The Bootstrapping Service*

Márk Jelasity†
University of Bologna, Italy
jelasity@cs.unibo.it

Alberto Montresor
University of Trento, Italy
montreso@dit.unitn.it

Ozalp Babaoglu
University of Bologna, Italy
babaoglu@cs.unibo.it

Abstract

We outline a lightweight architecture to support novel application scenarios for P2P systems. These scenarios include merging and splitting of large networks, or multiplexing relatively short-lived applications over a pool of shared resources. In such scenarios, the architecture needs to be quickly and efficiently (re)generated frequently, often from scratch. We propose the bootstrapping service *abstraction as a solution to this problem. We present an instance of the service that can jump-start any prefix-table based routing substrate quickly, cheaply and reliably from scratch. We experimentally analyze the proposed bootstrapping service, demonstrating its scalability and robustness.*

1. Introduction

Structured overlay networks are increasingly seen as a key layer (or service) in peer-to-peer (P2P) systems, supporting a wide variety of applications. Index-based lookup is generally considered to be a "bottom" layer (e.g., [2,12]), based on the assumption that the life cycle of supported systems is similar to grassroots file sharing networks: there exists at least one functional network, membership can change due to churn, and the network size can also fluctuate, but relatively smoothly. Join operations are assumed to be uncorrelated. Most simulation and analytical studies also reflect these assumptions, since they are often based on traces collected from real file sharing networks.

While this scenario may be appropriate for many important applications, we believe that overlay networks can be important design abstractions in radically different scenarios that have not yet been considered by the P2P research community. In particular, *massive joins* to a large overlay network are not supported by known protocols very well, and many protocols have trouble dealing with *massive departures* as well. Other related scenarios that are important yet underemphasized include *bootstrapping* a large network from scratch, *merging* two or more networks, *splitting* a large network into several pieces, and *recovering from catastrophic failure*.

If these scenarios were to be supported efficiently, we could build a fully open and flexible computing infrastructure that points well beyond current applications. In this paper we envision scenarios that involve (virtual) organizations with (possibly) large pools of resources organized in overlay networks. We want to allow these overlay networks to freely and flexibly merge with and split from networks of other organizations on demand, and we want to admit allocation (or sale) of pools of resources for relatively short periods to users who could then build their own infrastructures on demand and abandon them when they are done. This vision is in line with current efforts to enhance the flexibility of Grid infrastructures using P2P technology [4].

To support the above vision, we propose a P2P architecture with two main components: the *peer sampling service* and a dedicated *bootstrapping service*. Merging several large networks or starting an application from scratch within its time-slice are unusual and radical events that many existing P2P protocols are not designed to cope with. To provide a reliable platform in the face of massive joins and departures, we propose the *peer sampling service* [6] as a lightweight bottom-most layer of our P2P architecture. The bootstrapping service is then built on top of this peer sampling service. In the proposed architecture, large collections of resources can be readily aggregated into global structured overlays rapidly and efficiently. This then allows the use of existing, well-tuned protocols without modification to maintain the overlays once they have been formed. As a concrete example of the bootstrapping service, we present

*In *Proc. 26th ICDCS WORKSHOPS*, 2006. doi:10.1109/ICDCSW.2006.105. This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).

†Also with RGAI, MTA SZTE, Szeged, Hungary.

a novel protocol that can efficiently build prefix-based overlay routing substrates such as Pastry [13], Kademlia [8], Tapestry [17] and Bamboo [11] from scratch.

The outline of the paper is as follows. Section 2 presents the architecture to support the scenarios mentioned above. Section 3 briefly describes the bottom layer: the peer sampling service. Section 4 describes the protocol implementing the bootstrapping of routing substrates, while Section 5 presents experimental results. Finally, Sections 6 and 7 compare our contribution to related work and conclude the paper.

2. The Architecture

Our ultimate goal is to design a P2P architecture that allows for large pools of resources to behave almost like a liquid substance: it should be possible to merge large pools, or split existing pools into several pieces easily. Furthermore, it should be possible to bootstrap potentially complex architectures on top of these liquid pools of resources quickly on demand.

The architecture is outlined in Figure 1. The lowest layer, the peer sampling service, implicitly defines a group abstraction by allowing higher layers to obtain addresses of random samples from the actual set of participating peers; even shortly after massive joins or catastrophic failures. The basic idea of the architecture is that we require *only* this lowest layer to be liquid, that is, persistent to the radical scenarios we described, and we propose to build all other overlays on demand. In other words, the sampling service functions as a last resort that provides a very basic, but at the same time extremely robust service, which is sufficient to enable jump starting or recovering all higher layers of the architecture.

