Computational Puzzles as Sybil Defenses

Nikita Borisov
Department of Electrical & Computer Engineering
University of Illinois at Urbana-Champaign
Email: nikita@uiuc.edu

***Abstract*— We consider the problem of defending against Sybil attacks using computational puzzles. A fundamental difficulty in such defenses is enforcing that puzzle solutions not be reused by attackers over time. We propose a fully decentralized scheme to enforce this by continually distributing locally generated challenges that are then incorporated into the puzzle solutions.**

Our approach consists of an all-to-all broadcast of challenges, with a combining function to ensure this can be done efficiently. The combining function generates certificates that can be used to prove that each node's challenge was delivered to and used by each other node, therefore proving the freshness of each puzzle. We show how our distribution and verification mechanisms can be implemented on top of the the Chord [21] overlay.

I. Introduction

Peer-to-peer research has created an architecture for creating completely decentralized distributed systems with large numbers of participants. There are many applications where a centralized point of control is undesirable, whether it be because of a trust issue, such as with anonymous networks, or for emerging applications where a centralized point has not yet been established. However, the lack of a central point of makes it difficult to control admission to the network and therefore enables the Sybil attack [7]. Because the cost of participating in a peer-to-peer network is typically low, it is easy for an attacker to create a large number of phantom nodes, potentially overwhelming the membership of the peer-to-peer network and allowing abuse of or attacks on the network.

Sybil attacks are not unique to peer-to-peer networks; other contexts where the low cost of using a service can lead to abuse are email (spam) and web services (denial of service attacks). A common solution explored in all of these cases is artificially raising the cost of using a service, through computation (client puzzles [13]), human time (CAPTCHAs [22]), or even through imposing direct financial costs on participation. These solutions have been more effective in non-p2p contexts because of the difficulty of imposing this cost in a fully decentralized system. It is impractical to offer financial compensation to or perform a CAPTCHA for every one of the thousands peer-to-peer node you interact with.

This problem could be solved with a level of indirection, using a centralized service that issues membership certificates while imposing some cost on the participants. Such a service, however, must be scalable, reliable, and trustworthy. Part of the success of p2p networks has relied precisely on avoiding such centralized points — a decentralized, self-organizing system is much easier to deploy, and decentralized trust is essential to some applications of p2p overlays, such as anonymous networks [19], [9].

We propose a completely decentralized Sybil solution involving computational puzzles. Such puzzles involve posing a challenge that requires a large amount of computation to solve but is easy to verify. The difficulty in using puzzles in a decentralized context is in the process of issuing challenges. A node could issue a challenge before it first communicates with a node, but that would introduce undue latency; as well, the cost imposed on attackers is still only proportional to the amount of legitimate traffic, rather than the number of phantom nodes. On the other hand, it is impractical to issue a challenge to potential communication partners *before* communicating with them, since the set of potential partners can be as large as the whole network.

Our solution involves an efficient broadcast of a challenge from each peer-to-peer node to each other node. By combining challenges as they are broadcast, the overhead for this all-to-all communication can be accomplished with only $O(k)$ overhead, where k is the degree of the node. The combination operation itself is verifiable; the proof that a node's challenge was included in the puzzle answered by a node is distributed along the communication path towards that node, so it can naturally and efficiently be collected and verified any time a communication between two nodes is required.

II. Background

A. Computational Puzzles

The idea of using computation to restrict abuse dates back to Naor and Dwork [8], who suggested using it to fight spam. An email sender would take a hash of a message contents, as well as the address of the sender and destination, and then feed it as input to a a *moderately hard function* (or a "pricing function"): a function that is (moderately) difficult to compute but the result of which is easy to verify. The output of the function would be included with the message. An email recipient would only accept a message upon verifying the result of the pricing function. In this way, a new pricing function computation would need to be performed per each individual message, imposing a minimal cost on legitimate email users but a heavy one on spammers.

A similar scheme has been used to prevent denial of service. Juels and Brainard [13] suggested issuing client puzzles to anyone requesting service and requiring a solution before the service is provided. The puzzle consists of a random input to a pricing function, so client puzzles are effectively an interactive equivalent of the non-interactive email puzzle protocol.

