Using Rhythmic Nonces for Puzzle-Based DoS Resistance

Ellick M. Chan, Carl A. Gunter, Sonia Jahid,
Evgeni Peryshkin, and Daniel Rebolledo
University of Illinois

ABSTRACT

To protect against replay attacks, many Internet protocols rely on nonces to guarantee freshness. In practice, the server generates these nonces during the initial handshake, but if the server is under attack, resources consumed by managing certain protocols can lead to DoS vulnerabilities. To help alleviate this problem, we propose the concept of *rhythmic nonces*, a cryptographic tool that allows servers to measure request freshness with minimal bookkeeping costs. We explore the impact of this service in the context of a puzzle-based DoS resistance scheme we call "SYN puzzles". Our preliminary results based on mathematical analysis and evaluation of a prototype suggests that our scheme is more resistant than existing techniques.

1. INTRODUCTION

Cryptographic nonces are used to help ensure freshness in many distributed applications. They are widely applied in common Internet protocols to help thwart replay attacks, and hence they play a central role in the security of the network as a whole. Traditionally, such nonces are issued from a service provider to clients as part of the handshake protocol, however, recent resource depletion attacks game the system, as the act of nonce issuance in certain protocols can be costly when a server is under heavy load or attack [6]. In this paper, we postulate the existance of a global nonce broadcast system and explore its applicability to DoS resistance with client puzzles.

Client puzzles have been proposed as a countermeasure to DDoS attacks. When a server is under heavy load, clients whom intend to access services must commit to the transaction first by performing work. The cost of this work acts as a payment and proof of intent: legitimate clients can afford to commit some resources to access a service whereas malicious computers cannot do so without sacrificing their attack capability to a certain extent. Although this approach does not directly solve the problem, it at least increases the number of nodes necessary to launch a successful resource-based DDoS attack, ideally to an unacceptable level for the attacker. Puzzle schemes have evolved over time, but the fundamental concept remains the same: a client makes a request, gets a puzzle, solves it, and sends the result back as part of a hash cash. This means that the puzzle issuance system is itself a potential vulnerability. It is possible to off-load responsibilities to some degree, for instance by creating a special puzzle issuer distinct from the main server, but this puzzle issuer then itself becomes a potential target.

This paper explores the idea of puzzles that can be obtained without a puzzle issuer server *per se*. Instead we postulate the existence of a cryptographic tool called a "rhythmic nonce" which is basically a stream of random numbers broadcast by a secure global source with the property that the numbers are unpredictable but the time intervals between them can be efficiently calculated. The idea is similar to a secure synchronized global timestamp, but focuses on the intervals between nonces rather than absolute times and provides a source of randomness. Such a service does not exist in the Internet now, but services such as DNS and GPS or protocols such as secure multicast could be extended to provide rythmic nonces.

In our approach, clients solve puzzles with parameters harvested from broadcasts of the rhythmic nonces rather than requesting them directly from a puzzle issuer. The server admits new connections based on puzzle cost and freshness which are deduced from the rhythmic nonce used in the puzzle. Salting the rythmic nonce with information about the client and server limits the ability of attackers to share puzzle solutions, and the global nature of the rhythmic nonce stream (which might be available from many sources) limits the effectiveness of an attack on the rhythmic nonces themselves. Our design thus aims to assure that new DoS vulnerabilities are not created by minimizing server-side state and computational costs. We substantiate our claims through a theoretical analysis of the protocol and empirical evaluation which demonstrates that our system is capable of processing requests at about forty times line rate over 100 Mbps ethernet.

CSAW'08, October 31, 2008, Fairfax, Virginia, USA.
.

Our contributions in this work are:

1. The introduction of Rhythmic Nonces.
2. The application of Rhythmic Nonces to puzzle-based DoS countermeasures.
3. The evaluation of a Rhythmic Nonce prototype.

This paper is organized into seven sections. Section 2 overviews the work on puzzles as a DDoS defense mechanism. Section 3 introduces the concept of rhythmic nonces and sketches ideas about their implementation. Section 4 introduces SYN puzzles, which is our application for rhythmic nonces. Sections 5 and 6 provide theoretical and experimental treatments respectively of the SYN puzzles based on rhythmic nonces. Section 7 summarizes conclusions.

