Building Latency-aware Overlay Topologies with QuickPeer

Andrea Ceccanti[*] and Gian Paolo Jesi[†]
Dept. of Computer Science
Università di Bologna
Via Mura Anteo Zamboni, 7, 40127 Bologna
{ceccanti,jesi}@cs.unibo.it

Abstract

This work presents a gossip-based protocol, termed QuickPeer, which builds and maintains latency-aware overlay topologies. Such topologies are useful for several distributed applications, like distributed online gaming, context-aware P2P applications and QoS-aware publish/subscribe systems. The distinctive feature of QuickPeer is that it can manage large scale overlay topologies providing each host in the overlay with its closest or furthest neighbour, according to network distance (RTT), in few gossip rounds. We present experimental results that prove that QuickPeer is a scalable and robust solution for large-scale latency-aware overlay topology management.

Keywords: Epidemic algorithms, Network-aware topology management, Virtual coordinates, Peer-to-Peer systems, Internet

1 Introduction

Recent years have witnessed a growing interest in the area of application-layer overlay protocols and peer-to-peer (P2P) systems. Examples include popular file-sharing applications [15, 29], end-to-end multicast [16, 7], multimedia streaming applications [5, 19] as well as publish/subscribe systems [4, 6, 23].

In P2P systems, nodes maintain *logical* connections to a small subset of other participating nodes (typically called *neighbors*). These connections define the topology of the overlay network, which is usually built on top of exisiting network infrastructure providing end-to-end connectivity (i.e., IP). Building and maintaining scalable overlay networks that organize many thousands of nodes is a challenging task. Two main approaches have been proposed to deal with this problem. On the one hand, structured overlays [28, 24, 27, 30] organize nodes in hierarchical structures that enable scalable and efficient generic key lookup of distributed information; however, structured overlays do not support well highly transient nodes and complex queries on distributed content. On the other hand, unstructured overlays [11, 18, 15, 7] do not impose additional structuring on participating nodes, and exhibit better resilience to failures and highly transient peer populations[1].

In this paper, we study the topology management problem in unstructured overlay networks. In particular, we present a protocol, termed Quick-Peer, that can effectively build and maintain large scale *latency-aware* overlay topologies. Such topologies reflect the underlying IP-level network topology, and provides each peer with the knowledge of the closest (or furthest) neighbour, according to network distance (i.e., latency), present in the overlay network at a given time.

QuickPeer is an epidemic or gossip-based protocol. Epidemic protocols have shown to be a robust and scalable solution for information dissemination [12], resource monitoring [26] and distributed state management [10].

This paper is structured as follows. Section 2 introduces the latency-aware overlay topology construction problem and introduces QuickPeer. Section 3 defines the experimental setup and presents simulation results proving QuickPeer scalability and adaptiveness to dynamic environmental conditions. In section 4 we discuss related work. Finally, Section 5 concludes the paper.

[*]This work has been partly supported by the Italian FIRB project entitled WEBMINDS and by INFN-CNAF in the context of the EU projects EGEE (IST-2003-508833) and CoreGRID (IST-2004-004265).

[†]This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Projects BISON (IST-2001-38923) and DELIS (IST-001907).

[1]usually called *churning* in P2P jargon.

```
do forever
  wait(Δt)
  neighbour = view.selectPeer()
  send(myView,neighbour)
  peerView = receive()
  update(myView,peerView,trimPolicy)
```

(a) Active thread

```
do forever
  peerView = receive()
  send(myView,peerView.sender)
  update(myView,peerView,trimPolicy)
```

(b) Passive Thread

Figure 1. QuickPeer protocol pseudo-code.

2 Latency-aware topology management

2.1 System Model

In our model, peers communicate via message exchanges, exploiting the connectivity provided by an underlying routed network (i.e., the Internet). Each peer knows a set of other peers (its *neighbours*) that defines its *view* of the system. Since we consider networks of large size, partial membership information at each peer is required for scalability and manageability purposes.

