

# Impeding Attrition Attacks in P2P Systems

Petros Maniatis

Intel Research, Berkeley, CA

TJ Giuli

Stanford University, CA

Mema Roussopoulos

Harvard University, Cambridge, MA

David S. H. Rosenthal

Stanford University Libraries, CA

Mary Baker

HP Labs, Palo Alto, CA

**Abstract**—P2P systems are exposed to an unusually broad range of attacks. These include a spectrum of denial-of-service, or *attrition*, attacks from low-level packet flooding to high-level abuse of the peer communication protocol. We identify a set of defenses that systems can deploy against such attacks and potential synergies among them. We illustrate the application of these defenses in the context of the LOCKSS digital preservation system.

## 1. INTRODUCTION

Peer-to-peer (P2P) systems are exposed to an unusually broad range of attacks because of their lack of central control or administration. In earlier work [20], we classify these attacks according to their intent and describe techniques for thwarting some of them. Here we focus on attacks aimed at *Denial of Service* (DoS), in the broad sense of the term introduced by Needham [22]. More recently, Denial of Service has come to mean high-bit-rate network-level attacks such as SYN-flooding [7] that rapidly degrade the usefulness of the victim system. We use the term *attrition* to indicate our equal concern with moderate- or low-bit-rate application-level attacks that gradually waste the victim system’s resources over a long period, impairing its function.

The contribution of this work is to bring together a broad range of techniques, none wholly original, that can help to resist attrition attacks on P2P systems, identify synergies among them, and describe how they can be implemented in the context of a real P2P application with promising preliminary results.

## 2. THE ADVERSARY

We classify adversaries according to the intent of their attack on the victim system into the following categories:

- *Stealth* adversaries attempt to modify, subvert or otherwise compromise the integrity of the content or service of the system undetected. In a file system, this adversary would seek to modify files unobtrusively without authorization.
- *Theft* adversaries attempt to access restricted system services. In a file system, this adversary would seek to read restricted files without authorization.
- *Nuisance* adversaries attempt to cause many apparently false alarms to discredit intrusion detection and monitoring systems.
- *Free-loader* adversaries attempt to benefit from the system’s services while contributing nothing in return. In a file system, a free-loader would use up the disk space of others while refusing to make his own space available.

- *Attrition* adversaries attempt to prevent clients of the system from obtaining timely service.

Here we focus on the attrition adversary, whose goal may be to consume resources at peers in general so as to reduce the usefulness of the system as a whole, or to consume resources at individual but critical peers, either to prevent them from contributing to the system or to facilitate another attack. To these ends, the adversary can consume or cause waste of network-level resources (bandwidth, buffer space, connection descriptors), and application-level resources (computation, memory).

The attrition adversary has three potential “modes” of operation representing increasing levels of sophistication in exploiting the victim system’s protocol:

- *Pipe stoppage*. Through massive traffic abuse, the adversary saturates the victim peers’ network connections, preventing them from sending or receiving valid protocol messages. We assume this mode of operation to be sustainable for short time periods, on the order of hours.
- *Anomalously high rates of requests*. With more understanding of the target protocol, an attrition attacker can send well-formed requests to victims at a rate that, while not saturating any network connections, causes resource exhaustion as the victims try to respond. Such obviously anomalous traffic rates can lead to the identification of the attack sources and eventually stem the tide within days (e.g., with packet marking and subsequent filtering).
- *Seemingly innocuous rates of requests*. The adversary sends requests at rates no greater than the highest expected from other loyal peers in the absence of an attack. Through even more understanding of the target protocol, these requests are crafted and timed to exhaust the victims’ resources or to prevent them from contributing to the system. We must assume that such a mode of adversary operation is sustainable almost indefinitely, absent other tell-tale signs of anomalous behavior.

The next section demonstrates our strategy: persevere during short-term pipe stoppage, “discourage” the adversary from using high-rate attacks by making them less effective than low-rate ones, and rely on more sophisticated defenses for raising the cost of low-rate attacks.

