

# Query Protocols for Highly Resilient Peer-to-Peer Networks

Suresh Jagannathan, Gopal Pandurangan, Sriram Srinivasan  
Department of Computer Science  
Purdue University  
West Lafayette, IN 47906  
Email: {suresh,gopal,ssriniv}@cs.purdue.edu

**Abstract**—The decentralized and *ad hoc* nature of peer-to-peer (P2P) networks means that both the structure of the network, and the content stored within it are highly variable. Real-world studies indicate that only a small number of peers remain persistent over significant time periods, and that the perceived importance of objects stored in the network, measured in terms of access or update frequency, may not follow a uniform distribution.

In this paper, we present WARP, a P2P system that exploits these distinctions as an integral part of its design. WARP employs a novel fault-tolerant mechanism to manage the dynamic nature of node arrivals and departures by allowing multiple physical nodes to service data mapped to a single node in the overlay. Moreover, the overlay supports different query types, distinguishing queries to popular or valuable data from queries to unpopular or less valuable data.

We prove via a rigorous stochastic analysis that any query, regardless of type, will be successfully serviced with high probability. Further, we show that for a network with  $N$  nodes, the hop complexity of the protocol is  $O(\log N)$  with high probability. We also define bandwidth complexity, a measure of congestion at any node, and prove that it is  $O(\log^3 N)$  with high probability. We provide a detailed simulation of the system and show that it conforms closely to our theoretical guarantees.

## I. INTRODUCTION

A P2P networked system is a collaborating group of Internet nodes which overlay their own special-purpose network on top of the Internet. Such a system performs application-level routing on top of IP routing. These systems, like the Internet itself, can be large, require distributed control and configuration, and have a routing mechanism that allows each node to communicate with the rest of the system. P2P networks are emerging as a significant vehicle for providing distributed services (e.g., search, content integration and administration) [6], [7], [10], [20]. Some of the benefits of these systems are: decentralized computing (e.g., search), sharing data and resources, and serverless computing [2]. Various research groups have recently designed a variety of P2P systems including those that support fast look-up services [4], [23], provide large-scale network storage [15], anonymous publishing [7], and application-level multicast [5].

An important feature of P2P networks is their dynamically changing topology [4], [11]: peers enter and leave the network for various reasons (including failures) and connections may be added or deleted at any time. Recent measurement studies [29], [30] show that the peer turnaround rate is quite

high: nearly 50% of peers in real-world networks can be replaced within an hour, though the total number of peers in the network is relatively stable. These studies also indicate that P2P networks exhibit a high degree of variance in terms of the traffic volume, query distribution, and bandwidth usage generated by peers over time, and that many node and data characteristics may follow Zipfian or lognormal distributions [19].

Thus, to be useful, these systems must address a number of important and complex issues that ensure efficient and reliable routing in the presence of a dynamically changing network:

- 1) The overlay must exhibit good topological properties (e.g., connectivity, low diameter, low degree, etc.) even if the composition of the underlying physical network exhibits significant change.
- 2) Queries for data objects in the system must be serviced efficiently, and should scale with network size. Thus, non-scalable techniques such as network broadcast or flooding [8], [24] which may be appropriate in more constrained centralized environments, would be ineffective for wide-area deployment.
- 3) Because the system dynamics of these networks is also highly asymmetric with only a small number of peers persistent over significant time periods, providing fault-tolerance in the presence of mostly short-lived peers is essential.
- 4) The decentralized and *ad hoc* nature of these networks means that both the structure of the network, and the content stored within it are highly variable. Devising an overlay sensitive to the statistics of node characteristics (such as bandwidth, serving capacity, and “on” times – duration of connectivity) and of temporal characteristics of queried objects (such as access frequency and update frequency) is therefore critical.

These issues have each been addressed separately to some degree in previous work. For example, Ledlie *et al.* [16] discusses self-organization schemes for P2P systems driven by changing global characteristics of the network (issue 1). Pandurangan *et al.* give a protocol to build connected, low-diameter, constant degree unstructured P2P networks [21] under a realistic dynamic setting (issue 1). Structured P2P systems such as Chord [4], CAN [23], Pastry [26], Tapestry

[32], Viceroy [18] and the Plaxton *et al.* protocol [22] use distributed hashing to tightly couple the content of an object with the node in the P2P overlay where it should reside, enabling search algorithms to scale efficiently with network size (issue 2). Aspnes *et al.* [3] gives a general scheme to construct an overlay network by embedding a random graph in a metric space (issue 2). Cohen and Shenker [8] discuss replication strategies for improving performance in unstructured P2P networks such as Gnutella and Freenet (issue 2). Saia *et al.* [27] discuss techniques to improve fault-tolerance by devising a topology that creates multiple highly-redundant routes among peers; Liben-Nowell *et al.* [17] presents maintenance protocols that continuously repairs a Chord overlay as nodes leave and enter the system (issue 3). Xu *et al.* [31] defines a two-level overlay to take advantage of node and bandwidth heterogeneity in the underlying physical network; Kaaza [13] uses a multi-level overlay for similar reasons (issue 4).

In this paper, we present WARP, a novel peer-to-peer system that explicitly addresses each of these issues as integral features of its design. Among the efforts cited above, WARP is closest in spirit to the virtual content addressable network described by Fiat and Saia [9] and Saia *et al.* [27]. For an  $O(N)$  sized network, they define a  $O(\log N)$  latency and  $O(\log^3 N)$  degree fault-tolerant overlay. Their work guarantees that a large number of data items are available even if a large fraction of peers are deleted, under the assumption that, in each time step, the number of peers deleted by an adversary must be smaller than the number of peers joining. WARP, on the other hand, guarantees that *every* search succeeds with high probability<sup>1</sup> at any time, rather than simply a large fraction, assuming a natural and general  $M/G/\infty$  model [25] (i.e., the holding times of nodes can have any distribution, and is thus much more general than, for example, the  $M/M/\infty$  model of [21]). The construction of our overlay is also simpler as described in the following section.

The remainder of this paper is structured as follows. The next section presents an overview of the system. Section III describes the overlay and the query protocols. Section IV analyzes its complexity. Section V presents simulation results that validate our theoretical bounds. Conclusions are given in Section VI.