B. Sybil Attacks

A Sybil attack [7] consists of an attacker introducing a large number of phantom nodes into a network. Without centralized admission control, it is difficult to distinguish multiple nodes operated by a single attacker from several independent nodes. Since the cost of participation in a p2p network is usually low, resourceful attackers can introduce enough phantom nodes such that a very large fraction of all nodes belong to them. This can be used for denial of service or other abuse of the network.

Of course, denial of service can still be carried out even if the number of malicious nodes are limited. However, in this case, effective countermeasures can be used: Castro *et al* [4] describe a set of secure routing primitives that are able to significantly improve resilience to DoS attacks in structured P2P networks. They advocate a certificate-based approach for restricting participation, wherein each node must obtain a certificate from a central authority before it is allowed to join the p2p network.

The certificate is used to both prove that a node is allowed to participate and to assign it a unique *position* within the Pastry structured overlay [20]. The positions are then used to verify that routes and connections are made according to the Pastry algorithms and not directed towards malicious nodes. If misbehavior is suspected, several messages are sent along a diverse set of routes to ensure delivery in the face of further attacks.

C. Precomputation

Our approach is to impose a computational cost on occupying a position within the overlay. This will limit the number of positions that attackers are able to compromise, and hence enable some of the mechanisms developed in [4] to be effective without any centralized requirements.

Castro *et al* in fact consider one simple computational puzzle approach. They suggest that participants in the network generate a public/private key pair such that the SHA1 hash of the public key has p "0" bits at the end. The work required to find such a pair is $O(2^p)$. A node's position is then determined by taking another hash of the public key. It is easy for participants to prove that they have done the work to occupy a particular position simply by presenting the public key and authenticating a message with the corresponding private key.

This approach has the disadvantage that it is amenable to precomputation. Attackers can continually generate new key pairs of the appropriate form to gain an increasing number of positions. Over time, an arbitrary fraction of nodes will belong to attackers.

Note that client puzzles used to prevent denial of service do not suffer from this problem, because each server transaction requires solving a new puzzle, created by the server. Castro *et al* suggest that this could be imitated by using a public source of randomness to generate a new challenge every time period. However, such a source would be both a centralized point of trust — predicting the random outputs could be used for precomputation — and failure. In particular, a targeted denial-of-service attack against the randomness source itself could be used to disable the network. Our contribution consists of a way to use the overlay itself to create this source of randomness

D. Limitations

Before we discuss our approach, we want to address a common criticism of the computational puzzle approach. The approach breaks down when there is a large disparity between the computational ability of would-be attackers and legitimate users. If, for example, some legitimate users are stuck with out-of-date hardware, while attackers have access to high-performance computing clusters, the attackers may be able to solve puzzles several orders of magnitude faster than legitimate users. Although the hardness of puzzles is easily parameterizable, any choice of difficulty will either restrict legitimate users too much or not block attackers enough under such a scenario.

We do not hope to solve this fundamental problem here, and as such our solution is ultimately limited. However, we believe that for a wide range of applications, a puzzle approach is appropriate. With a large enough number of users, even a small computational requirement can stop all but the most resourceful of attackers from attaining a significant fraction of nodes in the network. In medium-sized networks, the costs of mounting Sybil attacks will still be increased out of the reach of casual attackers, improving the resilience of the p2p network. And proxies can help severely computationally constrained users participate in the network.

Finally, any defense against Sybil attacks hits a fundamental limit if the amount of resources attackers are willing to expend exceeds those of legitimate users. For example, attackers could pay people to donate computing time, obtain certificates, or perform CAPTCHAs. A proper comparison of the efficacy of Sybil defenses will depend on a detailed cost analysis — an important question, but one we leave for future research. Our goal in this paper is to demonstrate the technical feasibility of a computation-based Sybil defense.

III. Our Scheme

The key to our design is a mechanism to create random challenges based on inputs from all the participants in the p2p network. The challenges are used as input to computational puzzles that are required to participate in the network. Whenever a node A contacts another node B, A can verify both that B has solved the puzzle and that A's random input has been included in the computation. This way, A is sure that B has recently expended some amount of computation in order to secure its participation in the network.