2. RELATED WORK

The Internet was designed to forward packets efficiently. However, the finite nature of network bandwidth and server resources sometimes causes lapses in service availability due to congestion resulting from flash flooding, which involves excessive levels of legitimate requests, or Denial-of-Service (DoS) attacks, where malicious attackers attempt to overwhelm the network or the server. DoS attacks can take the form of brute force attacks on network resources or semantic attacks that exploit vulnerabilities of specific protocols, algorithms, or implementations [8, 12]. Effective responses include traceback and filtering techniques and protocol modifications such as SYN cookies. However, the effectiveness of these strategies is diminished by Distributed Denial-of-Service (DDoS) attacks that disguise attack origins by mimicing legitimate flash flood traffic through the use of multiple diverse hosts.

Seminal work involving client puzzles is described in [11, 5] where computational puzzles are used to defend against connection depletion attacks. The most widespread puzzle format is, given a bit string b and a secure hash function h, finding a bit string x such that $h(b||x)$ has a certain number of leading zeros ($||$ denotes concatenation). Since this means that puzzle difficulties can only be powers of two, it has also been proposed to require multiple puzzles, different puzzles of different difficulties, or a hash inversion given a hint.

In most studies, puzzles are generated on-the-fly by the server. For example, the client puzzles proposed in [6] are created by the server as part of an extended TLS handshake. In puzzle auctions [14], clients use the puzzle difficulty to bid for server resources. Clients which solve puzzles of the highest difficulty win the auction to gain prioritized access to server resources. To prevent pre-computation and replay, [4] requires the server to periodically generate a server nonce and send it to interested clients.

Puzzle schemes do not always require processing time as payment and may use memory resources or properties instead: [3] exploits memory latency for memory-based puzzles to provide some degree of fairness for machines of varying computational capability, from low-end handheld devices to high-end servers. The authors in [7] also propose using memory-bound search algorithms to this end.

[15] proposes the use of Diffie-Hellman to outsource puzzles to an external *bastion* (which signs them digitally) in an attempt to protect the overloaded server from extra puzzle traffic, however the bastion itself may become a point of failure. [10] proposes to harvest challenges from existing Internet data sources such as stock prices and RSS feeds to generate puzzles without depending on the server itself. Rhythmic nonces are based on a similar principle, but provide a deliberate strategy rather than attempting to exploit sources for puzzles incidentally. However, the techniques in [10] could be viewed as a practical implementation strategy for rhythmic nonces to the extent the techniques are the same.

All but the last approach requires the server to generate puzzle parameters upon receiving a request. They also require the server to keep some form of state whose size or computational cost ultimately depends on the number of client puzzle requests and therefore leaves the server open to attack. Motivated by these limitations, we propose a technique where clients themselves generate puzzles without involving the server at all. The only task the server must perform is to verify the puzzle solution attached to the client's initial SYN packet.

3. RHYTHMIC NONCES

"Rhythm" comes from a Greek word that means a measured flow or movement. A sequence of random numbers $(u_n) \in A^{\mathbb{N}}$ is a *rhythmic nonce stream* if finding u_{n+1} knowing u_n, u_{n-1}, ..., u_0 is an intractable problem, and if for any natural j there exists an easy-to-calculate function δ_j such that

$$\delta_j(u_n, u_m) = \min(|n - m|, j).$$

Such a sequence is made available using a global secure broadcast at instants $\tau\mathbb{N}$ in such a way that the time interval between successive rhythmic nonces $|\tau(\mathbb{N}+1) - \tau(\mathbb{N})|$ is known to within a stipulated threshold. This interval is the "rhythm" of the nonces; it can be combined with the unpredictability of the nonces to show freshness and as a source of common random numbers. It can be varied according to application, for instance, one stream of rhythmic nonces might be provided in a allegro of milliseconds and another in an andante of seconds.

