Controlling the Cost of Reliability in Peer-to-Peer Overlays

Ratul Mahajan† Miguel Castro‡ Antony Rowstron‡

†University of Washington
Seattle, WA

‡Microsoft Research
Cambridge, UK

Abstract—**Structured peer-to-peer overlay networks provide a useful substrate for building distributed applications but there are general concerns over the cost of maintaining these overlays. The current approach is to configure the overlays statically and conservatively to achieve the desired reliability even under uncommon adverse conditions. This results in high cost in the common case, or poor reliability in worse than expected conditions. We analyze the cost of overlay maintenance in realistic dynamic environments and design novel techniques to reduce this cost by adapting to the operating conditions. With our techniques, the concerns over the overlay maintenance cost are no longer warranted. Simulations using real traces show that they enable high reliability and performance even in very adverse conditions with low maintenance cost.**

I. Introduction

Structured peer-to-peer (p2p) overlay networks (e.g., [6, 12, 7, 14]) are a useful substrate for building distributed applications because they are scalable, self-organizing and reliable. They provide a hash table like primitive to route messages using their keys. These messages are routed in a small number of hops using small per-node routing state. The overlays update routing state automatically when nodes join or leave, and can route messages correctly even when a large fraction of the nodes crash or the network partitions.

But scalability, self-organization, and reliability have a cost; nodes must consume network bandwidth to maintain routing state. There is a general concern over this cost [10, 11] but there has been little work studying it. The current approach is to configure the overlays statically and conservatively to achieve the desired reliability and performance even under uncommon adverse conditions. This results in high cost in the common case, or poor reliability in worse than expected conditions.

This paper studies the cost of overlay maintenance in realistic environments where nodes join and leave the system continuously. We derive analytic models for routing reliability and maintenance cost in these dynamic conditions.

This work was done while Ratul Mahajan was visiting Microsoft Research. Emails: ratul@cs.washington.edu, {mcastro,antr}@microsoft.com

We also present novel techniques that reduce the maintenance cost by observing and adapting to the environment. First, we describe a self-tuning mechanism that minimizes the overlay maintenance cost given a performance or reliability target. The current mechanism minimizes the probe rate for fault detection given a target message loss rate. It estimates both the failure rate and the size of the overlay, and uses the analytic models to compute the required probe rate. Second, we present mechanisms to effectively and efficiently deal with uncommon conditions such as network partitions and extremely high failure rates. These mechanisms enable the use of less conservative overlay configurations with lower maintenance cost. Though presented in the context of Pastry [7, 2], our results and techniques can be directly applied to other overlays.

Our results show that concerns over the maintenance cost in structured p2p overlays are not warranted anymore. It is possible to achieve high reliability and performance even in adverse conditions with low maintenance cost. In simulations with a corporate network trace [1], over 99% of the messages were routed efficiently while control traffic was under 0.2 messages per second per node. With a much more dynamic Gnutella trace [10], similar performance levels were achieved with a maintenance cost below one message per second per node most of the time.

The remainder of the paper is organized as follows. We provide an overview of Pastry and our environmental model in Section II, and present analytic reliability and cost models in Section III. Techniques to reduce maintenance cost appear in Section IV. In Section V we discuss how our techniques can be generalized for other structured p2p overlays. Related work is in Section VI, and conclusions in Section VII.

II. Background

This section starts with a brief overview of Pastry with a focus on aspects relevant to this paper. Then, it presents our environment model.

Pastry Nodes and objects are assigned random identifiers from a large sparse 128-bit *id space*. These identifiers are called *nodeIds* and *keys*, respectively. Pastry provides a primitive to send a message to a key that routes the message to the live node whose nodeId is numerically closest to the key in the id space.

The routing state maintained by each node consists of the *leaf set* and the *routing table*. Each entry in the routing state contains the nodeId and IP address of a node. The leaf set contains the $l/2$ neighboring nodeIds on either side of the local node's nodeId in the id space. In the routing table, nodeIds and keys are interpreted as unsigned integers in base 2^b (where b is a parameter with typical value 4). The routing table is a matrix with $128/b$ rows and 2^b columns. The entry in row r and column c of the routing table contains a nodeId that shares the first r digits with the local node's nodeId, and has the $(r+1)$th digit equal to c. If there is no such nodeId or the local nodeId satisfies this constraint, the entry is left empty. On average only $log_{2^b} N$ rows have non-empty entries.