## II. OVERVIEW

The WARP overlay is defined as an embedding of  $k$  copies of a complete binary tree on itself in a random fashion. Nodes which correspond to a root of the tree are designated to hold highly valuable data. Non-roots serve as caches for root nodes, and may hold less valuable data. We leave unspecified the mechanism by which nodes are mapped to vertices in the graph, observing that regardless of the exact algorithm used to formulate the mapping, there is no fixed *a priori* determined route between the source of a query and a target since any node, including a root, may leave the overlay at any time.

<sup>1</sup>Throughout this paper w.h.p. (with high probability) denotes probability at least  $1 - N^{-b}$ , for some constant  $b \geq 1$ .

WARP’s assymmetric overlay structure make it reasonable to expect that nodes with high availability and bandwidth characteristics get mapped to roots or nodes near them, and that nodes with poor availability and bandwidth get mapped to leaves. While devising effective mappings is likely to be important in practice, our analysis does not consider node heterogeneity in deriving the overlay’s latency and complexity bounds. Indeed, we show that even with uniform random placement, the structure of the overlay provides sufficient redundancy to support an efficient query protocol that ensures any request will be successfully serviced with high probability.

Queries are logically classified as either *centralized*, for queries that target data stored on a root, or *distributed*, for queries that access data on non-roots using a content-based distributed hashing scheme [12]. A centralized query could be internally generated by a non-root node in response to a distributed query targeted to it. This may occur if the target does not have the data of interest because of insufficient storage (i.e., cache miss), or if the data is never stored locally (i.e., it is non-cacheable). We assume that applications running on WARP initially determine the query class to which a particular data object belongs. Although we only consider centralized and distributed queries here, there is no reason why more refined classes cannot be supported. For example, objects which are initially the targets of centralized queries may over time be reclassified as targets for distributed queries if their importance to the application diminishes. Such refinements add no interesting complications to the protocol or analysis.

When nodes depart the WARP overlay, they leave a *hole* in the graph that can be subsequently filled by a replacement. Informally, a hole represents a placeholder that can be occupied by a number of nodes. Any node in the set of nodes which cover a hole can service queries for data mapped to the hole. The cardinality of this set,  $k$ , defines a measure of the fault-tolerance provided by the overlay. Data mapped to a hole is thus replicated on the  $k$  nodes which cover it. Although  $k$  is a tunable parameter of the network, we show that small  $k$ , logarithmic in the total number of nodes in the underlying network, is sufficient to ensure that at least one path exists from a query source to a root for both centralized and distributed queries, with high probability.

Support for high availability and fault-tolerance is an important distinguishing design feature of WARP; the same mechanism used to manage the dynamic nature of node arrival and departures also permits queries to be serviced successfully with high probability. The topology of the network achieves a high degree of fault-tolerance in two ways: (1) the embedding ensures that every data object is recorded in each of the tree’s  $k$  copies; (2) a given node connects to a distinct parent in each of the tree’s  $k$  copies.

Latency and bandwidth overheads are measures of the network’s efficiency in servicing a query. Latency overhead measures the cost of resolving a query in terms of the number hops taken by a query from source to target. Bandwidth complexity measures the number of messages serviced by a given node in a fixed time interval. Traffic complexity bounds

are not immediately obvious if queries can target any node. For a single tree,  $O(n)$  queries may need to be serviced by a single node in a fixed time interval. However, we show that by exploiting the structure of the embedding in which there are multiple random paths connecting nodes found in different subtrees, and by adding a small number of links among nodes in nearby subtrees (i.e., enforcing a small-world like network structure [11], [14]) to improve convergence, the system need only provide bandwidth for nodes to handle  $O(\log^3 N)$  queries at each timestep.

### A. Example

To motivate WARP’s design, imagine a distributed service that provides real-time information on stock prices. A client initiates a query to some node using a distributed hashing scheme; the hash may be computed based on the stock symbol, and the message may define whether a realtime or delayed quote is desired. The target node generates a centralized query for realtime quotes to a root. To ensure scalability, there is no point-to-point connection between non-root and root nodes; thus, centralized queries propagate through the network in the same way that distributed queries do. Delayed quotes are cached on non-roots and periodically refreshed. Since prices may frequently change, it is critical that there is some globally consistent view of what the latest quote is among all users of the service; this view is provided by centralized queries serviced by the system’s roots, and initiated by other nodes in response to client queries.

## III. THE WARP OVERLAY NETWORK AND PROTOCOL

Our protocol is based on an underlying randomized topology defined as follows. Consider a graph  $G = (V, E)$  of size  $S^2$  determined by embedding  $k$  copies of a complete, size  $S$  binary tree  $T$  on itself in a random fashion as explained below. Assume that vertices of  $T$  are labeled by a unique number according to a simple scheme: level  $i$  ( $i = 0$  corresponds to the root) in a tree is numbered by  $2^i - 1$  to  $2(2^i - 1)$  in a left to right fashion. A node<sup>3</sup>  $j$ ,  $1 \leq j \leq S$ , in  $G$  has a label called the *node-id* which is a  $k$ -tuple where the  $i$ th component corresponds to a numbered vertex in the tree  $T$ . The  $k$  components of a node-id are determined by independently sampling uniformly at random<sup>4</sup> from 1 to  $S$ . We say that the node *covers* the tree vertices corresponding to its  $k$  components. The nodes which cover the root (i.e., have 0 in any of its components) are called *root nodes*. There is an edge between two nodes  $x = \langle x_1, \dots, x_k \rangle$  and  $y = \langle y_1, \dots, y_k \rangle$  if for some  $i, j$ ,  $1 \leq i, j \leq k$ , there is a tree edge between  $x_i$  and  $y_j$  or  $x_i = y_j$ . We say that  $x$  is a parent (or child) node of  $y$  if for some  $i$ ,  $x_i$  is the parent (child) vertex of some  $y_j$  in the tree. We call these edges *random edges* and the adjacent nodes as *random neighbors*.

<sup>2</sup>Informally,  $S$  stands for an estimate of the number of peers in the network.

<sup>3</sup>We use the term *node* to denote a peer of the overlay. We reserve the term *vertex* for the vertices of the tree.

<sup>4</sup>We assume sampling with replacement for simplicity of analysis, although in practice it would be more efficient to do sampling without replacement.

