to the server $\{h_0, h_{sol}, t_0, t_L, i_1\}$ along with its service request, and the server checks to see if h_0 and h_{sol} that it computes using formulas (1) and (6) matches the h_0 and h_{sol} submitted by the client. If both hash values match, the server allocates resources to process the client's request.

C. Thwarting Simultaneous Tours

The sequential completion of a single tour can be achieved via cryptographic hash chains, as shown above. However, a malicious client can still take multiple tours simultaneously. To prevent simultaneous tours, the GTP scheme limits the number of tours that can be carried out within each time interval. The details are described next.

The time in the GTP protocol is divided into time intervals of length Δ, and T_i is used to denote the i-th time interval. The token that the client receives during the time interval T_i for completing a tour can only be used during the time intervals T_i and T_{i+1}. This restriction can be achieved easily by using a clock tick value of Δ for the coarse timestamp t_s used in the GTP protocol. The interval length Δ must be set to a value that provides enough time for completion of at least a single tour.

Acquiring the token in T_i and using it in T_{i+1} eliminates the additional service delay incurred by using GTP scheme. But, there must be limits to how many tokens a client can acquire per interval and how much service a single token can buy. The policy concerning the per-token resource allocation at the server can be decided by the owner of the service, hence it is beyond the scope of this article. However, it is worth noting that reuse of a token can be prevented by caching only the verified tokens in a Bloom filter [33], and check for the existence in the Bloom filter whenever a new token arrives.

To limit how many tokens a client can acquire per interval, we propose the following solution. During each time interval T_i, a tour guide keeps a pair of counting Bloom filters (in a counting Bloom filter the array positions, or buckets, are extended from being a single bit, to an n-bit counter), one for each interval T_{i-1} and T_i. The tour guide counts the number of tours a client x is taking during each time intervals using the bloom filter, and ignores the client if it has already taken C_{T_i} tours for that period. The value of C is decided by the server for each time interval based on its load, and it is included in the hash chain computation to protect against manipulation by the client. The number of tours are accounted for two time intervals (T_{i-1} and T_i), as a single tour may span two time intervals and using a single counter does not capture that situation. The GTP scheme uses the server timestamp t_0 to decide, which interval a tour belongs to; if $t_0 = T_{i-1}$, the tour belongs to the interval T_{i-1}, and vice versa. This scheme for limiting tours requires the clocks of the tour guides and the server to be synchronized. However, the accuracy of the synchronization is coarse-grained enough (in the order of seconds) such that synchronizing the server and the tour guide clocks independently with a time server using NTP protocol [34] suffices.

D. Increasing Tour Guide Robustness

To prevent attackers from indirectly launching DoS on the server by attacking one of the tour guides, tour guides should be robust against attacks on themselves. As tour guides perform very simple operation, i.e., computing a hash function, they are very light-weight and far less susceptible to DoS attacks. Also due to their simple operation, securing the tour guides against compromise attempts also becomes much simpler. Furthermore, the basic guided tour puzzle scheme is designed to localize the impact of a compromised tour guide. Due to the all-pair pair-wise shared keys, compromising one tour guide only gives the attacker a free ride for the leg of the tour that starts with the compromised tour guide, and the attacker still has to complete the majority of the tour.

In terms of DoS attacks on the tour guides, we propose a simple solution to thwart DoS attacks. In this solution, whenever a tour guide receives a puzzle solving request from the client, it checks to see if the client's request is already in its service queue (the priority queue when the puzzle solving time adjustment mechanisms is used), and it simply drops the request if another request from the same client is already in the queue. This will prevent a malicious client from unfairly taking up more than one slot in the tour guide's service queue, and subsequently minimizes the effect of a DoS attack on legitimate clients.

Although the tour guides are highly immune to DoS attacks, it is still possible for a tour guide to be down due to internal failure or a very strong DoS attack that involves millions of nodes. To operate gracefully even when one of the tour guides is down, all tour guides exchange heartbeat messages with each other and with the server, such that unavailability of a tour guide is immediately known by the

server and other tour guides, and forwarding of clients to the failing or unavailable tour guide is avoided. The heartbeat messages should be adequately protected to thwart attacks on the tour guides.

VI. Analysis

In this section, we use analytical reasoning and experimental results to demonstrate that guided tour puzzles meet all of our proposed design goals.

A. General Puzzle Properties

For each property, we briefly explain how that property is achieved in guided tour puzzle.

Computation guarantee. Each client is required to complete L round-trips in order to obtain a token that proves to the server that it has completed a length L guided tour. A client cannot skip any one of the required round-trips, because doing will be detected by the tour guides immediately. Therefore, guided tour puzzles achieve a strict computation guarantee that enforces the same number of operations for computing the same puzzle answer at all clients.

