Exploring Proximity Based Peer Selection in a BitTorrent-like Protocol

Asfandyar Qureshi
(asfandyar@mit.edu)

May 7, 2004

Abstract

BitTorrent [1] is popular file-sharing tool, accounting for a significant proportion of Internet traffic. Files are divided into fragments and transferred out of order among nodes trying to download them. Individual nodes penalize and reward other nodes, adjacent in the BitTorrent overlay, depending on how willing others are to share data. This incentives scheme and the subsequent enforcement of sharing is credited with making BitTorrent outperform other contemporary file-sharing systems.

Nonetheless, BitTorrent builds its overlays by randomly selecting peers, a fact that has the potential to seriously handicap both individual performance and waste global network resources. This paper investigates a scheme which attempts to build a more intelligent overlay network, particularly using synthetic network coordinates to select overlay peers that are *close by* in the underlying network. We evaluate our techniques both from the network's perspective (resources used) and from the individual's perspective (average time to complete a download). In both cases, our results show that better peer selection can lead to improved performance with no major changes to the basic BitTorrent protocol. Our evaluation is based on real-world experiments performed over Planet-Lab [2].

1 Introduction

Traffic logs on the Abilene network [3] show that in March 2004 BitTorrent accounted for almost as much logged traffic as HTTP. Additionally, BitTorrent transfers accounted for about three times the amount of octets associated with all other file sharing applications combined. Weekly reports through March indicate a steady increase in BitTorrent traffic.

With the development of ways to search BitTorrent networks for files and the adoption of BitTorrent-variants for legal activities—such as the distribution of Linux ISOs and online game binaries—this usage is likely to continue to rise.

This paper focuses on the BitTorrent overlay and how it is used to transfer data. We largely ignore other aspects of the protocol, such as its incentives scheme. Our goal is to focus on one specific problem (the overlay construction), evaluate an alternative using real-world globally distributed experiments, and to design a robust and distributed algorithm that requires only minor modifications to the current protocol.

BitTorrent avoids the P2P search problem and builds a different overlay networ for each file that is being shared. Clients contact a known server, the *tracker*, to join the overlay network. There is a different tracker for each file being shared. BitTorrent divides a file into fragments and clients download those fragments individually, possibly out of order.

Each node is assigned a set of peers as soon as

it joins the overlay. This set is static, in the sense that a node x never proactively seeks out new peers, but new nodes may request to peer with node x and some of node x's peers may leave the overlay. A node will only try to download file fragments directly from its peers; it will never try to search for a desired fragment using the overlay.

What sets BitTorrent apart from other P2P designs is its enforcement of sharing. In the abstract, it models the P2P file-sharing domain as a repeated evolutionary prisoner's dilemma and incentivizes rational individuals in the P2P network to contribute some upload bandwidth to others.

Each BitTorrent client tries to acquire the rarest fragment among its peers, this leads to the desirable property that any set of peers can simultaneously download and upload to each other. When a client notices that one of its peers is not uploading to it that client *chokes* the connection between the two, refusing to upload to that peer. Alternatively, a client rewards peers by priortizing uploads for peers from which download rates are high.

While this works quite well in practice, the formation of a highly sub-optimal BitTorrent overlay network is not avoided in the present design of the protocol. When a node joins a *torrent*, the tracker gives it a list of peers, placing the new node in the overlay network. The present approach in BitTorrent trackers is to return a random list of peers, from all the known nodes that are part of the overlay. This randomization can result in there being a significant divergence between the logical overlay network and the underlying network. Two nodes that are peers in the overlay may be on different coasts, in different countries, or even on different planets. The present overlay construction algorithm would not consider either case worse than the other.

We believe that the BitTorrent overlay network should be constructed so as to better reflect the underlying network topology. Clients in the *torrent* who are close by in the *real world*, particularly those within the same AS, should be close by in the overlay network. Such an overlay building approach would both reduce average latencies between peers and is likely to boost the average bandwidth, since clients on the same subnet are now far more likely to be paired together than they were before.