## 3. ATTRITION DEFENSES

We describe general defenses against attrition that we have found useful. In Section 4, we apply these defenses to a real system that preserves access to web-published documents.

*Effort Balancing.* If the effort needed by a requester to procure a service from a supplier is less than the effort needed by the supplier to furnish the requested service, then the system can be vulnerable to an attrition attack that consists simply of large numbers of ostensibly valid service requests. We can use provable effort mechanisms such as Memory-Bound Functions [12] to inflate the cost of relatively “cheap” protocol operations by an adjustable amount of provably performed but otherwise useless effort. By requiring that at each stage of a multi-step protocol exchange the requester has invested more effort in the exchange than the supplier, we raise the cost of an attrition strategy that defects part-way through the exchange.

This effort balancing is applicable not only to consumed resources such as computations performed, memory bandwidth used or storage occupied, but also to resource commitments. For example, if an adversary peer issues a cheap request for service and then defects, he can cause the supplier to commit resources that are not actually used and are only released after a timeout (e.g., SYN floods [7]). The size of the provable effort required in a resource reservation request should reflect the amount of effort that could be performed by the supplier with the resources reserved for the request.

*Rate Limitation.* A peer can use its resources judiciously and slow down attacks if its decisions about participation are autonomous and free from external coercion. Peers should satisfy requests *no faster than necessary* rather than *as fast as possible*; for example, this policy can effectively slow the spread of viruses [30]. A peer can follow this policy by maintaining a fixed rate at which it requests services from others or supplies services to others, linking the two through reciprocation when the application domain allows. This policy is easier to apply when requests carry expiration times; peers can make informed choices of whether or when to supply the requested service. The policy is also easier to follow in applications without bursty communication patterns during regular operation, either thanks to statistical multiplexing or by definition; for example, auditing applications usually operate on fixed periodic schedules that can, collectively, offer good estimates of the expected work rate per peer.

*Admission Control.* Rate limitation implies that a peer must reject or even drop unacknowledged some incoming requests. Ideally, a strategy for doing so discriminates against the attrition adversary by selectively dropping his requests. Strategies that have been used in similar circumstances include random drops (which, unfortunately, tend to penalize legitimate requesters during a DoS attack), session-based classification [6], and reputation-based classification [14].

*Redundancy.* If an individual peer is essential to the function of the overall system, then the attrition adversary can focus attacks on that peer, for example by flooding its network connection. Otherwise, when an outage at an individual peer affects only that peer but not the system as a whole, the adversary must attack a large proportion of the peers simultaneously. This can be arranged using self-healing overlays (e.g., in SOS [19]) and voting mechanisms (e.g., BFT [5] and LOCKSS [20]).

*Compliance Enforcement.* There is usually some cost for a legitimate requester to process the result of its request, which an adversary would like to avoid. It is possible for a requester

to prove to the supplier that the operation for which the request was made has actually been performed via an unforgeable *evaluation receipt*. Proving the requester’s compliance with the behavior expected of it in this way has similarities with “uncheatable computations” [17].

*Desynchronization.* Sometimes a peer requesting a service must find more than one peer simultaneously available to supply that service (e.g., in a read-one-write-many fault-tolerant system [5], [25]). Sometimes multiple peers inadvertently synchronize (e.g., TCP sender windows at bottleneck routers, clients waiting for a busy server, and periodic routing messages [15]). When this happens, even absent an attack, moderate levels of peer busyness can prevent the system from delivering services. An attacker in this situation may benefit by increasing peer busyness only slightly (see Section 4.1). P2P system designers should only opt for synchrony if it is necessary; accidental synchrony should be prevented by randomization, back-off, turn-taking, etc.