Efficiency. In guided tour puzzle, construction of a puzzle takes only two hash operations (to compute h_0 and m_0) at the server, and verification of a puzzle answer also takes two hash operations (to compute h_0 and h_L). This efficiency can be further improved at the cost of a small fixed size memory for caching h_0 values. Transferring of puzzle from server to the client requires $2 * B/8$ plus few extra bytes, where the size of hash digest B is usually $160 \sim 256$ bits.

Adjustability of difficulty. The difficulty of a tour puzzle is adjusted by adjusting the tour length L, which can be increased or decreased by one as needed. So, guided tour puzzle provides linear adjustability of difficulty.

Correlation-free. Attackers may try to correlate previousy seen puzzles and puzzle solution to directly obtain a new valid puzzle solution or compute new puzzle solutions after obtaining the secret key used in the hash computations. In guided tour puzzle scheme, resistance to such correlation attacks are achieved through the pre-image resistance and collision resistance properties of the cryptographic hash function used. It is important to pick a hash function that is proven in practice to be secure against various cryptanalytic attacks.

Stateless. Guided tour puzzle does not require the server to store any client or puzzle related information, except for the cryptographic keys that are used for the hash calculation. Puzzle answer verification using memory lookup require few megabytes of memory, but the size of this memory is constant and does not increase as the number of clients increase. The Bloom filter memory used by tour guides is also roughly constant in size and small enough such that it is not susceptible to memory depletion attacks.

Tamper-resistance. The server can guarantee a limited validity period of a puzzle answer, by checking if the difference between the t_L and t_0 is within an acceptable value t_D, i.e. $t_L - t_0 \leq t_D$. Recall that t_0, t_L are submitted by the client to the server in the verification phase of the puzzle protocol, and the client cannot change these values since they are protected by the hash values h_0 and h_L. Meanwhile, the puzzle answer computed by one client cannot be used by any other client, since a value unique to each client is included in the computation of h_0 and h_s.

Non-parallelizability. A guided tour puzzle cannot be completed in parallel, because at each stop s, the tour guide G_{i_s} requires the client to submit the hash values m_{s-1} from the previous stop, and replies to the client if only m_{s-1} is valid. As such, the puzzle scheme strictly enforces sequential completion of a guided tour and achieves non-parallelizability.

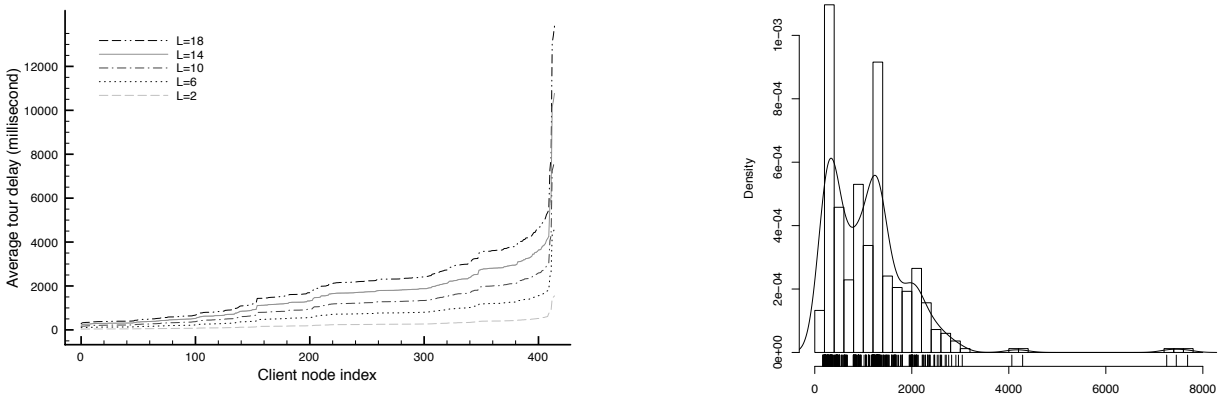
B. Achieving Puzzle Fairness

The guided tour puzzle scheme is not affected by the disparity in the computational powers. It is because the round trip delays that consist the puzzle solving time of a guided tour puzzle are mostly decided by the intermediate network between the client and the tour guides, and the clients' CPU, memory, or bandwidth resources have minimal impact. In terms of the uniformity of puzzle solving time across clients, the guided tour puzzle scheme provides a better guarantee compared to the computation-based puzzle schemes as shown in next subsections.

1) Experimental Analysis: In the following, we use *tour delay* to refer to the sum of all round trip delays occurred in a single tour. We use a two-week long measurement data collected on PlanetLab [35] to show that the variation in tour delay across clients is within a small factor for a large distributed system. PlanetLab has a collection of over $1,000$ nodes distributed across the globe, and provides a realistic network testbed that experiences congestion, failures, and diverse link behaviors [35]. We used about 40% (over 400) of the nodes that had complete measurement data available throughout the two-week period.