If the number of clients were small, one could use network proximity measurements (e.g. round-trip times) between each client pair to build an overlay minimizing latencies between peers. Such a simple approach does not scale well, and real-world BitTorrent overlays are quite large (we assume they are on the order of thousands of nodes). Section 3 presents an approach to building overlays which tries to avoid this scalability problem. We base our approach on a distributed algorithm for calculating synthetic network coordinates [4] coupled with a probabistic flooding algorithm borrowed from ad hoc sensor network research [5]. Our overlay retains some degree of randomization to avoid hot-spots from developing pathologically.

An important side-effect of chosing peers in the overlay that are close by emerges from the *rarest fragment first* algorithm used by the nodes for deciding which fragment to download next. Essentially, each node looks at all the file fragments held by his peers and then tries to download the rarest one. Thus, uncommon fragments are replicated more quickly than common ones. Further, among any set of peers, all nodes are connected to the *seed* node, the original server having all the fragments. Ideally, due to the coordinated replication strategy, a duplicate request will never be sent to the seed for the same fragment, as long as at least one peer has already requested that fragment and notified all its peers of that request.

When peers are on different coasts, common fragments must still be shipped across the country when they are downloaded from peers in the overlay. When peers are selected—and therefore *clustered*—according to real-world proxim-

ity, only fragments that are rare among peers will normally need to be transferred from far away. In an ideal case, with the improved overlay, for a group of peers on the MIT subnet, each fragment would only be transferred from outside MIT exactly once and all other transfers would occur on the internal MIT subnet. With a random overlay, the MIT nodes may quite reasonably be placed on the overlay many (overlay) hops apart.

We believe that selecting an overlay network that corresponds more closely to the underlying network proximity between the peers can result both in increased individual performance (downloads finish earlier, because of better peer selection) and a more efficient usage of network resources (packets need to travel shorter distances between peers). Both are desirable goals, making the overlay construction problem important.

The rest of this paper is structured as follows: §2 is a more formal statement of the problem and an outline of the P2P protocol we will be using; §3 describes our approach to building a proximity based overlay; §4 evaluates our implementation, comparing it against the random graph building used today; §5 provides our conclusions; and finally §6 surveys relevant related work.

2 Problem Statement

This section first outlines the basic peer-to-peer protocol we will be modifying and then explains the goals that drive our modification decisions.

2.1 Basic Protocol Model

The peer-to-peer protocol we are experimenting with is based on BitTorrent, but incompatible since we use a different wire protocol and a relaxed threat model. This incompatibility is a result of the need for rapidly building a prototype, not a fundamental flaw in our design. Later (§3) we present our overlay construction algorithm as a set of simple modifications to the protocol outlined here.

Node Arrivals and the Tracker A node joins the overlay network by contacting a known *tracker* node. The only job of the tracker is to maintain the list of nodes who have joined the network and to provide a random set of peers to each new node. This set is a random subset of the set of all nodes that have contacted the tracker before the newly joining node.

Downloading/Uploading The goal of every client in the network is to acquire the complete set F of fragments that constitute the file the overlay has been created for. All fragments are of the same length. A node only downloads fragments from each of its direct peers. A node also simultaneously uploads any requested fragments to each of its peers. The maximum upload rate provided to any given peer is the same as the download rate from that peer, based on moving averages. To avoid the pathological case where both peers stop uploading to each other due to transient congestion or the like, if a peer is limiting uploads to another, it will try to optimistically increase its upload rate to the other peer and see if the other peer responds with a symmetric increase.

Rarest Fragment First Each node shares with all its peers the set of fragments it has, all the fragments it is downloading and their expected arrival times. Each node uses this information to decide which fragment to download next and which peer to download each fragment from. A node will prefer to download the rarest fragment among its peers. It will try to download from nodes that offer the lowest expected arrival time for that fragment.

Peer Ranking A node continuously ranks all of its peers, only activating the top k of them at any given time—a ranking based on measured moving throughput averages. Downloads and uploads proceed only for active peers. Occassionally, a node randomly and optimistically activates inactive peers, in order to ensure that a

better peer will be used in preference to one in use, if a better peer exists.