There are three general strategies for freshness proofs in security protocols, round-trip nonces, timestamps, and sequence numbers. Each of these has similarities and differences compared to rhythmic nonces. Like round-trip nonces, rhythmic nonces are based on unpredictable random numbers, but unlike round-trip nonces they are not intended for a round trip, but rather for general broadcast so they can be shared by all nodes. Like timestamps they are shared by all nodes generally, but unlike timestamps they add a random element and they are not based on a global absolute time but only on a predictable temporal interval between nonces. They

are similar to sequence-numbered passwords, but they are used quite differently since they are released globally on a temporal schedule. Rhythmic nonces could have many security applications and similar ideas have been considered in the cryptography literature. For example, Moran, Shaltiel and Ta-Shma [13] consider the idea of using a broadcast of a random number too long for storage as a basis for non-interactive timestamping. Their aim is to eliminate the need to send messages to a timestamping service when generating a timestamp on a document. Our aim is to eliminate the need to send messages to a puzzle server when providing proof-of-work for a request.

There are a variety of ways to represent the values and interval function for rhythmic nonces. For instance, a stream of consecutive integers could be encrypted under a public key, or a reverse hash stream salted with a secret can also serve the same purpose. In this paper, we examine an implementation of Rhythmic Nonces using repeated encryption. In this context the encryption process is performed by applying a one-way trapdoor function e to the cleartext. Its inverse is a decryption function d, and if we define $u_{n+1} = d(u_n)$ then we obtain a rhythmic nonce. Indeed, knowing u_n, u_{n-1}, and u_0 is equivalent to knowing u_n because $u_{n-k} = e^k(u_n)$ (for $k = 0, 1, \ldots, n$), and in a public-key cryptosystem the problem of finding u_{n+1} knowing $u_n = e(u_{n+1})$ is intractable. The distance functions δ_j can be generated as follows:

$$\begin{aligned}\delta_j(x,y) &= \min(\{n \in [\![0,j]\!] | e^n(x) = y\} \\ &\quad \cup \{n \in [\![0,j]\!] | e^n(y) = x\} \cup \{j\})\end{aligned}$$

These functions are easy to calculate since they only rely on encryption. We may, in some cases, have an incorrect estimation (for example if $e^3(x) = y$ and $e^5(y) = x$) but for sufficiently small values of j, this does not happen very often because if it did it would be a security flaw in the encryption algorithm (as you could basically decrypt with a few successive encryptions).

This result shows that we can use RSA or elliptic curve cryptography to generate the rhythmic nonces and obtain a very high degree of security. Indeed [9] states that a 256-bit ECC key can provide security for the foreseeable future (barring significantly capable quantum computers).

As for the broadcast aspect of rhythmic nonces, there is, of course, a practical challenge to accomplishing this at the scale of the Internet. However, a variety of services already have some or most of what is required. For instance, GPS satellites provide times that are accurate to within nanoseconds; perhaps these services could also provide rhythmic nonces. DNS servers are often cited as a scalable infrastructure for the distribution of virtually anything; using them to supply rhythmic nonces does not seem technically difficult. Real time Seismic data and observations of solar activity [2, 1] can also serve as out of band data sources. Let us add to this a short sketch of an infrastructure based on a gossip protocol as illustrated in Figure 1. The idea is that each router keeps track of the most recent rhythmic nonce and, upon receiving another from an adjacent router, it checks that

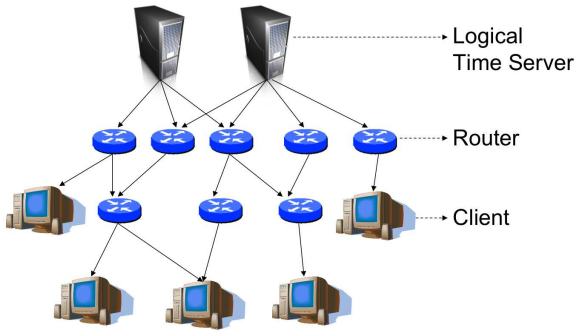

Figure 1: Rhythmic Nonces communicated via router gossip

it is more recent than its own and that it is authentic. In that case, it updates its own rhythmic nonce and sends it on to all other adjacent routers and clients. The rhythmic nonces originate at a small number of root timeservers, analogous to the DNS root servers. While this strategy might seem implausible for deployment in the Internet as a whole, it could be used in conjunction with other mechanisms. For instance, it might be implemented at an enterprise using as its root an enterprise server that connects via some other architecture to the global rhythmic nonce distribution system.

