New Approach to Mitigating Distributed Service Flooding Attacks

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

***Abstract*—Distributed denial of service (DDoS) attacks pose great threat to the Internet and its public services. Various computation-based cryptographic puzzle schemes have been proposed to mitigate DDoS attacks when detection is hard or has low accuracy. Yet, existing puzzle schemes have shortcomings that limit their effectiveness in practice. First, the effectiveness of computation-based puzzles decreases, as the variation in the computational power of clients increases. Second, while mitigating the damage caused by the malicious clients, the puzzle schemes also require the benign clients to perform the same expensive computation that doesn't contribute to any useful work from the clients' perspective. In this study, we introduce *guided tour puzzles*, a novel puzzle scheme that addresses these shortcomings. The guided tour puzzle scheme uses latency — as opposed to computational delay — as a way of forcing sustainable request arrival rate on clients. We evaluate the DoS mitigation effectiveness of the scheme in a realistic simulation environment, and show that guided tour puzzle scheme provides a strong mitigation of request flooding DDoS and puzzle solving DDoS 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].

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 compute, 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 [6]. They have then been extended to defend against other attacks, including DoS [1] [2] [5] [7] [8] and Sybil attacks [9] [10]. Furthermore, different ways of constructing and distributing puzzles have been explored [5] [11] [12] [13] [14]. 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 compu-

 ISBN: 978-1-61208-184-7

tational penalty on legitimate clients, when the server load becomes heavy and increasing 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 paper, we propose a novel, latency-based puzzle scheme, referred to as *guided tour puzzle*, to address the shortcomings of current cryptographic puzzle schemes in dealing with DoS attacks. The guided tour puzzle scheme is the first to use network latency to control the rate of client requests and prevent potential DoS attacks on the server.

The rest of the paper is organized as follows. Section II describes the system model and the threat model used. Section III introduces the guided tour puzzle scheme. In Section IV, we use analysis and measurement to show that guided tour puzzles satisfy our requirements and design goals. The effectiveness of the guided tour puzzles in mitigating DDoS attacks is evaluated in Section V. Future plans for extending the guided tour puzzle scheme and conclusion of the paper are presented in Section VI.

II. System 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 runs 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 the attacker may possess the best commercially available hardware and bandwidth resources. 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.

III. Guided Tour Puzzle

A. Overview

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 guided tour puzzle 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 does not know the order of the tour beforehand, and each tour guide has to direct the client towards the next tour guide for the client to complete the tour in the correct order. 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 tour.

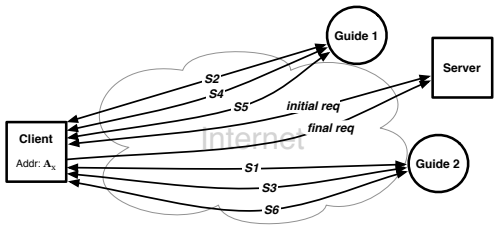


Figure 1. Example of a guided tour; the tour length is 6, and the order of visit is: $G_2 \rightarrow G_1 \rightarrow G_2 \rightarrow G_1 \rightarrow G_1 \rightarrow G_2$.

The 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 to visit. 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 that it completed the tour. The client then sends this

 ISBN: 978-1-61208-184-7

proof to the server, along with its service request, and the server grants the client service if the proof is valid. Figure 1 shows an example of a guided tour with two tour guides and 6 stops.

B. Basic Scheme

We set up N tour guides in the system, where $N \geq 2$. 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 between 2 and 20. 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 guided tour puzzle protocol is described below.

Table I
NOTATION SUMMARY.

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	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

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 — initial puzzle generation.

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 timestamp, h_0 is a hash value, m_0 is a message authentication code (MAC). The value of h_0, m_0 are computed as follows:

$$h_0 = hash(A_x \mid\mid L \mid\mid i_1 \mid\mid t_0 \mid\mid k_S) \quad (1)$$

$$m_0 = hash(A_x \mid\mid L \mid\mid i_1 \mid\mid t_0 \mid\mid h_0 \mid\mid k_{Si_1}) \quad (2)$$

where, $\mid\mid$ means concatenation, A_x is the address (or any unique value) of the client x, and $hash$ is a cryptographic hash function such as Secure Hash Algorithm - 1 (SHA-1) [15]. 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:

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

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

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 hash 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 hash 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 value h_{sol} that can prove to the server that the client did complete a tour of length L. The hash value h_{sol} is computed as follows:

$$h_{sol} = hash(h_0 \mid\mid A_x \mid\mid L \mid\mid t_L \mid\mid k_{Si_1}) \quad (6)$$

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.

