Wired Geometric Routing

Jonathan Ledlie, Michael Mitzenmacher, Margo Seltzer
Harvard University
{jonathan,michaelm,margo}@eecs.harvard.edu

Peter Pietzuch
Imperial College London
prp@doc.ic.ac.uk

Abstract

Routing substrates for overlay networks are an important building block for large distributed applications. Many existing substrates are based on a random identifier space and therefore do not respect node locality when routing data. This can lead to lower performance for locality-sensitive applications, such as web caching, distributed gaming, and resource discovery.

This paper examines the problem of building a locality-aware routing substrate on top of a locality-based coordinate system, where the distance between coordinates approximates network latencies. As a starting point, As a starting point we take the scaled θ-routing proposal for geometric routing in a Euclidean space. We address the practical problems of forming routing tables with imperfect node knowledge and churn and examine query performance on non-Euclidean data sets.

1 Introduction

Routing is a basic primitive that must be addressed in any distributed system. Over the previous decade, overlay networks based on the random key paradigm (*e.g.* [21]) have become ubiquitous; they have moved from the research playground to being the core of several systems with millions of users (*e.g.* [2, 4]). Routing with random keys is simple: destinations are defined in the same key namespace as nodes; each node stores routing entries that refer to other nodes across the key space; and nodes forward messages to the neighbor whose key is closest to the target key.

However, the performance of applications, such as distributed web caching, scalable anycast, and resource discovery, often depends on the latency and reliability of the paths taken by messages. For example, in a distributed multi-player game, it is important to host the game server at a node close to the centroid of all client locations to ensure faireness in access latency. Similarly, a web caching infrastructure will exhibit lowest latency when client requests are routed to the closest existing cache.

While the random key paradigm has many intrinsic benefits, such as load-balancing and resilience to churn, by its very nature it does not respect physical geography or location of nodes when routing data. To be sure, there exist techniques for augmenting random key routing schemes with a limited degree of proximity (*e.g.* [19]), but random key routing destroys locality between nodes when building routing tables.

In contrast to this dominant body of work on random key routing, a series of theoretical papers have begun to examine routing schemes for wired networks explicitly based on the underlying physical topology [1, 8, 11]. Instead of assigning random keys to nodes, these proposals assume that each node has a d-dimensional *network coordinate* and that the set of coordinates encode the positions of nodes in the network [3, 17, 18, 20, 22]. Previous work made the assumption that a coordinate substrate existed and asked: what algorithms route on top of such a substrate efficiently? This paper begins where this more theoretical research leaves off.

In this paper, we take a first look at both the practical problems associated with coordinate-based routing and the performance one could expect from deployment on a distributed system. In particular, we examine the challenges of (a) forming and maintaining locality-aware routing tables, (b) routing to the node *nearest* a coordinate (the analog of finding the root for a distributed hash table query), and (c) the primary parameter tradeoffs in coordinate-based routing.

The rest of the paper is organized as follows. We introduce the problem of overlay routing in wired overlay networks in Section 2. We describe proposals for Euclidean plane routing algorithms in Section 3. In Section 4, we address the central challenges for this work to be applicable in a real network setting, for example, by proposing a construction for routing tables through a join protocol and background gossip. In Section 5, we evaluate the performance of routing in a d-dimensional coordinate space using simulation; in particular, we analyze the state-accuracy trade-off and accuracy under churn. We discuss related work in Section 6 and conclude in Section 7.

2 Background

Routing techniques that take advantage of network geometry first arose in the context of *wireless networks* [5, 9, 14]. In a wireless network, deployed nodes are capable of communicating directly with only those nodes in their vicinity through radio communication. Taking advantage of a node's local knowledge of its neighbors and a "sense of direction" for routing decisions results in efficient routing algorithms. Fundamentally, such wireless geometric routing is locality-preserving because the network does not permit long-distance hops due to the limited range of radio links.

To apply similar ideas to *wired overlay networks*, nodes must obtain knowledge of their location within the network. For the reminder of this paper, we define node locations based on communication latencies between nodes, as opposed to geographic location. This is because one metric for routing efficiency is the delay caused by message routing. Wired overlay networks support long-distance hops for direct communication between nodes outside of each others' vicinities. Both the lack of knowledge of one's immediate neighbors and the potential for intelligently placed long-distance edges make routing in the wired and wireless domains significantly distinct.