The rhythmic nonce message format we use in our study has the form $(n, u_n, s(\{n, u_n\}))$, representing a sequential identifier n, the nonce u_n and a signature of these two values. Therefore, checking freshness and authenticity is easy, and furthermore this format foils spoofing and replay attempts. It also allows new routers to bootstrap efficiently because, as long as there is a path from a root timeserver and the new router, it will receive the correct rhythmic nonce. Further, given the redundant nature of Internet links, this protocol would be highly resilient to DoS attacks. The root servers cannot be attacked as they receive no traffic and the amount they send is constant.

4. SYN PUZZLES

A proper implementation of client puzzles must take into account several constraints: it must not only ensure that a client cannot forge a puzzle solution without actually solving it, but it must also guarantee that puzzles cannot be solved in advance, that they cannot be easily shared by several clients, and that the puzzle scheme does not introduce any additional DoS vulnerabilities. Since network state (viz. SYN or TCP tables) has been repeatedly exploited to perform DoS attacks, one of our innovations is that we force the client to send the puzzle solution on the very first packet to the server using the TCP SYN packet. This can be contrasted with approaches like [6] where the puzzle exchange occurs in the TLS handshake after the server has already allocated significant resources for the handshake itself. Moreover, the use of SYN puzzles works for any type of TCP link, not just for those running TLS.

To gain access to a protected server, the client must solve a puzzle. Assuming that it knows the required puzzle difficulty k, it listens for the most recent rhythmic nonce t, chooses a random session nonce n, and the task is then to find x such that $h(t||n||x)$ has k leading zeros. As mentioned, although processing costs seem to increase geometrically, we can exert finer control over them by requesting more than one, in the same way we can pay thirty-three dollars with one, two, five and ten dollar bills.

The client then solves the puzzle and sends its parameters and the solution to the server. If the puzzle is valid, has sufficient difficulty, and is fresh enough, it accepts the connection. Naturally, the server decides how recent a puzzle needs to be in order to be considered fresh. The server can also use the session nonce n or a hash thereof to create virtual channels and so thwart replay attacks or to fingerprint botnet clusters.

To discover the puzzle difficulty, the client first sends a puzzle solution of nonzero difficulty. If, by the time the server receives it, the puzzle is valid but the packet is not fresh enough then it replies with a modified reset packet indicating this error condition. If, on the other hand, the puzzle is valid and fresh enough but of insufficient difficulty, the server replies with a modified reset packet indicating the required difficulty for access. This approach limits the use of SYN-puzzle-enabled servers as reflectors (since RST packets are small when compared to SYN+puzzle packets) and, at the same time, avoids any resource commitment on the server side until the client has performed enough work, thereby limiting secondary DoS vulnerabilities.

So far, our scheme prevents cheating (as long as we use a secure hashing function), pre-computation (since puzzles must be relatively fresh), we've been careful not to expose any state that might induce additional DoS vulnerabilities and, as we will see later, experimental results show that a typical server can process SYN+puzzle packets at about forty times line rate over 100 Mbps ethernet. However, our scheme is still vulnerable to replay: a malicious client can send the same puzzle solution in the small window during which it is still fresh. Intuitively, if we make this window smaller, the server becomes less vulnerable to attacks, but this may put clients located further away at a disadvantage (in terms of latency). However, we claim that by carefully choosing the size of the window we can ensure that a large percentage of legitimate clients get access even in an attack scenario.

5. THEORETICAL EVALUATION

We now perform a theoretic evaluation of our scheme, and our goal is threefold: first, we want to know how often we should broadcast the rhythmic nonces; second, we would like to see whether in the absence of DoS attacks a large percentage of users can send their puzzle solutions in time; and finally, we evaluate the same percentage in a DoS attack scenario.