A. Overlay Basis

Our design is intended to function in concert with the secure routing primitives by Castro *et al* [4]. Their description is based on the Pastry structured overlay [20], but the concepts can easily be translated to other overlay types. Most of our mechanisms similarly translate naturally to common types of overlays; for concreteness, we will use Chord [21] as an example and we will discuss other overlay choices below.

Briefly, each node in Chord is assigned an identifier between 0 and $2^k - 1$, where $k = 128$ or $k = 160$. Chord uses the concept of a *successor* node: for $x \in [0, 2^k - 1]$, $\mathrm{succ}(x)$ is the node with the next highest identifier larger than x (wrapping around at 0). A node with identifier y has neighbor links (or *fingers*) to the successors of $y + 2^i \pmod{2^k}$ for $i = 0, \ldots, k-1$. In a network of N nodes, only about $O(\log N)$ of these fingers will be distinct. Routing to a destination z proceeds by recursively contacting the finger whose identifier is closest to z, using the distance metric of $z - y \pmod{2^k}$. Such routing is expected to take $O(\log N)$ hops to reach any destination.

B. Distributing Challenges

To maintain the structure of the distributed overlay, each node in Chord will periodically send a ping to each of its neighbors, to ensure they are still available. We modify this ping message to include two numbers: each node A will send a sequence number $n^{(A)}$ and a challenge $c_{n^{(A)}}$ in each message. The sequence number increases with each message, while the challenge is calculated as follows:

Let $n^{(B_1)}, \ldots, n^{(B_m)}$ be the sequence numbers from the last ping message received from each of the node A's m neighbors, and let $c_{n^{(B_1)}}, \ldots, c_{n^{(B_m)}}$ be the corresponding challenges. Then:

$$\begin{aligned} c_{n^{(A)}} &= H(B_1 || n^{(B_1)} || c_{n^{(B_1)}} || \ldots \\ &\quad || B_m || n^{(B_m)} || c_{n^{(B_m)}} || r_{n^{(A)}} || c_{n^{(A)}-1}) \end{aligned}$$

where $H()$ is the SHA1 hash function [18], $||$ represents concatenation, and $r_{n^{(A)}}$ is a random number chosen by node A. The node records $c_{n^{(A)}}$, $r_{n^{(A)}}$ and $n^{(B_i)}, c_{n^{(B_i)}}$ for each i in a table $State_{n^{(A)}}$. The table only needs to keep the last $2t + d$ entries; the values t and d will be discussed below.

The challenge that A sends out is based on the challenges it received from all of its neighbors. Similarly, in turn, A's neighbors' challenges in the next round of pings they send out will be based on A's challenge, and so on. In this way, A's challenge will flood the entire overlay through ping messages. The diameter of Chord networks is $O(\log N)$, so if a ping is sent every second, all nodes' challenges will be based on A's challenge in less than a minute for any reasonable network size.

C. Challenge Verification

The purpose of using a cryptographic hash function is to confirm that a challenge was computed correctly. If A's neighbor B_i wants to confirm that $c_{n^{(B_i)}}$ was included in the computation of $c_{n^{(A)}}$, A can present the values $B_j, n^{(B_j)}, c_{n^{(B_j)}}$, for all its other neighbors, as well as $c_{n^{(A)}-1}$ and $r_{n^{(A)}}$, as recorded in $State_{n^{(A)})}$, and B_i can confirm that the computation was performed correctly. The property of the SHA1 hash function ensures that A could not have computed $c_{n^{(A)}}$ before learning $c_{n^{(B_i)}}$, since, given c and h, it is infeasible to find strings a and b such that $H(a||c||b) = h$. Therefore, the hash function enforces a verifiable temporal order on the challenges.

Once B_i learns $c_{n^{(A)}}$ and verifies that $c_{n^{(B_i)}}$ was included in its computation, it can then contact one of A's neighbors, C, and verify that $c_{n^{(C)}}$ is based on $c_{n^{(A)}}$, which is in turn based on $c_{n^{(B_i)}}$ using the same proof procedure. It can then iteratively follow the overlay links and for each node obtain a temporal link between its challenge and the current challenge of any node.