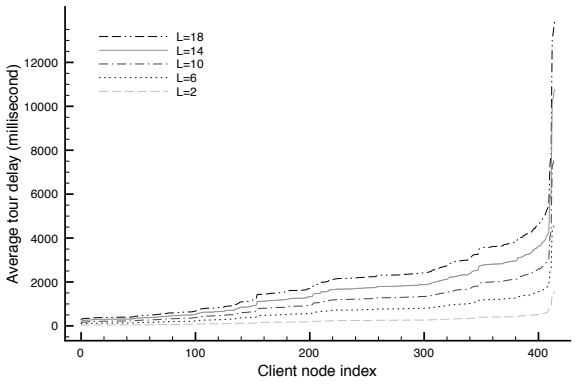
IV. ANALYSIS

In this section, we use analytical reasoning and experimental results to demonstrate that guided tour puzzles are not effected by the disparity in the clients' computational power and minimizes the useless work required for clients.

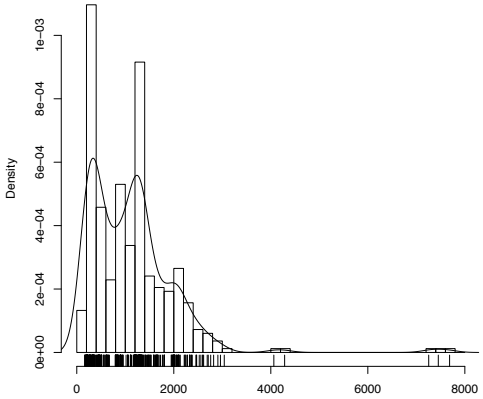
A. Minimize the Effect of Computational Power Disparity

The guided tour puzzle scheme is not effected by the disparity in the computational power of clients. This is because the round trip delays that consist the puzzle solving time of a puzzle are mostly determined by the network(s) between the client and the tour guides, and the clients' CPU, memory, or bandwidth resources have minimal impact on them. As the data that needs to be transferred between client and tour guides is trivial in size, the bandwidth of the end hosts does not effect the round trip delay.

 ISBN: 978-1-61208-184-7



(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.

Since it is possible that the round trip delays from different clients to a tour guide may have significant variation, it is possible that the sum of round trip delays — referred to as *tour delay* — for different clients differ significantly. Next, we use experimental and analytical analysis to show that this variation is small, compared to the variation in the puzzle solving times of computation-based puzzles.

1) Experimental Analysis: We use a two-week long measurement data collected from little over 400 machines on PlanetLab [16] to show that the variation in tour delay across clients is within a small factor for a large distributed system.

We first randomly chose 20 nodes, out of the 400 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) setting, guided tours are generated for all client nodes. The tour delay at a given time is computed buy summing the corresponding round trip delays in that time period. To find the average tour delay of a client for a specific (N, L) setting, the two-week long tour delays of that client for that setting is averaged. Next, the average tour delays are sorted by least-to-most to provide a better view of the delay variation across clients. Figures 2(a) 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 in Figures 2(a). The ratio of the largest and the smallest tour delays is around 5, when 5% 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 [11] [12]). Figure 2(b) shows that majority of tour delays are clustered within a small area of delay and the distribution of tour delays closely reflect a normal distribution. The probability density curve is concave around 1000 milliseconds, as fewer nodes complete the tour in around 1000 millisecond compared to nodes that do in about 500 and 1500 milliseconds.

2) Analytical Analysis: Since 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 is several times 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 [17], $15 \sim 20ms$ for ADSL and cable [18], [19], 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 [20], 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

 ISBN: 978-1-61208-184-7

[21]. 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 effects all packets, regardless of whether they are from malicious clients or legitimate clients.

B. Minimize Interference and Useless Work

In guided tour puzzle scheme, a client has to perform only one type of operation: sending packets to 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 a data payload less than 100 bytes per packet. Since L is usually a small number below 30, this creates negligible CPU and bandwidth overhead even for resource-constrained devices such as cellular phones.

V. Study of DDoS Defense Efficacy

In this study, we focus our evaluation on the ability of guided tour puzzles in preventing the application layer DDoS (also referred to as *distributed service flooding*) 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.

A. Simulation Setup

We use Network Simulator 2 (NS-2) [22] to achieve a practical simulation environment. A topology with $5,000$ nodes is generated using Inet-3.0 [23] to closely simulate large-scale wide area networks, such as the Internet. The bandwidth and the link delay values are calculated based on the Inet-3.0 generated link distance values. The link and queueing delays are set differently for different links, therefore the round trip delays, and consequently the tour delays, of different nodes will be very different.

As clients, tour guides, and server nodes will be located in the edge in real networks, we use degree-one nodes in the topology as the client, server, and tour guide nodes. The topology contains 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 candidates for tour guides. The remaining $1,901$ degree-one nodes are all used as client nodes, including legitimate and malicious client nodes. The number of malicious client nodes is varied from 0% to 90% with an increment of 10%, and the server load is increased from 0.96 to 8.74 correspondingly.