First we will assume, without loss of generality, that rhythmic nonces are sent at instants $\tau\mathbb{N}$ and their sequence numbers start at 0. If a client c contacts a server

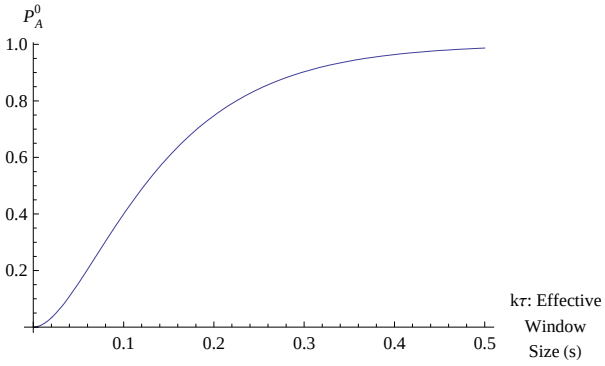

Figure 2: Packet accept rate (P_A^0) without DoS as a function of the effective window $k\tau$

s at time t, he has access to rhythmic nonces up to $\left\lceil \frac{t-l_c}{\tau} \right\rceil$ (seq. number) where l_c is the latency between the machine and the nearest root timeserver. The server receives the solution at time $t + c + l$ where c is the cost (in time) to solve a puzzle and l is the latency between the client and the server. The server's rhythmic nonce is then $\left\lceil \frac{t+c+l-l_s}{\tau} \right\rceil$ and it accepts the connection if

$$\left| \left\lceil \frac{t+c+l-l_s}{\tau} \right\rceil - \left\lceil \frac{t-l_c}{\tau} \right\rceil \right| \leq \eta$$

Where η is called the *window*. This condition is the same if we take t modulo τ, so we can assume that $t \in [0, \tau)$. We now give probability distributions to each of these variables: first it is reasonable in a first approximation analysis to assume that all these variables are independent, that c is constant, that, for a given server, l_s is constant and that t is uniform. Further, we model l and l_c as exponential variables with averages $1/\alpha = 100ms$ and $1/\beta = 50ms$ (as that there are many geographically-spread root timeservers): probabilities, in this context, therefore represent percentages of packets.

With this model, we can give a lower bound to the probability of accepting a packet independently of the (nonzero) window size η:

$$1 - \frac{\alpha^2 e^{-\beta\tau}(1 - e^{-\beta\tau}) - \beta^2 e^{-\alpha\tau}(1 - e^{-\alpha\tau})}{\alpha\beta(\beta - \alpha)\tau}$$

We observe that this function of τ increases and tends towards 1. Now, τ must be as small as possible to allow the server to fine-tune the window, but large enough to maximize the overall number of legitimate packets accepted. Luckily, this expression has an inflection point, so values close to this point are reasonable trade-offs between these two constraints, and we may choose the frequency of the rhythmic nonces as the inverse of this value.

We also proved that, without attacks, the probability that a packet gets accepted by the server is

$$P_A^0 = 1 - \frac{\alpha^2(1 - e^{-\beta})e^{-\beta k} - \beta^2(1 - e^{-\alpha})e^{-\alpha k}}{\alpha\beta(\alpha - \beta)\tau}$$

where we call $k = \eta - p = \eta - \frac{c-l_s}{\tau}$ the (reduced) size of

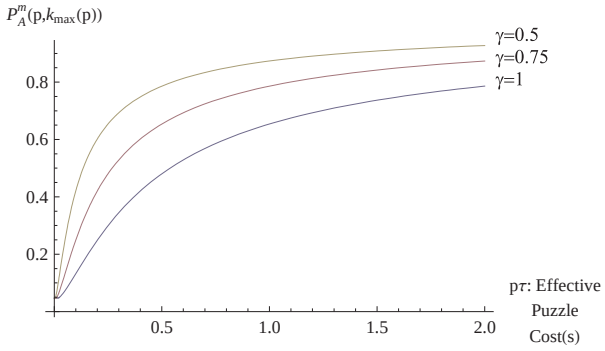

Figure 3: Maximum packet accept rate ($P_A^m(p, k_{\max}(p))$) under DoS as a function of the effective cost $p\tau$ with $\gamma = 1, 0.75, 0.5$

the effective window on the server side. This is plotted in Figure 2). Indeed, we call $p = \frac{c-l_s}{\tau}$ the (reduced) effective cost of the puzzle as seen by a client: even though a puzzle takes c seconds to solve, each client gets bonus time from the fact that the server's clock running l_s seconds late, and we use the word *reduced* because this cost is expressed in units of τ. As we can see, even with a very small window we can ensure that a high percentage of clients get serviced.