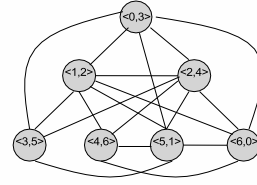


Fig. 1. A simple overlay graph with  $k = 2$ . In this graph, nodes labeled  $\langle 0, 3 \rangle$  and  $\langle 6, 0 \rangle$  are roots. Only random edges are shown.

In addition to the random edges we have the following, somewhat, non-intuitive set of edges. Let the level of a vertex  $x$  be  $l$  in tree  $T$ . Further, let  $T(p_i(x))$ ,  $0 \leq i \leq l - 1$  denote the subtree of  $T$  rooted at the  $i$ th ancestor of  $x$  ( $0$ th ancestor is the root of the tree), but not including the subtree hanging from  $i + 1$  itself. For example,  $T(p_{l-1}(x))$  denotes the tree rooted at the parent of  $x$ , but excluding the subtree hanging from  $x$  itself. We call these *ancestor subtrees*. For each vertex  $x$  in  $T$ , we have an edge between  $x$  and a random node in each of the trees  $T(p_i(x))$ ,  $0 \leq i \leq l$ . We call these *small-world edges* and the adjacent nodes as *small-world neighbors*. Two nodes  $x$  and  $y$  have an edge between them if there is a small-world edge between a vertex covered by  $x$  and a vertex covered by  $y$  (or if they share a common vertex).

The motivation comes from [14]; these edges in our scheme, as in that paper, are crucial in routing. However, there is an important difference: in [14], the “long-range” edges reduce the latency or the hop complexity of routing; in our scheme they help in reducing the bandwidth complexity (defined precisely in section IV). Intuitively, small-world edges allow routing of messages among nodes to bypass nodes at higher-levels in the tree, thus avoiding network congestion at roots and nodes close to them. In the above graph, if there is a small-world edge between vertex 3 and 6, then there will be edges between the node with node-id  $\langle 3, 5 \rangle$  and nodes with node-ids  $\langle 6, 0 \rangle$  and  $\langle 4, 6 \rangle$ . Similarly, there will be edges between the node with node-id  $\langle 0, 3 \rangle$  and nodes with node-ids  $\langle 6, 0 \rangle$  and  $\langle 4, 6 \rangle$ .

We call a vertex in  $T$  to be *occupied* if there is a peer or node in the network (i.e., a live peer) which covers this vertex, i.e., the vertex is one of the components of its node-id; otherwise we call it to be a *hole*.

### A. Joining and Leaving the Network

An incoming node (say  $v$ ) chooses a  $k$ -tuple node-id where each component of the tuple is an independent random sample between 1 and  $S$ <sup>5</sup>. Because of the numbering scheme,  $v$  can determine (by itself) the node-ids of its (potential) random neighbors. The node-id’s of small-world neighbors is chosen by  $v$  as follows: for each component vertex in the node-id, its small-world neighbors are chosen by sampling uniformly at random from the vertex numbers of the corresponding ancestor

<sup>5</sup>If  $v$  does not know  $S$ , it can find out an accurate estimate with high probability; see Section IV.

subtrees; this is easy, since the tree numbering is known. Thus,  $v$  determines the node-id's of its (potential) neighbors without any global knowledge.

Then,  $v$  contacts any one of the nodes in the network (found by some external mechanism<sup>6</sup>) and uses the distributed querying protocol (see Section III-B) to find its neighbors (i.e., their IP addresses) and joins by connecting to them. For every component vertex in its node-id, data is copied from any other node which shares this vertex.

A node can simply leave the network at any time; the node's data need not be transferred.

### B. Querying Schemes in WARP

The WARP protocol supports two types of querying schemes: *centralized* and *distributed*. Centralized queries go to one of the roots. The protocol is simple: a node sends a query to one of its live ancestors nodes which in turn forwards it to one of its ancestors, till a root node is reached. If all the ancestors of a node are not live (all the corresponding vertices are holes), then the query fails.

Distributed queries are handled by a distributed hashing scheme with a randomized routing strategy as follows. The data (or key) is hashed to a random number between 1 to  $S$  and inserted to all nodes having this number in any one of the components of its node-id.<sup>7</sup> Query for this data is thus directed to a node with one of its components equal to the data's hashed value. Actually, since all nodes sharing a vertex id are connected to each other, search will succeed even if only one node covering this vertex is live in the network. This is easily achieved because of the unique numbering scheme.

The routing for distributed queries is handled as follows. Suppose we have a query from a source node  $i$  to a target data hashed to a value  $t$ . Pick any vertex  $s$  covered by  $i$ . Assume  $t$  is not in the subtree rooted at  $s$ . Let  $lca(x, t)$  denote the least common ancestor of  $x$  and  $t$  in  $T$ . Let  $T(lca(x, t))$  denote the subtree rooted at  $lca(x, t)$  excluding the subtree (rooted at  $lca(x, t)$ ) containing  $x$  itself. The routing uses the small-world edges crucially – in step 2.2 they guarantee a neighbor in  $T(lca(x, t))$ . In step 2.2,  $i$  itself can cover a vertex in  $T(lca(x, t))$ , in which case the step is trivial. Note also that the numbering scheme easily allows us to find  $x$ . For simplicity, when we say "a neighbor of a vertex" we mean a node covering the neighbor of a vertex.

```

1  $x = s$ 
2 while  $x \neq t$  do
2.1 if  $t$  is in the subtree rooted at  $x$  then
2.1.1 route to  $t$  via the unique path to  $t$ 
2.1.2 break
2.2 if there is a live neighbor

```

<sup>6</sup>For example, in Gnutella [10] there is a central server that maintain list of host IP addresses which clients visit to get entry points into the P2P network; for example, <http://www.gnutella.com/> is a website which maintains a list of active Gnutella servants. New clients can join the network by connecting to one or more of these servants.