Network distance between peers is modeled using the Vivaldi network coordinate system [9]. This system creates an accurate geometric model where a point in multi-dimensional space is assigned to each peer in the overlay network. In this virtual space, Euclidean distance between two points predicts, with good accuracy, network round-trip time (RTT) between peers in the overlay network, without requiring all-pairs RTT probes. Each peer in the overlay has an associated *peer identifier* (PID), which contains the peer's IP address and port and its virtual coordinates.

In our model, we consider a dynamic overlay network, where peers may join or leave the network at any time.

2.2 The QuickPeer algorithm

QuickPeer (QP) is an epidemic protocol that, within few gossip rounds, provides each peer with the *closest* (or *furthest*) peer identifiers (PIDs) available in the overlay. The basic idea underlying the protocol, inspired by the work presented in [17], is as follows. Each peer maintains a fixed-size view containg k PIDs. The view is sorted according to the network distance estimates provided by Vivaldi coordinates. So, at any time, the first position in the view holds the closest peer known so far.

At bootstrap time, QP views needs to be initialized with a random sample of nodes taken from the whole overlay. For this purpose, QP relies on a *peer randomizer*, i.e., an epidemic protocol that builds and maintains an approximately random-graph overlay topology. In this work, we adopted the Newscast protocol [18] as a peer randomizer. Starting from a first random snapshot, QP basically evolves the mirrored overlay towards the desired latency-aware topology.

In order to evolve the topology, peers exchange views in an epidemic fashion. Periodically, each peer actively selects a neighbour and starts a view exchange process (see pseudocode in Figure 1). Once the remote peer's view has been received, it is merged with the local one. Note that this merge operation preserves the ordering of the local view, i.e., newly received PIDs are sorted according to the distance from the *local* peer coordinates.

After the views have been merged, a trimming policy selects the k PIDs (out of the possible $2k$) that are kept in the local view. Currently, QP supports two distinct trimming policies:

1. **ClosePolicy(k)**: selects the first k PIDs in the view (i.e., the closest neighbours seen so far);

2. **CloseFarPolicy(k)**: selects the first and the last $k/2$ PIDs in the view (i.e., the closest and furthest neighbours seen so far).

Merging and trimming operations described above are performed in the `update()` method shown in Figure 1.

QP uses distinct picking strategies to select the neighbour for the view exchange, according to which trimming policy is in use. When **ClosePolicy** is used, the neighbour is picked in the first half of the view only (i.e., among the first $k/2$ PIDs). Experimental results [17] show that this strategy leads to faster convergence to a latency-aware optimal overlay. In contrast, when the **CloseFarPolicy** is used, the neighbour is selected randomly from the whole view.

Note that QuickPeer lets each node exchange at most once for each gossip round (actively or passively). This ensures that, on the average, all peers exchange views the same number of times during a QuickPeer session.

2.3 Failure detection

QuickPeer detects failed nodes at picking time. If a neighbour selected for the view exchange does not answer a probe message in a limited amount of time, it is considered failed and its PID is removed from the view. In case of massive node failures, however, the second half of the view will still be populated with references to failed peers, since the picking "cleans" only the first half of the view. To overcome this limitation, QuickPeer periodically triggers a **cleanView** procedure that probe peers that appear in the second half of the view. The frequency at which this procedure is activated may be adaptively tuned to limit the network traffic generated by the probes.

3 Evaluation

We validate QuickPeer effectiveness in building latency-aware overlay topologies using simulation. Experiments are run on Peersim, a Java-based cycle-driven simulator developed in the Bison project [2]. We consider three different network sizes: 2^{12}, 2^{13} and 2^{14} nodes. Network topologies are generated with the Brite Internet topology generator [3], using the Waxman algorithm on a flat router model. The output of this phase is a weighted graph, where weights represents latencies between routers in the generated topology. We then run all-pairs shortest paths on the generated graph to obtain a matrix of RTT distance between all pairs of routers in the network. Creating RTT matrix offline speeds up simulations and allows us to simulate larger networks. A Vivaldi simulation is then ran offline on this data to build the static, five-dimensional coordinates used in QuickPeer experiments.

