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 [12], 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 [14]. More recently, Denial of Service has come to mean high-bit-rate network-level attacks such as SYN-flooding [20] 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 impair the victim system over a long period.

The contribution of this paper 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 classes:

- *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 only 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 telltale 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 instantiate these defenses for LOCKSS, a real system for preserving 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. Provable effort mechanisms (e.g., Memory-Bound Functions [7] or client puzzles [6]) can ensure that the requester exerts more effort than the supplier. They inflate the cost of 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 [20]). 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 [22]. 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. Note also that the effort balancing defense described above also provides inherent rate limitation.

Admission Control. Rate limitation implies that a peer must reject or even drop 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 (e.g., in web services, preferentially drop requests signifying a new service session, instead of requests that continue longer-running sessions with greater potential for a purchase [4]), and reputation-based classification (prefer requesters with a good subjective or global history [8]).

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 [11]) and voting mechanisms (e.g., BFT [3] and LOCKSS [12]).

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. Using techniques similar to Golle and Mironov's [10] "uncheatable computations," it is possible to implement an unforgeable *evaluation receipt* that proves to the supplier that the requester performed the operation for which it made the request, demonstrating its compliance with the expected behavior of loyal peers.

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 [3], [17]). Sometimes multiple peers inadvertently synchronize (e.g., TCP sender windows at bottleneck routers, clients waiting for a busy server, and periodic routing messages [9]). 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, backoff, turn-taking, etc.

4. A Concrete Example: LOCKSS

To illustrate these defenses, we describe them in the context of the LOCKSS[1] digital preservation system, originally described in [16]. Briefly, LOCKSS peers cooperate to audit their copy of an online 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 hashes a nonce and its copy of the document to produce a vote. The poller tallies these votes and if it is outvoted in a landslide, it assumes its copy is corrupt and repairs it from a disagreeing voter.

Unfortunately, the original protocol implementing this simple concept is vulnerable to many attacks [13]. In subsequent work [12], we defend against the stealth and nuisance adversaries with a more complex series of exchanges between a poller and each participating voter. Using *Rate Limitation*, *Effort Balancing* and *Redundancy*, we render attacks by these adversaries ineffective. Here we outline further protocol refinements that address attrition attacks while retaining resistance to the stealth and nuisance adversaries.

4.1 Vulnerabilities

We identify a number of ways in which the LOCKSS protocol is vulnerable to the attrition adversary.

Defection. Creating a vote requires expensive hashing subject to a deadline, making it necessary to schedule the time for this effort. Repairs also consume bandwidth. An adversary intending to waste a victim peer's resources can start a protocol exchange with the victim then abort it at the point of maximum waste. An attacker can waste a voter's computation by inviting it to create a vote and then ignoring that vote; he can waste a poller's time by falsely committing to compute a vote but never delivering it; and he can waste a peer's bandwidth by requesting unneeded repairs.

Synchrony. A LOCKSS poller requires a quorum of $Q > 1$ voters (10 or so) before it can accept the outcome of a poll.

[1]Lots Of Copies Keep Stuff Safe. LOCKSS is a trademark of Stanford University.

Finding them can be difficult. They must be chosen at random to make directed subversion hard for the adversary; they 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. 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 $P_l = Qp_l = Qp_a^Q = P_a p_a^{Q-1} \leq P_a$. In conjunction with a defection attack that increases the busyness b of available peers during high contention, this vulnerability gives the adversary a formidable advantage.

Garbage Flooding. The first two messages between a poller and a voter perform a Diffie-Hellman key exchange that is used to prevent spoofing of and eavesdropping on the remaining messages. If a message appears to be part of a poll, the recipient must pay the cost of decrypting it which, though small in comparison to the hashing costs, 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. Some of these defenses are also applicable to our earlier considered adversaries, stealth and nuisance, while others are uniquely targeted at the attrition adversary.

4.2.1 Effort Balancing: Every protocol message that may cause the recipient to consume effort E is padded with provable, useless effort greater than E, forcing the attrition adversary to expend at least as much effort as his victim. More specifically, 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. Furthermore, votes are computed in rounds that interleave provable effort and data hashing; the poller who evaluates a vote can detect a bogus vote with no more effort than was required to generate it.

Vote requests supply a deadline by which the vote must be returned to prevent time-shifting. A voter must reserve time in its schedule to compute the vote in time to meet the deadline. We therefore include in the poll request at least as much provable effort as required by the voter to establish a session key via a Diffie-Hellman exchange and reserve time for voting; we also size this effort to be proportional to the size of the reserved time. This forces the attrition adversary to expend effort commensurate with that inflicted on his victim directly (through resource consumption) and indirectly (through unavailability due to a time reservation).