We realize that this idea is rather radical, and depends crucially on the existence of a lightweight and fast bootstrapping service, and a robust sampling service. The latter has been provided in our previous work and is briefly outlined in Section 3. In this paper we focus on the bootstrapping service assuming that the peer sampling service is available.

As shown in Figure 1, the architecture supports other components in addition to structured overlays. For example, a number of components rely only on random samples, like probabilistic broadcast (gossip) or aggregation [3, 7]. The architecture can also support other overlays, such as proximity based ones [5, 15].

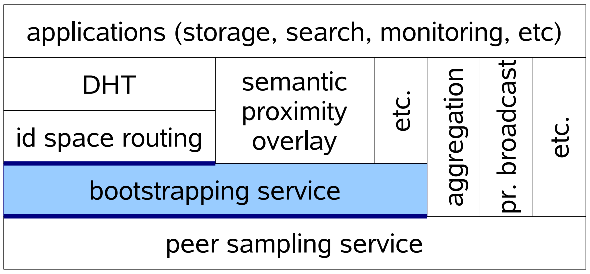

Figure 1. The layers of the proposed architecture. The highlighted part is discussed in this paper.

3. The Peer Sampling Service

The bottom layer of the proposed P2P architecture is the peer sampling service [6]. The purpose of this layer is to provide random peer addresses from the set of participating nodes. In addition, the layer implicitly defines membership as being the pool from which the samples are drawn. Our previous work confirms that the layer can be implemented in such a way that it provides high quality (i.e., sufficiently random) samples, even immediately after massive joins and departures. This is essential for dealing with the scenarios described in the Introduction and allows us to focus on bootstrapping over a network in which the sampling service is already functional.

In this paper we consider an instantiation of the peer sampling service based on the NEWSCAST protocol [6]. The basic idea of NEWSCAST is that each node periodically sends a small, locally available random set of node addresses to a member of this random set. After receiving such a message, the node keeps a fixed number of freshest addresses (based on timestamps). In the following we briefly summarize the most important properties of the protocol.

Cost: Each node sends one message to one other node during a fixed time interval. Implementations exist in which these messages are small UDP messages containing approximately 30 IP addresses, along with the ports, timestamps, and descriptors such as node IDs. The time interval is typically long, in the range of 10 seconds. The cost is therefore small, similar to that of heartbeat messages in many distributed architectures.

Self-healing: The protocol provides sufficiently random samples very quickly after catastrophic failures (up to 70% nodes may fail) and during massive churn. Besides, even if the sets of random samples that are available at the nodes locally are initialized in a non-random way (e.g., all nodes have the same samples) the protocol very quickly randomizes the samples.

Due to its low cost, extreme robustness and minimal assumptions, this protocol is an ideal bottom layer that makes the bootstrap service feasible. The sampling service is useful (and, in fact, sufficient) for gossip-based protocols that are based on sending information periodically to random peers.

4. The Bootstrapping Service

As argued earlier, our architecture crucially relies on the existence of a lightweight and efficient implementation of the bootstrapping service, that in turn relies on peer sampling. Here we develop a protocol that fulfills these requirements. Note that we have already addressed bootstrapping Chord [9] that is based on a sorted ring, and additional fingers that are defined based on distance in the ID space. However, an important alternative design decision of DHT-s is applying *prefix-based routing tables*, which have some important advantages, such as independence of ID distribution, but which are a significantly different task to build and maintain. The protocol that we present here constructs prefix-based routing tables at all participating nodes simultaneously, and from scratch. The key idea is to build a sorted ring, and during the process, collect entries to fill the prefix tables at all nodes.

The prefix table is defined as follows. We assume that all nodes have unique numeric IDs. An ID is represented as a sequence of digits in base 2^b — each digit is encoded as a b-bit number. The prefix table of a given node contains up to k IDs for all pairs (i, j), where i is the length (in digits) of the longest common prefix of the ID and the node's own ID, and j is the first differing digit. The entries may be less than k if there are not enough node IDs with the desired prefix and digit among the participating nodes. Many overlay routing substrates are based on this prefix table: for example Pastry [13], Kademlia [8], Tapestry [17] and Bamboo [11]. Using the prefix tables and the leaf sets (that define the sorted ring) the protocol constructs the routing tables of all these networks can be bootstrapped.

The protocol executed at all nodes is shown in Figure 2. Each node has a prefix table to fill and a leaf set, that is being evolved to contain the nearest neighbors in the sorted ring of node IDs. The size of the leaf set is c. The components of the protocol work as follows.