The QuickPeer view size k is set to 40 in all the experiments discussed in this section. An instance of the Newscast protocol boostraps the QuickPeer views at beginning of the simulations[2].

Our experiments focus on the evaluation of the following QuickPeer aspects:

1. protocol scalability: how well the protocol scales as the network size increase;

2. robustness: how QuickPeer reacts to fluctuations in the peer population.

All the results presented here have been averaged over 10 simulation runs.

3.1 Static scenario

In this section we present experiments that evaluates QuickPeer scalability and convergence rate in static overlays (i.e., no nodes joining or leaving the networks). We presents results obtained using two distinct view trimming policies: **ClosePolicy** and **CloseFarPolicy**. These results show that QuickPeer scales well and is fast in constructing optimal, large-scale latency-aware topologies.

We measured the convergence of the protocol in terms of *optimal nodes* achieved over time (in cycles). A node becomes “optimal” when it discovers and collects its closest (*and* furthest, when **CloseFarPolicy** is used) neighbour identifier (PID) in its local view.

[2]Newscast view size is also set to 40

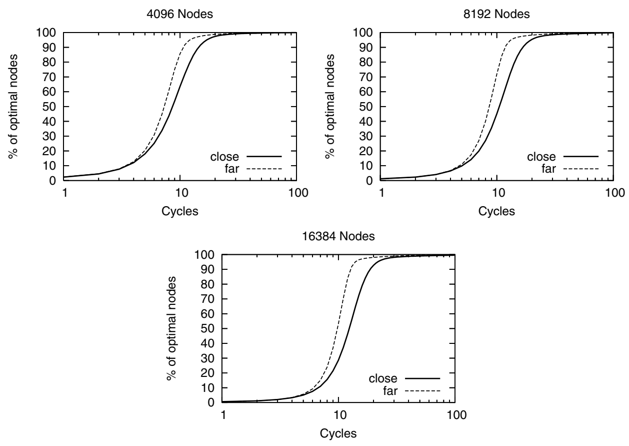

Figure 3. QuickPeer convergence performance for each network size. The *CloseFar* policy is used to trim the node views. Two kinds of optimality are considered: close convergence (black line) and far convergence (dotted line).

3.1.1 ClosePolicy

Figure2 presents the QuickPeer convergence performance using the **ClosePolicy** trimming policy with parameter $k = 40$. The first thing to note is that QuickPeer convergence rate does not depend on the network size. For all the three scenarios, more than 99.5% of the peers have their closest neighbor in view at cycle 20. However, QuickPeer reaches 100% optimality only around cycle 60. In fact, as the protocol clusters close neighbours together, it becomes harder and harder for those peers that did not reach optimality to find their closest neighbour.

To improve the convergence speed in the final phase, we implement the following optimization. At each view exchange, the randomized view maintained by the underlying peer randomizer is added to the merging process. This optimization yields 100% convergence at about cycle 30, as can be seen in figure 2. Note that this feature comes at no added cost in terms of network usage since the merge process is local at each node. For these reasons, we have decided to keep this feature always on during all the other tests.

3.1.2 CloseFarPolicy

Figure 3 shows the convergence performance obtained with the **CloseFar** trim policy. With this policy, QuickPeer provides each peer with the closest and the furthest neighbours present in the overlay. To obtain faster convergence time, the node picking strategy has been slightly modified: the node is selected randomly in the whole view and not only in the first half as when **ClosePolicy** is used. As can be seen