Pastry routes a message to a key using no more than $log_{2^b} N$ hops on average. At each hop, the local node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the local node's nodeId. If no such node is known, the message is forwarded to a node whose nodeId is numerically closer to the key and shares a prefix with the key at least as long. If there is no such node, the message is delivered to the local node.

Pastry updates routing state when nodes join and leave the overlay. Joins are handled as described in [2] and failures are handled as follows. Pastry uses periodic probing for failure detection. Every node sends a keep-alive to the members of its leaf set every T_{ls} seconds. Since the leaf set membership is symmetric, each node should receive a keep-alive message from each of its leaf set members. If it does not, the node sends an explicit probe and assumes that a member is dead if it does not receive a response within T_{out} seconds. Additionally, every node sends a liveness probe to each entry in its routing table every T_{rt} seconds. Since routing tables are not symmetric, nodes respond to these probes. If no response is received within T_{out}, another probe is sent. The node is assumed faulty if no response is received to the second probe within T_{out}.

Faulty entries are removed from the routing state but it is necessary to replace them with other nodes. It is sufficient to replace leaf set entries to achieve correctness but it is important to replace routing table entries to achieve logarithmic routing cost. Leaf set replacements are obtained by piggybacking information about leaf set membership in keep-alive messages. Routing table maintenance is performed by periodically asking a node in each row of the routing table for the corresponding row in its routing table, and when a routing table slot is found empty during routing, the next hop node is asked to return any entry it may have for that slot. These mechanisms are described in more detail in [2].

ENVIRONMENT MODEL Our analysis and some of our cost reduction techniques assume that nodes join according to a Poisson process with rate λ and leave according to an exponential distribution with rate parameter μ (as in [4]). But we also evaluate our techniques using realistic node arrival and departure patterns and simulated massive correlated failures such as network partitions. We assume a fail-stop model and conservatively assume that all nodes leave ungracefully without informing other nodes and that nodes never return with the same nodeId.

III. RELIABILITY AND COST MODELS

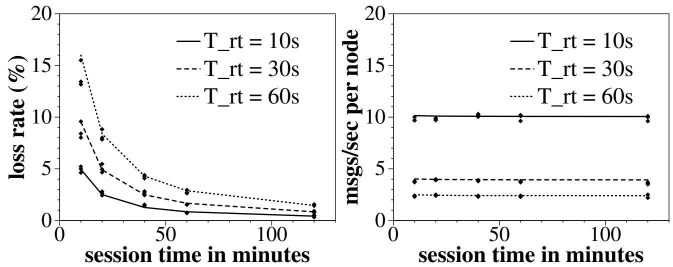
Pastry forwards messages using UDP with no acknowledgments by default. This is efficient and simple, but messages forwarded to a faulty node are lost. The probability of forwarding a message to a faulty node at each hop is $P_f(T,\mu) = 1 - \frac{1}{T\mu}(1 - e^{-T\mu})$, where T is the maximum time it takes to detect the fault. There are no more than $log_{2^b} N$ overlay hops in a Pastry route on average. Typically, the last hop uses the leaf set and the others use the routing table. If we ignore messages lost by the underlying network, the message loss rate, L, is:

$$L = 1-(1-P_f(T_{ls}+T_{out},\mu)).(1-P_f(T_{rt}+2T_{out},\mu))^{log_{2^b} N - 1}$$

Reliability can be improved by applications if required [9]. Applications can retransmit messages and set a flag indicating that they should be acknowledged at each hop. This provides very strong reliability guarantees [2] because nodes can choose an alternate next hop if the previously chosen one is detected to be faulty. But waiting for timeouts to detect that the next hop is faulty can lead to very bad routing performance. Therefore, we use the message loss rate, L, in this paper because it models both performance and reliability – the probability of being able to route efficiently without waiting for timeouts.

We can also derive a model to compute the cost of maintaining the overlay. Each node generates control traffic for five operations: leaf set keep-alives, routing table entry probes, node joins, background routing table maintenance, and locality probes. The control traffic in our setting is dominated by the first two operations. So for simplicity, we only consider the control traffic per second per node, C, due to leaf set keep-alives and routing table probes:

$$C = \frac{l}{T_{ls}} + \frac{2 \times \sum_{r=0}^{\frac{128}{b}}((2^b - 1) \times (1 - b(0; N, \frac{1}{(2^b)^{(r+1)}})))}{T_{rt}}$$



(a) loss rate (b) control traffic

Fig. 1. Verifying loss rate and control traffic models.