This introductory effort must come before a voter has committed to participate. If the voter declines, perhaps because it is busy, the 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.

Another form of effort balancing is *nonce chaining*, a technique related to SYN-cookies [1] for encryption. A sender of a protocol message includes a nonce in the encrypted portion of the message that both the sender and the recipient must 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 message 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.

4.2.2 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; it must wait to be invited by the potential victim. A peer decides autonomously whether to supply a vote, based on its own resource schedule (see Section 4.2.3). 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.

4.2.3 Admission Control: The process of considering a poll invitation is relatively cheap but not free; it involves checking local resource commitments to determine availability, establishing a Diffie-Hellman public key for subsequent exchanges, and verifying an introductory effort proof. An adversary can flood a victim peer with garbage poll invitations containing bogus introductory effort proofs, which cost little to generate but more for the victim to detect as bogus. We prevent this by using rate limitation as the basis of an admission control mechanism. Each peer limits the rate at which it considers poll invitations from *unknown* peers according to a blanket policy and from *known* peers according to their history.

The rate limit for invitations from unknown pollers is implemented 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 peer implements the limit for known pollers by maintaining for each a history of mutual voting on shared documents to determine whether the poller is in *debt* (i.e., it has supplied the invitee fewer votes than it has received from the invitee) or *credit* (in the opposite case). An invitation from a poller in debt is subject to both random rejection with a fixed probability, and to the rate limit for invitations from unknown peers. Otherwise, invitations from the known poller are always inspected. This behavior approximates a reciprocative strategy [8] in which the cost of producing the introductory effort is wasted ("lost") when the invitation is rejected or dropped.

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. 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 poller's effort to consider the adversary's eventually *admitted* invitation. 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 effort of proving them bogus is not a significant drain on the peer's resources.

4.2.4 Redundancy: Massive redundancy and "opinion poll" fault tolerance 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. To reduce the extent to which stealth or nuisance adversaries could do this, we originally [12] 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 [23] 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: Distributed Hash Tables

Some of the attrition defenses we identify have already been proposed for Distributed Hash Tables (DHTs), which provide an efficient hash table abstraction for P2P applications. Others may well be useful also. Peers in a DHT maintain a portion of the hash table and local routing tables. They satisfy application queries that match their portion of the hash, and otherwise route those queries to peers with the appropriate portion. DHTs are subject to a number of attacks, but here we focus on attrition attacks intended to disrupt the system rather than those intended to control or influence it.

As *churn* (the rate at which the peer population changes) increases, both the latency and the probability of failure of queries to a DHT increases [15]. An attrition attack might consist of adversary peers joining and leaving fast enough to destabilize the routing infrastructure.

The attrition adversary might cause pipe stoppage at selected peers, preventing them from forwarding messages. If all possible routes for a message pass through one of the victims, the query fails.

Alternatively, the adversary might attempt to flood the DHT's storage resources with garbage and thereby prevent useful content from being added. If peers' identities are strong, for example supported by smartcards [18], or a central authority [2] quotas may be enforceable to prevent this; for loosely connected communities, however, where strong identities may be unrealistic, Sybil attackers can defeat quotas by creating new identities as needed.

Rate Limitation. Rate limits on peers joining a DHT have been suggested [2], [21] as a defense against attempts to control parts of the hash space, for example to control the placement of certain data objects or for misrouting. Limiting both joins and stores to empirically determined safe rates will also be needed to thwart the attrition adversary. At least for file sharing, studies [19] have suggested that users' behavior may not be sensitive to latency. The increased storage latency that rate limits create is probably unimportant.

Effort Balancing. In the absence of strong peer identity, rate limits alone do not prevent the churn and useless storage attacks. The adversary can generate join and store requests at a rate sufficient to swamp the loyal peers, creating new identities as needed. If requests to join or store content must include a proof of enough effort to compensate for the cost of the operation, there can be a limit on the rate at which a resource-constrained adversary can generate such requests, and thus on the effectiveness of the attack.

Redundancy. Routing along multiple redundant paths in the DHT overlay has been suggested as a way of increasing the probability that a message arrives at its intended recipient despite nodes dropping messages due to malice [2] or pipe stoppage [11].