A simulation model of the guided tour puzzle scheme is developed in NS-2. As the Internet traffic is self-similar and the self-similar traffic can be generated by multiplexing ON/OFF sources that have fixed rates in the ON periods and heavy-tail distributed ON/OFF period lengths [24] [25], each client is implemented as an ON/OFF source with ON/OFF period lengths are taken from a Pareto distribution to simulate the Internet traffic. On average, each legitimate client sends 1 request every 2 seconds, and each malicious client sends 10 times the rate of a legitimate client. The server capacity is set to $1,000$ requests per second, such that the server's full capacity can be reached when setting all clients as legitimate. The server load increases by 96% with each 10% increase of the percentage of malicious clients. 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 clients are legitimate. We achieved a utilization of 0.9656 for this setting in our experiments, which validates the correctness of our simulation setup.

Three evaluation metrics are used: average completion time per legitimate request, legitimate utilization of the server, and legitimate request drop rate. The average completion time is calculated by averaging the time spent between sending of a request and the receiving of its response, including the time spent on solving puzzles, for all completed requests of all the legitimate clients. The legitimate utilization of the server is computed as the fraction of the time the server's CPU is processing the requests of legitimate clients. The legitimate request drop rate is computed by dividing the total number of dropped legitimate requests by the total number of legitimate requests sent.

We experimented with two types of attacks: the flooding attack and the puzzle solving attack. In a flooding attack, a malicious client sends requests at a high rate and ignores the server's request for solving puzzles; whereas in the puzzle solving attack, a malicious client solves puzzles as fast as they can to send requests at the maximum speed possible.

B. Simulation Results

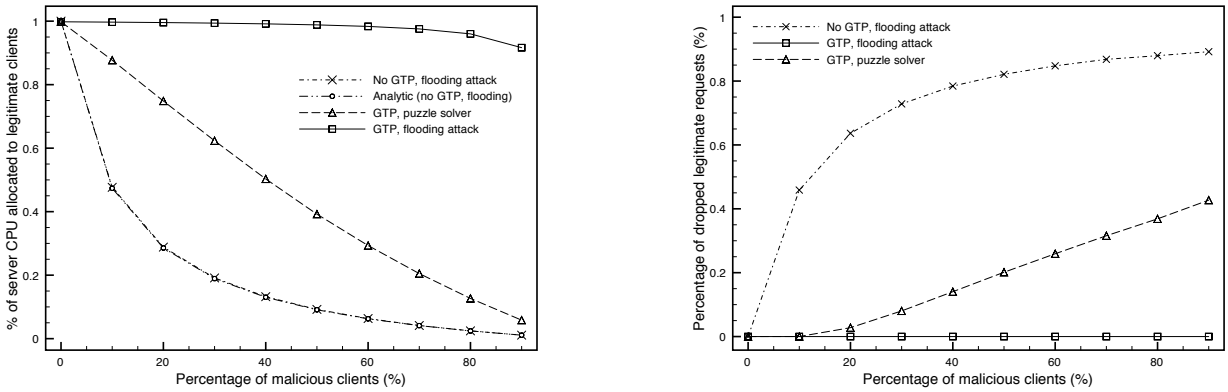
The first set of simulations are conducted with a fixed tour length of 8 and using 4 tour guides. The results are reported in Figure 3.

1) Server CPU utilization: Figure 3(a) illustrates the improvement in the legitimate utilization of the server. As the curve "No GTP (), flooding attack" (GTP stands for 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 legitimate utilization of the server 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 of legitimate requests for different settings 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, x is the percentage of malicious nodes, and $10 \times r$ is the malicious request rate. The curve "Analytic (no GTP, flooding attack)" is then

 ISBN: 978-1-61208-184-7



(a) Legitimate utilization of the server

(b) Legitimate request drop rate

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

computed using Formula (7), and it overlaps perfectly with the experiment results from the NS-2 simulation for the case of "No GTP, flooding attack".

The curve "GTP, flooding attack" 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 protrude the cost of puzzle solution verification.

The last curve "GTP, Puzzle solver" in Figure 3(a) is corresponding to the attack targeted at the guided tour puzzle scheme itself. It shows that, when the guided tour puzzle scheme is used, the legitimate utilization of the server is roughly equal to the percentage of legitimate clients in the system. We argue that without being able to identify malicious clients, the best a DoS mitigation scheme can achieve is to treat every client equally and fairly allocate the server's CPU to all clients that are requesting service.