We first randomly chose 20 nodes, out of the 400 selected nodes, as candidates for tour guides. The remaining nodes are used as client nodes. The number of tour guides N is varied from 4 to 20, and the tour length L is varied from 2 to 18. For each (N, L) pair, guided tours are generated for all client nodes. The tour delay at a given time is computed based on the round trip delays for corresponding time periods.

To give a better idea of how the tour delays vary across clients on average, we averaged tour delays of all clients over two-week period. To find the average tour delay of a client for a specific (N, L) setting, all tour delays of the client for a given (N, L) configuration is averaged over the two-week period to get the average tour delay of a client



(a) Average tour delays of two-week period, $N = 4$.

(b) PDF of tour delay (unit: millisecond) when $N = 4$

Figure 2: The tour delays of clients when 4 tour guides are used.

for that (N, L) setting. Next, the average tour delays are sorted by least-to-most to provide a better view of the delay variation across clients. Figures 2a shows the average tour delays computed using this method for all client nodes when N = 4. Results for other values of N are skipped due to the space limitation, but they are very similar to the results shown here. The ratio of the largest and the smallest tour delays is around 5, when outliers are excluded. This disparity is several orders of magnitude smaller when compared to the disparity in available computational power (which can be in thousands [12] [13]). Figure 2b shows that majority of tour delays are clustered within a tight area of delay and the distribution of tour delays closely simulates a normal distribution.

2) Analytical Analysis: Although the PlanetLab data provides a somewhat good approximation of the delay characteristics of the Internet, it certainly has limitations. Due to the fact that the majority of the PlanetLab machines are connected to the Internet through campus networks, the delay data may not sufficiently reflect the diverse access network technologies that are used for connecting end hosts to the Internet. Next, we use latency data from the existing literature to show that even when clients are connected to the Internet using access technologies that provide very different delay properties, the disparity in their end-to-end round trip delays will be still smaller than the disparity in the computational power.

Let us take four very common access network technologies with very different delay characteristics: 3rd generation mobile telecommunications (3G), Asymmetric Digital Subscriber Line (ADSL), cable, and campus Local Area Network (LAN). The average access network delays are $200ms$ for 3G [36], $15 \sim 20ms$ for ADSL and cable [37], [38], and in the order of $1ms$ or negligible for campus LANs. Here, we refer to the access network delay as the round-trip delay between the end host and the edge router of the host's service provider. This latency is usually measured by measuring the round-trip delay to the first pingable hop. Based on the measurement analysis of the Internet delay space [39], the delay space among edge networks in the Internet can be effectively classified into three major clusters with average round trip propagation delays of about $45ms$ for the North America cluster, $135ms$ for Europe cluster, and $295ms$ for Asia cluster. Using these edge to edge propagation delay values and the average access network delay values, we can compute an average end to end round trip delays of 245, 335, and 495 ms for 3G hosts, 65, 155, and 315 ms for DSL & cable hosts, and 45, 135, and 295 ms for campus LAN hosts. The biggest disparity occurs between the hosts in the Asia cluster that connect through 3G and the hosts in the US cluster that connects through campus LAN, and the ratio of their round trip delays is $495ms/45ms = 11$. This disparity is about 4 times smaller than the low estimate of computational disparity provided in [26]. The round trip delays may get higher than $495ms$ due to congestion and high queuing delays in the intermediate routers. However, these congestions and high queuing delays affects all packets, regardless of whether they are from malicious clients or legitimate clients. Being able to persistently decrease the tour delay requires the attacker to compromise majority of the intermediate routers between itself and the tour guides, which is hard compared to minimizing computational puzzle solving time by adding more computing power.

C. Achieving Minimum Interference

In guided tour puzzle scheme, a client only has to send/receive packets to/from tour guides. To complete a guided tour puzzle with tour length L, a client only needs to send and receive a total of $2 \times L$ packets with about less than few hundred bytes (depending on the output size of the cryptographic hash function) of data payload. Since L is usually a small number in the order of tens, this creates negligible CPU and bandwidth overhead even for small

devices such as cellular phones.

D. Limitation

As with any other solution, GTP scheme has its limitation. First of all, GTP is a mitigation scheme that reduces the impact of DDoS, hence does not eliminate the effects of the DDoS entirely. The optimal defense that is achievable without being able differentiate malicious requests or detect malicious clients is to service all clients equally. The GTP scheme aims at being very close to such optimal defense. Secondly, the GTP scheme only works at the application/service layer, and relies on other solutions to provide protection against bandwidth flooding DoS attacks. Lastly, the tour delay seen by different clients for the same tour length can be different in the GTP scheme, and this may lead to different levels of service response time experience by different clients. However, note that this disparity in the tour delay does not discriminate against clients that possess less computational resources, and both malicious clients and legitimate clients are equally likely to experience longer tour delays. Having more computational power does not particularly help malicious clients complete a guided tour faster.