Node Departure In our experiments we assume that a node departs the network as soon as it has completed its download of the file. Furthermore, we assume a silent failure model: when a node departs it does so without notifying the tracker or any of its peers. The peers can determine departure since the relevant TCP sockets will generate events, but the tracker's list can become stale.

Failure Model We ignore the problem of Byzantine failures. Essentially, we use a stopping failure model instead of the Byzantine failure one assumed by the original BitTorrent protocol. Consequently, the protocol we experiment with does not implement explicit penalties or verify each fragment against a hash. This is acceptable, since we are only interested in how much better we can do with a good overlay.

2.2 Goals

Our modifications to the protocol previously described are driven by a very specific set of goals.

Proximity Based Overlay We want to build a *network proximity* based overlay. Informally, all things being equal, a node should preferentially select those other nodes as peers with which it has low observed communication latencies. Formally, our overlay building algorithm should have the following property: given a random node arrival process, if any two nodes x and y have a round-trip communication latency $rtt_{[x,y]}$, the nodes x and z have a communication latency $rtt_{[x,z]}$, $rtt_{[x,y]} < rtt_{[x,z]}$ and $(rtt_{[x,z]} - rtt_{[x,y]}) > \varepsilon$, then $P(x,y) > P(x,z)$, where $P(x,y)$ is the probability that x and y are peers in the overlay.

Tracker Limitations In the protocol design, the tracker is a single point of failure. It is required, unfortunately, to introduce any new node to others in the overlay. In pursuit of scalability, we do not want to increase the responsibilities of the tracker and so our protocol modifications do not include any additional processing at the tracker.

Individual Download Time Perhaps the most important property to users of a peer-to-peer file sharing program is the time it takes to completely download an individual file. Our protocol modifications are geared towards—and evaluated on their ability to—minimize the average time to download a file. Another important metric is the time taken by the fastest downloaders. Consequently, intend to evaluate our approach by considering the distribution of download times rather than just the mean.

Network Resources Used Although users do not really care about whether or not network resources are being wasted, reducing such waste is a highly desirable goal. Our approach of proximity peer selection was initially inspired by the desire to optimize the use of available network resources. It is a little harder to evaluate the impact of our algorithms with respect to this metric, given that we will be running globally distributed experiments on Planet-Lab.

3 Design

This section describes a distributed proximity peer selection algorithm as a set of modifications to the previously described protocol. Our algorithm tries to minimize the communication latencies between peers, using synthetic network coordinates [4] and probabilistic flooding to minimize the overhead introduced by having to search for nearby nodes.

At a high level, the peer selection algorithm consists of two phases:

Phase 1: Synthetic Network Coordinates
A node must first approximate its *D*-

dimensional synthetic network coordinates using the peers randomly assigned to it when it joined the overlay.

Phase 2: Nearby Neighbour Discovery
Using its own synthetic network coordinates, a node discovers, and connects to, an additional set of peers that are nearby (by euclidean distance) in the synthetic coordinate space.

We discuss each phase in detail in the next two sections. We name this two phase algorithm *Giraud*, after Anna Giraud[1], in homage to the fact that we are combining *Vivaldi* and *Gossip*.

Our algorithm is meant to run in the background, while the rest of the protocol works on downloading the file. Even before phase 1 is complete, the node knows about some peers and can start downloading fragments right away. Phase 2 continues indefinitely, but runs at a controlled low frequency in the background, as explained later. New, better peers, are discovered and connected to by *Giraud*. The rest of the protocol need not worry about how the overlay is changing. The peer ranking system already in place will use the best peers from among all known ones, weeding out bad choices made by *Giraud*.

3.1 Network Coordinates

A joining node must quickly approximate its *position* in the network. This approximation is then used to find *nearby* neighbours. Before we continue, we must decide on a definite notion of *network position*. To simplify the problem of defining position, we define the *network distance* between two hosts as the measured packet transmission latency between them[2].