Method updateLeafSet takes a set of node descriptors (addresses and corresponding IDs) and tries to improve the leaf set using these descriptors. First, it merges the set given as a parameter, and the current leaf set, and then sorts this set according to distance from the node's own ID in the ring of all possible IDs. Note that all IDs can be classified as successors and predecessors: if an ID is closer in the increasing direction, it is a successor, otherwise it is a predecessor. Then, in an effort to collect an equal amount of successors and predecessors, the method attempts to keep an equal number $(c/2)$ of closest successors and predecessors. If there are not enough successors or predecessors, then the leaf set is filled with the closest elements in the other direction.

```
loop
    wait(Δ)
    $q \leftarrow$ selectPeer()
    $m_p \leftarrow$ createMessage($q$)
    send $m_p$ to $q$
    $m_q \leftarrow$ receive($q$)
    updateLeafSet($m_q$)
    updatePrefixTable($m_q$)
end loop
```

(a) active thread

```
loop
    $m_q \leftarrow$ receive($*$)
    $m_p \leftarrow$ createMessage($q$)
    send $m_p$ to sender($m_q$)
    updateLeafSet($m_q$)
    updatePrefixTable($m_q$)
end loop
```

(b) passive thread

Figure 2. Bootstrapping protocol at node p.

Method selectPeer() also sorts the leaf set according to distance from the node's own ID in the ring of all possible IDs, and then picks a random element from the first half of the sorted list.

These components of the protocol are similar to the application of T-Man for building a sorted ring, as described in [5]. The rest of the components are responsible for building the prefix table. The basic idea is that the gradually improving prefix table is fed back into the ring building process, so that the two components mutually boost each other. That is, the ring-building process fills in most of the entries in the prefix tables, however, the prefix tables—even before completed—can already fulfill a kind of routing function, just like in DHT-s. Especially in the end phase, when most of the nodes have found their place in the ring, but a few still have an incorrect neighborhood, the gossip mechanism of T-Man and the almost complete prefix tables together make sure these last nodes also find their correct neighborhood quickly, essentially as if they were routed by the routing substrate under construction.

The simplest method is UPDATEPREFIXTABLE that takes a set of node descriptors and fills in any missing table entries from this set.

The key component of the algorithm is CREATEMESSAGE, which is responsible for generating a set of node descriptors to be sent to the peer node. Knowing the ID of the peer, the method optimizes the information to be sent as follows. First it takes the union of the leaf set, c_r random samples taken from the sampling service, the current prefix table, and its own descriptor (in other words, all locally available information). It orders this set according to distance from the peer node, and keeps the first c entries. In addition, it adds to the message all node descriptors that are potentially useful for the peer for its prefix table (i.e., have a common prefix with the peer ID). The size of this additional part is not fixed but is bounded by the size of the full prefix table, and usually is smaller in practice.

Finally, the protocol needs to be started in a loosely synchronized manner, that is, the nodes have to start the execution of the protocol with a relatively small time difference, for example, within an interval of Δ time units, which defines the frequency of communication (see Figure 2). At start time, all nodes use the peer sampling service to initialize their leaf sets with a set of random nodes, and clear their prefix table. For simplicity, we assume here that the protocol is started by a system administrator, using some form of broadcasting or flooding on top of the peer sampling service (e.g., [3]).

Let us summarize the parameters of the protocol. The prefix table is defined by b (the number of bits in a digit) and k, the number of entries for a specific prefix length and first differing digit. The size of the leaf set is c. Parameter Δ defines the frequency of communication. Finally, c_r is the number of random samples used for improving the messages to be sent. Note that these samples are "free" (if c_r is not too large) since the generic peer sampling layer is assumed to function independently of the bootstrapping service.

5. Simulation Results

Both the sampling service and the bootstrapping service were implemented for the PEERSIM simulator developed by us [10]. We focus on two aspects of the protocol: scalability and fault tolerance. To this end, we fix all the parameters of the protocol, except the network size and failure model. In our simulations IDs are 64-bit integers. Although typical definitions of the ID space consider 128-bit integers, using only 64 bits for our simulations is not limiting since the length of the largest common prefix is much less than 64 bits for all node pairs in networks of any practical size. The extra bits play no role in this protocol.

The parameters of the prefix table were chosen to match common settings: $b = 4$ and $k = 3$. For networks that do not require multiple alternatives of a given table entry, setting $k > 1$ is still useful because it allows for optimizing the routes according to proximity. The leaf set size was $c = 20$ and the parameter c_r was set to be 30. We experimented with network sizes (N) of 2^{14}, 2^{16} and 2^{18} nodes.