The first term is the cost of leaf set keep-alives: l keep-alives every T_{ls} seconds. The second is the cost of routing table probing: two messages (probe and response) for each routing table entry every T_{rt}. The summation computes the expected number of routing table entries, where $b(k;n,p)$ is the binomial distribution.

We verified these equations using simulation. We started by creating a Pastry overlay with 10,000 nodes. Then we let new nodes arrive and depart according to a Poisson processes with the same rate to keep the number of nodes in the system roughly constant. After ten simulated minutes, 500,000 messages were sent over the next ten minutes from randomly selected nodes to randomly selected keys. Figure 1a shows the message loss rate for three different values of T_{rt} (10, 30 and 60 seconds) with T_{ls} fixed at 30 seconds. The *x-axis* shows the mean session lifetime of a node ($\mu = 1/lifetime$). The lines correspond to the values predicted with the loss rate equation and the dots correspond to the simulation results (three simulation runs for each parameter setting). Figure 1b shows a similar graph for control traffic. The results show that both equations are quite accurate. As expected, the loss rate decreases when T_{rt} (or T_{ls}) decrease but the control traffic increases.

IV. REDUCING MAINTENANCE COST

This section describes our techniques to reduce the amount of control traffic required to maintain the overlay. We start by motivating the importance of observing and adapting to the environment by discussing the characteristics of realistic environments. Then, we explain the self-tuning mechanism and the techniques to deal with massive failures.

A. Node arrivals and departures in realistic environments

We obtained traces of node arrivals and failures from two recent measurement studies of p2p environments. The first study [10] monitored 17,000 unique nodes in the Gnutella overlay over a period of 60 hours. It probed each node every seven minutes to check if it was still part of the overlay. The average session time over the trace was approximately 2.3 hours and the number of active nodes in the overlay varied between 1300 and 2700. Figure 2a shows the failure rate over the period of the trace averaged over 10 minute intervals. The arrival rate is similar. There are large daily variations in the failure rate of more than a factor of 3.

The Gnutella trace is representative of conditions in an open Internet environment. The second trace [1] is representative of a more benign corporate environment. It monitored 65,000 nodes in the Microsoft corporate network, probing each node every hour for a month. The average session time over the trace was 37.7 hours. This trace showed large daily as well as weekly variations in the failure rate, presumably because machines are switched off during nights and weekends.

These traces show that failure rates vary significantly with both daily and weekly patterns, and the failure rate in the Gnutella overlay is more than an order of magnitude higher than in the corporate environment. Therefore, the current static configuration approach would require not only different settings for the two environments, but also expensive configurations if good performance is desired at all times. The next sections show how to achieve high reliability with lower cost in all scenarios.

B. Self-tuning

The goal of a self-tuning mechanism is to enable an overlay to operate at the desired trade-off point between cost and reliability. In this paper we show how to operate at one such point – achieve a target routing reliability while minimizing control traffic. The methods we use to do this can be easily generalized to make arbitrary trade-offs.

In the loss rate and control traffic equations, there are four parameters that we can set: T_{rt}, T_{ls}, T_{out}, and l. Currently, we choose fixed values for T_{ls} and l that achieve the desired resilience to massive failures (Section IV-C). T_{out} is fixed at a value higher than the maximum expected round trip delay between two nodes; it is set to 3 seconds in our experiments (same as the TCP SYN timeout). We tune T_{rt} to achieve the specified target loss rate by periodically recomputing it using the loss rate equation with the current estimates of N and μ. Below we describe mechanisms that, without any additional communication, estimate N and μ.