Note that B_i it may need to pick an older challenge to start with, since its newer challenges may not have propagated to the entire network. If we expect the diameter of the network to be about d, then B_i can begin with $c_{n^{(B_i)}-d}$. Then A would look up $c_{n^{(B_i)}-d}$ in one of its $State$ records, and return a corresponding $c_{n^{(A)}-l}$ challenge with a proof that $c_{n^{(B_i)}-d}$ was included in it. It would then query C about $c_{n^{(A)}-l}$, and so on. If B_i sends a new ping every second, it would know that for any node X that it reached this way, the challenge $c_{n^{(X)}-l'}$ was computed at most d seconds ago.

Whenever a node B_i routes a message, it will first contact one of its neighbors, say X_1, then one of X_1's neighbors, say X_2, and so on. At each step of the route, X_j, B_i can verify that its challenge $c_{n^{(B_i)}-d}$ was included in $c_{n^{(X_j)}-l_j}$ for some l_j. We will next use this ability to impose computational puzzles on nodes.

D. Computing Puzzles

Every t time steps, each node will compute a puzzle based on the current value of $n^{(A)}$ and $c_{n^{(A)}}$. The puzzle consists of finding a value $s_{n^{(A)}}$ such that:

$$H(ID_A || PK_A || n^{(A)} || c_{n^{(A)}} || s_{n^{(A)}}) = h$$

with the last p bits of h equal to 0, where id_A is the Chord identifier assigned to node A, and PK_A is its public key. To solve this puzzle, A needs to perform $O(2^p)$ hash function evaluations with random values for $s_{n^{(A)}}$; by changing the parameter p, the amount of work A needs to perform can be varied arbitrarily. The main constraint on p is that the slowest node that wants to participate in the network should be able to compute $O(2^p)$ hashes in fewer than t time steps.

Using the mechanism from the last subsection, whenever a node X routes a message to or through node Y, it can find out a temporal relationship between $c_{n^{(X)}-d}$ and $c_{n^{(Y)}-l}$ for some l. By starting with $c_{n^{(X)}-d-2t}$ instead, X can find some $c_{n^{(Y)}-l'}$ where $l' \geq 2t$. Let $n^{(Y)} - q$ be the sequence number of the last challenge that node Y used for a puzzle, and $n' = n^{(Y)} - q - t$ be the previous one. Since it takes at most t time steps to solve a puzzle, the puzzle based on $c_{n'}$ must have already been solved and Y can present the solution, $s_{n'}$. Further, since $q < t$, Y can show that $c_{n'}$ is based on $c_{n^{(Y)}-l'}$, since:

$$c_{n'} = H(x_1 || H(x_2 || \ldots || H(x_{l'-t-q} || c_{n^{(Y)}-l'}) \ldots))$$

for some strings x_i. So X can be assured that Y has performed $O(2^p)$ hash function evaluations in the last $2t + d$ time steps.

Since the puzzle solution incorporates both ID_Y and PK_Y, X can be sure that some node with public key PK_Y has solved a puzzle allowing it to occupy position ID_Y. Therefore, the corresponding private key can be used to authenticate messages, such as those used in the routing failure test in [4]. Since a different puzzle is used for every identifier, the amount of work Sybil attackers have to do is proportional to the number of positions they try to occupy. And the challenge distribution and verification mechanisms ensure that they cannot use precomputation to build up solutions and occupy more positions.

IV. Discussion

A. Dealing with Churn

The distribution and verification mechanisms described so far assume a static network topology, where message routes follow the same links as the pings that precede them. In reality, the overlay membership and structure may have changed in the last $2t+d$ time steps, and the route may take a link $C \rightarrow D$ that has only recently appeared. In this case, it will be impossible to relate $c_{n(C)-l}$ and $c_{n(D)-l'}$ except for very small values of l' (namely, $l' < 2t$).

We first deal with additions to the network. When a new node C joins, other nodes will change some of their fingers to point to it. We modify the neighbor maintenance algorithm such that the old fingers that would be replaced with links to C are maintained for $2t + d$ time steps. During that time, any queries that would normally be sent to C are sent along the old link, since there is a temporal relationship between the old challenge values that is necessary to verify puzzle computations.

