Defending against Eclipse attacks on overlay networks

Atul Singh[1]* Miguel Castro[2] Peter Druschel[1] Antony Rowstron[2]

[1]Rice University, Houston, TX, USA.†
[2]Microsoft Research, Cambridge, UK.

Abstract

Overlay networks are widely used to deploy functionality at edge nodes without changing network routers. Each node in an overlay network maintains pointers to a set of neighbor nodes. These pointers are used both to maintain the overlay and to implement application functionality, for example, to locate content stored by overlay nodes. If an attacker controls a large fraction of the neighbors of correct nodes, it can "eclipse" correct nodes and prevent correct overlay operation. This Eclipse attack is more general than the Sybil attack. Attackers can use a Sybil attack to launch an Eclipse attack by inventing a large number of seemingly distinct overlay nodes. However, defenses against Sybil attacks do not prevent Eclipse attacks because attackers may manipulate the overlay maintenance algorithm to mount an Eclipse attack. This paper discusses the impact of the Eclipse attack on several types of overlay and it proposes a novel defense that prevents the attack by bounding the degree of overlay nodes. Our defense can be applied to any overlay and it enables secure implementations of overlay optimizations that choose neighbors according to metrics like proximity. We present preliminary results that demonstrate the importance of defending against the Eclipse attack and show that our defense is effective.

1 Introduction

Overlay networks are widely used to deploy functionality at edge nodes without changing network routers. Many popular applications are deployed as overlay networks, for example, BitTorrent [2], Gnutella [8], Kazaa [13], Overnet and eDonkey [16]. There is also a large amount of research on using overlays to implement application level multicast[12, 11, 1] and distributed hash tables[17, 22, 20, 24]. This paper discusses a general attack on overlay networks that we call the *Eclipse attack*.

Each node in an overlay network maintains overlay links to a set of neighbor nodes. These links are used both to maintain the overlay and to implement application functionality, for example, to locate content stored by overlay nodes or to multicast messages. Correct overlay operation requires that correct nodes be able to communicate by forwarding messages along overlay links. If an attacker controls a large fraction of the neighbor sets of correct nodes, it can "eclipse" correct nodes by dropping or rerouting messages that attempt to reach them. In the extreme, the Eclipse attack provides the attacker with full control over all overlay traffic.

The Eclipse attack is more general than the well-known Sybil attack [6]. An attacker can use a Sybil attack to launch an Eclipse attack by creating a large number of seemingly distinct overlay nodes to populate the neighbor sets of correct nodes. However, a defense against the Sybil attack may be insufficient to defend against the Eclipse attack. Even if attackers control only a small fraction of overlay nodes, they may be able to launch an Eclipse attack by exploiting the overlay maintenance algorithm. For example, in an overlay like Gnutella, nodes replace faulty neighbors with nodes obtained by traversing neighbor links. If the attacker controls a fraction f of the nodes in the overlay, attacker nodes can return other compromised nodes whenever they are asked to for a neighbor and correct nodes may still return a compromised node with probability at least f. Therefore, the fraction of neighbors of correct nodes that is controlled by the attacker tends to grow until the attacker has full control over all overlay traffic. We will present results of simulations that demonstrate the effectiveness of this attack.

Castro et. al identify the Eclipse attack as one of the security problems in structured overlays [3]. They use strong structural constraints on the overlay to defend against this attack. Nodes are assigned identifiers and each node's overlay neighbors are the overlay nodes with identifiers closest to particular points in the identifier space. This defense is effective but it removes the flexibility necessary to implement optimizations like proximity neighbor selection [4, 18, 9] and works only for structured overlays. More recent work [10, 21] proposes defenses to the Eclipse attack that allow proximity neighbor selection. They assume that it is hard for the attacker to pretend to be close to all nodes in the network when only

*Work started during a summer internship at Microsoft Research, Cambridge.

†This research was supported by Texas ATP (003604-0079-2001), by NSF (ANI-0225660), http://project-iris.net and a gift from Microsoft Research.

a small fraction of overlay nodes is compromised. These defenses apply only to specific structured overlays and rely on the ability to measure network delays securely. This may be difficult because attacker nodes can interfere with delay measurements by causing artificial congestion in the network. Moreover, many nodes tend to appear equidistant in real networks [9].