We use the density of nodeIds in the leaf set to estimate N. Since nodeIds are picked randomly with uniform probability from the 128-bit id space, the average distance between nodeIds in the leaf set is $2^{128}/N$. It can be shown that this estimate is within a small factor of N with very

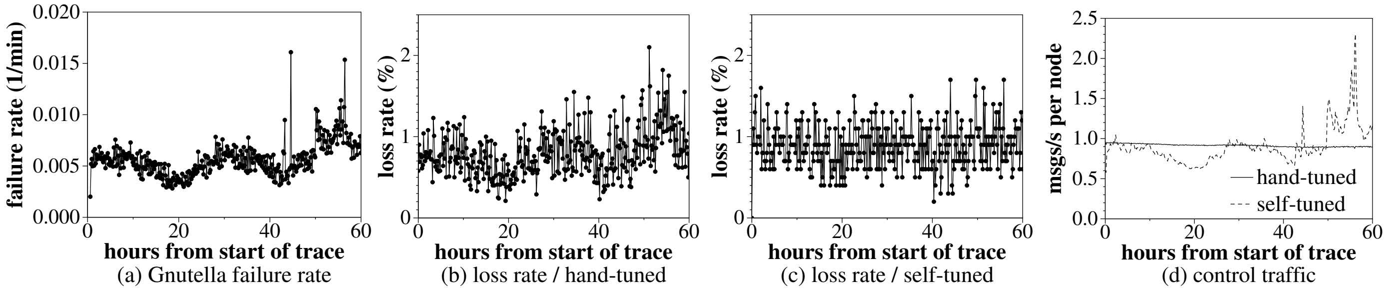


Fig. 2. Self-tuning: (a) shows the measured failure rate in the Gnutella network; (b) and (c) show the loss rate in the hand-tuned and self-tuned versions of Pastry; and (d) shows the control traffic in the two systems.

high probability, which is sufficient for our purposes since the loss rate depends only on $log_{2^b} N$.

The value of μ is estimated by using node failures in the routing table and leaf set. If nodes fail with rate μ, a node with M unique nodes in its routing state should observe K failures in time $\frac{K}{M\mu}$. Every node remembers the time of the last K failures. A node inserts its current time in the history when it joins the overlay. If there are only $k < K$ failures in the history, we compute the estimate as if there was a failure at the current time. The estimate of μ is $\frac{k}{M \times T_{k_f}}$, where T_{k_f} is the time span between the first and the last failure in the history.

The accuracy of μ's estimate depends on K; increasing K increases accuracy but decreases responsiveness to changes in the failure rate. We improve responsiveness when the failure rate decreases by using the current estimate of μ to discard old entries from the failure history that are unlikely to reflect the current behavior of the overlay. When the probability of observing a failure given the current estimate of μ reaches a threshold (e.g., 0.90) without any new failure being observed, we drop the oldest failure time from the history and compute a new estimate for μ.

We evaluated our self-tuning mechanism using simulations driven by the Gnutella trace. We simulated two versions of Pastry: *self-tuned* uses the self-tuning mechanism to adjust T_{rt} to achieve a loss rate of 1%; and *hand-tuned* sets T_{rt} to a fixed value that was determined by trial and error to achieve the same average loss rate. Hand-tuning is not possible in real settings because it requires perfect knowledge about the future. Therefore, a comparison between these two versions of Pastry provides a conservative evaluation of the benefits of self-tuning.

Figures 2b and 2c show the loss rates achieved by the self-tuned and the best hand-tuned versions, respectively. The loss rate is averaged over 10 minute windows, and is measured by sending 1,000 messages per minute from random nodes to random keys. T_{ls} was fixed at 30 seconds in both versions and T_{rt} at 120 seconds for hand-tuned. The results show that self-tuning works well, achieving the target loss rate independent of the failure rate.

Figure 2d shows the control traffic generated by both the hand-tuned and the self-tuned versions. The control traffic generated by hand-tuned is roughly constant whereas the one generated by self-tuned varies according to the failure rate to meet the loss rate target. It is interesting to note that the loss rate of hand-tuned increases significantly above 1% between 52 and 58 hours due to an increased failure rate. The control traffic generated by the self-tuned version clearly increases during this period to achieve the target loss rate with the increased failure rate. If the hand-tuned version was instead configured to always keep loss rate below 1%, it would have generated over 2 messages per second per node all the time.

We simulated the Microsoft corporate network trace also and obtained similar results. The self-tuned version achieved the desired loss rate with under 0.2 messages per second per node. The hand-tuned version required a different setting for T_{rt} as the old value would have resulted in an unnecessarily high overhead.

C. Dealing with massive failures

Next we describe mechanisms to deal with massive but rare failures such as network partitions.

BROKEN LEAF SETS Pastry relies on the invariant that each node has at least one live leaf set member on each side. This is necessary for the current leaf set repair mechanism to work. Chord relies on a similar assumption [12]. Currently, Pastry uses large leaf sets ($l = 32$) to ensure that the invariant holds with high probability even when there are massive failures and the overlay is large [2].