VII. Study of DDoS Defense Efficacy

In this study, we focus our evaluation on the ability of guided tour puzzles in mitigating the application layer DDoS attacks. We show that the guided tour puzzle scheme provides an optimal defense against request flooding attacks and a near optimal defense against puzzle solving attacks, when the server does not have the capability to differentiate the malicious clients from the legitimate ones.

A. Simulation Setup

To evaluate the effectiveness of guided tour puzzle in a practical simulation environment, we used Network Simulator 2 (NS-2) [40]. To create a network topology that can closely simulate large-scale wide area networks, such as the Internet, a topology with $5,000$ nodes is generated using the Internet Topology Generator 3.0 (Inet-3.0) [41]. The bandwidth and the link delay values are calculated based on the Inet-3.0 generated link distance values. Note that the link and queueing delays are different from one link to another, therefore the round trip delays, and consequently the tour delays, of different clients will be different.

Since client nodes, tour guides, and server nodes will be located in the edge in real networks, we use only degree-one nodes from the generated topology as client, server, and tour guide nodes. From a total of $1,922$ degree-one nodes, we randomly choose a degree-one node as the server node and another 20 degree-one nodes as potential tour guides. The remaining $1,901$ degree-one nodes are all used as client nodes, which includes both legitimate and malicious client nodes. The percentage of malicious client nodes is varied

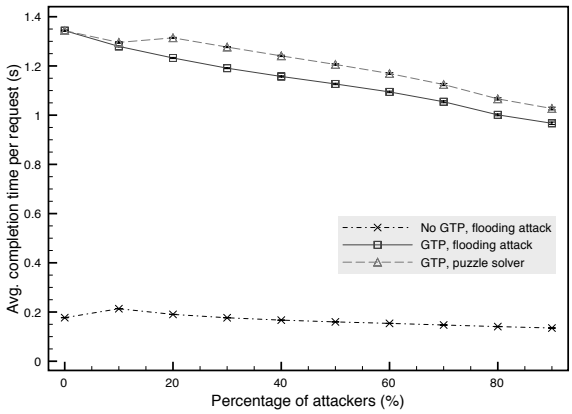
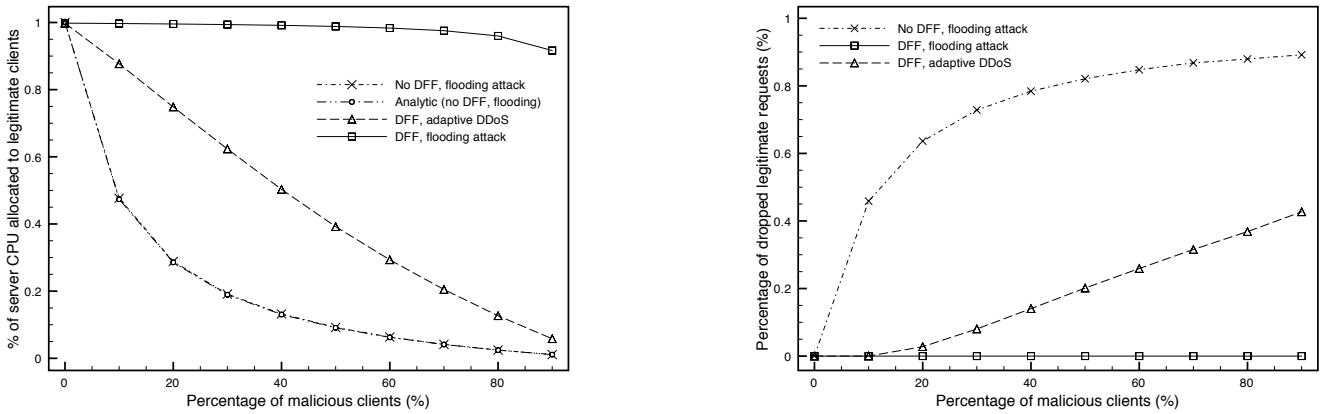


Figure 4: The cost of guided tour puzzles in terms of request completion times.

from 0% to 90% with an increment of 10%, and the server load is increased from 0.96 to 8.74 correspondingly. The server load is calculated as the ratio of the number of incoming requests per second to the server CPU capacity in requests per second.

A simulation model of the guided tour puzzle scheme is developed in NS-2. Clients in the simulation model generate self-similar traffic to closely mimic the Internet traffic. Since the self-similar traffic can be produced by multiplexing ON/OFF sources that have fixed rates in the ON periods and heavy-tail distributed ON/OFF period lengths [42] [43], each client application is implemented as an ON/OFF source with ON/OFF period lengths are taken from a Pareto distribution with shape parameter α equals to 1.5 (NS-2 default). The average ON and OFF times are set to 2 seconds. Each legitimate client sends at an average rate of 8000 bits per second. The average client request size is set to 1000 bytes, thus each legitimate client essentially sends requests at one request per second during on times, and 0.5 request per second on average. The average ON and OFF times, the client request size, and the server response size values selected based on the Web workload model introduced by Barford and Crovella [44].