<sup>7</sup>It will follow from our analysis (4.4) that this replication scheme guarantees availability of data w.h.p at any point of time.

of  $x$  (say  $y$ ) in  $T(lca(x, t))$  then  
(If there is more than one such neighbor  
choose one of them randomly)

```

2.2.1 send query to  $y$ 
2.2.2  $x = y$ 
2.3 else
2.3.1 if  $\text{parent}(x)$  is live
        send query to  $\text{parent}(x)$  in  $T$ 
         $x = \text{parent}(x)$ 
2.3.2 else
        send query to  $r$ , a random neighbor of  $x$ 
         $x = r$ 
3 endwhile

```

## IV. ANALYSIS

In evaluating the performance of our protocol we focus on the long term behavior of the system in which nodes arrive and depart in an uncoordinated, and unpredictable fashion. We model this setting by a stochastic continuous-time process: the arrival of new nodes is modeled by Poisson distribution with rate  $\lambda$ , and the duration of time a node stays connected to the network is independently determined by an *arbitrary* distribution  $G$  with mean  $\mu$ . This is also called the  $M/G/\infty$  model in queuing theory. Recent measurement studies of real P2P systems [29], [30] indicate that the above model approximates real-life data reasonably well, especially since the holding time distribution is arbitrary. (these studies indicate that the holding times may follow Zipfian or lognormal distributions). The Poisson model has been used in [17] to motivate the *half-life* concept and in analyzing the dynamic evolution of P2P systems.

Let  $G_t$  be the network at time  $t$  ( $G_0$  has no vertices). We are interested in analyzing the evolution in time of the stochastic process  $\mathcal{G} = (G_t)_{t \geq 0}$ . Since the evolution of  $\mathcal{G}$  depends only on the ratio  $\lambda/\mu$  we can assume w.l.o.g. that  $\lambda = 1$ . To demonstrate the relation between these parameters and the network size, we use  $N = \lambda/\mu$  throughout the analysis. We justify this notation by showing that the number of nodes in the network rapidly converges to  $N$ . We use the notation  $G_t = (V_t, E_t)$  be the network at time  $t$ .

Throughout our analysis we use the Chernoff bounds for the binomial and the Poisson distributions. Let the random variable  $X$  denote the sum of  $n$  independent and identically distributed Bernoulli random variables each having a probability  $p$  of success. Then,  $X$  is binomially distributed with  $\mu = np$ . We have the following Chernoff bounds [1]: For  $0 < \delta < 1$

$$\Pr(X > (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}$$

$$\Pr(X < (1 - \delta)\mu) \leq e^{-\mu\delta^2/2}$$

We have identical bounds even when  $X$  is a Poisson random variable with parameter  $\mu$ .

The following theorem characterizes the network size and is a consequence of the fact that the number of nodes at any time  $t$  is a Poisson distribution (despite the fact that the

holding times follow an arbitrary distribution) [25, pages 18-19]; applying the Chernoff bound for the Poisson distribution gives the high probability result. We omit a formal proof here.

*Theorem 4.1 (Network Size):* 1) For any  $t = \Omega(N)$ , w.h.p.  $|V_t| = \Theta(N)$ .  
2) If  $\frac{t}{N} \rightarrow \infty$  then w.h.p.  $|V_t| = N \pm o(N)$ .

The above theorem assumed that the ratio  $N = \lambda/\mu$  was fixed during the interval  $[0, t]$ . We can derive a similar result for the case in which the ratio changes to  $N' = \lambda'/\mu'$  at time  $\tau$ .

*Theorem 4.2:* Suppose that the ratio between arrival and departure rates in the network changed at time  $\tau$  from  $N$  to  $N'$ . Suppose that there were  $M$  nodes in the network at time  $\tau$ , then if  $\frac{t-\tau}{N'} \rightarrow \infty$  w.h.p.  $G_t$  has  $N' + o(N')$  nodes.

The following lemma is a consequence of the randomized construction of the overlay topology. Thus the high probability bounds are with respect to this randomization.

*Lemma 4.1:* Let  $S = \Theta(N)$  and  $k = \Theta(\log N)$ . Then

- 1) The number of node-ids covering a given vertex is  $\Theta(\log N)$  w.h.p.
- 2) The routing table size of any node is bounded by  $O(\log^3 N)$  w.h.p.

**Proof:**

- 1) There are  $S$  node-ids generated by random sampling. Let  $Y_j$  be the indicator random variable for the event that node-id  $j$  covers a given vertex  $v$ . Then  $\Pr(Y_j = 1) = (1 - (1 - 1/S)^k)$ . Thus, by linearity of expectation, the expected number of node-ids covering a given vertex is  $S(1 - (1 - 1/S)^k) = \Theta(\log N)$ . Applying the Chernoff bound gives the high probability result.
- 2) Let  $x = (x_1, \dots, x_k)$  be a node-id of a node. The number of random edges incident on this node is  $\Theta(3k \log N) = \Theta(\log^2 N)$  since each vertex  $x_i$  is adjacent to at most 3 other vertices of the tree and each is covered by  $\Theta(\log N)$  node-ids w.h.p. There are  $\log S$  number of small-world neighbors of a given vertex. Thus, the number of small-world edges is bounded by  $\Theta(\log S k \log N) = \Theta(\log^3 N)$  w.h.p.

□

The following theorem follows from Lemma 4.1 and Theorem 4.1.

*Theorem 4.3 (Routing table size):* At any time  $t$  such that  $t/N \rightarrow \infty$ , the routing table size of any node (or the degree) is bounded by  $O(\log^3 N)$  w.h.p.

We state a key theorem about the presence of holes in our protocol.

*Theorem 4.4 (Occupancy of Holes):* Let  $S = cN$ , for some positive constant  $c$  and let  $k = a \log N$ , for a sufficiently large constant  $a$ . Then, at any time  $t$ , such that  $t/N \rightarrow \infty$ , w.h.p. every vertex in the overlay network is occupied.

**Proof:** When  $t/N \rightarrow \infty$  nodes depart the network according to a Poisson process with rate 1. Also from theorem 4.1, w.h.p.

the number of nodes in the network is at least  $N - o(N)$ . Since, every hole has an equal probability of getting filled at any time step  $t$ , the probability that a vertex is not covered is at most

$$\left(1 - \frac{1}{S}\right)^{k(N - o(N))} (1 - o(1/N)) \leq e^{\Theta(1 - o(1))} \leq 1/N^2$$

for a suitable choice of constant  $a$ . Applying Boole's inequality, the probability that no vertex is unoccupied is at most  $1/N$ . □

The following theorem on the success probability of a query is a consequence of the previous theorem and the way nodes link to each other.