We describe a new leaf set repair algorithm that uses the entries in the routing table. It can repair leaf sets even when the invariant is broken. So it allows the use of smaller leaf sets, which require less maintenance traffic.

The algorithm works as follows. When a node n detects that all members in one side of its leaf set are faulty, it selects the nodeId that is numerically closest to n's nodeId on that side from among all the entries in its routing state. Then it asks this seed node to return the entry in its routing state with the nodeId closest to n's nodeId that lies between the seed's nodeId and n's nodeId in the id space. This process is repeated until no more live nodes with closer nodeIds can be found. The node with the closest nodeId is then inserted in the leaf set and its leaf set is used to complete the repair. To improve the reliability of the repair process, we explore several paths in parallel starting with different seeds.

The expected number of rounds to complete the repair is $log_{2^b}N$. We improve convergence by adding a *shadow leaf set* to the routing state. The shadow leaf set of node n contains the $l/2$ nodes in the right leaf set of its furthest leaf on the right, and the $l/2$ nodes in the left leaf set of its furthest leaf on the left. This state is acquired at no additional cost as leaf set changes are already piggybacked on keep-alive messages, and is inexpensive to maintain since nodes in it are not directly probed. Most leaf set repairs complete in one round using the nodes in the shadow leaf set. We also use the shadow leaf set to increase the accuracy of the estimate of N.

The experiments in this paper use a small leaf set with $l = 8$. This is sufficient to ensure reliable operation with high probability even when half the nodes fail simultaneously. At the same time it ensures that the leaf set repair algorithm does not need to be invoked very often. For example, the probability of all nodes in half of a leaf set failing within $T_{ls} = 30s$ is less than 10^{-10} with the average session time in the Gnutella trace.

MASSIVE FAILURE DETECTION The failure of a large number of nodes in a small time interval results in a large number of faulty entries in nodes' routing tables, which increases loss rate. This is resolved when nodes detect that the entries are faulty and repair the routing tables. But it can take very long in environments with low average failure rate because the self-tuning mechanism sets T_{rt} to a large value. We reduce the time to repair routing tables using a mechanism to detect massive failures.

The members of a leaf set are randomly distributed throughout the underlying network. So a massive failure such as a network partition is likely to manifest itself as failure of several nodes in the leaf set within the same probing period. When a node detects a number of faults in the same probing period that exceeds a specified fraction of l, it signals the occurrence of a massive failure and probes all entries in its routing table (failures discovered during this probing are not used for estimating μ as the background failure rate has not changed). This reduces the time to repair the routing tables after the failure to $T_{ls} + T_{out}$ seconds. We set an upper bound on the value of T_{ls} that achieves the desired repair time. In the experiments described in this paper, the threshold on the number of faults for failure detection is 30% and $T_{ls} = 30s$.

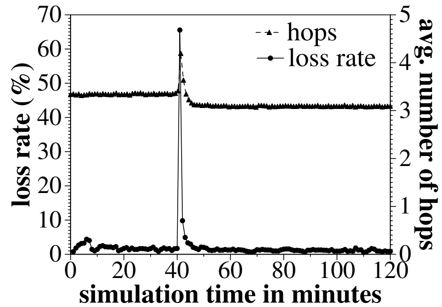


Fig. 3. Impact of half the nodes in the overlay failing together.

We evaluated our mechanisms to deal with massive failures using a simulation with 10,000 nodes where we failed half the nodes 40 minutes into the simulation. In addition to this failure, nodes were arriving and departing the overlay according to a Poisson process with a mean lifetime of 2 hours. 10,000 messages per minute were sent from random sources to random keys. Figure 3 shows the average loss rate and number of overlay hops in each minute. There is a large peak in the loss rate when the massive failure occurs but Pastry is able to recover in about a minute. Recovery involved not only detecting the partition and repairing routing tables but also repairing several broken leaf sets.

V. APPLICABILITY BEYOND PASTRY

The work presented in this paper is relevant to other structured p2p overlay networks as well. In this section, we briefly outline how it applies to other networks. Due to space constraints, we only describe how it can be applied to CAN [6] and Chord [12], and we assume that the reader has a working knowledge of CAN and Chord.