This paper proposes a new defense that prevents Eclipse attacks by bounding the degree of overlay nodes. The idea is simple: the indegree of attacker nodes is likely to be higher than the average indegree of correct nodes when the attacker launches an Eclipse attack. Therefore, correct nodes choose their neighbors from the subset of overlay nodes whose indegree is below a threshold. This defense introduces a new attack because attacker nodes can consume the indegree of correct nodes and prevent other correct nodes from pointing to them. Thus, it is also necessary to bound the outdegree so correct nodes choose neighbors from the subset of overlay nodes whose indegree and outdegree are below a threshold. We describe an efficient auditing technique to prevent attacker nodes from lying about their indegree and outdegree.

Our defense can be applied both to structured and unstructured overlays because it does not rely on any specific structure. Moreover, it allows overlay optimizations that choose neighbors according to metrics like proximity, which are important for overlay efficiency.

In the following section, we discuss different overlays designs and their vulnerability to the Eclipse attack. Section 3 describes our defense against the Eclipse attack and Section 4 provides a preliminary evaluation of our defense. Section 5 presents conclusions and a discussion of future work.

2 Vulnerability to Eclipse attacks

Different overlays impose different constraints on the members of a given node's neighbor set. Such constraints affect the resilience of an overlay to the Eclipse attack and they also determine the effectiveness of optimizations that choose overlay neighbors based on performance metrics like network proximity. This section discusses the impact of structure and performance optimizations on the effectiveness of the Eclipse attack. We assume that these systems implement a defense against Sybil attacks that bounds the fraction of overlay nodes controlled by the attacker to f.

2.1 Unstructured overlays

Unstructured overlays like Gnutella [8] do not impose any constraints on the members of a node's neighbor set. They are the most vulnerable to the Eclipse attack.

These overlays use floods or random walks to find overlay neighbors. For example, nodes can use a random walk through the overlay graph to select a node uniformly at random from the set of overlay nodes if the length of the walk is greater than the diameter of the graph. However, an attacker can bias the selection to nodes that it controls. If the random walk visits a node controlled by the attacker, it returns another compromised node as the result of the search. The probability of visiting a node controlled by the attacker during a walk of length l is at least $1-(1-f)^l$, which is greater than f. Thus, the average fraction of neighbor set members controlled by the attacker increases, which results in an increased probability of visiting an attacker node during a future random walk. The fraction of attacker nodes in the neighbor sets of correct nodes keeps increasing until the attacker has full control over all overlay traffic.

2.2 Structured overlays

Structured overlays (e.g., [17, 22, 20, 24]) impose constraints on a node's neighbors. Each node has a unique identifier and selects its neighbors from the set of nodes whose node identifiers satisfy certain constraints relative to its own identifier. For example, the neighbor sets in Tapestry and Pastry are organized as a matrix; a node x can be a neighbor in slot (i,j) of y's neighbor matrix if the first i digits in x' and y's identifiers are the same and the $(i+1)$th digit in x's identifier is j.

These constraints limit the expected fraction of attacker nodes in a node's neighbor set, assuming a defense against Sybil attacks that prevents attackers from obtaining many node identifiers or choosing their node identifiers. For example, unless the attacker controls a very large fraction of the overlay, she won't be able to supply neighbors that can fit in the bottom rows of the neighbor matrices. Unfortunately, the Eclipse attack is still very effective because the attacker can easily supply neighbors to fit in the top rows, as shown in Section 4.

It is possible to strengthen these structural constraints to prevent the Eclipse attack. For example, in overlay networks like CAN [17], the original Chord [22] and Pastry with a constrained routing table [3] each node's overlay neighbors are the overlay nodes with identifiers closest to particular points in the identifier space. For example, the neighbors of a node with identifier i in Chord are the nodes whose identifiers are the successors of $(i+2^{i-1}) \bmod 2^{160}$ for positive integers i. Since the identifiers of attacker nodes are uniformly distributed throughout the identifier space, the attacker has probability f of controlling the node with identifier closest to a particular point in the identifier space. So it can control only an expected fraction f of the neighbor set members of correct nodes.

In addition to enforcing strong structural constraints, a secure mechanism is needed to locate the nodes with identifiers closest to particular points in the identifier space. Techniques to achieve this are described in [3].