The scenario of an experiment is as follows. We assume that we are given a network where the sampling service is already functional. We start the bootstrapping protocol at each node at a different random time within an interval of length Δ. For convenience, we call the consecutive intervals of length Δ *cycles*. The first interval corresponds to cycle 0.

The protocol is then run until the *perfect* leaf sets and prefix tables are found at all nodes, based on the actual set of IDs in the network. This cannot be decided locally, and indeed, the protocol has no stopping criterion. However, since our protocol is cheap and needs only a small number of iterations, in practice, after initialization it can be run simply for a fixed number of cycles that are known to be sufficient for convergence.

To test scalability, in the first set of experiments (shown in Figure 3) there are no failures and all messages are delivered reliably. For network sizes 2^{14}, 2^{16} and 2^{18}, we performed 50, 10 and 4 independent experiments, respectively. The plots show the results of each individual experiment, ending when perfect convergence is obtained.

From the top graph of Figure 3 we observe that the time required to reach a desired quality of the leaf sets increases by an additive constant despite a four-fold increase in the network size. This is a strong indication that the time needed for convergence is logarithmic in network size. In addition to being logarithmic, the actual convergence times are also rather small. Convergence of the leaf sets clearly follows an exponential behavior.

The convergence of the prefix tables is rather surprising (bottom graph of Figure 3): the network of 2^{18} nodes converges *faster* in the final phase than a network that is four times smaller, with the same parameters. Note that in this final phase, the vast majority of the entries are already available (less than 1 out of 1000 entries are missing). This slight difference has to do with the scarcity of suitable IDs for the remaining positions

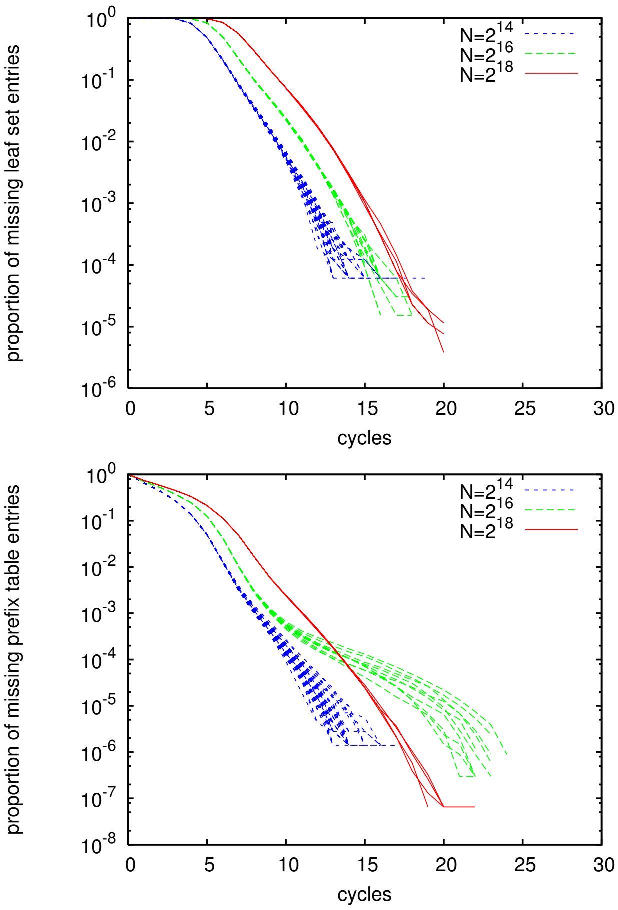

Figure 3. Results in the absence of failures. When a curve ends, the corresponding tables are perfect at all nodes.