#### 4. A CONCRETE EXAMPLE: LOCKSS

To illustrate these defenses, we describe them in the context of the LOCKSS<sup>1</sup> digital preservation system, originally described in [24]. Briefly, LOCKSS peers cooperate to audit their copy of an on-line document replicated at many peers by voting in “opinion polls.” A *poller* invites a sample of the peer population into a poll, in which each invitee individually produces a vote on the contents of that document. An invitee votes on a poll by cryptographically hashing a poller-supplied random challenge with its local copy of the audited document and returning the resulting hash value. The poller tallies these votes and if its copy agrees with almost all votes, the poller declares success and schedules another audit in the future. If, however, the poller’s copy disagrees with almost all votes, it assumes its copy is corrupt; to repair, the poller obtains the copy of one of the disagreeing voters and reevaluates all previously received votes so as to verify that, with that obtained copy, it agrees with most of the votes. If the outcome of the poll after the tally is inconclusive — for example, the split between votes that disagree and those that agree with the poller is close to 50-50 — then the poller raises an inconclusive poll alarm indicating malicious activity, and requires the help of a human operator to resolve the situation.

Peers decline invitations if they lack the resources to vote, for example if they are busy voting in another poll or are running their own poll. Pollers only use the results of a poll if that poll obtains at least a minimum quorum of voters, typically 10; they discard results of inquorate polls. Communication between a poller and each of its voters is protected against eavesdropping and tampering via encryption, using a symmetric session key derived from an anonymous Diffie-Hellman key exchange.

Unfortunately, the original protocol implementing this simple concept [24] is vulnerable to many attacks [21]. In subsequent work [20], we defend against the stealth and nuisance adversaries described in Section 2 with a more complex series of exchanges between a poller and each participating voter. Using *Rate Limitation*, *Effort Balancing* and *Redundancy*, we render

<sup>1</sup>Lots Of Copies Keep Stuff Safe.

likely attack strategies by these adversaries ineffective. Here we outline an improved protocol design that addresses attrition attacks while retaining resistance to the stealth and nuisance adversaries we have studied before.

#### 4.1 Vulnerabilities

We identify a number of ways in which the LOCKSS protocol as described in [20] is vulnerable to the attrition adversary.

*Defection.* Creating a vote requires expensive hashing, which takes up sizable portions of a voter’s resources. Similarly, supplying a repair, which is a copy of an entire document, consumes bandwidth. An adversary intending to waste a victim peer’s resources can start a poll, a multi-step protocol exchange, with the victim and then abort it at the point of maximum waste: an attacking poller can waste a voter’s computation by inviting it to create a vote and then ignoring that vote; the attacking poller can also waste a peer’s bandwidth by requesting unneeded repairs; an attacking voter can waste a poller’s time by falsely committing to compute a vote but never delivering it.

*Synchrony.* A LOCKSS poller requires a quorum of  $Q > 1$  voters (10 or so) before it can accept the outcome of a poll. Finding them can be difficult. They must be chosen at random to make directed subversion hard for the adversary; otherwise, an adversary with deterministic knowledge of which poller invites which voter into a poll can plan his subversions with maximum overall benefit [18]. Furthermore, the potential voters that the poller chooses at random must have free resources at the specified time, in the face of resource contention from other peers who are also competing for voters on the same or other documents at the same time. The adversary is under no such requirement, but can find and invite an individual victim into a futile poll.

This is a great advantage for the adversary. Intuitively, if the probability that a loyal peer is busy is  $b$ , then the probability that  $i$  peers are available for a poll is  $p(i) := (1 - b)^i$ . The probability that the adversary can make progress is that of finding one available peer  $p_a := p(1)$ , whereas the probability that the loyal peer can make progress is that of finding  $Q$  concurrently available peers  $p_l := p(Q) = p_a^Q$ . Even assuming that contacting  $Q$  voters in parallel is no more time-consuming than contacting 1, the expected number of peers the adversary can engage per try is  $P_a := Qp_a$  whereas the expected number of peers a loyal peer can engage per try is  $P_l := Qp_l = Qp_a^Q = P_a p_a^{Q-1} \leq P_a$ . The adversary can enhance this advantage with a defection attack that increases the busyness  $b$  of available peers during high contention, exacerbating the situation.