To deal with departing nodes, we use the the *flexible route selection* and *static resilience* properties of Chord [10]. While the normal routing algorithm in Chord uses the finger link that is closest to the destination for the next hop, it is possible to choose a different finger, as long as it still makes progress towards the destination. Therefore, the routes in Chord are *flexible*, and this kind of selection of non-optimal next hops has only a minimal effect on the average lengths of the routes. Whenever the optimal hop from a node D for destination y has left the network in the last $2t+d$ steps, we cannot use the replacement finger to forward the query since it is too new. However, we can simply use the next best neighbor link to follow.

How well does this work? We can answer this question by measuring the *static resilience* of Chord. Static resilience describes the percentage of queries that could be successfully routed if some fraction of nodes failed and no repair action was taken. This corresponds exactly to the scenario of departed nodes, since any repair actions taken in the last $2t + d$ steps cannot be used. Chord has been shown to have a very high level of static resilience [10]: even with 50% of unrepaired node failures, flexible routing can successfully repair all but 20% of the queries. This is our main motivation for using the Chord overlay as the basis of our system; other overlays could be used, but would result in a higher percentage of undelivered queries.

Of course, the restriction that we cannot use any of the links established over the last $2t + d$ time steps introduces inefficiencies into the operation of the overlay. However, if the p2p overlay stores self-certifying data, and if most queries are expected to succeed, we can first send a query on the most efficient path and not verifying puzzle solutions. In the rare case that the data is not found, the full scheme using older links can be used to verify this fact, similar to the two-level solution in [4].

B. Choice of t

The restriction that links formed in the last $2t + d$ time steps cannot be used imposes a limit on the length of t that is reasonable. Peer-to-peer networks have been shown to have mean node lifetimes on the order of one hour, so a reasonable choice for t may be 3 minutes, so that during $2t + d$ time steps, about 10% churn is experienced. This results in a small number of unroutable queries, due to the static resilience of Chord.

We recommend against setting t much lower, since otherwise attackers may be able to solve the puzzles online in the process of answering a query. This could give attackers a significant advantage in networks where the query volume is much lower than the number of nodes in a system. An attacker could only compute those challenges that correspond to a query and still occupy an arbitrary number of positions in the network. With longer puzzles, the attackers are forced to commit to their positions before queries are issued and therefore have less flexibility in their attack strategy.

C. Position Flexibility

A certificate, as described in [4], fixes the position that a node can occupy in the network. Our scheme, in contrast, gives a node flexibility to chose its position, and even to switch it from time to time; as a result, our scheme is compatible with load-balancing algorithms.

A random allocation of nodes to positions results in some variation in the distance between two successive nodes. In networks such as Chord, such distance will vary by a factor of $O(\log^2 n)$ [17]. Balancing algorithms [2], [17], [1], [14], [16] can reduce this figure to $O(1)$, sometimes as small as 2 [16].

The primary motivation for using these algorithms is to balance load on each p2p node, which is proportional to the distance to the next node. An imbalance in the distribution of nodes can also result in less efficient routing; a load-balancing algorithm could shrink the diameter of the p2p overlay and therefore we could use a smaller d value in our scheme. Load balancing can also improve the performance of the density test used to determine whether routing was successful [4], reducing the number of false negatives and positives for such a test.

Some load-balancing algorithms [17], [1], [14] are based on assigning $O(\log n)$ random positions to each node and having the node switch between them to maintain balance.

Such algorithms could be compatible with a certificate-based Sybil defense, as each certificate could represent the right to occupy one of $O(\log n)$ positions, rather than a single one. However, this would allow attackers to introduce a factor of $O(\log n)$ more nodes than certificates; also, the most efficient load-balancing algorithms [16] rely on explicitly assigning positions to nodes.

Position flexibility in our scheme does make it easier for attackers to *selectively* deny service to a certain position in the network by occupying all locations around that position. Selective DoS is probably most relevant to p2p publishing systems, where attackers may wish to block a particular document. In other systems that use unpredictable, random looking keys, attackers will have a hard time picking a position to attack. Even with a certificate-based system, selective denial of service can be carried out by directly attacking the node at a particular position; techniques that hide the correspondence of nodes and positions may be used to make such attacks more difficult [12], [5].