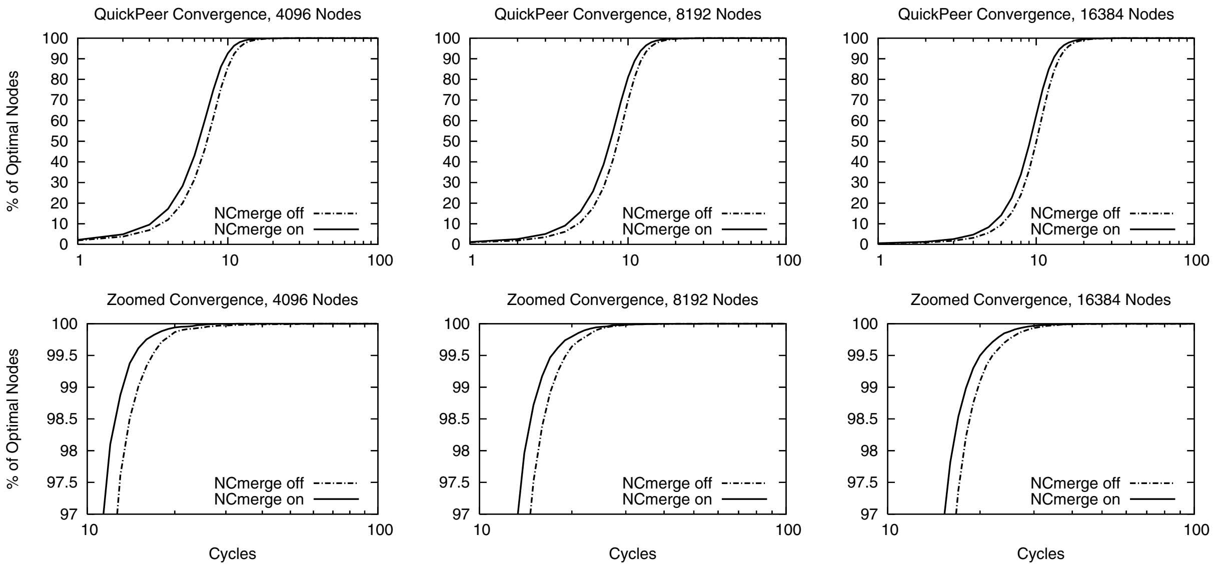

Figure 2. QuickPeer convergence rate expressed in percentage of optimal nodes for each network size over time (simulation cycles). The second line of pictures highlights the final phase convergence details.

in Figure 3, close and far convergence rate are pretty similar. However, in all our experiments, we experienced that QuickPeer locates more easily furthest nodes in the initial convergence phase.

3.2 Dynamic scenarios

In this section, we present experiments that evaluate QuickPeer scalability and convergence rate in dynamic overlays. We consider a massive node crash scenario in which half of the nodes are killed during QuickPeer convergence phase and show that the protocol handles the failures gracefully. In addition, we evaluate QuickPeer behaviour in a scenario where a large number of nodes join the overlay during the convergence phase. Even in this case, QuickPeer adapts to mutated environmental conditions.

3.2.1 Nodes crash

To evaluate the protocol robustness in case of a massive node crash, we ran the following experiment. The experiment starts with a network of 2^{14} nodes. At cycle 5, right in the middle of the Quickpeer convergence process, 50% of the active nodes fail.

In this catastrophic scenario, QuickPeer is still performing well, as depicted in Figure 4. Note that QuickPeer convergence is still increasing even one cycle after the massive node crash and optimality is reached in about the 30 cycles. This behaviour is expected, since now QuickPeer has an easier job to accomplish given the smaller size of the overlay.

After the node crash, each node holds in its view, with high probablility, references to failed nodes. In this experiment, the **cleanView** procedure is triggered every 3 simulation cycles.