The same client application model is used for malicious clients except that 1) malicious clients can choose to follow or not to follow the puzzle process, whereas legitimate clients always follow the puzzle process; and 2) malicious clients send requests at a higher rate than legitimate clients. We experimented with two different types of attacks — the flooding attack and the attack against the puzzle scheme. In a flooding attack, a malicious client sends requests at a high rate and ignores the server's request for solving puzzles. In the attack against the puzzle scheme, a malicious client solves puzzles as fast as they can to send requests at the maximum speed possible. The latter is a much stronger attack, since a server that deploys guided tour puzzle scheme can trivially filter out a malicious request that contains an



(a) Percentage of CPU allocated to legitimate clients

(b) Percentage of dropped legitimate requests

Figure 3: The effectiveness of guided tour puzzle against flooding attacks and puzzle solving attacks (N=4, L=8).

incorrect puzzle solution, while a malicious request that includes a correct puzzle solution consumes significantly more resources at the server.

The server's behavior is modeled as described in Section V. Since NS-2 does not provide a CPU model, the server's CPU is modeled as a link between the server node and a dummy node that is connected only to the server. When a client request arrives at the server, the server injects a packet with its size equals to the server response size into the link toward the dummy node. And when the packet is pinged back by the dummy node (which implies the completion of the server processing of the request), the server sends a response to the client. The capacity of this link is set to 80 Mbps to simulate a CPU processing capacity of $\frac{80Mbps}{8\times 10{,}000bytes} = 1{,}000$ requests per second, where the $10,000$ bytes is the average size of a server response. The server capacity of $1,000$ requests per second is used so that the server's full capacity can be reached when all clients are legitimate, and the server load can be increased by 100% with each increase of the percentage of malicious clients. A round-robin queue is used to model the CPU's round-robin process scheduling.

Using the average estimated client request rate of 0.5 request per second and the server CPU rate of $1,000$ requests per second, we can compute that the expected utilization of the server is $\frac{0.5\times 1901}{1000} = 0.9505$ when all the clients are legitimate clients. We achieved a utilization of 0.9656 for this setting in our experiments, which validates the correctness of our simulation setup.

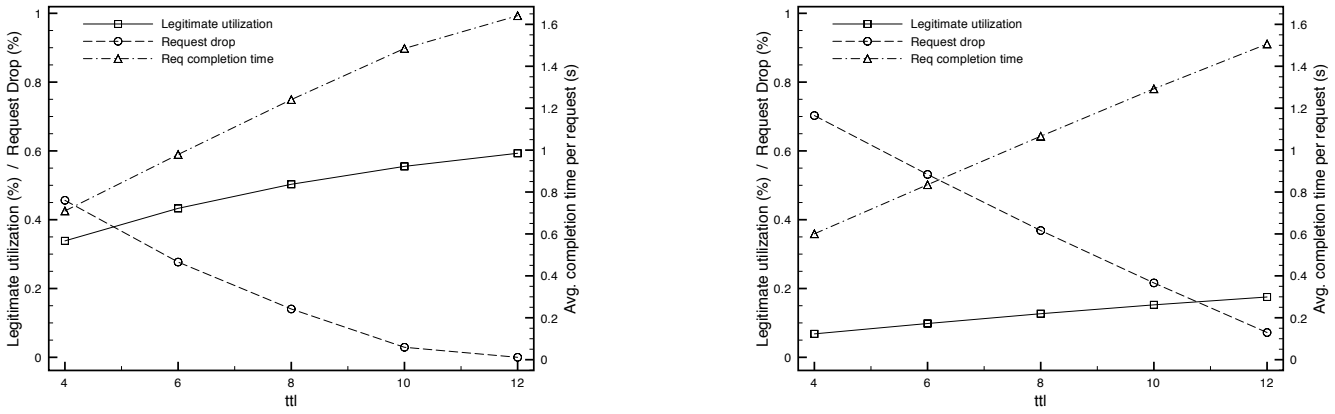
To keep the simulation simple, instead of using an adaptable tour length, a fixed tour length is used within a single run of the simulation. For each solved puzzle, clients are granted service for a single request. We may achieve significantly better protection against the denial of service attack by dynamically adapting the tour length and the number of granted requests per completed puzzle.

We use three evaluation metrics — average completion time of a single legitimate request, percentage of the server CPU allocated to legitimate requests, and percentage of dropped legitimate requests. The average completion time is calculated by recording the time spent between sending of a request and the receiving of its response, which includes the time spent on solving puzzles, for all completed requests of all the legitimate clients and taking the average. The percentage of the server CPU allocated to legitimate requests is computed as the fraction of the time the server's CPU is processing the requests of legitimate clients. The percentage of dropped legitimate requests is computed by dividing the total number of dropped legitimate requests by the total number of legitimate requests sent. For the results we report here, we set the simulation length of each run to 1000 seconds. For each simulation a warmup period of 100 seconds is used, after which recording of the evaluation metric measurement is started. Each experiment is repeated 10 times, and the average of 10 runs is reported along with a 99% confidence interval.