*Garbage Flooding.* Since a poll exchange between a poller and a voter is encrypted, if a message appears to be part of a poll by arriving on the correct port from the correct sender, the recipient must pay the cost of decrypting it. Although small compared to the hashing costs, this is not negligible. The attrition adversary can spoof the IP address of other poll participants and flood a victim with garbage messages costing little to generate, but much more for the victim to decrypt and identify as garbage.

#### 4.2 Defenses

We continue by describing LOCKSS defenses of each type proposed in Section 3. We proposed early versions of some of these defenses to tackle stealth and nuisance adversaries in prior work [20]. In this paper we concentrate on why these defenses help with an attrition adversary and describe others that are uniquely targeted at the attrition adversary. The full details on the resulting protocol are described elsewhere [16].

*4.2.1 Effort Balancing:* The LOCKSS opinion poll protocol requires that every protocol message that may cause the recipient to consume effort  $E$  be padded with provable, useless effort greater than  $E$ ; this means that the sender of the message has expended at least as much effort in producing that message as the recipient does due to the message. In the case of an attrition adversary, this ensures that his attack costs him as much effort as it costs his victim.

This effort balancing technique appears in several portions of the LOCKSS opinion poll protocol. First, a poller must supply to a voter participating in its poll a proof of at least as much effort as that required by that voter to produce the vote (i.e., to verify the poller’s effort proof and to produce the vote itself). Second, a voter must supply to a poller, along with its vote, as much provable effort as it takes the poller to evaluate that vote (i.e., to hash the local copy of the audited document with the same challenge so as to compare the digests and to detect a bogus vote).

These two instances of effort balancing correspond to the two “transitions” between the three basic steps of the poller-voter exchange: poller asks for a vote, voter constructs the vote, poller evaluates the vote. Effort balancing in the first transition protects a loyal voter from a frivolous poll invitation by a malicious poller, whereas in the second transition it protects a loyal poller from a bogus returned vote by a malicious voter.

For scheduling efficiency, we augment the poller-voter exchange with two extra scheduling steps: poller requests a vote by time  $t$  supplying an *introductory* effort proof, voter reserves appropriate voting resources, poller submits effort proof to balance the actual vote (as above), voter constructs the vote, poller evaluates the vote. In addition to the effort balancing described before, now the poller must also supply an introductory proof of effort as large as the effort required by the voter to reserve voting resources, augmented by the length of the reservation. This third instance of effort balancing protects a loyal voter from a malicious poller who seeks to tie up resources at a voter for the duration of the reservation. Note that if the voter declines, perhaps because it has no available resources to meet the poller-requested voting deadline, the poller’s introductory effort is wasted. Thus, the larger the introductory effort required, the greater the potential for waste when the system is busy, but also the greater the discouragement to the adversary who invites a victim into a poll and then defects, leaving the victim’s resources reserved but idle.

In summary, we use effort balancing to force the attrition adversary to expend effort commensurate with that inflicted on his victim directly (through resource consumption) and indirectly (through unavailability due to reservations of future resources).

*4.2.2 Rate Limitation:* Peers interact with each other by requesting and supplying votes. A peer decides autonomously

when to call a poll and from which peers to request votes, based on a fixed target rate. An adversary posing as a voter cannot attack at will; he must wait to be invited by the potential victim. A peer decides autonomously whether to supply a vote, based on its own resource schedule and on past experience with the specific poller (see Section 4.2.3). Consequently, an adversary posing as a poller can attack at will but cannot force the victim to respond. Peers also decide autonomously whether to supply a requested repair based on a maximum rate for each requester. In all cases, the peer limits its rate of participation without regard to external factors.

Rate limitation poses a challenging trade-off between adaptability of the rate limit when document collection sizes change and susceptibility to adversary manipulation. On one hand, a rigid rate limit inflexibly prevents the system from gracefully adapting to growing document collections: more documents to be audited means more polls run per time unit, all other parameters being equal. On the other hand, allowing rate limits to adapt automatically to the *perception* of document collection growth means that if a stealth modification adversary can convince peers to run polls more frequently, he can then subvert them faster. For this reason, we have chosen to fix rate limits and specify that the system has a fixed document capacity; higher capacities can be handled by provisioning additional instances of the system that run in parallel. For realistic workloads in the context of the preservation of digital scientific publications by libraries, this is a reasonable and practical approach.