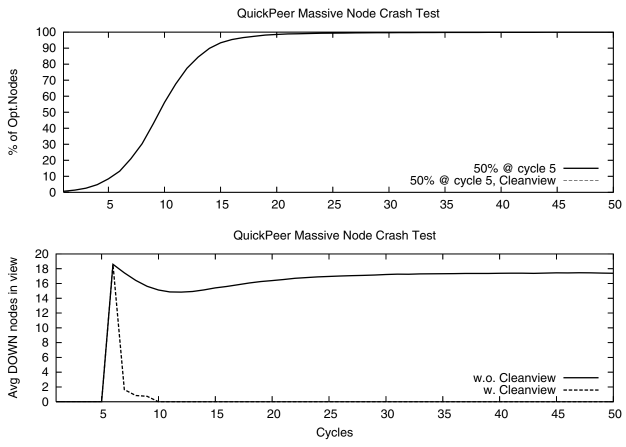

Figure 4. Massive crash scenario: 50% of nodes are randomly crashed at cycle 5. The two sub-figures depict respectively the convergence rate and average node view pollution per node.

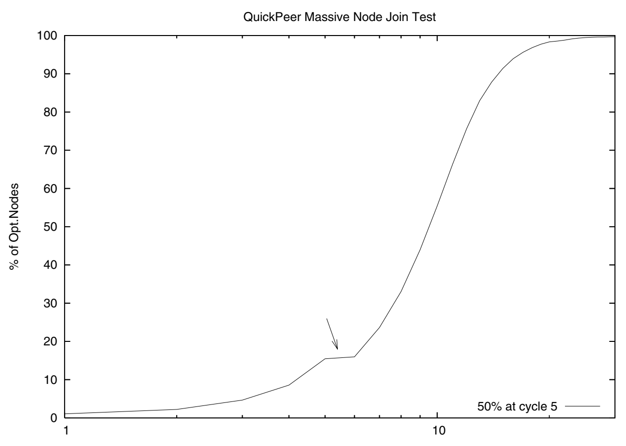

Figure 5. QuickPeer convergence rate for the massive nodes join experiment: starting from a 2^{13} nodes network, 4096 new nodes are added at cycle 5. The arrow (between cycle 5 and 6) indicates a transient slow down in convergence rate due to the massive node join.

The bottom sub-figure in Figure 4 shows that, after the crash, the node views are getting populated with failed PIDs. Without the **cleanView** procedure, the number of failed PIDs initially tend to decrease due to the cleaning process associated with the picking in the first half of the view. However, after a few cycles, the average pollution stabilizes around 45% of the view size (i.e., it fills nearly half of the view). The **cleanView** procedure stops this pathological behaviour at cycle 10.

3.2.2 Nodes join

QuickPeer reaction to a massive node join scenario is depicted in Figure 5. The experiment starts with an overlay network of 8192 nodes. At cycle 5, 4096 new nodes join the overlay.

The convergence rate slows down at cycle 5, just after the massive join. After this step, the rate grows exponentially as in previous experiments until full convergence is achieved at cycle 30.

4 Related Work

In this section, we review research results related with topology-aware overlay construction.

T-Man [17] is a generic, gossip-based framework for managing and building large-scale overlay topologies that inspired our work on QuickPeer. However, in [17], T-Man performance is evaluated only on "geometric" topologies (e.g., torus, ring or binary tree). In contrast, we evaluate QuickPeer using more realistic topology models and in dynamic environments to illustrate QuickPeer self-healing and adaptive behaviour.

In [25], the authors propose a scheme to partition overlay nodes into "bins" according to network proximity information. This information is gathered from DNS and delay measures against a set of landmark nodes. Our approach, in contrast, does not need any infrastructure services and exploits a synthetic virtual coordinates system (Vivaldi [9]) to obtain distance measurements.

In [21], the authours propose an epidemic protocol, the *Localiser*, to optimise an unstructured overlay network built using SCAMP [14]. Such protocol prove to be scalable and tolerates failures. However, no massive node join scenarios are evaluated.