Geographical location seems to be a possibile proxy for network location[6]. However, this information is hard to obtain without either involving the user or relying on non-standard hardware (e.g. GPS). Furthermore, given the vagaries of AS relationships and network connectivity in the Internet, geographical distance will not likely be a good proxy for network distance.

Using the enclosing Autonomous System (AS) to define a node's location is another viable possibility. Although, without relying on an internal AS-connectivity model to predict inter-AS network distances, the best we can do is find nodes in the same AS, missing those nearby but in different AS's. Furthermore, treating an AS as a single point is a bad approximation for nodes inside large AS's (e.g. Sprint).

There are other approaches we could take. Landmark based location approximation algorithms exist [7], and we could plausibly use `traceroute`'s to known nodes (e.g. the tracker) in order to gather information about a node's position.

We use a variant of the Vivaldi distributed algorithm [4] to generate synthetic network coordinates. These coordinates are only meant to be consistent within our overlay. The only constraint for picking node coordinates is that the euclidian distance between a pair of nodes in the synthetic coordinate system should be proportional to the measurable network distance between those two nodes.

When a client wishes to join the overlay, it acquires a random list of peers from the tracker as before. Since the node has no knowledge about its relationships to other nodes in the overlay, it cannot do better than selecting a random list of peers. Using these initial peers, their synthetic coordinates and measured communication latencies, the client simulates a force directed spring system to calculate its own coordinates. After some number of measurements, and consequent simulated iterations of the spring system, the client moves into the next phase and begins a search for nearby nodes. As in Vivaldi, the peers will also update their own coordinates. Furthermore, we piggy-back latency measurements on data packets.

We expect lower quality synthetic coordinates

[1] An actress, a member of Antonio Vivaldi's entourage and the cause of much Venetian gossip.

[2] We simplify further by ignoring asymmetric latency measurements.

than those produced by the version of Vivaldi presented by Cox et al, since our variant of Vivaldi uses fewer probes and does not wait as long for the system to stabilize. Nonetheless, we believe these coordinates will be *adequate* hints when it comes to finding nearby nodes. We are not attempting to build the *optimal* overlay.

3.2 Nearby Neighbour Discovery

Once a node has approximated its position, it proactively searches for other nodes in the overlay which are close to it.

A centralized approach to finding nearby nodes is to contact the tracker again, provide it the calculated coordinates and request a list of nearby nodes. However, even with clever lookup algorithms, this would—at least—double the load on the tracker (each node now queries the tracker twice). We avoid contacting the tracker at all during the search phase.

The simplest distributed search would be for the new node to flood a coordinate announcement through the overlay (a message containing the originator's network address and the synthetic coordinates). Whichever nodes realized that the new node was *close by* (using D-dimensional euclidian distance) and were willing to accept more peers, would initiate a direct connection with the announcing node. Although this form of flooding seems likely to waste more network resources than is reasonable, it is otherwise a simple and viable solution to our search problem.

Instead of simple flooding, we propose to use a low-rate probabilistic flooding mechanism similar to that proposed by Haas et al in their `GOSSIP` protocols [5] for ad hoc wireless networks. The basic idea behind their simplest protocol `GOSSIP1(p)` is that when a node receives a route (or, in our case, coordinate) announcement it discards that announcement with uniform probability $(1-p)$ and forwards it on to all its neighbours (overlay peers) with probability p. Using heuristics, this basic protocol can be tweaked to deliver the message to over 90% of the nodes, with the authors demonstrating savings of up to 35% in message overhead, compared to flooding. p can be varied to increase the savings at the expense of the fraction of nodes that receive each announcement.

Giraud's search phases presently uses `GOSSIP1(0.75)`, but further limits the rate of the generated gossip. Every minute a timer fires on the client causing it to send out a bounded size *gossip* message to all of its active peers. This message is a probabilistic aggregation of all the *gossip* messages heard since the last such timer. The content is controlled by the rules of `GOSSIP1(0.75)` and some additional rules apply when too much information has been received. We believe that using a maximum of 32 bytes per second or so for gossipping on each communication channel is acceptable. This information can always be piggybacked onto data packets. $p = 0.75$ was selected based on the results cited by Haas et al [5].