Now we determine the efficiency of our scheme under a distributed denial-of-service attack. The scenario is as follows: a botnet of m agents is trying to attack a server and they all have the most up-to-date rhythmic nonce (as this is a worst-case scenario). We assume that the agents cannot cooperate because latencies between them would make this impractical (as real-world botnets are spread all around the world). We now denote c_s the server's capacity, and therefore its maximum number of virtual channels. Of course $m \leq c_s$, as we do not claim to solve DoS attacks that masquerade as flash floods.

Since the client session nonces are mapped to the virtual channels, changing channels means re-computing puzzle solutions and therefore we assume each agent tries to attack a single channel at a time. To do so, each one solves a puzzle and replays it until it expires: this means the attack is successful during a fraction k/p of the time. Therefore, a legitimate packet gets rejected if it would normally be rejected or if it would normally be accepted but arrives at an attacked channel during the attack, so with total probability $(1-P_A^0)+\frac{m}{c_s}\frac{\min(k,p)}{p}P_A^0$. This means the packet gets accepted with probability

$$P_A^m = \left(1 - \frac{m}{c_s}\frac{\min(k,p)}{p}\right) P_A^0$$

We show that, if we view P_A^m as a function of the (reduced) effective puzzle cost p and the (reduced) effective window size k (so $P_A^m(p,k)$), for each value of p there exists a $k = k_{max}(p)$ that maximizes $P_A^m(p, \bullet)$. We now plot these maxima as a function of p for different values of the attacker-to-capacity ratio $\gamma = \frac{m}{c_s}$ in Figure 3.

As we can see, even under very heavy distributed denial-of-service attacks we can ensure that a high percentage of legitimate packets are accepted, and in fact the server can decide how high this percentage should be by choosing higher puzzle costs.

Now, we see that malicious machines can only cooperate by inverting the hash function together. However, if a machine finds an inverse, it has very little time to communicate it to the other nodes in the botnet as the rhythmic nonce is short-lived. In practice, other botnets in the network will only be able to use this inverse to mount an attack if their latencies are small. We have therefore effectively split the botnet into small geographically-localized groups where the potential for cooperation exists. As botnets are distributed throughout the entire world, this considerably slashes their attack capability.

To further inhibit attackers, puzzle solutions may be salted with various parameters to limit their domain. For example, salting the client IP address may help to prevent widescale reuse of a solution. However some clients, such as those behind a NAT firewall or proxy, may not know their IP addresses and will not be able to include this parameter in the request packet. In this situation, request packets fortified with additional IP address information will receive priority access. With this scheme, clients are no worse off than before, since a degraded level of service is provided to certain clients, rather than a total service blackout. Spoofing these request addresses does not pose any additional threat, as the salting mechanism requires an IP attribute match as a precondition, however, a clique of nodes in a botnet can cooperate amongst themselves to solve puzzles for a particular node in the botnet and use that host as a concentrator; this is not distinguished from the case where each bot node individually creates puzzle requests.

If the puzzle protocol omits the server address as a parameter, the same solution can be reused horizontally among multiple servers. To prevent this type of abuse, server attributes must be considered as part of the hash parameters. The use of this technique can help to inhibit the attacker's ability to magnify the attack intensity by constraining the domain of the solution to particular services. Finally, due to the salt characteristics, cooperating botnets form a unique fingerprint that can be used to trace and locate cooperating cliques within the net.