The average number of hops in CAN is $\frac{d}{4}N^{1/d}$ (where d is the number of dimensions) and is $\frac{1}{2}log_2 N$ in Chord. We can use these equations to compute the average loss rate for a given failure rate (μ) and number of nodes (N) as we did for Pastry. The average size of the routing state is $2d$ in CAN and $log_2 N + l$ (where l is the successor set size) in Chord. We can use these equations and the probing rates used in CAN and Chord to compute the amount of control traffic.

Self-tuning requires an estimate of N and μ. Our approach for estimating N using density can be generalized easily – the size of local and neighboring zones, and the density of the successor set can be used respectively for CAN and Chord. We can estimate μ in CAN and Chord using the failures observed in the routing state exactly as we did in Pastry.

The leaf set repair algorithm applies directly to Chord's successor set. A similar iterative search mechanism can be used when a CAN node loses all its neighbors along one dimension (the current version uses flooding). Finally, partition detection can be done using the successor set in Chord and neighbors in CAN.

VI. Related Work

Most previous work has studied overlay maintenance under static conditions but the following studied dynamic environments where nodes continuously join and leave the overlay. Saia et al use a butterfly network to build an overlay that routes efficiently even with large adversarial failures provided that the network keeps growing [8]. Pandurangan et al present a centralized algorithm to ensure connectivity in the face of node failures [5]. Liben-Nowell et al provide an asymptotic analysis of the cost of maintaining Chord [12]. Ledlie et al [3] present some simulation results of Chord in an idealized model with Poisson arrivals and departures. We too study the overlay maintenance cost in dynamic environments but we provide an exact analysis in an idealized model together with simulations using real traces. Additionally, we describe new techniques to reduce this cost while providing high reliability and performance.

Weatherspoon and Kubiatowicz have looked at efficient node failure discovery [13]; they propose that nodes further away be probed less frequently to reduce wide area traffic. In contrast, we reduce the cost of failure discovery through adapting to the environment. The two approaches can potentially be combined though their approach makes the later hops in Tapestry (and Pastry) less reliable, with messages more likely to be lost after having been routed for a few initial hops.

VII. Conclusions and Future Work

There are general concerns over the cost of maintaining structured p2p overlay networks. We examined this cost in realistic dynamic conditions, and presented novel techniques to reduce this cost by observing and adapting to the environment. These techniques adjust control traffic based on observed failure rate and they detect and recover from massive failures efficiently. We evaluated these techniques using mathematical analysis and simulation with real traces. The results show that concerns over the overlay maintenance cost are no longer warranted. Our techniques enable high reliability and performance even in adverse conditions with low maintenance cost. Though done in the context of Pastry, this work is relevant to other structured p2p networks such as CAN, Chord and Tapestry.

As part of ongoing work, we are exploring different self-tuning goals and methods. These include $i)$ operating at arbitrary points in the reliability vs. cost curve and using different performance or reliability targets; $ii)$ choosing a self-tuning target that takes into account the application's retransmission behavior, such that total traffic is minimized; and $iii)$ varying T_{ls} (has implications for detecting leaf set failures since keep-alives are unidirectional) along with T_{rt} under the constraint that T_{ls} has an upper bound determined by the desired resilience to massive failures. We are also studying the impact of failures on other performance criteria such as locality.

Acknowledgements

We thank Ayalvadi Ganesh for help with mathematical analysis, and John Douceur and Stefan Saroiu for the trace data used in this paper.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *ACM SIGMETRICS*, June 2000.

[2] 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.

[3] J. Ledlie, J. Taylor, L. Serban, and M. Seltzer. Self-organization in peer-to-peer systems. In *ACM SIGOPS European Workshop*, Sept. 2002.

[4] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *ACM Principles of Distributed Computing (PODC)*, July 2002.

[5] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. In *IEEE FOCS*, Oct. 2001.

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

[7] 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.

[8] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *IPTPS*, Mar. 2002.

[9] J. Saltzer, D. Reed, and D. Clarke. End-to-end arguments in system design. *ACM TOCS*, 2(4), Nov. 1984.

[10] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, Jan. 2002.

[11] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, Nov. 2002.

[12] 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.

[13] H. Weatherspoon and J. Kubiatowicz. Efficient heartbeats and repair of softstate in decentralized object location and routing systems. In *ACM SIGOPS European Workshop*, Sept. 2002.

[14] 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.