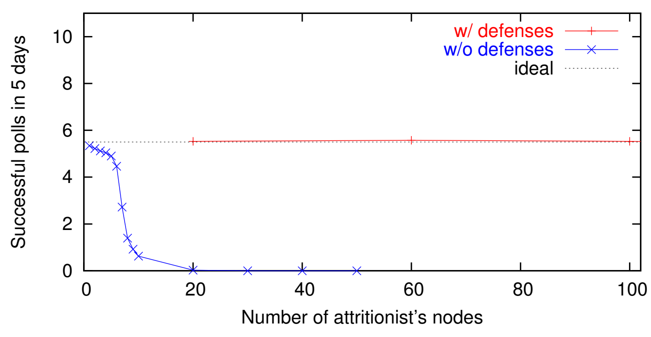

Fig. 1. Preliminary data. A LOCKSS system of 100 peers is attacked by attrition adversaries of increasing strength. The graph shows the average number of polls a loyal peer successfully completes in 5 days.

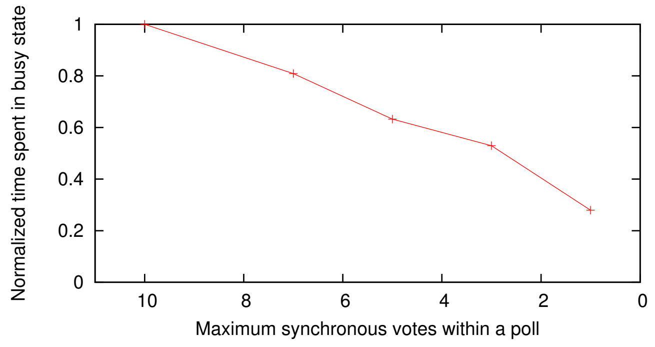

Fig. 2. Preliminary data. Absent an attack, the busyness (average time spent in a busy state) of the system of 100 peers decreases as the maximum synchronous votes within a poll decreases from 10 to 1. We normalize by the maximum synchrony experiment.

Compliance Enforcement. Some researchers have proposed storing useless content in exchange for having content be stored 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 [5], [21].

Desynchronization. Waves of synchronized routing updates caused by joins or departures cause instability during periods of high churn [15]. Breaking the synchrony through lazy updates (e.g., in Bamboo [15]) 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 [12], 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" that a deployed system should run in.

Also, new protocol revisions require us to re-validate against past adversary strategies. We have run simulations against the stealth adversary and found no major differences in results.

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

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 these funding agencies. We would like to thank Kevin Lai and Yanto Muliadi for their comments and help.

References

[1] D. J. Bernstein. Syn cookies. `http://cr.yp.to/syncookies.html`, 1996.
[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *OSDI*, 2002.
[3] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *OSDI*, 1999.
[4] L. Cherkasova and P. Phaal. Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites. *IEEE Trans. on Computers*, 51(6), 2002.
[5] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *SOSP*, 2003.
[6] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *USENIX Security Symp.*, 2001.
[7] C. Dwork, A. Goldberg, and M. Naor. On Memory-Bound Functions for Fighting Spam. In *CRYPTO*, 2003.
[8] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *ACM Electronic Commerce*, 2004.
[9] S. Floyd and V. Jacobson. The Synchronization of Periodic Routing Messages. *ACM Trans. on Networking*, 2(2), 1994.
[10] P. Golle and I. Mironov. Uncheatable Distributed Computations. *Lecture Notes in Computer Science*, 2020, 2001.
[11] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *SIGCOMM*, 2002.
[12] P. Maniatis, M. Roussopoulos, TJ Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting. In *SOSP*, 2003.
[13] N. Michalakis, D-M. Chiu, and D. S. H. Rosenthal. Long Term Data Resilience Using Opinion Polls. In *22nd IEEE Intl. Performance Computing and Communications Conference*, Phoenix, AZ, USA, April 2003.
[14] R. Needham. Denial of Service. In *ACM Conf. on Computer and Communications Security*, 1993.
[15] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. Technical Report UCB//CSD-03-1299, UC Berkeley, December 2003.
[16] D. S. H. Rosenthal and V. Reich. Permanent Web Publishing. In *USENIX, Freenix Track*, 2000.
[17] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, 2001.
[18] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *SOSP*, 2001.
[19] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *OSDI*, 2002.
[20] Computer Emergency Response Team. CERT Advisory CA-1996-21 TCP SYN Flooding Attacks. `http://www.cert.org/advisories/CA-1996-21.html`, Sept 1996.
[21] D. Wallach. A Survey of Peer-to-Peer Security Issues. In *Intl. Symp. on Software Security*, 2002.
[22] M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. In *Annual Computer Security Applications Conf.*, 2002.
[23] D. Xuan, S. Chellappan, X. Wang, and S. Wang. Analyzing the Secure Overlay Services Architecture under Intelligent DDoS Attacks. In *ICDCS*, March 2004.