2.3 Overlay optimizations

Strong structural constraints can prevent Eclipse attacks but they also prevent important performance optimizations that exploit flexibility in the choice of overlay neighbors. For example, proximity neighbor selection (PNS) [4, 9] enables low delay routing in structured overlays by selecting neighbor nodes that are nearby in the physical network from among all candidate nodes that satisfy the structural constraints. Unfortunately, strong structural constraints prevent PNS because they leave no flexibility to choose neighbors.

Recent work [10] proposed an interesting defense against the Eclipse attack that does not prevent proximity neighbor selection. This defense is based on the idea that it is difficult for the attacker to pretend to be close to all good nodes in the network. It relies on a mechanism that exploits the Tapestry overlay structure to locate nearby nodes. This mechanism is secure under certain assumptions about the distribution of network delays and the accuracy of delay measurements.

However, it is unclear to what extent these assumptions hold in real networks. Recent results suggest [9] that the distribution of delays from a node to other nodes in the Internet is such that a large number of nodes lie within a fairly narrow delay band. Therefore, it may be hard to use delay as a constraint to prevent an attacker from biasing neighbor selection. Additionally, the defense relies on the ability to securely measure network distances, which may be difficult because the attacker can cause artificial congestion in the network to increase measured delays to good nodes.

There are also important optimizations that exploit heterogeneity. For example, Kazaa [13] and the latest versions of Gnutella [8] select some high capacity nodes as super-peers and ordinary nodes attach only to super-peers. Similarly, nodes in GIA [5] have a capacity value and the overlay is built such that indegree is proportional to capacity. These optimizations are important to achieve high performance in these unstructured overlays but they make them even more vulnerable to the Eclipse attack; the attacker can simply pretend to be a high capacity node to increase the fraction of members it controls in the neighbor sets of correct nodes. It is hard to prevent the Eclipse attack in these systems without controlling the capacity that each node is allowed to advertise.

3 Defense: degree bounding

We describe a new defense against the Eclipse attack that relies neither on structural constraints nor accurate proximity measurements. The defense can be applied to both structured and unstructured overlays and it permits performance optimizations like proximity neighbor selection (PNS).

The basic idea behind our defense is simple: the indegree of attacker nodes must be higher than the average indegree of nodes in the overlay during an Eclipse attack. Therefore, correct nodes can bound the indegree of attacker nodes by choosing their neighbors from the subset of overlay nodes whose indegree is below a threshold. Unfortunately, this defense introduces a new attack because attacker nodes can consume the indegree of correct nodes and prevent other correct nodes from pointing to them. Therefore, it is necessary to bound both the indegree and the outdegree of attacker nodes. Correct nodes choose neighbors from the subset of overlay nodes whose indegree and outdegree are below a threshold. One of the difficulties is how to enforce the indegree and outdegree bounds. We outline a technique to securely enforce these bounds.

3.1 Auditing to enforce degree bounds

We use auditing to enforce degree bounds. We rely on certified node identifiers [3] to defend against Sybil attacks and to bootstrap authentication. A node generates a public-private key pair that can be used to encrypt and sign messages. The node identifier certificate binds the node's random identifier with the public key. It is difficult for attackers to obtain many certified identifiers or to choose the identifiers that they obtain.

Each node x in the overlay is required to maintain a list with all the nodes that have x in their neighbor set. We refer to this as the back pointer list of x. Periodically, x challenges each member of its neighbor set by asking it for its back pointer list. If the number of entries in the returned back pointer list is greater than the indegree bound or x is not present in the back pointer list, x removes that member from its neighbor set. Node x only forwards traffic from nodes in its backpointer list.

To prevent an attacker from consuming the indegree of correct nodes, each node x periodically challenges each member of its back pointer list by asking it for its neighbor set. If x is not in the returned neighbor set or the size of the returned neighbor set is greater than the outdegree bound, x removes the node from its back pointer list.

To ensure that replies are fresh and authentic, x includes a random nonce in the challenge. The node being challenged includes the nonce in its reply and signs it. When x receives the reply, it checks the signature and the nonce before accepting the reply.