6. EXPERIMENTAL EVALUATION

Our implementation is based on the TCP protocol over IP version 4. Puzzle messages are embedded into the TCP headers, which enables us to withstand connection-oriented attacks like SYN floods or NAPTHA. We distribute simple UDP packets to gossip rhythmic nonces with 96-bit RSA values. We use the TCP options field to transmit the puzzle solution (96 bits), the rhythmic nonce (96 bits), the client nonce (32 bits), the puzzle difficulty k (8 bits), and some other implementation-related bits totaling to 248 bits. We chose these sizes to be able to fit all the data into the TCP options field. Naturally, 96-bit RSA rhythmic nonces are notoriously insecure. Nonetheless, our implementation should be considered a proof of concept as other techniques exist to embed more secure (perhaps

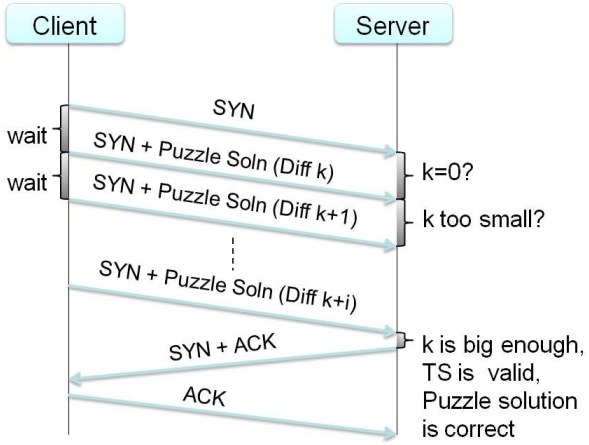

Figure 4: Message sequence chart of protocol negotiation

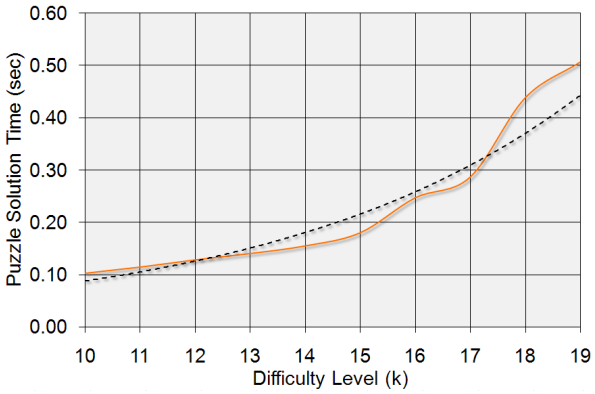

Figure 5: Puzzle solution time is exponentially related to difficulty level

ECC) rhythmic nonces into IPv4 and IPv6 packets.

Our prototype client relies on a user-space TCP stack. Raw IP packets are sent using the sendto() call. The server runs a modified Linux kernel which has support for client puzzles: the networking stack only admits SYN packets that have valid puzzle solutions of sufficient difficulty. When a server is not under load, the server accepts all SYN packets. When under heavy load, the difficulty level is increased by setting a kernel parameter.

The client sends a SYN message to the server with the aforementioned parameters in the options field. We have not yet implemented modified RST packets so we use an auctioning mechanism: the server, if under heavy load, discards unacceptable packets. The client, on the other hand, waits for a timeout period before retrying with a harder puzzle. This goes on until the client receives a SYN+ACK from the server or until the difficulty reaches a threshold. The server, upon receiving a puzzle solution, first checks the difficulty level; if it is sufficient, it then checks the rhythmic nonce for freshness, and if that test succeeds, it checks the puzzle solution. If every check is successful, the server proceeds with the normal connection sequence by sending back a SYN+ACK. The message sequence chart for our protocol is depicted in Figure 4.

As the implementation is still in its early stages, we only have preliminary performance metrics. We attacked our server with a conventional SYN flood over a 100 Mbps LAN. With client puzzles enabled, the server successfully refused connection requests from clients that didn't provide a solution or that provided a weak solution. It only accepted legitimate requests from clients that solved hard enough puzzles. We also verified that solving a puzzle of difficulty k (that is, finding a solution with k leading zeros) takes a time proportional to 2^k (Figure 5), and therefore so does establishing a connection. As mentioned, we can easily have more fine-grained control over this cost by modifying the protocol slightly. Further, we determined that checking a rhythmic nonce for authenticity (at the client or router level) takes in the order of tens of microseconds (32.06 μs with a standard deviation of 53%) on a standard 2 GHz processor, which is an acceptable overhead and does not significantly alter internet latencies; and finally, we were able to determine that it takes on average 9.3 ns to verify a puzzle solution, which means that with our implementation we can process SYN+puzzle packets at line rate (100 Mbps) using only 2.5% of CPU time.