**4.2.3 Admission Control:** Effort balancing can help with malicious peers who defect from a protocol exchange part way through, but cannot help when a malicious peer defects at the first protocol message. Consider, for instance, an adversary who floods a victim peer with garbage poll invitations containing bogus introductory effort proofs; such poll invitations cost the adversary little to generate but cost the victim much more to detect as malformed or bogus: the victim has to scan the invitation, check whether it has available resources to handle the request, and verify the introductory effort proof included in the message (described in Section 4.2.1).

To mitigate the effects of this flooding attack, we control how poll invitations are admitted for consideration. Our admission control mechanism differentiates between poll invitations from *known* pollers (i.e., previously seen) and those from *unknown* pollers, according to the source address of the invitation; the populations we are considering, in the tens of thousands of peers, allow a hash table of all known peer addresses to be maintained in memory for easy and cheap access. We rate limit invitations from unknown pollers by dropping all such invitations during a fixed period (the *refractory* period) after the last such invitation is considered, and rejecting invitations randomly with a fixed probability at other times.

A potential voter admits poll invitations from a known poller according to a local record of its interactions with that poller in the past, as either a voter or a poller. If the inviting poller is in *credit* with the invited voter — the poller has supplied more votes than it has received from the invitee — then its invitation is admitted for consideration. In the opposite case, i.e., when the inviting poller is in *debt*, its invitation is subject to both ran-

dom rejection and to the same rate limit imposed on unknown peers via the refractory period. This behavior approximates a reciprocative strategy [14] in which the cost of producing the introductory effort is wasted (“lost”) when the invitation is rejected.

To assist discovery and to facilitate the initial operation of new but loyal peers, we allow voters to introduce other peers to the poller. A peer treats an invitation from an introduced peer as if it were from a known peer in credit if the last vote it received from the introducer was valid. Only one introduction is honored for each such valid vote.

The parameters of this mechanism can vary; we are currently exploring the parameter space. However, we have some heuristics to help determine approximate combinations of values. First, to discourage whitewashing of identities, the fixed rejection probability for unknown peers is higher than that for known peers in debt. Second, the maximum rate limit applied to unknown peers is a small multiple of the expected rate for the system (obtained out of band). Third, the fixed drop probability for unknown peers is set so that the cumulative introductory effort expended by the adversary on dropped invitations is more than the voter’s effort to consider the adversary’s eventually *admitted* invitation, consistently with our effort balancing philosophy. Fourth, the maximum rate of admitting invitations from unknown peers and the cost of verifying an introductory effort are set so that, even if invitations with bogus introductory efforts arrive at a peer at that maximum admission rate, the sustained effort of proving them bogus is not a significant drain on the peer’s resources.

Another form of admission control is *nonce chaining*, an adaptation of SYN-cookies [3] to encryption. A sender of a protocol message includes a nonce in the encrypted portion of the message that both the sender and the recipient store. The response to that message must contain the nonce in its unencrypted portion. Before a peer decrypts a protocol message, it checks that the unencrypted nonce in that message matches what it expects from the sender. This means that a peer can drop spoofed garbage messages with a simple hash lookup instead of a decryption. Without being a participant in the exchange, the adversary must guess a nonce, must decrypt an observed message in transit, or must intercept and substitute a message in flight, before he can convince a peer to decrypt a message.

**4.2.4 Redundancy:** The problem we seek to solve, that of auditing a population of complete replicas, offers the great advantage of massive redundancy “for free.” Several real-world instances of this problem exist; for example, libraries maintain local copies of on-line journals so as to serve their patrons regardless of networking conditions. This massive redundancy afforded by our problem and opinion poll fault tolerance offered by our protocol make efforts to focus attacks on individual peers ineffective in reducing the usefulness of the system as a whole: when a poller fails to obtain a vote from a chosen voter, it just asks someone else and tries again later. To succeed, a pipe stoppage attack must target a large proportion of peers and Rate Limitation ensures that the attack can succeed only if it persists for weeks or longer.