Several architecture for global distance estimation services that expolits synthetic coordinates have been proposed recently. IDMaps [13] and GNP [22] rely on deployment of infrastructure nodes. In contrast, Vivaldi [9], PIC [8] and PCoord [20] provide latency estimates using distance measurements gathered only between end-hosts in the overlay network. We opted for Vivaldi because of its fully distributed nature and simple implementation. However, Quick-Peer is not tied to a specific coordinate system and can be used with any of the systems cited above.

5 Conclusions and Future Work

This work presents a gossip-based protocol, termed QuickPeer, which builds and maintains latency-aware overlay topologies. Such topologies are useful for several distributed applications, like distributed online gaming, context-aware P2P applications and QoS-aware publish/subscribe systems. The distinctive feature of QuickPeer is that it can manage large scale overlay topologies providing each host in the overlay with its closest or furthest neighbour, according to network distance (RTT), in few gossip rounds.

Experimental results proves Quickpeer scalability, robustness to failures and adaptiveness to scenarios in which large numbers of nodes join the overlay concurrently.

Future investigations will develop further our dynamic scenarios, including extensive churning experiments. In addition, we plan to evaluate QuickPeer on a distributed testbed such as PlanetLab [1].

Acknowledgements

The authors would like to thank Özalp Babaoğlu, Márk Jelasity, Giorgia Lodi, Alberto Montresor, Fabio Panzieri and Spyros Voulgaris for helpful discussion and precious comments on earlier versions of this paper.

References

[1] Planetlab home page, 2004. http://www.planet-lab.org.

[2] The BISON project. http://www.cs.unibo.it/bison.

[3] BRITE Topology Generator. http://www.cs.bu.edu/brite/.

[4] A. Carzaniga, D. S. Rosemblum, and A. L. Wolf. Design and evaluation of a Wide-Area event notification service. *ACM Trans. on Computer Systems*, 19(3):332–383, August 2001.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal in Selected Areas of Communications*, 20(8):1489–1499, October 2002.

[7] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of the ACM SIGCOMM*, August 2001.

[8] M. Costa, M.Castro, A.Rowstron, and P.Key. Pic: Practical internet coordinates for distance estimation. In *Proc. of ICDCS'04*, 2004.

[9] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM '04 Conference*, Portland, Oregon, August 2004.

[10] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC87)*, pages 1–12, 1987.

[11] P. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and L. Massoulié. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[12] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, May 2004.

[13] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: a global internet host distance estimation service. In *Proc. of IEEE Infocom '99*, 1999.

[14] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protcols. *IEEE Transactions on Computers*, 52, 2 2003.

[15] Gnutella web site. http://www.gnutella.com.

[16] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. USENIX OSDI*, Oct 2000.

[17] M. Jelasity and O. Babaoglu. T-man: fast gossip-based construction of large-scale topologies. Technical Report UBLCS-2004-7, Dept. of Computer Science, University of Bologna, 2004.

[18] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast Computing. Technical Report IR-CS-006, Vrije Universiteit, Department of Computer Science, Amsterdam, 2003.

[19] D. Kostić, A. Rodriguez, J. Albrecht, and A.Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[20] L. Lehman and S. Lerman. PCoord: network position estimation using peer-to-peer measurements. In *Proc. of the 3rd IEEE International Symposium on Network Computing and Applications (NCA'04)*, 2004.

[21] L. Massoulié, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proc. of the 22nd International Symposium on scalable and distributed systems (SRDS'03)*, 2003.

[22] T. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. of IEEE Infocom*, 2002.

[23] P. Pietzuch and J. Bacon. Hermes: a distributed event-based middleware architecture. In *Proc. of the First Intl. Workshop on Distributed Event-Based Systems (DEBS'02)*, pages 611–618, June 2002.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM'01*, pages 161–172, 2001.

[25] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM*, June 2002.

[26] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakhrisnan. Chord: a scalable, peer-to-peer lookup protocol for internet applications. In *Proc. of ACM SIGCOMM 2001*, pages 149–160, 2001.

[29] Freenet. http://freenet.sourceforge.net.

[30] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient, global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.