B. Simulation Results

The first set of simulations are conducted with a fixed tour length of 8 and using 4 tour guides. The results are reported in Figure 3 and 4.

1) Server CPU utilization: Figure 3(a) illustrates the improvement in the percentage of the server's effective CPU capacity that is allocated to processing the requests of legitimate clients. As the line "No GTP, flooding" (GTP means guided tour puzzle) indicates, the legitimate clients' share of the server's CPU capacity drops rapidly as the percentage of attackers increases when no guided tour puzzle is used. The percentage of server CPU allocated to processing legitimate requests in this case is predominantly decided by the ratio of total number of legitimate requests to the total number of requests. This can be validated by computing the percentage



(a) 40% attacker, N = 4 (b) 80% attacker, N = 4

Figure 5: The effect of the tour length on the effectiveness of the guided tour puzzle defense.

of legitimate requests for different percentage of malicious clients using the following formula.

$$\frac{r \times (1-x) \times N_c}{r \times (1-x) \times N_c \; + \; 10 \times r \times x \times N_c} = \frac{1-x}{1+9x} \qquad (7)$$

where, r denotes the request rate of legitimate clients, N_c is the total number of client nodes, and x is the percentage of malicious nodes. The line "Analytic (no GTP, flooding)" is then computed using the Formula (7), and it overlaps perfectly with the experiment results from the NS-2 simulation for the case of "No GTP, flooding attack".

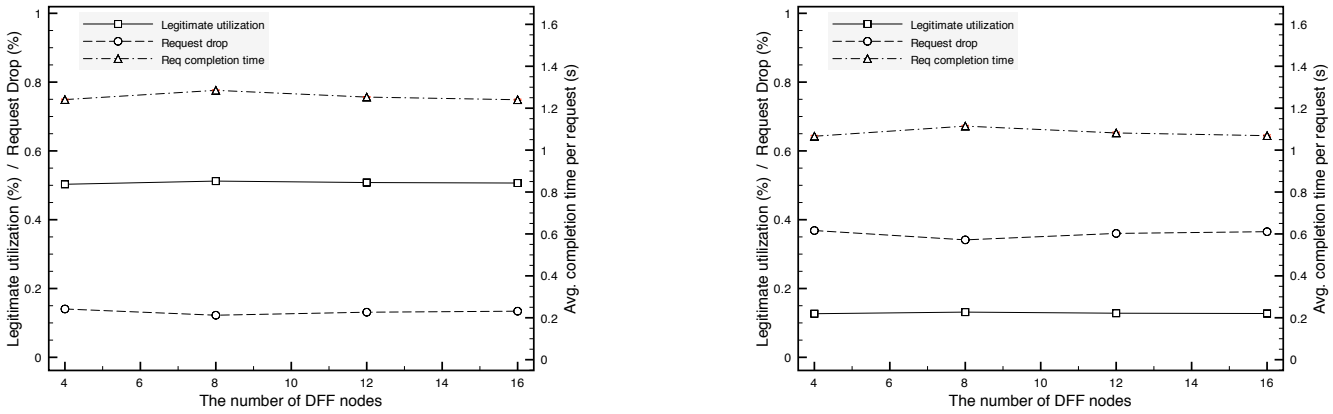
The top line "GTP, flooding attacker" in the Figure 3(a) shows that using guided tour puzzle eliminates the impact of flooding attackers entirely. In this scenario, the malicious clients do not solve any puzzle, but send requests that include fake puzzle solutions at a high rate in an attempt to consume as much server CPU capacity as possible. The slight decrease in the legitimate clients' utilization of the server CPU as the percentage of attackers increases is due to the increase in the percentage of server's CPU capacity allocated to verifying puzzle solutions. We intentionally used a low estimate of 10^6 hash operation per second as the server's hash computation rate to highlight the cost of puzzle solution verification.

The last line "Puzzle, solver" in Figure 3(a) is corresponding to the attack targeted at the guided tour puzzle scheme itself. It shows that the percentage of server CPU allocated to legitimate clients is roughly equal to the percentage of legitimate clients in the system when the guided tour puzzle scheme is used. We argue that without being able to differentiate legitimate clients from the malicious ones, the best a DoS prevention scheme can achieve is to treat every client equally and fairly allocate the server CPU to all the clients that are requesting service. Therefore, the optimal protection that a defense mechanism can provide without being able to differentiate malicious clients is to guarantee the legitimate clients the amount of server CPU that is equal to the percentage of legitimate clients in the system.