We do not believe the choice of `GOSSIP1` is particularly special, nor is it integral to the functioning of *Giraud*. Certainly more sophisticated protocols exist. The optimal flooding protocol by Paruchuri et al [8] is such a protocol, which could be used in place of `GOSSIP1`.

4 Evaluation

4.1 Methodology

Original Plan Initially, the plan was to write a fully functional real-world client for the protocol that we have described above. Then the client could be evaluated using a set of nodes distributed across Planet-Lab [2] to run our implementation. The set was to consist of hundreds of randomly selected nodes. Using experimental runs of our client on this test-bed we were hoping to determine the nature of the performance improvements, if any, our improved overlay would bring. The primary metric we were concerned with optimizing was individual download times. Since these would be real-world ex-

periments using a completely implemented P2P client program, we hoped this roadmap would produce something that was immediately useful to people outside of 6.824.

Unfortunately, things did not work out so well due to time constraints and some regrettable decisions on our part[3]. While the P2P client code is presently around 80% complete and the Planet-Lab slice is primed, the experiments could not be run in time.

Reality While we believe we will have to run the Planet-Lab tests, or at least `ns2` packet-level simulations, to accurately gauge the impact of an improved overlay on the P2P protocol's performance, simulation can shed some light on how good an overlay *Giraud* actually builds. The results we present here use an implementation of *Giraud* and simulations that ignore packet-level semantics to explore the quality of the overlay built by our algorithm.

4.2 Simulation

This section describes the simulations we ran. Using the King data set [9], an n^2 matrix of measured round-trip-times between n hosts on the internet, we generated a random poisson arrival process at the tracker. Using this process we simulated the following three overlay building algorithms on various subsets of the King data set.

Random When a node x arrived, it was assigned a random subset of peers selected uniformly over the set of all previously arrived nodes[4].

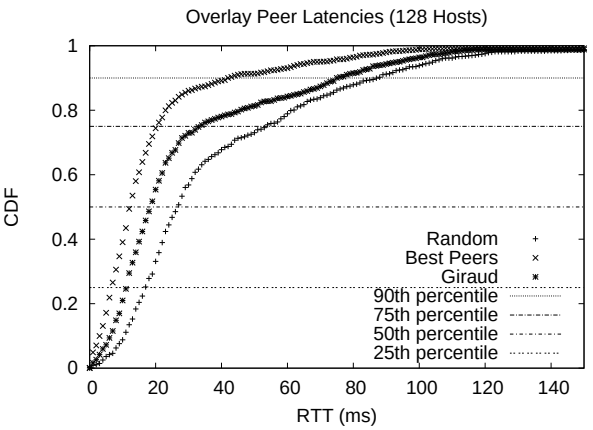

Figure 1: The peer-to-peer round-trip-time CDF measured during simulations for each of the three overlay building techniques. Using *Giraud* instead of *Random* almost doubles the number of peers who are within 20ms of each other in the overlay.

Best-Peers When a node x arrived, it was assigned the *best* set of peers, from all previously arrived nodes. This set was determined by picking the peers which had the K lowest RTT's to x.

Giraud When a node x arrived, it was assigned a random subset of peers from all previously arrived nodes. The number of these peers was about a third of the number assigned in *Random*. x ran about 30 iterations of our vivaldi variant in 7-Dimensional space and then began to gossip. Whenever x heard, through gossip, of a node y such that the euclidian distance to y was less than the average euclidian distance to x's best 8 peers (lowest actual RTTs), then x would connect to y. The gossip timer fired every

[3]Time sinks: 3. Worrying about malicious clients and writing code to handle them; 2. Designing and implementing a `libasynch` style helper library in Java so that the application runs in essentially a single thread, instead of going for the simpler–less efficient–multithreaded alterntive; 1. Worrying about writing well designed and documented code so that I could look my 6.170 students in the eye afterwards. I will probably get round to finishing this after I wrap up my MEng thesis in July.