There exist several proposals for assigning coordinates to nodes so that the distance between two coordinates estimates communication latency [3, 17, 18, 20, 22]. These embeddings of inter-node latencies into a Euclidean space are imprecise because Internet routing often violates the triangle inequality [15]. However, for many practical purposes it is possible to achieve adequate accuracy [13]. Originally conceived as a way to reduce the overhead of latency measurements [6], network coordinates make it possible to apply geometric routing techniques to wired overlay networks.

We consider the problem of wired geometric routing as follows. Given (1) that nodes have coordinates that encode their location and (2) each node has a *routing table* containing a small set of other nodes, we want an efficient method to (a) setup routing tables and (b) route messages to a target location via multiple hops. A target location may be either the exact coordinate a node or a coordinate for which the (approximate) closest node should be found. This *approximate nearest neighbor routing* facilitates the implementation of many locality-dependent applications, such as a distributed web cache, where a client request for an external web page is routed to the closest existing web cache in the network.

The recent theoretical work on Euclidean routing algorithms described in the next section provides the foundation for coordinate-based routing. However, given that we can now construct coordinate systems of low dimensionality that are accurate on many networks, a number of interesting questions remain to be resolved: (1) how can routing tables be constructed in a decentralized manner? (2) how can they be maintained under churn? (3) how do the existing methods need to change to support nearest neighbor queries? (4) how does the error inherent in network coordinates affect the utility of routing results? We build on this prior work by highlighting the practical problems with wired geometric routing and examine what can be expected of its performance in realistic settings.

3 Scaled θ-routing

There exist several proposals for efficient geometric routing in a two-dimensional Euclidean plane [1, 8, 11]. We focus on scaled θ-routing because its assumptions make it directly applicable to wired overlay networks with network coordinates. In particular,

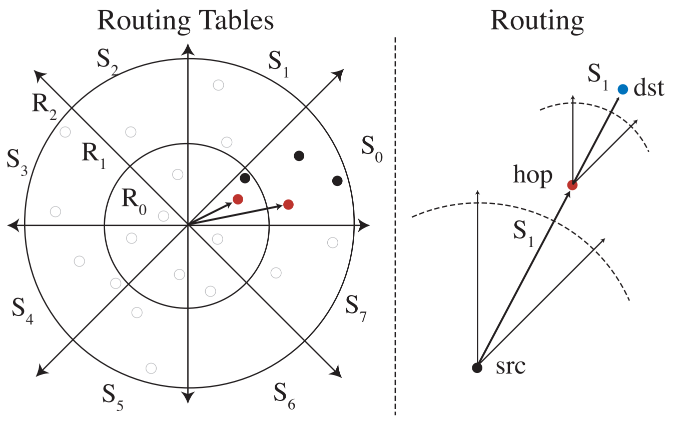


Figure 1: **$\hat{\theta}$-routing** subdivides the coordinate space into $\frac{2\pi}{\theta}$ sectors and r rings. We show how a node selects its closest neighbors in sector S_0 for rings R_0 and R_1 in a network where $\theta = \frac{\pi}{4}$ and $r = 3$. Nodes route to destinations by (a) calculating the zone of the destination and (b) greedily forwarding the message to the furthest hop does not exceed the target's zone.

it assumes that (1) each node has a coordinate in a Euclidean space and (2) nodes do not have a maximum communication range: any node may connect with any other.

Scaled θ-routing is based on Keil and Gutwin's θ-routing [10]. θ-routing takes advantage of the "sense of direction" in a Euclidean plane by routing "towards" a given target. and is based on the construction of a θ-graph spanner. Such a graph has the property that each pair of nodes are connected by a "short" path. More formally, for any two nodes n_1 and n_2, the length of the routing path (n_1, n_2) is no more than a constant factor times the Euclidean distance $\mid n_1 n_2 \mid$. The construction of a two-dimensional θ-graph spanner is simple. Each node subdivides the area around it into $\frac{2\pi}{\theta}$ *sectors* of angle θ. For each sector, it inserts the nearest neighbor into its routing table. Messages are forwarded by sending them to the neighbor in the same sector as the target location. The advantages of θ-routing are that the routing table size is small and routing paths have low delay stretch, which is the relative penalty of the route taken compared to direct IP routing (because they hold closely to the line between the source and the target). However, it suffers from a linear hop count that is proportional to the diameter of the network.