**4.2.5 Compliance Enforcement:** Voters generate votes and pollers evaluate them using very similar processes: hashing

blocks of the local copy of the document and either generating or validating effort proofs. At the end of the evaluation process, the poller decides whether the vote was valid or invalid by the effort proofs and it decides whether it was agreeing or disagreeing by the hashes. A valid vote shows the poller that the voter performed the necessary effort. Conveniently, the process of generating a proof of effort produces about 160 bits of byproduct in addition to the proof itself. This byproduct is regenerated by the process of validating the proof. We use it as a receipt that the poller sends to the voter after evaluating the vote. If the receipt matches the byproduct of generating the vote, the voter knows the poller performed the necessary effort.

*4.2.6 Desynchronization:* An adversary can use limited resources more effectively by “time-shifting,” voting multiple times in a single poll under different identities. To reduce the extent to which stealth or nuisance adversaries can do this, we originally [20] made all voters in a poll start voting together and finish by some deadline. Even without an attrition attack, this synchrony reduces throughput under moderate load. It is easy for the adversary to exploit this, especially with a short-term estimate of how loyal peers have allocated their resources obtained through subversion [31] or observation.

Fortunately, our simulations show the system performing well even when the stealth adversary has unlimited resources and thus no need to time-shift. As we include the attrition adversary in our threat model, we allow the poller to invite voters independently, perhaps even serially, rather than simultaneously. A poll now consists of a sequence of two-party interactions rather than a single multi-party interaction. The attrition adversary suffers a lot while the stealth adversary gains little.

## 5. RELATED WORK

*Effort Balancing.* Effort balancing has been used as a defense against high-rate application-level denial of service such as email spam. Dwork and Naor [13] first articulated the concept of pricing via processing as a concrete tool to capture the “investment” of a player in the correct execution of a protocol. This investment can be measured in memory cycles [1], [12], CPU cycles [2], [13], or even Turing tests [28]. Douceur [11] argues that, in a pricing via processing scheme, if the difference in speed between the fastest and the slowest effort producer is large, then the scheme may become either “unaffordable” for legitimate slow players or unconstraining for illegitimate fast players. This makes memory-bound schemes, with their narrower gap between commercial low-end and high-end technologies [12], preferable to CPU-bound schemes. Crosby et al. [9] show that worst-case behavior of application algorithms can be exploited in application-level DoS attacks; effort balancing may be an effective protection against such attacks when fixing the worst-case behavior of algorithms is not an option, or when average-case behavior is, itself, expensive.

*Rate Limitation.* Rate limits are effective in slowing the spread of viruses at the network layer [30] or at the user-kernel boundary [27]. They have also been applied to peers joining a DHT [4], [29] as a defense against attempts to control part of the hash space. For many applications, rate limitation does not affect user behavior or experience [26].

*Admission control.* Admission control has been used to improve the usability of overloaded services. Cherkasova et al. [6] propose admission control strategies that help protect long-running web service sessions (i.e., related sequences of requests) from abrupt termination, since long-running sessions have greater potential for a purchase. In a P2P context, Daswani et al. [10] use admission control to mitigate the effects of a query flood against super-peers in unstructured file-sharing networks such as Gnutella. Reputation, i.e., an opinion formed by a peer about another peer, based on first-hand (subjective) or third-party information, can be invaluable in performing admission control. Feldman et al. [14] conduct a game-theoretic analysis of peer behavior to show that a reciprocative strategy (that is, “do unto others as others have done unto you”) in admission control can motivate cooperation among selfish peers.

*Redundancy.* Routing along redundant paths in an overlay has been suggested as a way of increasing the probability that a message arrives at its intended recipient despite forwarders dropping messages due to malice [4] or pipe stoppage [19].