When the neighbor set has structure, it may be necessary to enforce bounds on specific neighbor set components. For example, Tapestry and Pastry routing tables have rows. If an attacker controls a large fraction of entries in the top rows of routing tables, it will be able to control most communication. With a bound on the total number of backpointers, attackers can choose to attract only pointers from top level entries to cause the most damage.

We prevented this problem by enforcing a bound on the number of backpointers for each row number, for example, a node x can appear in at most $2^b - 1$ entries in row i of the routing tables of other nodes (where $2^b - 1$ is the number of entries per routing table row in Pastry). We also enforced the same outdegree bounds per row. A bound of $2^b - 1$ ensures that the fraction of entries controlled by the attacker in each row is bound to $f/(1-f)$ when the attacker controls a fraction f of the overlay.

3.2 Anonymous auditing

In order for the auditing to work, we need to ensure that the node being challenged does not know the identity of the challenger. Otherwise, the node being challenged can easily produce a back pointer list (or neighbor set) with the correct size that includes the challenger. We need an anonymous channel between the challenger and the node being challenged.

We could use existing implementations of anonymous channels [19, 7, 14] but the anonymity requirements of our auditing mechanism are weaker than those provided by off-the-shelf techniques. We only require sender anonymity for auditing and a node should receive challenges from all of its overlay neighbors and only from them. Therefore, it is sufficient to ensure that the challenge is equally likely to come from any neighbor, which is easier than providing anonymity with general communication patterns. Additionally, it is sufficient to ensure that the identity of the challenger is obscured most of the time. An occasional failure of sender anonymity merely increases the time to detect malicious behavior. We implemented an anonymous channel on Pastry that exploits the weaker requirements to improve performance.

Our anonymous auditing process involves two steps: (1) discovery of intermediate nodes and (2) relaying of challenges through intermediate nodes. It is important to ensure that the first step does not expose the challenger. For example, if the attacker observes the traffic issued by a challenger to discover an intermediate node, it may be able to use this information while responding to challenges forwarded by that intermediate node. We avoid this problem by ensuring that all nodes auditing a node x relay their challenges to x through the same set of intermediate nodes. This is a general solution to prevent attacks that correlate the identity of the intermediate node with the identity of the challenger.

In our current implementation, the intermediate nodes used to audit node x are the a nodes with identifiers closest to x's identifier. We call these nodes the anonymizer nodes for x. Each node that wants to audit x uses the redundant routing technique described in [3] to discover the set of anonymizer nodes for x.

To audit x, a node picks a random node from the set of anonymizers and relays the challenge through that node. If the anonymizer is correct, it will forward the request to x without revealing the identity of the challenger. If x is correct, it will reply to the challenge but if it is compromised, it may choose not to reply because it would risk exposure by responding with a back pointer list or neighbor set without the challenger. On the other hand if the anonymizer and x are both compromised, the anonymizer may reveal the identity of the challenger to allow x to respond correctly. Conversely, a compromised anonymizer may drop the challenge if x is honest. So it is important to repeat the challenge through different anonymizer nodes before dropping neighbors.

We challenge each node multiple (a) times through different anonymizer nodes. If the number of correct replies to the challenges is below a threshold t, the node being audited is classified compromised and dropped from the neighbor set or back pointer set as appropriate. We can choose a and t to achieve a desired tradeoff between overhead and accuracy for a given bound on the fraction of malicious nodes in the overlay.

We randomize the period between challenges from the same node to prevent an attacker from correlating the arrival time of the challenge with the identity of the challenger. Similarly, a node waits for a random delay after discovering the anonymizer nodes and before issuing a challenge.

4 Preliminary evaluation

In this section, we present preliminary results on the impact of Eclipse attacks and evaluate the effectiveness of bounding node degrees to defend against this attack.

4.1 Experimental setup

We used MSPastry [15] and a packet-level discrete-event simulator with a transit-stub network topology model [23]. This model has 5050 routers arranged hierarchically. There are 10 transit domains at the top level with an average of 5 routers in each. Each transit router has an average of 10 stub domains attached, and each stub has an average of 10 routers. Routing is performed using the routing policy weights of the topology generator [23]. The simulator models the propagation delay on the physical links. The average delay of core links was 40.7ms. Each end system was attached to a randomly selected stub router with a link delay of 1ms.