[4]Even though selection is uniformly random during any one subset selection, nodes with earlier arrival times tend to have a higher expected number of selections than later nodes, since earlier ones are included in more subset selections. We did not compensate for this effect. This effect causes the first few nodes to have a higher average degree than later ones.

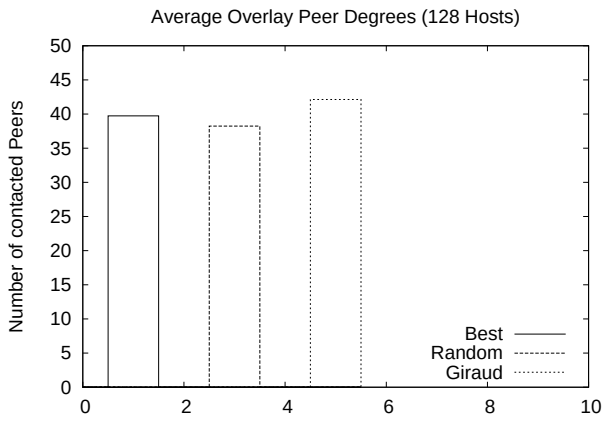

Figure 2: The average degree of a peer in an overlay. There is not much difference between the three methods. Our improved results for *Giraud* are therefore not a consequence of picking a larger set of random peers, but the result of picking a *better* set of peers.

60 seconds.

Simulation Results Figure 1 shows, for each simulated protocol, the CDF of latencies between two peers. The CDF is based on observed frequencies in the following set $R = \{rtt_{i,j}\}$ where $rtt_{i,j} \in R$ iff $rtt_{i,j}$ is the RTT between i and j and either i is one of j's best 8 peers or j is one of i's best 8 peers. The result in the figure is for simulations on a set of 128 nodes. Larger node sets produce similar results.

The first thing to note is how big of a difference there exists between *Random* and *Best-Pairs*. While around 75% of links in the *Best-Pairs* constructed overlay have a latency less than $20ms$, around 25% of links in *Random* have a latency below $20ms$. *Giraud*'s overlay quality lies between the other two. *Giraud* has almost twice the number of links below $20ms$ compared to *Random*.

One possibility for *Giraud*'s improvement is that gossipping is producing a larger average sized set of peers than *Random* has. A larger set could account for all of the improvement in the overlay, since we are only considering the best 8 peer links. Figure 2 shows that this is not the case. In our experiments, *Random* and *Best-Peers* start off with a set of peers $3x$ as large as that initially assigned to a *Giraud* node. The average number of *known* peers turns out to be approximately equal for all three protocols.

Figure 2 shows the average number of other nodes any given node knows about. We assume only the best 8 peers are active at any given time, so even a bloated number here would not be too bad. Gossipping limits the rate at which this set builds up, and our simulations run for over a few hours of simulated time, so a large set does not necessarily imply many wasted resources communicating with them.

5 Conclusions

Although we were not able to demonstrate that a proximity based overlay would visibly effect the distribution of individual download times or lead to a more efficient utilization of global network resources, we did construct, implement and present a reasonably simple distributed algorithm which constructs an intelligent proximity based overlay. Much of the design decisions were driven by the need to make minimal changes to the present BitTorrent protocol. We feel that it should not be too hard to integrate *Giraud* into BitTorrent.

Our protocol can be improved significantly and we hope to finish implementing and experimenting with the clien over this summer.

6 Related Work

Various proposals have been put forward for network coordinate system mechanisms [10] [6] [11] [7] [4]. Of these, one [6] tries to make the coordinate system conform to external geographic realities. As we noted earlier, geographic distance is a bad proxy for network distance, which is what we really want to estimate using the network coordinates. While IDMaps [10], the Internet Coordinate System [11], and Global Net-

work Positioning [7] all promise to estimate network distance well, neither of them are truly distributed, in the sense that we need them to be. All of them use *landmark based* calculations to derive node coordinates; and the BitTorrent overlay does not have enough stable nodes to be used as landmarks. Vivaldi [4] offers a practical distributed algorithm for building a synthetic coordinate space without landmarks, one that we can easily integrate into BitTorrent.