V. Related Work

A potential alternative to address Sybil attacks is to use IP addresses as a scarce resource. However, it is sometimes possible for attackers with access to /A or /B network to obtain a large number of addresses; conversely, many users behind network address translation gateways share the same IP address. These issues have been explored in the special case of anonymous p2p networks in MorphMix [19] and Tarzan [9] anonymous networks. These designs ensure that any network path traverses IP addresses from different /16 prefixes, while allowing in principle participation of any number of nodes from a given prefix, or even a particular IP address, but they do not easily generalize to blocking Sybil attacks on generic distributed hash tables or other p2p overlays.

Danezis *et al* propose to use social information to address Sybil attacks [6]. Their design uses a "bootstrap" graph that maintains a record of which node introduced each node to the network. The intuition is that this graph reflects the out-of-band social relationships between nodes in the network and provides a diverse set of paths that can be used to reach most nodes. Their algorithms survive even a majority of Sybil nodes, but the routing efficiency depends on the shape of the bootstrap graph and is linear in the number of nodes in the worst case.

Our approach of using computational puzzles as Sybil defenses was inspired by the ongoing work by Halderman and Waters that aims to build random challenges using public sources of randomness [11].

There have been a number of application of computational puzzles [3], [8], [13]. Perhaps the most closely related to our work is outsourcing puzzle creation to bastions [23]. As in our scheme, bastions allow clients to solve puzzles *before* contacting a service, and this puzzle solution can later be verified. The bastions, however, have to be a trusted source of randomness, and the cryptographic protocols for generating puzzles based on this source are significantly more complex than ours.

Our mechanism for spreading challenges parallels the tools used in secure timeline entanglement [15], which also use hashes to prove a partial order of events in a distributed systems.

VI. Conclusion

We have described a fully decentralized mechanism to defend against Sybil attacks in structured p2p overlays based on computational puzzles. We showed how to address the problem of puzzle precomputation through distributing fresh challenges to nodes in the network. Our mechanism allows each participant to verify that its challenge was delivered to each other participant and therefore locally verify the freshness of the puzzle solution. Our solution can be combined with the secure routing primitives from [4] to create a p2p network resilient to denial of service attacks without any point of centralization.

VII. Acknowledgments

We would like to thank Alex Halderman and Brent Waters for motivating our work in this area and for helpful discussions about computational puzzles and peer-to-peer networks. We would also like to thank the anonymous referees for their helpful suggestions.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2003.

[2] M. Adler, E. Halperin, R.M. Karp, and V.V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003.

[3] A. Back. Hashcash — a denial-of-service countermeasure, 2002. http://www.hascash.org/hashcash.pdf. Original system developed in 1997.

[4] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, December 2002.

[5] G. Ciaccio. Recipient anonymity in a structured overlay. In *International Conference on Web Applications and Services*, 2006.

[6] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoekk, and R. Anderson. Sybil-resistant DHT routing. In *10th European Symposium on Research in Computer Security*, 2005.

[7] John Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop*, March 2002.

[8] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.

[9] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[10] Krishna P. Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of ACM SIGCOMM 2003*, August 2003.

[11] J.A. Halderman and B. Waters. Personal communication. 2005.

[12] Steven Hazel and Brandown Wiley. Achord: A variant of the Chord lookup service for use in censorhip resistant peer-to-peer publishing systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[13] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[14] D.R. Karger and M. Ruhl. New algorithms for load balancing in peer-to-peer systems. In *IRIS Student Workshop*, 2003.

[15] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *USENIX Security Symposium*, 2002.

[16] G. S. Manku. Balanced binary trees for ID management and load balance in distributed hash tables. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, 2004.

[17] M. Naor and U. Wieder. Novel architectures for P2P applications: The continuous-discrete approach. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[18] National Institute of Standards and Technology. Secure hash standard (SHS). Federal Information Processing Standards Publication 180-1, April 1995.

[19] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, November 2001.

[21] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, August 2001.

[22] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Eurocrypt*, 2003.

[23] B. Waters, A. Juels, J.A. Halderman, and E.W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *ACM Conference on Computer and Communications Security*, 2004.