*Theorem 4.5:* Let  $k = O(\log N)$ . Then for any time  $t$ , such that  $t/N \rightarrow \infty$ , w.h.p. any central or distributed query will be successful. Furthermore, the number of hops needed is  $O(\log N)$  w.h.p.

**Proof:** We focus on centralized querying. The proof for distributed querying is similar. Consider a query emanating from the node  $x$  with node-id  $(x_1, \dots, x_k)$ . This query will be successful if there is a path to one of the root nodes. In terms of the underlying tree  $T$ , this will occur if there is a path from any of the  $x_i$ 's to the root, in particular the path  $P = x_1, \text{parent}(x_1), \dots, \text{root}$ . From our previous theorem, since every hole of the tree is occupied w.h.p. every vertex in  $P$  is covered by some (live) node in the network. Furthermore, from our construction of the random edges, there is an edge between any node covering a vertex to any node covering the parent of the vertex. Thus, w.h.p. the query will take  $O(\log N)$ . □

The above corollaries also imply the following theorem on the connectivity and diameter of the network.

*Corollary 4.1:* Let  $k = O(\log N)$ . Then for any time  $t$ , such that  $t/N \rightarrow \infty$ , the network is connected and has a diameter of  $O(\log N)$  w.h.p.

*Corollary 4.2:* Let  $k = O(\log N)$ . Then for any time  $t$ , such that  $t/N \rightarrow \infty$ , the work needed when a node joins the network is  $O(\log^3 N)$  w.h.p.

**Proof:** For each vertex component in its node-id, an incoming node has to locate a node covering this vertex; then it can find all the random neighbors corresponding to this component. Since there are  $O(\log N)$  components w.h.p. and finding neighbors corresponding to one component takes  $O(\log N)$  time (Theorem 4.5), the total time needed to find all random neighbors is  $O(\log^2 N)$ . Since there are  $O(\log^2 N)$  small-world neighbors a similar argument yields the result. □

**Remarks.** We conclude with important remarks about the protocol, its implementation, and extensions.

- 1) From the proof of the above theorem it is clear that we need only a "reasonable" estimate (upto a constant factor) of the network size, as alluded to before. Then

choosing  $k = \Theta(\log n)$  will still be sufficient to guarantee the theorem. Thus, henceforth, we assume that  $S = cN$ , where  $c > 0$  is a constant.

- 2) When an incoming node joins the network we assumed that it knows  $S$  (Section III). This is actually not required: a node can sample a small subset of nodes (for example, by first contacting a node and then searching) and use Theorem 4.1. It is not difficult to show that only  $O(\log N)$  nodes need to be searched to get an accurate estimate with high probability.
- 3) The idea used in WARP (Theorems 4.4 and 4.5) can be applied to other underlying topologies as well; thus it can be used to “convert” any static topology into a dynamic fault-tolerant network. For example, we can show that applying the scheme to a butterfly network (i.e., the underlying template topology is a butterfly network instead of a tree) yields a  $O(\log N)$  latency (i.e., every search succeeds in  $O(\log N)$  time w.h.p.) and  $O(\log^2 N)$  degree network. This is an improvement in the degree size over the network of Saia et al. [27] as described in Section I. Thus, the additional  $O(\log N)$  factor in the degree of WARP is due to the presence of the small-world edges which are needed for reducing bandwidth complexity, as described in the following section, and not required for providing fault-tolerance *per se*. We explore an interesting variant of the WARP protocol which has  $O(\log^2 N)$  degree in Section V. This scheme reduces routing table size from  $\Theta(\log^3 N)$  to  $\Theta(\log^2 N)$  by allowing edges between two nodes  $x = \langle x_1, \dots, x_k \rangle$  and  $y = \langle y_1, \dots, y_k \rangle$  only if  $\exists_{1 \leq j \leq k} j$  such that there is a tree edge between  $x_j$  and  $y_j$ .
- 4) It can be shown that bad events (such as the network size exceeding  $S$ ) happen with minuscule probability. In such cases, temporary remedial measures can be taken such as generating new node-ids (by random sampling) or rejecting new connections till the situation self-corrects itself. Our analysis can be extended to handle such situations.

#### A. Bandwidth Complexity

We define the *bandwidth* complexity of the protocol as the worst-case expected number of queries that go through any node (i.e., use the node as an intermediate node) in a time step. We assume a uniform query distribution for analyzing distributed querying<sup>8</sup>:  $N$  queries are generated per time step, one per node, each query has a random destination independent of other queries. This is a natural distribution to analyze for two reasons: (1) the query rate, i.e., the number of queries per time step is much more than the rate of change of the network (i.e., the arrival and leaving rate), and every node is likely to generate a query in the worst case (2) under uniform hashing it is reasonable for a query to have

<sup>8</sup>The bandwidth complexity of centralized querying is  $O(1)$  since we assume that queries that go to the roots can be aggregated.

a random destination if queries are for different data, which is the appropriate scenario for doing a distributed search as opposed to a centralized search.

Let  $Q(x)$  be the number of queries that use  $x$  as an intermediate node under uniform query distribution (in one time step). Then the bandwidth complexity is  $B = \max_{x \in V} E[Q(x)]$ . We show that  $B$  is  $O(\log^3 N)$  for our protocol. This is somewhat non-intuitive – although it appears that the top nodes (near the root) will get  $O(N)$  queries, this happens with very low probability: most of the queries converge to their destinations by using the small-world edges, and thus avoiding the “usual” route of going through the top nodes. We also show, that using the random edges alone does not guarantee low traffic complexity. This is because, since the edges are randomly distributed, only the ancestor subtrees which are farther away from the destination are favored. On the other hand, the small-world edges favors all the ancestor subtrees uniformly.

*Theorem 4.6:* Let  $k = O(\log N)$  and  $S = cN$ , for a constant  $c > 1$ . At any time  $t$ , such that  $t/N \rightarrow \infty$ , w.h.p. the bandwidth complexity of the protocol is  $B = O(\log^3 N)$  for distributed querying under the uniform query distribution.

**Proof:** Since theorem 4.4 guarantees that w.h.p. there will be no hole in the network, it is enough if we show w.h.p. the bandwidth complexity is  $O(\log^3 N)$  assuming no holes.