2) Request drops: Figure 3(b) shows the percentage of dropped legitimate requests. When no guided tour puzzle is used, the flooding attack caused legitimate clients to drop most of their requests as the line "No puzzle, flooding" indicates. When the percentage of attacker is increased to 90%, almost all legitimate requests are dropped as a result of the flooding attack. After switching to use guided tour puzzles (line "Puzzle, flooding"), the percentage of dropped requests becomes zero under the flooding attack even when the 90% of the clients are malicious. In the case of puzzle solving attacks, guided tour puzzle scheme reduces the legitimate request drops by more than half in all cases, and reduces the request drops to zero in some cases. In fact, the legitimate request drops can be eliminated entirely even in the case of puzzle solving attacks, as the simulation results in "effect of tour length" subsection show.

3) Request completion time: Of course, the benefit of using the guided tour puzzle scheme comes at the cost increased average request completion time, similar to any other "proof of work" based DoS defense mechanism. This cost is shown in the Figure 4. When guided tour puzzle is utilized, the average completion time of a request increased significantly in both flooding attack and puzzle solver attack cases, due to the extra delay introduced by the puzzle solving process. Nonetheless, the increase in the request completion time is within an acceptable range of degradation of service quality. Moreover, the guided tour puzzle scheme provides an easy way to achieve a better trade-off between two mutually restricting sets of quality of service goals by varying the tour length.

4) Effect of tour length: The tour length in guided tour puzzles is critical for the optimality of the guided tour puzzle defense, especially for the legitimate clients' utilization of server CPU in the case of puzzle solving attacks. The next set of simulation experiments are conducted to measure the



(a) 40% attacker, L = 8 (b) 80% attacker, L = 8

Figure 6: The effect of the number of tour guides on the effectiveness of the guided tour puzzle defense.

effect of tour length on utilization, request completion time, and request drops in the case of puzzle solving attacks. Configurations of 40% and 80% malicious clients are used in these experiments, and the number of tour guides N is set to 4.

The response of various metrics to the change in tour length is illustrated in Figure 5. As the tour length increases, the CPU allocated to legitimate clients ("Legitimate utilization") and the request completion time ("Req completion time") increase while the percentage of dropped legitimate requests ("Request drop") decreases. After increasing the tour length to 12, the percentage of dropped legitimate requests becomes zero, and the server CPU allocated to legitimate clients becomes optimal in both cases of 40% and 80% malicious clients. Here the optimal means legitimate clients are granted the amount of server CPU capacity that is equal to the percentage of legitimate clients in the system. Further increasing the tour length does not improve the utilization and request drop metrics and decreases the total utilization of the server CPU, while increasing the request completion time. The increase in the request completion time is evident since larger tour length means more round trips between clients and tour guides. These observations tell us that choosing the right tour length is important in achieving optimal DoS prevention results and providing better trade-off between mutually restricting metrics.

5) Effect of the number of tour guides: The last set of experiments are conducted to determine the effect of the number of tour guides on the effectiveness of guided tour puzzles. The 40% and 80% malicious clients are used, while the tour length L is set to 8. As the results in Figure 6 show, increasing the number of tour guides in the system does not produce any significant change in terms of all three metrics we are measuring. We can conclude from these results that guided tour puzzle can provide a good protection against the DDoS attack with just a few tour guides. Since the tour guides have a single function, which is replying to every request with the hash of the input message contained in the request, it is much easier to protect and maintain. The cost of hardware devices that can be used as tour guides likely to be significantly cheaper than over-provisioning by adding new servers. Moreover, a set of tour guides can be used to protect multiple servers, which further minimizes the cost per server by amortizing the total cost of tour guides over multiple servers.

VIII. Conclusion and Future Work

In this article, we showed that most existing cryptographic puzzle schemes do not consider the resource disparity between clients. We argued that resource disparity reduces or even nullifies the effectiveness of cryptographic puzzle schemes as a defense against denial of service attacks. To this end, we introduced the guided tour puzzle scheme, and showed that it achieves the desired properties of an effective and efficient cryptographic puzzle scheme. In particular, we showed how guided tour puzzles achieve puzzle fairness, minimum interference properties, and how it can achieve better defense against denial of service attacks. Meanwhile, using extensive simulation studies we showed that guided tour puzzle is very effective in mitigating distributed denial of service attacks, and that it is a practical solution to be adopted.

As future work, we would like to further improve the guided tour puzzle scheme in terms of the following. First, we would like to eliminate the need for the server's involvement in the puzzle generation process. Second, further investigation is needed to find out optimal ways to position tour guides in the network. Last but not least, adopting guided tour puzzle to defend against other application layer attacks, such as Sybil attack and spam, is also desirable.

References

[1] M. Abliz and T. Znati, "New approach to mitigating distributed service flooding attacks," in *the 7th International Conference on Systems (ICONS '12)*, Reunion Island, 2012.

[2] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *NDSS '99*, San Diego, CA, 1999, pp. 151–165.