We created Pastry overlays with different sizes. These overlays were created by having all nodes join at the same time and there were no failures or additional node arrivals in the experiments. For all the experiments Pastry was configured with $b = 4$, $l = 16$, and nodes join from 16 distinct bootstrap nodes [3].

The fraction of malicious nodes was $f = 0.2$ and they all collude to maximize the fraction of malicious nodes in the neighbor set of correct nodes; they misroute join messages to other malicious nodes and always supply malicious neighbor set entries.

4.2 Eclipse attacks with no defense

The first set of experiments evaluates the impact of Eclipse attacks without our defense. Figure 1 shows the fraction of Pastry routing table entries that point to malicious nodes for different overlay sizes with and without proximity neighbor selection (PNS). It also shows the fraction of malicious entries in the top level of the routing tables. Even if the fraction of all routing table entries controlled by the attacker is low, the attacker can control most overlay communication if the fraction of top level routing table entries that point to malicious nodes is high.

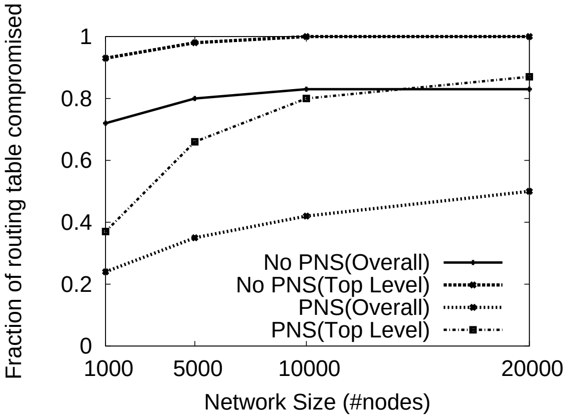

Figure 1: Fraction of malicious nodes in correct routing tables and in top row of routing table for different network sizes with and without PNS

The results show that the Eclipse attack is more effective without PNS. Since we did not model attacks on delay measurements, PNS replaces malicious neighbors by correct nodes that are closer in the network. However, the Eclipse attack is extremely effective for large overlays even when using PNS. Since the attacker controls about 80% of the neighbors in the top levels of routing tables, it will be able to intercept more than 80% of the communication. This happens because the delay distribution changes when the overlay size increases: the number of nodes that are equidistant from a target increases. Therefore, it is less likely for malicious neighbors to be replaced by closer correct nodes.

4.3 Defending against Eclipse attacks

Our previous results show that it is important to defend against Eclipse attacks. We ran experiments to evaluate the effectiveness of enforcing degree bounds to defend against the Eclipse attack. We enforce indegree and outdegree bounds per routing table row (as discussed in Section 3.1).

Auditing oracle The first set of experiments uses an oracle to determine if nodes exceed the per row degree bounds. They provide an evaluation of our defense that is independent from any particular implementation of anonymous auditing. We experimented with bounds of 16, 32, 48, and 64 ($2^b = 16$ is the routing table row size). Table 1 presents results with PNS. It shows the average fraction of malicious routing table entries in the whole routing table. The fraction of malicious routing table entries for each row number was very similar.

bound	1000	5000	10000	20,000
16	0.24	0.24	0.24	0.24
32	0.24	0.29	0.31	0.37
48	0.24	0.31	0.35	0.45
64	0.24	0.33	0.38	0.48
no defense	0.24	0.35	0.42	0.5

Table 1: Fraction of malicious nodes in correct routing tables with different degree bounds per row.

The results show that bounding degrees is effective at maintaining the fraction of malicious routing table entries low. As expected, this fraction is approximately $f/(1-f) = 0.25$ when the bound is equal to the average number of entries per row. However, the effectiveness of our defense decreases significantly when the bound is higher than the average number of entries per row.

We also evaluated the impact of our defense technique on routing delays. Since our technique constrains the choice of neighbors, it may increase delays in the absence of attacks. We observed a delay penalty of approximately 25% in the overlay with 20,000 nodes with $f = 0$ and a degree bound of 16 per row. The penalty decreases to about 8% for a bound of 32.