We calculate the number of queries that go through an arbitrary node  $i$ , with respect to each vertex it covers. Let  $i$  cover a vertex  $x$  in level  $l$ . We denote the left subtree and the right subtree of  $x$  in  $T$  by  $T_l(x)$  and  $T_r(x)$  respectively. Let  $h = O(\log N)$  denote the height of the tree  $T$ .

We calculate case by case the expected number of queries that go through  $i$  depending on the source and destination of queries. We count the queries that go through  $i$  due to  $i$  covering  $x$ ; the total (expected) number of intermediate queries through  $i$  is multiplied by  $k$  since  $i$  covers  $k$  vertices.

- Case 1: We consider messages that have destination in the subtree of  $x$ , i.e., the target value is hashed to a vertex that is in the subtree rooted at  $x$ . There are two subcases depending on the origin:

(a) origin in a node (say  $s$ ) which is in the subtree rooted at  $x$ , i.e.,  $s$  covers a vertex (say  $y$ ) in the subtree rooted at  $x$ . Without loss of generality, let the message originate in  $T_l(x)$  and its destination be in  $T_r(x)$  (otherwise the message will never go through  $x$ ). Then the message will go through  $x$  if the small-world edge connects  $y$  to  $x$  (in  $T$ ) and  $i$  is chosen (among all the nodes covering  $x$ ). The expected number of messages is bounded by

$$O\left(\frac{2^{h-l-1}}{N}\right)2^{h-l-1} \frac{1}{2^{h-l-1}} \frac{1}{\Theta(\log N)} = \frac{1}{\Theta(\log N)}$$

(b) Messages which originate in  $T(p_{l-1}(x)), \dots, T(p_0(x))$ . The expected number which go through  $x$  is bounded by:

$$O\left(N \frac{2^{h-l}}{N} \frac{\lg N}{2^{h-l-1}}\right) = O(\log N)$$

since the message will pass through  $x$  if it reaches  $x$  or any of the ancestor nodes of  $x$ .

- Case 2: Messages which have destination in  $T(p_{l-1}(x)), \dots, T(p_0(x))$ . We note that only messages that originate in  $T(p_i(x))$  with destination in  $T(p_j(x))$ , where  $j > i$  have a chance of getting routed through  $x$ . Consider such messages. The expected number of messages that go through  $x$  is upper bounded by (note that messages that end in  $T_l(x)$  and  $T_r(x)$  do not get routed through  $x$ )

$$\sum_{i=0}^{l-1} \frac{|T(p_i(x))|}{N} \left( \sum_{j=i}^{l-1} (|T(p_j(x))| \frac{1}{|T(p_j(x))|}) \right) = O(\log^2 N)$$

Since  $x$  covers  $O(\log N)$  vertices w.h.p. the total upper bound is  $O(\log^3 N)$ .

□

### B. Why small-world edges?

We show that just having the random edges alone is not sufficient to guarantee polylogarithmic bandwidth complexity.

*Theorem 4.7:* Let  $k = O(\log N)$  and assume that we have only the random edges. Then at any time  $t$ , such that  $t/N \rightarrow \infty$ , the traffic complexity of distributed querying is  $\Omega(N)$  under the uniform query distribution.

**Proof:** Consider an arbitrary node  $x$  which covers a vertex at level  $l$ . We calculate a lower bound on the number of messages that go through  $x$ . Consider messages which originate in  $T_l(x)$  - the left subtree of  $x$ , and having a destination in the right subtree of  $x$ . Then the expected number of nodes which go through  $x$  is ( $h$  is the height of  $T$ )

$$\begin{aligned} & \sum_{d=1}^{h-l-1} \left(1 - \frac{2^{h-l}}{N}\right)^{kd} 2^d \frac{2^{h-l-1}}{N} \\ &= \frac{1}{2^{l+1}} \sum_{d=1}^{h-l-1} \left(2\left(1 - \frac{1}{2^l}\right)^k\right)^d \sim 2^{h-2l} e^{-k(h-l)/2^l} = \\ & \quad \Omega(N) \end{aligned}$$

if  $l = \Theta(\sqrt{\log N})$ . □

## V. SIMULATION RESULTS

To validate the analysis presented in the previous section and to obtain an estimate of the hidden constants in the analysis, we simulated the protocol by varying  $N$ , the size of the network, and the parameter  $k$ . We implemented a discrete event simulator in which node arrival and departure follows a Poisson distribution. Each simulation run consists of series of time steps. Nodes join and leave the network at the beginning of each time step. Queries are assumed to be made by randomly chosen active nodes before the beginning of next time step, and after all leave and join events have been handled. Queries are assumed to be successful if they reaches the destination; responses are not routed back to the requesting

node. Administrative messages exchanged between leaving (joining) nodes, roots and their neighbors are not accounted for in bandwidth calculations to adhere to the analysis presented. We consider three experiments. The first studies the fault tolerance of the overlay under different replacement rates; the second explores bandwidth and latency complexity of the protocol; and, the third investigates an alternative overlay structure with lower routing table size overheads.

To study the fault tolerant aspects of the overlay, we start with a network with  $2N$  holes, and subsequently fill it with  $N$  nodes. Such a network can be obtained by constructing a  $2N$  network, forcing  $N$  nodes to leave initially. Holes in the network, so obtained, are on average only 50% filled and thus the probability of having a discontinuous path is higher than the network configurations considered so far. The replacement rate was varied from 0.1 to 0.5. The graphs in Figure 5 show the fraction of successful queries over different replacement counts assuming  $N = 10^4$  nodes. The replacement count indicates the number of nodes that remain in the network between consecutive timesteps; high replacement counts thus imply large change in the underlying network. Figure 2 indicates that there is enough redundancy imposed by the embedding to ensure 100% delivery when  $k$  is  $(\log N)$ .