*Compliance Enforcement.* Researchers have proposed the practice of storing useless content in return for having content be stored elsewhere, as a way to enforce symmetric storage relationships. Compliance enforcement is achieved by asking the peer storing the file of interest to hash some portion of the file as proof that it is still storing the file [8], [29]. Golle and Mironov [17] propose more general “uncheatable computations” based on the state of a protocol player; unless the player is in the state it seeks to prove to others, it cannot forge or guess the outcome of the uncheatable computation with significant probability. Such constructs can be effective for building compliance enforcement tools. This practice is also similar to the “trust but verify” vision [32] for detecting accountably the compliance of protocol entities to an expected service specification.

*Desynchronization.* Waves of synchronized routing updates caused by joins or departures in a DHT cause instability during periods of high churn [23]. Breaking the synchrony through lazy updates (e.g., in Bamboo [23]) can absorb the brunt of a churn attack.

## 6. FUTURE WORK

We are evaluating LOCKSS defenses against attrition attacks. Preliminary results are encouraging. In contrast with the protocol of [20], Figure 1 shows the system performing well even with as many attackers as defenders. Figure 2 suggests that less synchrony is better. We fake asynchronous voting by running more but smaller polls to obtain the same number of total votes and we observe reduced overall busyness.

We plan to run experiments perturbing protocol parameters to measure the sensitivity of the system and to see if there is a “sweet spot” for a deployed system.

Also, new protocol revisions require us to re-validate against past adversary strategies. We are currently running simulations against the stealth and nuisance adversaries.

Finally, a major goal of the LOCKSS research effort is to merge our findings into the currently running LOCKSS production system.

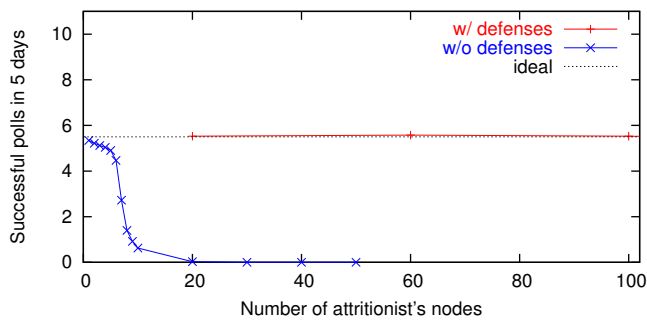


Fig. 1. Preliminary data. A LOCKSS system of 100 peers is attacked by attrition adversaries of increasing strength. The graph shows the average number of polls that loyal peers successfully complete per sliding 5-day period.

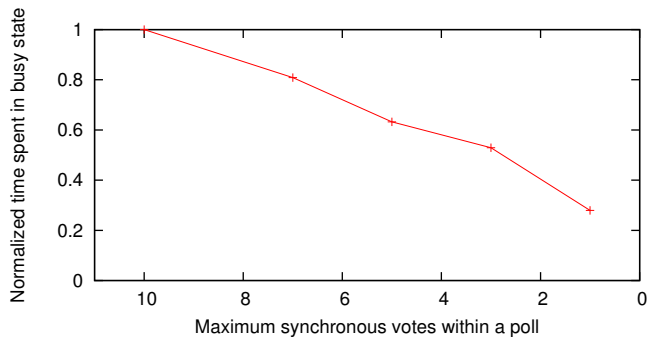


Fig. 2. Preliminary data. Absent an attack, the busyness (average time spent in a busy state) of the system of 100 peers decreases as the maximum synchronous votes within a poll decreases from 10 to 1. We normalize by the maximum synchrony experiment.

## 7. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation (Grant No. 0205667). Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the NSF. We would like to thank Kevin Lai, Yanto Muliadi, and Athena Markopoulou for their comments and help.