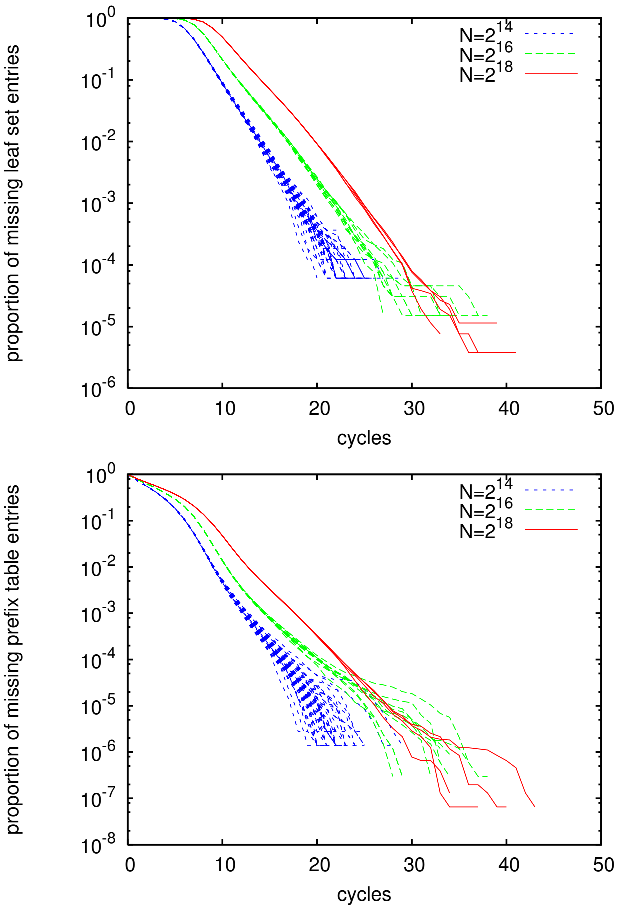

Figure 4. Results with 20% of the messages dropped. When a curve ends, the corresponding tables are perfect at all nodes.

to fill.

In the second set of experiments we tested the robustness of our protocol by dropping messages with a uniform probability (Figure 4). This failure model is appropriate for study because we designed the protocol with a cheap, unreliable transport layer in mind (UDP). The drop probability was chosen to be 20%, which is unrealistically large. Since the protocol is based on message-answer pairs, if the first message is dropped, then the answer is not sent either. Taking this effect into account, elementary calculation shows that the expected overall loss of messages is 28%.

The main conclusion of these experiments is that the behavior of the protocol is very similar to the case when there are no failures, only convergence is slowed down proportionally.

The protocol is not sensitive to churn either (not shown). In short, the quality of the routing tables generated by our protocol is similar to that obtained by known routing substrates in the presence of similar churn. Furthermore, since our protocol is based on cheap UDP messages and can be completed in a small number of cycles, the effect of churn during this short time is naturally limited.

6. Related Work

Massive joins to already running overlays have been addressed previously (e.g., [12, 16]) proposing a form of periodic repair mechanism for maintaining the leaf set, not unlike the one presented here. More recently the bootstrapping problem has been addressed as well, focusing on specific overlays [1,9,14]. Our contribution with respect to related work is twofold. First, we propose an *architecture* that can support a protocol that jump-starts an entire overlay *from scratch*. Our protocol is independent of the protocol that manages the routing substrate: we singled out the abstract bootstrap service as an important architectural component. Second, our protocol is efficient and lightweight, and supports overlays based on *prefix-tables* and leaf sets.

7. Conclusions

We proposed a P2P architecture that relies on a robust peer sampling service and a bootstrapping service. Although the functionality of the sampling service is basic, its implementation is more robust and flexible than those of currently available structured overlays. The architecture we presented here, and in particular, the bootstrapping service, bridges the robustness and flexibility of the sampling service and the functionality of structured overlays.

Based on our simulation results, the proposed instantiation of the bootstrapping service can build a *perfect* prefix table and leaf set at all nodes, in a logarithmic number of cycles, even in the presence of message delivery failures. This performance, in combination with the support of the sampling layer, enables the on-demand deployment of complex (multi-layered) P2P applications in short time-slices over large pools of shared resources, in addition to allowing large pools of resources to be merged or split temporarily.

References

[1] Karl Aberer, Anwitaman Datta, Manfred Hauswirth, and Roman Schmidt. Indexing data-oriented overlay networks. In *Proceedings of 31st International Conference on Very Large Databases (VLDB)*, Trondheim, Norway, August 2005. ACM.

[2] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *Proceedings of the SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[3] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Anne-Marie Kermarrec, and Petr Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[4] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[5] Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based overlay topology management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, 2005. to appear.

[6] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.

[7] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.

[8] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, 2001.

[9] Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Chord on demand. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pages 87–94, Konstanz, Germany, August 2005. IEEE Computer Society.

[10] PeerSim. http://peersim.sourceforge.net/.

[11] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, June 2004.

[12] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of ACM SIGCOMM 2005*, pages 73–84. ACM Press, 2005.

[13] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer-Verlag, 2001.

[14] Ayman Shaker and Douglas S. Reeves. Self-stabilizing structured ring topology p2p systems. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, pages 39–46, Konstanz, Germany, August 2005. IEEE Computer Society.

[15] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. In José C. Cunha and Pedro D. Medeiros, editors, *Proceedings of Euro-Par*, number 3648 in Lecture Notes in Computer Science, pages 1143–1152. Springer, 2005.

[16] Ben Y. Zhao, Ling Huang, Anthony D. Joseph Jeremy Stribling, and John D. Kubiatowicz. Exploiting routing redundancy via structured peer-to-peer overlays. In *Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP 2003)*, pages 246–257, 2003.

[17] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, April 2001.