Hassin and Peleg [8] improve on the linear hop count of θ-routing by adding long-distance links to traverse large distances in a single hop. In *scaled θ-routing*, abbreviated as $\hat{\theta}$-routing, the θ-graph spanner is augmented with exponentially expanding rings. To create its routing tables, each node partitions the space around it into $\frac{2\pi}{\theta}$ sectors and r rings. The areas obtained through the intersection of sectors and rings are called *zones*. Nodes keep as neighbors the nearest node that they know in each zone. We illustrate $\hat{\theta}$-routing in Figure 1. Compared to *θ-routing* , *$\hat{\theta}$-routing* reduces the hop count to logarithmic complexity, but increases the amount of routing state each node maintains.

4 Practical Wired Geometric Routing

Given Hassin and Peleg's algorithm and given the ability to construct stable and accurate live network coordinate systems [13], we needed to address numerous practical barriers that apply to $\hat{\theta}$-routing and other geometric routing proposals. In this section, we highlight three main challenges to coordinate-based routing and our initial proposals for how to address them. The challenges are: (a) mapping nodes to zones when $d > 2$, (b) efficiently learning neighbors, and (c) performing nearest neighbor queries instead of ones with a known target coordinate.

4.1 Zone Assignment

As illustrated in Figure 1, the process of assigning neighbors to sectors is intuitive in two dimensions: given the angle θ_i of the neighbor i and a set of rays each θ degrees apart, we simply find between which rays θ_i lies. However, this process proved surprisingly challenging in three or more dimensions.

A straightforward example in more than two dimensions is the three dimensional case when $\theta = \frac{\pi}{2}$. Here, one can readily visualize that there are 8 sectors with the axes acting as the rays that divide each sector. In addition, the assignment of neighbors to zones is fairly intuitive: we can find the spherical coordinates θ and ϕ and perform essentially the same method of assignment as we did in two dimensions. Note that one of the two angles will range from $(-\pi \ldots \pi)$ and the other from $(0 \ldots \pi)$; one completes the full circle but the other needs to traverse only half that to cover all possible directions. Each angle can then be dealt with separately, resulting in four possible sectors from the angle with the larger domain and two from the one with the smaller.

While harder to visualize, this method generalizes to an arbitrary number of dimensions. It is simpler to operate on Cartesian coordinates for network coordinate refinement processes such as Vivaldi [3]. For zone assignment, however, we convert the d-dimensional vectors from nodes to their neighbors into a set of hyperspherical coordinates $\phi_0, \ldots, \phi_{d-1}$, again with the proviso that one angle completes the full circle. Because each additional dimension effectively "slices" the sectors of prior dimensions, the total number of sectors increases exponentially with the number of dimensions $2 \times s \times s^{d-2}$, where s is the number of sectors per standard dimension.

The second component of zone assignment is determining the neighbor's ring. With simple Euclidean coordinates, the L_2 norm defines distance and therefore the appropriate ring. However, some schemes redefine distance using *height* to capture access link latencies: the delay that all communication must incur. In that case, the distances between the vector components remain Euclidean, but including the height component, they are not. Because height captures delay "above" not "across" a network, we only use the non-height component of our coordinates for ring assignment.

4.2 Maintenance Protocol

The routing algorithms described in Section 3 make greedy routing decisions by nature — that is, without global knowledge or knowledge of the prior path of the query — but rely on omniscient assignment of neighbors to routing tables. This assumption of global knowledge needs to be removed for use in a distributed setting.

In addition, we are not aware of any work on the resilience of coordinate-based routing schemes to imperfect knowledge of the network. This imperfect knowledge leads to incomplete or imperfect routing tables. For $\hat{\theta}$-routing, there are three possible error conditions for a given zone: (1) a neighbor is not the truly nearest node in that zone, (2) a neighbor has failed, or (3) there is not an entry for a neighbor when one exists. Intuitively, the most important entries in the routing table are the nearest nodes in each sector. With only these edges, routing proceeds correctly, just with a higher hop count. This has an analog in one-dimensional DHT routing: routing succeeds as long as every node knows its direct successor in the namespace. Thus, acquiring and maintaining this local knowledge first and then optimizing long-range links second seems a sensible approach to constructing neighbor tables for $\hat{\theta}$-routing.