## REFERENCES

- [1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately Hard, Memory-bound Functions. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, Feb. 2003. Internet Society.
- [2] A. Back. Hashcash - a denial of service counter measure, Aug 2002. <http://www.hashcash.org/hashcash.pdf>.
- [3] D. J. Bernstein. Syn cookies. <http://cr.yp.to/syncookies.html>, 1996.
- [4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation*, pages 299–314, Boston, MA, USA, Dec. 2002.
- [5] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186, New Orleans, LA, USA, Feb. 1999. USENIX Association.
- [6] L. Cherkasova and P. Phaal. Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites. *IEEE Transactions on Computers*, 51(6):669–685, June 2002.
- [7] Computer Emergency Response Team. CERT Advisory CA-1996-21 TCP SYN Flooding Attacks. <http://www.cert.org/advisories/CA-1996-21.html>, Sept 1996.
- [8] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 120–132, Bolton Landing, NY, USA, Oct. 2003.
- [9] S. Crosby and D. S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *12th USENIX Security Symposium*, 2003.
- [10] N. Daswani and H. Garcia-Molina. Query-Flood DoS Attacks in Gnutella. In *Proceedings of the ACM Conference on Computer and Communications Security*, Nov. 2002.
- [11] J. Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pages 251–260, Boston, MA, USA, Mar. 2002.
- [12] C. Dwork, A. Goldberg, and M. Naor. On Memory-Bound Functions for Fighting Spam. In *23rd Annual International Cryptology Conference*, Santa Barbara, CA, USA, Aug. 2003.
- [13] C. Dwork and M. Naor. Pricing via Processing. In *12nd Annual International Cryptology Conference*, pages 139–147, Santa Barbara, CA, USA, Aug. 1992.
- [14] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques For Peer-to-Peer Networks. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA, 2004. ACM Press.
- [15] S. Floyd and V. Jacobson. The Synchronization of Periodic Routing Messages. *ACM Transactions on Networking*, 2(2):122–136, 1994.
- [16] T. Giuli, P. Maniatis, M. Baker, D. S. H. Rosenthal, and M. Roussopoulos. Resisting Attrition Attacks on a Peer-to-Peer System. Technical Report arXiv:cs.CR/0405111, Computer Science Department, Stanford University, Stanford, CA, USA, May 2004.
- [17] P. Golle and I. Mironov. Uncheatable Distributed Computations. In D. Naccache, editor, *Proceedings of the RSA Conference, Cryptographers' track*, volume 2020 of *Lecture Notes in Computer Science*, pages 425–440, San Francisco, CA, USA, Apr. 2001. Springer.
- [18] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the Spread of Influence Through a Social Network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146. ACM Press, Aug. 2003.
- [19] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 61–72, 2002.
- [20] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 44–59, Bolton Landing, NY, USA, Oct. 2003.
- [21] N. Michalakakis, D.-M. Chiu, and D. S. H. Rosenthal. Long Term Data Resilience Using Opinion Polls. In *22nd IEEE International Performance Computing and Communications Conference*, Phoenix, AZ, USA, Apr. 2003.
- [22] R. M. Needham. Denial of Service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153. ACM Press, 1993.
- [23] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the Usenix Annual Technical Conference*, Boston, MA, USA, June 2004.
- [24] D. S. H. Rosenthal and V. Reich. Permanent Web Publishing. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track*, pages 129–140, San Diego, CA, USA, June 2000.
- [25] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov. 2001.
- [26] S. Saroiu, K. P. Gummadi, R. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, Dec. 2002.
- [27] A. Somayaji and S. Forrest. Automated Response Using System-Call Delays. In *Proceedings of the 9th Usenix Security Symposium*, Aug. 2000.
- [28] Spam Arrest, LLC. Take Control of your Inbox. <http://spamarrest.com>.
- [29] D. Wallach. A Survey of Peer-to-Peer Security Issues. In *International Symposium on Software Security*, 2002.
- [30] M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. In *Proceedings of the 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, Dec. 2002.
- [31] D. Xuan, S. Chellappan, X. Wang, and S. Wang. Analyzing the Secure Overlay Services Architecture under Intelligent DDoS Attacks. In *Proceedings of the International Conference on Distributed Computing Systems*, Tokyo, Japan, Mar. 2004. IEEE.
- [32] A. R. Yumerefendi and J. Chase. Trust but Verify: Accountability for Internet Services. In *Proceedings of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium, Sept. 2004. ACM SIGOPS.