Auditing implementation We implemented the anonymous auditing technique described in Section 3.2 in MSPastry and ran a simulation experiment to evaluate its effectiveness. The experiment started by creating an overlay with 1020 correct nodes using PNS. We added 244 malicious nodes to the overlay 20 minutes into the simulation (setting $f = 0.19$). Then, we measured the fraction of malicious routing table entries over time without additional changes in overlay membership. We used a per-row degree bound of 16 and an average auditing period of 1 minute.

We experimented with a *greedy attack* where malicious nodes attempt to get added to as many routing tables as possible but when audited through a correct anonymizer node do not reply to prevent exposure. This is a good strategy in the current implementation because neighbors are only marked malicious when they send bad replies. Neighbors that do not reply to challenges are marked suspicious and are replaced only if there is another node with enough indegree budget that is not suspicious or malicious.

The results of the experiment indicate that our anonymous auditing technique is an effective defense against the Eclipse attack. The attacker starts by controlling 35% of the routing table entries and auditing brings this fraction down to 20% within five hours with an overhead of approximately 4 messages per second per node. We are currently experimenting with marking suspicious nodes malicious if they are replying to routing table liveness probes. This should bring the fraction of malicious entries down in less than 30 minutes.

5 Conclusion

This paper has shown that Eclipse attacks are effective: attackers can disrupt overlay communication by controlling a large fraction of the neighbors of correct nodes even when they control only a small fraction of overlay nodes. It is important to defend against Eclipse attacks. We have proposed a novel defense that prevents Eclipse attacks by using anonymous auditing to bound the degree of overlay nodes. This defense can be used both in structured and unstructured overlays and it allows optimizations like proximity neighbor selection that bias the choice of neighbor based on some application metric. The results of preliminary experiments show that a prototype of our defense can prevent attacks effectively in a structured overlay.

The work presented in this paper is promising but it is not complete. There are several important issues that we have not addressed yet. We are still studying the ideal attacker strategy and evolving our anonymous auditing mechanism. The experiments we presented were run on static overlays. It is important to evaluate both the overhead and effectiveness of our technique with constant churn as well as large, sudden changes in overlay membership. We suspect that this will prompt changes to our prototype to optimize discovery of anonymizer nodes. Finally, our anonymous auditing prototype relies on overlay structure. It would be interesting to design an efficient auditing mechanism for unstructured overlays.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, Aug. 2002.

[2] Bittorrent, 2004. http://bitconjurer.org/BitTorrent/.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. OSDI 2002*, Boston, MA, Dec. 2002.

[4] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, May 2002.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *ACM SIGCOMM*, Aug. 2003.

[6] J. R. Douceur. The Sybil Attack. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, Massachusetts, Mar. 2002.

[7] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of IPTPS '02*, Cambridge, Massachusetts, Mar. 2002.

[8] The Gnutella protocol specification, 2000. http://dss.clip2.com/GnutellaProtocol04.pdf.

[9] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *ACM SIGCOMM*, Aug. 2003.

[10] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks. In *17th International Symposium on Distributed Computing*, Oct. 2003.

[11] Y. hua Chu, S. G. Rao, and H. Zhang. A Case For End System Multicast. In *Proc. of ACM Sigmetrics*, pages 1–12, Santa Clara, CA, June 2000.

[12] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. OSDI 2000*, San Diego, California, 2000.

[13] KaZaa. http://www.kazaa.com/.

[14] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. AP3: Anonymization of Group Communication. In *ACM SIGOPS European Workshop*, Sept. 2004.

[15] MSPastry. http://research.microsoft.com/˜antr/Pastry.

[16] OverNet, 2004. http://www.overnet.com/.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, Aug. 2001.

[18] S. Ratnasamy, S. Shenker, , and I. Stoica. Routing algorithms for DHTs: Some open questions. In *IPTPS*, Mar. 2002.

[19] M. K. Reiter and A. D. Rubin. Anonymous Web transactions with Crowds . *Communications of the ACM*, 42(2):32–48, Feb. 1999.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, Nov. 2001.

[21] A. Singh. Secure proximity aware routing. In *1st IRIS Workshop*, Aug. 2003.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *ACM SIGCOMM*, Aug. 2001.

[23] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM96*, San Francisco, California, 1996.

[24] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-area Location and Routing. Technical Report UCB-CSD-01-1141, U. C. Berkeley, Apr. 2001.