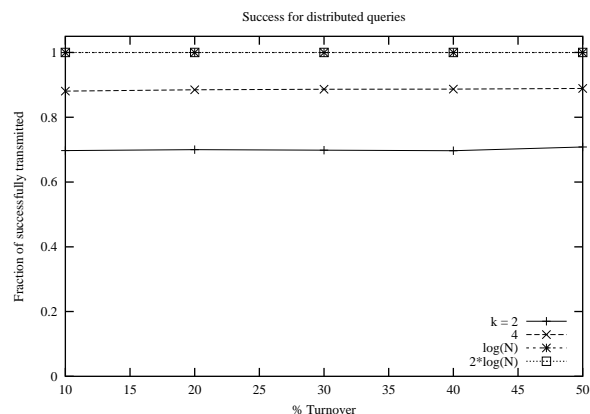


Fig. 2. Distributed query success characteristics for a network with 50% holes.

The second set of graphs (Figure 3) measure latency and bandwidth overhead of the system for distributed queries in which the replacement rate is 0.1 of the average number of nodes, i.e., a system in which roughly 10% of nodes, chosen at random, enter and leave at each time step. On average,  $N$  queries are generated by  $N$  nodes in each time interval. All simulation results are taken over 10 time steps. We consider values for  $k$ , ranging from 2 to  $2\log N$  (base 2). We studied latency, bandwidth, and failure behavior by varying  $\lambda$  from 0.1 to 0.5, but no substantial variation from the graphs presented here was found.

Not surprisingly, increasing  $k$  leads to noticeable reduction in latency, but even with small  $k$ , the number of hops required to service a query is low, logarithmic in the number of nodes. Latency behavior for centralized queries exhibits similar overhead.

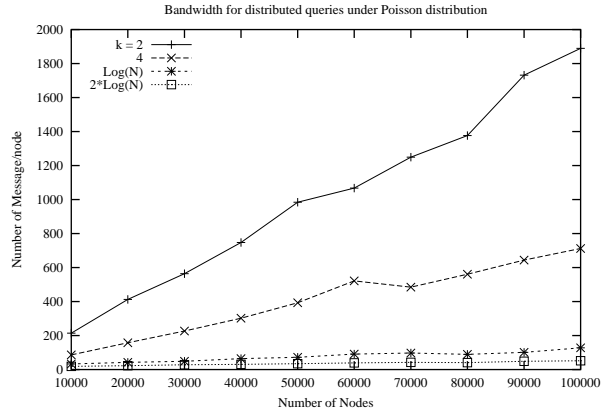
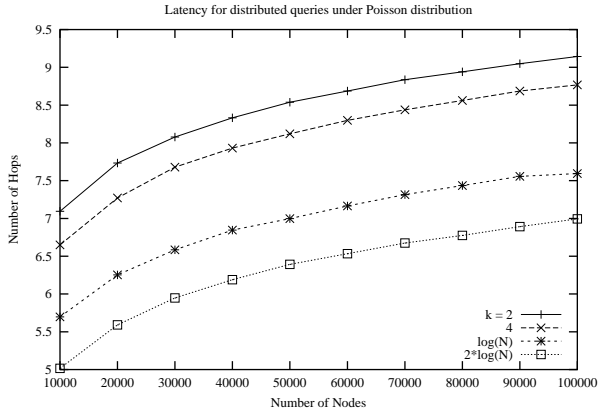


Fig. 3. Latency and bandwidth complexity for distributed queries.

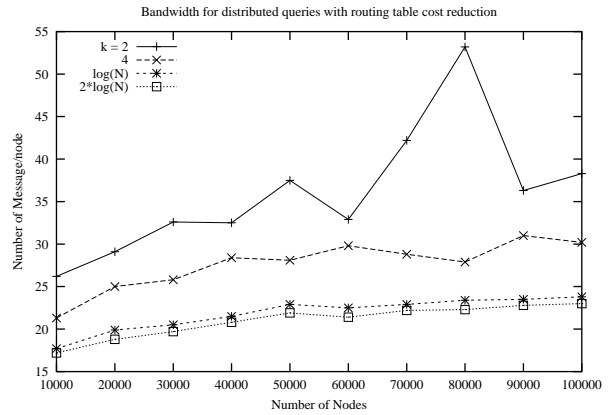
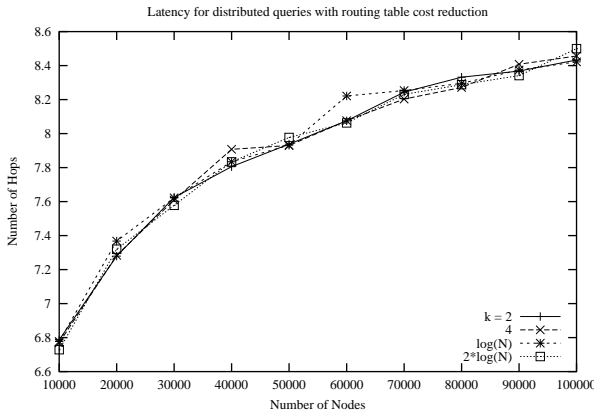


Fig. 4. Latency and bandwidth complexity for distributed queries with routing table size reduction.

We have redrawn the graph in Figure 3(b) in Figure 4 leaving out cases for  $k = 2$  and  $4$  to highlight the characteristics when  $k$  is  $\log N$  and  $2\log N$ . Because we assume centralized queries can be aggregated, their bandwidth requirements are easily shown to be bounded by a logarithmic factor in the number of nodes. The graphs reveal that the bandwidth requirements imposed by our overlay for distributed queries is low if we choose  $k$  to be  $c\log N$  for small  $c$ . It is clear that increasing  $k$  from  $4$  to  $\log N$  results in a tremendous reduction in bandwidth requirements. When  $k$  is  $\log N$ , the bandwidth requirements range from approximately 35 messages/node for  $N = 10,000$  nodes to 125 messages/node when  $N = 100,000$ . The requirements drop when  $k = 2\log N$ , ranging from 20 to 40 messages/node as  $N$  ranges from 10,000 to 100,000.

We have also studied latency and bandwidth performance under a Zipfian distribution in which a node leaves with probability inversely proportional to the square of its lifetime in the network. These results exhibit essentially identical characteristics to a Poisson replacement model and are consistent with our theoretical results which make no assumption on the type of the on-time distribution.