We assume a simple periodic gossip mechanism exists that allows nodes to exchange routing tables. Performed at random using an existing neighbor from the table, this strategy eventually leads to knowledge of one's local neighbors given enough time and limited churn. We assume that each node bootstraps into the system with a single randomly chosen neighbor. We further assume that dead nodes are discovered through failed routes or failed gossip.

Given the benefit of focusing on local neighbors primarily, we borrow a simple technique from Pastry [19]: instead of exchanging routing tables with a bootstrap neighbor, a node performs a query for its own coordinate using this neighbor. This special query is flagged so that each node on the path adds its routing table to the message. When the message is eventually delivered to the node, it is likely to contain at least one nearby neighbor, whose routing table will be similar to what the new routing table should be.

We define the performance metric *local knowledge* to be the fraction of sectors for which each node has correctly identified its nearest nodes, averaged over all nodes. We have found experimentally that this metric is strongly correlated with routing results, that is, correctly reaching a target. One could imagine a more complex metric of *weighted local knowledge* for which a gossip strategy could optimize: instead of attempting to learn only of the nearest nodes in each sector, the strategy would aim to learn of the larger vicinity, but give more weight to a node's immediate locality. Both metrics could be refined based on the number of sectors and the distance of each neighbor.

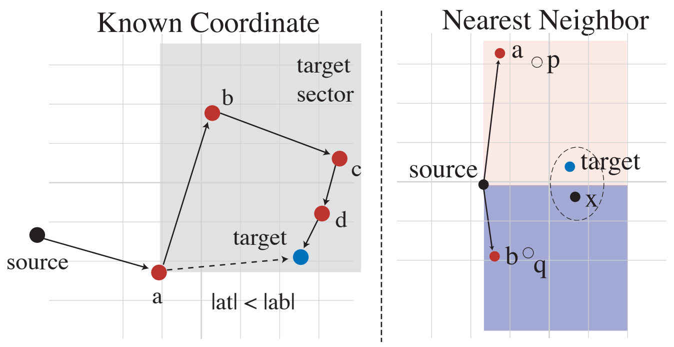


Figure 2: **Local minima with Greedy-by-Distance** occur even with "optimal" routing tables, where each node links to the nearest node in each of its zones. In the Known Coordinate case, with a source s and a target t, the path halts at a because it is nearer to the target than any of its neighbors ($\theta = \frac{\pi}{2}$). The query for the nearest neighbor t halts in a similar situation: at s when, in fact, x is the nearest node. Even if s stored its neighbors' routing tables (as in [16]), it would remain stuck assuming $\|st\| \leq \{\|at\|, \|bt\|, \|pt\|, \|qt\|\}$. Greedy-by-sector routing finds the correct target in both cases.

4.3 Known Coordinate vs. Nearest Neighbor

Euclidean routing algorithms make an assumption that is impossible to attain in a dynamic distributed system: that the source of each query knows precisely the target node's current coordinate. Because node locations change and coordinates are thus constantly under adjustment, one cannot assume that a target node's coordinate remains unchanged even over short periods of time. In a real-world scenario, without perfect knowledge, a natural basis for routing lies in using nearest neighbor queries. As mentioned in Section 2, nearest neighbor queries also aid in the discovery of computed target positions and are important for many locality-aware applications.

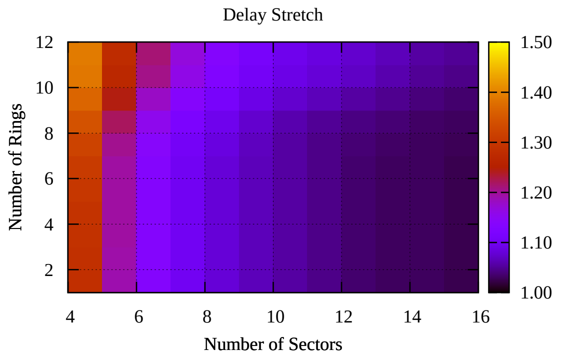
An obvious optimization to $\hat{\theta}$-routing frequently foundered on nearest neighbor queries. $\hat{\theta}$-routing is intended to operate greedily on sectors; that is, we forward to the neighbor that covers the maximum distance to the target while remaining in that sector. There may exist a node, not in the same sector as the target, but much closer to it than the in-sector node. This greedy-by-distance routing, however, leads to local minima as we illustrate in Figure 2. Because greedy-by-sector routing can also fail to find the *exact* nearest neighbor and because *all* queries in this context are forms of finding the nearest neighbor to a target point, we encourage theoretical work in this direction. In practice, we only used greedy-by-distance routing if no neighbor existed in the target's sector.