Flooding of announcements is a technique used by almost all unstructured peer-to-peer networks (e.g. Gnutella [12]). It is a provably effective, though wasteful, zero-knowledge routing mechanism. Researchers working on ad hoc wireless sensor networks have had to invent new variants on flooding, since resource wastage is all the more important in such sensor networks while global topology knowledge is quite limited. Broch et al [13] survey various ad hoc sensor network routing protocols. Furthermore, data dissemination protocols for sensor networks, such as Trickle [14], use randomized variants of flooding. We chose to use the gossip protocol presented earlier [5] primarily due to its simplicity. As we noted earlier there are better, more efficient, flooding protocols available. The Optimal Flooding Protocol is one such example [8]. Our choice of rate-limiting announcement floods is driven by the fact that these announcements are not integral to the performance of the protocol.

Content distribution networks, such as Akamai [15], use approaches similar to, but much simpler than, our clustering approach. Since the set of servers is fixed and heavily optimized, their job is considerably easier. Coral [16] uses a DHT-based approach to the CDN problem. Unlike our problem, Coral focuses on finding a *single* nearby copy for a query. In principle, we could have adapted some aspects of Coral's approach, but we decided to base our approach on network coordinates and controlled rate flooding, since these require simpler modifications to BitTorrent.

Rinaldi et al [17] use a variant of CAN and latency measurements to build their overlay network. Joining nodes measure their latencies to some small number of active nodes and then a spring embedder algorithm is used to pick an appropriate ID in the CAN d-dimensional ID space so that overlay distances are close to the measured latencies. While this approach seems to build good overlays, it relies heavily on the semantics of CAN, so it is hard to adapt to the BitTorrent overlay scheme.

References

[1] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer Systems*, 2003. http://bitconjurer.org/BitTorrent/bittorrentecon.pdf.

[2] "Planet-lab." http://www.planet-lab.org/.

[3] "Internet2: Netflow weekly reports." http://netflow.internet2.edu/weekly/20031117/.

[4] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, "Practical, distributed network coordinates," in *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, (Cambridge, Massachusetts), ACM SIGCOMM, November 2003.

[5] L. Li, J. Halpern, and Z. Haas, "Gossip-based ad hoc routing." citeseer.ist.psu.edu/haas01gossipbased.html.

[6] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," *Proceedings of SIGCOMM'2001*, p. 13, 2001. citeseer.ist.psu.edu/padmanabhan01investigation.html.

[7] T. S. E. Ng and H. Zhang, "Global network positioning: A new approach to network distance prediction." citeseer.ist.psu.edu/485391.html.

[8] V. K. Paruchuri, A. Durresi, D. S. Dash, and R. Jain, "Optimal flooding protocol for routing in ad-hoc networks." citeseer.ist.psu.edu/574649.html.

[9] "King data set." http://pdos.lcs.mit.edu/p2psim/kingdata/.

[10] 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," 2000. citeseer.ist.psu.edu/francis00idmaps.html.

[11] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing internet coordinate system based on delay measurement," in *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement*, pp. 129–142, ACM Press, 2003.

[12] "Gnutella." http://www.gnutella.com.

[13] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Mobile Computing and Networking*, pp. 85–97, 1998. citeseer.ist.psu.edu/broch98performance.html.

[14] P. Levis, N. Patel, S. Shenker, and D. Culler, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks.," in *First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004. http://www.cs.berkeley.edu/ pal/pubs/trickle-nsdi04.pdf.

[15] "Akamai technologies." http://www.akamai.com.

[16] M. J. Freedman, E. Freudenthal, and D. Maziéres, "Democratizing content publication with Coral," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 04)*, (San Francisco, CA), March 2004.

[17] R. Rinaldi and M. Waldvogel, "Routing and data location in overlay peer-to-peer networks," Research Report RZ–3433, IBM, July 2002.