We also consider an improvement to the overlay that reduces

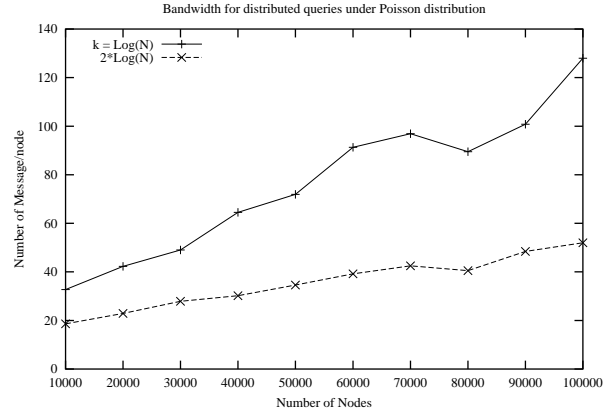


Fig. 5. Bandwidth complexity for distributed queries for  $k = O(c\log N)$  for small  $c$ .

routing table size maintained by nodes from  $O(\log^3 N)$  to  $O(\log^2 N)$ . Rather than preserving an edge to every node covering a vertex, this scheme simply records an edge to any one of the nodes covering a vertex, leading to an  $O(\log N)$  reduction in state information maintained at each node. Figure 5 measures latency and bandwidth characteristics for dis-

tributed queries under this alternative scheme. The latency characteristics of this alternate overlay is marginally worse than our original design for  $k = \log N$  or greater because failures (i.e. a node having no live neighbor) can occur more often with the smaller routing tables produced by this scheme, requiring retransmission of messages. For  $k = \log N$ , the bandwidth requirements imposed by the overlay is slightly superior to the original scheme. The primary reason for this is that the new scheme, unlike the original, no longer guarantees that every hop in a route will lead to forward progress in query distance; by relaxing this constraint, there is greater dispersion of queries among nodes in the overlay. We conjecture that the non-uniform spikes when  $k = 2$  is due to arbitrary congestion occurring because of low redundancy in the tree.

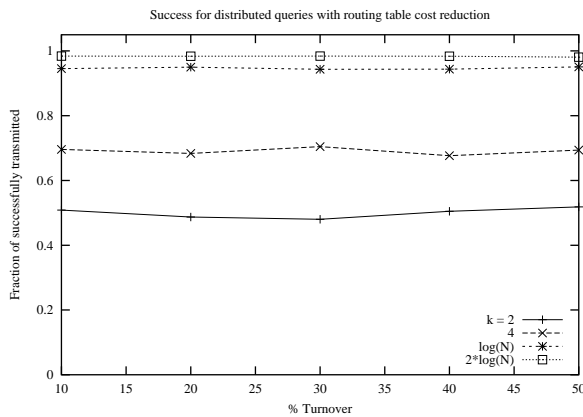


Fig. 6. Distributed query success characteristics for a network with 50% holes using routing table size reduction.

Figure 6 measures the query success characteristics for this scheme. As we would expect, the smaller routing table size incorporated in this scheme leads to reduced fault tolerance is  $k = 2$  or 4. However, there is no significant difference observed for  $k = \log N$  or greater. This is because the overlay has sufficient resiliency for higher values of  $k$  to ensure successful queries even though the number of edges in the overlay has been reduced by  $O(\log N)$  factor. These results give us confidence that the WARP overlay can effectively scale in practice.

## VI. CONCLUSIONS

This paper presents WARP, a fault-tolerant P2P system that is sensitive to the statistics of node characteristics (e.g., on-times) and query characteristics (e.g., popularity). We believe this is a first step in designing scalable and resilient P2P overlays whose theoretical properties conform closely to real-world behavior. We have not explicitly taken into account the statistics of other kinds of salient characteristics such as query access and update frequency or node capacities. Real data implies that these may follow a Zipfian distribution [28], [29]. For future work, we intend to incorporate these notions as part of our analysis.

## REFERENCES

- [1] Noga Alon and Joel Spencer. *The Probabilistic Method*. John-Wiley, 1992.
- [2] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
- [3] James Aspnes, Zoe Dimadi, and Guari Shah. Fault-Tolerant Routing in Peer-to-Peer Systems. In *Proceedings of the ACM Principles of Distributed Computing*, 2002.
- [4] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, pages 43–48, February 2003.
- [5] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, 2000.
- [6] David Clark. Face-to-Face with Peer-to-Peer Networking. *Computer*, 34(1), 2001.
- [7] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46+, 2001.
- [8] Edith Cohen and Steven Shenker. Replication strategies in unstructured peer-to-peer networks. In *ACM SIGCOMM'02 Conference*, 2002.
- [9] Amos Fiat and Jared Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proceedings of Symposium on Discrete Algorithms*, 2002.
- [10] Gnutella Protocol Specification v0.4. [http://www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf).
- [11] Theodore Hong. Performance. In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
- [12] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
- [13] <http://www.kaaza.com>.
- [14] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
- [15] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.
- [16] Jonathan Ledlie, Jacob Taylor, Laura Serban, and Margo Seltzer. Self-Organization in Peer-to-Peer Systems. In *SIGOPS European Workshop*, 2002.
- [17] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proceedings of ACM Principles of Distributed Computing*, 2002.
- [18] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *ACM Principles of Distributed Computing*, 2002.
- [19] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(1), 2003.
- [20] Napster. <http://www.napster.com>.
- [21] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building Low-Diameter P2P Networks. *IEEE Journal on Selected Areas in Communications*, 21(6):995–1002, 2003.
- [22] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.
- [23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM 2001*, 2001.
- [24] M. Ripeanu and I. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.
- [25] Sheldon Ross. *Applied Probability Models with Optimization Applications*. Dover Press, 1970.
- [26] Antony I. T. Rowstron and Peter Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

- [27] Jared Saia, Amos Fiat, Steve Gribble, Anna R. Karlin, and Stefan Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.
- [28] Stefan Saroiu, Krishan Gummadi, Richard Dunn, Steven Gribble, and Henry Levy. An Analysis of Internet Content Delivery Systems. In *ACM Conference on Operating System Design and Implementation*, 2002.
- [29] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.
- [30] Subhabrata Sen and Jia Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. *ACM Transactions on Networking*, to appear.
- [31] Zhichen Xu, Mallik Mahalingam, and Magnus Karlsson. Turning Heterogeneity into an Advantage in Overlay Routing. In *Symposium on File and Storage Technologies (FAST)*, 2003.
- [32] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.