5 Results

We began our investigation of geometric routing by understanding the parameter space tradeoffs and expected routing performance. In particular, explored the sector and ring parameter space, implemented and tested nearest neighbor routing with network coordinates derived from a real world data set, and examined how the maintenance method we proposed in Section 4.2 performed under churn. The main results from our experimentation are:

- The benefit of each additional ring is closely tied to the base of the ring radius compared to the network diameter; rings of wrong radii offer little benefit.
- Finding the true nearest neighbor to a node (in terms of latency) is almost entirely dependent on the accuracy of the underlying network coordinates; any improvement to these in turn will directly improve nearest neighbor accuracy.
- There exists a very strong correlation between knowing nodes in the local vicinity (the *local knowledge* metric) and finding the correct nearest neighbor under churn. Our join technique works well under high churn (when nodes are using it frequently), but a method to maintain strong local knowledge is needed.

We implemented $\hat{\theta}$-routing in a simulator that allowed us to easily vary its central parameters, inter-node latencies, and sets of network coordinates. We assume there is no message loss and that inter-node latencies are static during the experiments.



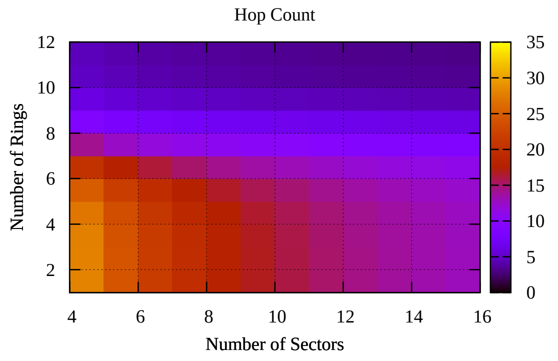


Figure 3: **For two-dimensional routing with "perfect" routing tables,** far reaching rings and few angles result in queries that zig-zag across the coordinate space. Additionally, ring radii need to be tuned to the size and shape of the network to reduce hop count.

5.1 Sector and Ring Parameters

Hop count and delay stretch are important metrics for any overlay routing strategy. We set up an experiment that captured the theoretical model for $\hat{\theta}$-routing: "perfect" routing tables and queries were for a participant's true coordinate. We show results from the two-dimensional case where 1024 coordinates were chosen from a unit square in Figure 3. The sector and ring parameters have distinct, interesting effects.

The data show that with long edges (large r) available from only a few sectors, routes zig-zag towards the target, resulting in a high delay stretch. As the number of sectors increases, each hop veers less from the direct line between the source and the target, diminishing stretch. But because increasing the number of sectors increases state exponentially, a decision to use small angles with $d > 2$ can cause high overhead. Because the dimensionality of network coordinates tends to range from three to about seven, very small angles may not be a practical option.

A base b determines the exponentially-increasing radius of each ring. With a unit square, a base of $\frac{1}{1000}$ expands to cover the network after twelve rings, but because the average distance between nodes is $\frac{1}{3}$, nearby rings are of little benefit (if they contain any nodes at all).

A further complication with choosing a good sector, angle, and base is the shape of the network. We have found that, in network coordinate systems composed of Internet nodes, one or two dimensions are dominant; they capture the physical distance across the Earth's surface while remaining dimensions serve as "wiggle room" that minimizes error [12]. A good parameter choice in one dimension may not be appropriate across all; designing asymmetric systems may be appropriate in practice and is an issue for future work.

5.2 Nearest Neighbors with Network Coordinates

While the performance of $\hat{\theta}$-routing routing on a hypercube confirms our intuition on the roles of different parameters, it does not illuminate what to expect from wired geometric routing on network coordinates derived from the Internet. In addition, the previous experiment does not suggest how nearest neighbor queries — not queries for active participants — would perform.