[3] W. Feng, E. Kaiser, and A. Luu, "The design and implementation of network puzzles," in *IEEE INFOCOM '05*, 2005.

[4] T. Aura, P. Nikander, and J. Leiwo, "DoS-resistant authentication with client puzzles," in *8th International Workshop on Security Protocols*, vol. 2133, 2000, pp. 170–181.

[5] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *10th USENIX Security Symposium*, 2001, pp. 1–8.

[6] X. Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *IEEE Symposium on Security and Privacy*, Oakland, 2003, pp. 78–92.

[7] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO '92*, 1992, pp. 139–147.

[8] X. Wang and M. K. Reiter, "Mitigating bandwidth-exhaustion attacks using congestion puzzles," in *CCS '04*, 2004.

[9] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New client puzzle outsourcing techniques for dos resistance," in *11th ACM CCS*, 2004, pp. 246–256.

[10] N. Borisov, "Computational puzzles as sybil defenses," in *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2006, pp. 171–176.

[11] H. Rowaihy, W. Enck, P. Mcdaniel, and T. L. Porta, "Limiting sybil attacks in structured p2p networks," in *the IEEE INFOCOM '07*, 2007, pp. 2596–2600.

[12] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," in *NDSS '03*, 2003.

[13] C. Dwork, A. Goldberg, and M. Naor, "On memory-bound functions for fighting spam," in *CRYPTO '03*, 2003.

[14] M. Ma, "Mitigating denial of service attacks with password puzzles," in *International Conference on Information Technology: Coding and Computing*, vol. 2, Las Vegas, 2005, pp. 621–626.

[15] B. Groza and D. Petrica, "On chained cryptographic puzzles," in *3rd Joint Symposium on Applied Computational Intelligence (SACI '06)*, Timisoara, Romania, 2006.

[16] D. G. Andersen, "Mayday: Distributed filtering for Internet service," in *4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, 2003.

[17] R. Thomas, B. Mark, T. Johnson, and J. Croall, "Netbouncer: Client-legitimacy-based high-performance DDoS filtering," in *3rd DARPA Information Survivability Conference*, 2003.

[18] S. M. Bellovin, M. Leech, and T. Taylor, "ICMP traceback messages," IETF Draft, 2003.

[19] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *ACM SIGCOMM '00*, vol. 30(4), Stockholm, Sweden, 2000, pp. 295–306.

[20] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 130–143.

[21] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in *ACM SIGCOMM*, 2005, pp. 241–252.

[22] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense," in *Proceedings of the SIGCOMM '06*, 2006, pp. 303–314.

[23] A. Stavrou, D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein, "WebSOS: An overlay-based system for protecting web servers from denial of service attacks," *Elsevier Journal of Computer Networks*, vol. 48, 2005.

[24] G. Price, "A general attack model on hash-based client puzzles," in *9th IMA Conference on Cryptography and Coding*, vol. 2898, Cirencester, UK, 2003, pp. 319–331.

[25] W. Feng, "The case for TCP/IP puzzles," in *ACM SIGCOMM Future Directions in Network Architecture*, 2003.

[26] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," in *ACM SIGCOMM*, 2007, pp. 289–300.

[27] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," MIT, Cambridge, Massachusetts, Tech. Rep., 1996.

[28] S. Tritilanunt, C. Boyd, E. Foo, and J. M. González, "Toward non-parallelizable client puzzles," in *6h International Conference on Cryptology and Network Security*, 2007, pp. 247–264.

[29] M. J. Coster, A. Joux, B. A. Lamacchia, A. M. Odlyzko, C. Schnorr, and J. Stern, "Improved low-density subset sum algorithms," *Computational Complexity*, vol. 2(2), 1992.

[30] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261(4), pp. 515–534, 1982.

[31] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks*, 1999, pp. 258–272.

[32] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997.

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

[34] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, Internet Engineering Task Force, Jun. 2010.

[35] "About planet lab," Planet Lab, [Accessed: Dec. 20, 2012]. [Online]. Available: http://www.planet-lab.org/about

[36] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," in *MobiSys '10*, 2010, pp. 165–178.

[37] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *IMC '07*, 2007, pp. 43–56.

[38] M. Yu, M. Thottan, and L. Li, "Latency equalization as a new network service primitive," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, p. 1, May 2011.

[39] B. Zhang, T. S. E. Ng, A. Nandi, R. H. Riedi, P. Druschel, and G. Wang, "Measurement-based analysis, modeling, and synthesis of the internet delay space," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 229–242, 2010.

[40] VINT, "The network simulator - ns-2," 2009.

[41] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," University of Michigan, Tech. Rep. CSE-TR-456-02, 2002.

[42] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, 1994.

[43] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226–244, 1995.

[44] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," *SIGMETRICS Perform. Eval. Rev.*, pp. 151–160, 1998.