7. CONCLUDING REMARKS

In summary, we have explored the concept of Rhythmic nonces in the context of DoS protection with client puzzles. Our preliminary evaluation of a puzzle scheme using this service suggests that these nonces can help improve certain aspects of DoS resistance that current schemes are unable to effectively address.

Since freshness and replay resistance are desired characteristics for many security protocols, we suspect that the introduction of Rhythmic nonces to the Internet will spur changes to existing protocols in a way that makes them more resilient and efficient.

The techniques we propose in this paper suggest that the inclusion of DoS countermeasures as part of the intrinsic design of the Internet would be beneficial to address today's systemic availability issues regarding the Internet. Furthermore, techniques involving distributed puzzles through systematic outsourcing can help to choke off malicious traffic early on upstream from the originating ISP. ISP-side cooperation to filter and throttle packets based on puzzle attributes can help shape traffic much in the same manner as push-back schemes which are used by current DoS countermeasures.

In conclusion, we have shown that the additional availability of freshness information via a pervasive internet broadcast has many potential benefits to addressing intrinsic availability problems.

Acknowledgements

This work was supported in part by NSF CNS 07-16626, NSF CNS 07-16421, NSF CNS 05-24695, ONR N00014-08-1-0248, NSF CNS 05-24516, DHS 2006-CS-001-000001, and grants from the MacArthur Foundation, Motorola and Boeing Corporation. Ellick Chan and Daniel Rebolledo were supported by a Siebel fellowship. The views expressed are those of the authors only.

8. REFERENCES

[1] National Solar Observatory/Sacremento Peak. Images and Current data. http://nsosp.nso.edu/data.

[2] USGS Earthquake Hazards Program. Latest earthquakes in the world - past 7 days. http://earthquake.usgs.gov/eqcenter /recentqsww/Quakes/quakes_all.php.

[3] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately Hard, Memory-Bound Functions. *ACM Transactions on Internet Technology (TOIT)*, 5(2):299–327, 2005.

[4] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant Authentication with Client Puzzles. *Proceedings of the 8th International Workshop on Security Protocols, Lecture Notes in Computer Science, Cambridge, UK, April*, 2000.

[5] A. Back. Hashcash - A Denial of Service Countermeasure. http://www.hashcash.org/hashcash.pdf, 2002.

[6] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. *Usenix*, 2001.

[7] S. Doshi, F. Monrose, and A. D. Rubin. Efficient Memory Bound Puzzles Using Pattern Databases. In *ACNS*, pages 98–113, 2006.

[8] C. Douligeris and A. Mitrokotsa. *Denial-of-Service Attacks, Network Security: Current Status and Future Directions*. Wiley, 2007.

[9] European Network of Excellence for Cryptology. Ecrypt yearly report on algorithms and keysizes 2006.

[10] J. Halderman and B. Waters. Harvesting Verifiable Challenges from Oblivious Online Sources. *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, 2007.

[11] A. Juels and J. Brainard. Client puzzles: A Cryptographic Countermeasure against Connection Depletion Attacks. *Proceedings of the Network and Distributed System Security Symposium*, pages 151–165, 1999.

[12] J. Mirkovic, J. Martin, and P. Reiher. A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms, 2001.

[13] T. Moran, R. Shaltiel, and A. Ta-Shma. Non-interactive timestamping in the bounded storage model. In *Advances in Cryptology (CRYPTO 04)*, volume 3152 of *Lecture Notes in Computer Science*. Springer, December 2004.

[14] X. Wang and M. Reiter. Defending against Denial-of-Service Attacks with Puzzle Auctions. *IEEE Symposium on Security and Privacy*, 3, 2003.

[15] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS '04*, pages 246–256, 2004.