To distinguish between the exigencies brought about by real network coordinates and those brought about by churn and other causes of partial network knowledge, in this experiment nodes are again assigned their "perfect" routing tables. We created a low-error (median $\approx 6\%$), four-dimensions-plus-height embedding from the inter-DNS server trace from Dabek *et al.* [3] gathered with the King method [7]. Here the average latency between two nodes is $180ms$ and the network diameter is $800ms$. We performed nearest neighbor queries by designating 10% of the 1740 nodes as non-participant targets. We let $r = 8$, $b = 4$, and $s = 6$. Experimenting with this system, we found that, although all queries found the nearest or second nearest node, this node was often not the truly closest node to the target in terms of latency. Because queries were finding the node with the correct coordinate, almost all of the latency penalty an application would experience using that node instead of the true nearest was due to the error inherent in the embedding process. Because queries took on average 4.18 hops, a simple, inexpensive optimization to this process was for each node on the routing path to measure its latency to the target and then to report the node with the lowest latency as the nearest node at the end of the query. We plot the results from $50k$ queries in Figure 4 (left). The results show that this technique reduced the absolute latency penalty by 43% at the median (to $7ms$). Because the latency penalty is so dependent on the embedding error, we anticipate that any improvement in coordinate accuracy will further improve these results (see Zhang *et al.* for a more theoretical discussion on nearest network coordinates [25]).

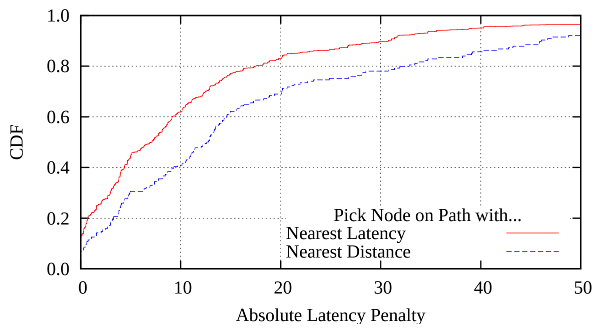
5.3 Finding Nearest Neighbors under Churn

Our final question was whether this process of finding nearby neighbors could be created and sustained under the more realistic setting of node arrival and departure. To simulate churn, we set nodes' lifetimes to a Poisson-distributed random variable and varied the mean. We assumed nodes gossipped with a neighbor from their routing table once every five minutes and let them either bootstrap using a random node or follow the Pastry-like join mechanism described in Section 4.2. In Figure 4 (right), we show the 80^{th} percentile absolute latency penalty (again compared to the node with the lowest latency to the target). The data exhibit two interesting characteristics. First, the join protocol reduces the latency penalty by 34% at the highest level of churn. Routing succeeds at this churn rate because nodes typically have knowledge of 54% of their immediate neighbors (their local knowledge). Second, as the churn rate falls, the join technique lessens in importance; again this is reflected in their local knowledge metric: it falls to 36% with average lifetimes of two hours. In fact, we found a very strong correlation between the this metric and the latency penalty ($R^2 = 0.87$). Together these characteristics suggest the routing table state may be significantly optimized with little or no loss in accuracy as long as a strong set of links to immediate neighbors is kept.

6 Related Work

In this section we provide a brief overview of other efforts to build locality-aware overlay networks and of other geometric routing strategies. To the best of our knowledge, none of the previous approaches have attempted to directly leverage theoretical work on routing in Euclidean spaces to build a general-purpose routing substrate. Instead, they were built with a specific application in mind.

The closest work to ours is *Mithos* [23], a locality-aware routing substrate on top of virtually assigned coordinates. Each node calculates a static coordinate depending on network locality and then maintains links to its immediate neighbors in each quadrant. It can thus be expressed as θ-routing with $\theta = \pi/2$. Messages



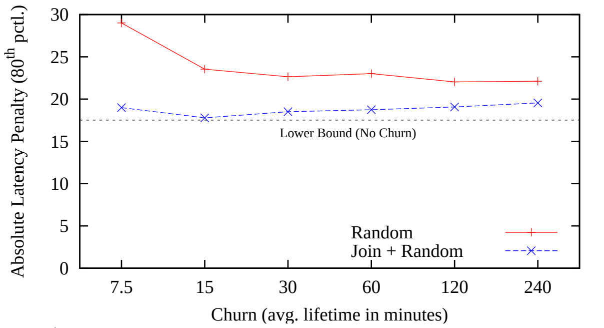


Figure 4: **Wired geometric routing on real-world network coordinates** performs approximate nearest neighbor queries without churn (left) and with it (right). Even with churn, the node with the nearest coordinate is typically found; the figures show the latency of the node returned relative to the node with the true lowest latency.

are routed greedily towards a target. Since routing tables do not contain long-distance links, hop count is linear unless a space with high dimensionality (and overhead) is used. Also, the coordinates are not updated dynamically to adapt to latency changes.