2) Request drops: Figure 3(b) shows the legitimate request drop rate. When no guided tour puzzle is used, the flooding attack caused legitimate clients to drop most of their requests as the curve "No GTP, flooding attack" indicates. When the percentage of attacker is increased to 90%, near 100% of legitimate requests are dropped as a result of the flooding attack. After switching to use guided tour puzzles (curve "GTP, flooding attack"), the percentage of dropped requests becomes zero under the flooding attack, including when the 90% of the clients are malicious. In the puzzle solving attacks, guided tour puzzle scheme reduces the legitimate request drops by more than half in all cases and to zero in some cases. This legitimate request drops can be eliminated entirely by optimally adjusting the tour length, as the simulation results in the next section show.

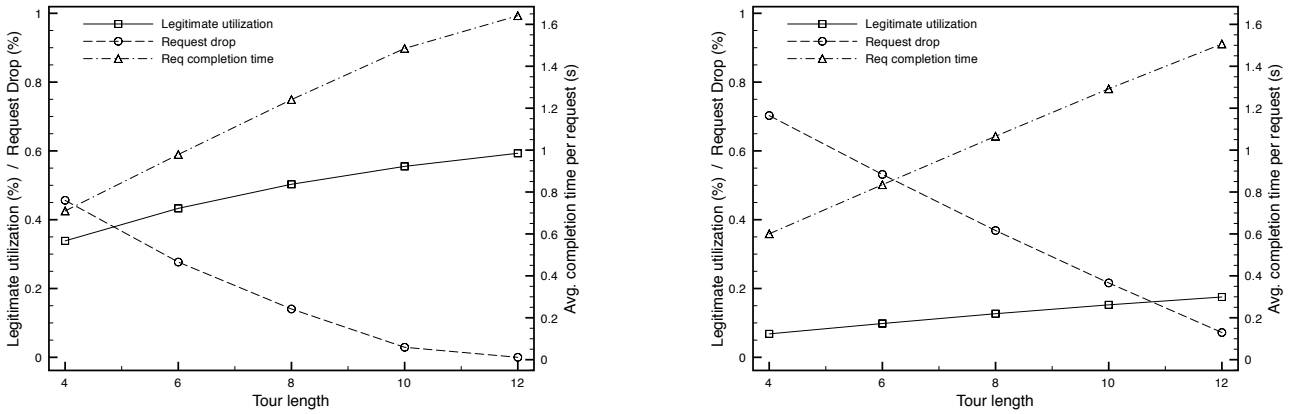
3) Effect of tour length: The tour length in guided tour puzzles is critical for the optimality of the guided tour puzzle defense, especially for the legitimate clients' utilization of server CPU in the case of puzzle solving attacks. The next set of simulation experiments are conducted to measure the effect of tour length on utilization, request completion time, and request drops in the case of puzzle solving attacks. These experiments are conducted using 4 tour guides and 40% and 80% of malicious clients respectively.

The response of various metrics to the change in tour length is illustrated in Figure 4. As the tour length increases, the legitimate utilization of the server (curve "Legitimate utilization") and the request completion time (curve "Req completion time") increase, while the legitimate request drop rate (curve "Request drop") decreases. After increasing the tour length to 12, the legitimate request drop rate becomes zero, and the legitimate utilization of the server becomes optimal in both cases of 40% and 80% malicious clients. Here "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, as well as increases 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 show that choosing the right tour length is important in achieving optimal DDoS mitigation results and providing better trade-off between mutually restricting performance metrics.

VI. Conclusion and Future Work

In this paper, we showed that existing cryptographic puzzle schemes become less effective as the variation in the computational power of clients increases, and that they

 ISBN: 978-1-61208-184-7



(a) 40% clients are malicious, $N = 4$

(b) 80% clients are malicious, $N = 4$

Figure 4. The effect of the tour length on the effectiveness of the guided tour puzzle defense.

require benign clients to perform the same expensive computation that doesn't contribute to any useful work. To this end, we introduced the guided tour puzzle scheme, and showed that it addresses the shortcomings of the existing puzzle schemes and achieves better protection against DDoS attacks. Meanwhile, using extensive simulation studies, we showed that guided tour puzzle is effective in mitigating distributed service flooding 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, we would like to devise an optimal strategy for adjusting the tour length.

REFERENCES

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

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

[3] 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.

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

[5] 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.

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

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

[8] 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.

[9] N. Borisov, "Computational puzzles as sybil defenses," in *the 6th IEEE International Conference on Peer-to-Peer Computing*, 2006, pp. 171–176.

[10] 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.

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

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

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

[14] B. Groza and D. Petrica, "On chained cryptographic puzzles," in *3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, Timisoara, Romania, 2006.

[15] *Secure Hash Standard*, National Institute of Standards and Technology (NIST) Std., 1995.

[16] "About planet lab," Planet Lab. [Online]. Available: http://www.planet-lab.org/about

[17] 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.

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

[19] 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.

[20] 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.

[21] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," in *SIGCOMM '07*, 2007, pp. 289–300.

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

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

[24] 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.

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

 ISBN: 978-1-61208-184-7