At the other end of the spectrum, *Meridian* [24] is specialized, locality-aware overlay that aims at providing a network location service. Neighbors are organized in concentric, non-overlapping rings with a fixed number of nodes per ring. As such, it can be regarded as a $\hat{\theta}$-routing scheme with $\theta = 2\pi$, where several nodes per zone are used. Routing to a target location proceeds by taking repeated latency probes to the target from neighbors in the associated ring and forwarding the request to the closest neighbor. In contrast to coordinate-based schemes, the active measurement approach results in higher accuracy but also in an increase in measurement overhead.

We picked $\hat{\theta}$-routing as the basic algorithm for a practical locality-aware routing substrate because of its good complexity properties and simple algorithmic construction. However, more intricate algorithms with better complexity exist. For example, *compact routing* proposed by Abraham and Malkhi [1] comes close to optimal space. However, this improvement in per-node state is balanced by a more complex construction that only amortizes for very large networks.

7 Conclusion

In this paper, we presented a practical locality-aware routing substrate that takes advantages of recent advances in algorithms for routing in a Euclidean plane. We addressed important challenges, such as routing table construction under imperfect knowledge, maintenance protocols for new nodes, and the support for nearest neighbor queries.

We believe that a geometric interpretation of wired routing problems opens up new avenues for solving many problems in distributed applications. As future work, we will explore the space of locality-aware, coordinate-based overlays more fully and expect that they will prove useful building blocks, similar to random key-based overlays. As a next step, we plan to analyze the properties of the set of routing hops taken in wired geometric routing. Some applications may depend on characteristics of the routing path, such as quick path convergence or uniform hop spread. For instance, in a web caching application, data may be stored at hops along the routing path to a target web server for faster future retrieval.

References

[1] I. Abraham and D. Malkhi. Compact routing on euclidian metrics. In *PODC*, July 2004.

[2] Azureus BitTorrent Client. azureus.sourceforce.net.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM*, Aug. 2004.

[4] eMule File Sharing Client. emule-project.net/.

[5] G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Tech. Report RR-87-180, ISI/USC, 1987.

[6] P. Francis et al. IDMaps: a global internet host distance estimation service. *IEEE/ACM Trans. Networking*, 9(5), 2001.

[7] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Lat. between Arbitrary Internet End Hosts. In *IMW*, Nov. 2002.

[8] Y. Hassin and D. Peleg. Sparse Communication Networks and Efficient Routing in the Plane. In *PODC*, July 2000.

[9] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MobiCom*, Aug. 2000.

[10] J. M. Keil and C. A. Gutwin. Classes of Graphs Which Approx. the Complete Eucl. Graph. *Discrete & Comp. Geom.*, 7(1), 1992.

[11] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *STOC*, May 2000.

[12] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. Technical report, Harvard University, Oct. 2006.

[13] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and Accurate Network Coordinates. In *ICDCS*, July 2006.

[14] B. Leong, B. Liskov, and R. Morris. Geographic Routing without Planarization. In *NSDI*, May 2006.

[15] E. K. Lua, T. Griffin, M. Pias, et al. On the Accuracy of Embeddings for Internet Coordinate Systems. In *IMC*, Oct. 2005.

[16] M. Naor and U. Wieder. Know thy Neighbor's Neighbor: Better Routing for Skip-Graphs and Small Worlds. In *IPTPS*, Feb. 2004.

[17] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, June 2002.

[18] M. Pias, J. Crowcroft, S. Wilbur, et al. Lighthouses for Scalable Distributed Location. In *IPTPS*, February 2003.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, DOLR for Large-Scale P2P Systems. In *Middleware*, Nov. 2001.

[20] Y. Shavitt and T. Tankel. Big-Bang Simulation for embedding network distances in Euclidean space. In *INFOCOM*, June 2003.

[21] I. Stoica, R. Morris, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, Aug. 2001.

[22] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *IMC*, Oct. 2003.

[23] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[24] B. Wong et al. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *SIGCOMM*, Aug. 2005.

[25] R. Zhang et al. Impact of the Inaccuracy of Distance Prediction Algorithms on Internet Applications. In *INFOCOM*, Apr. 2006.