Don't Love Thy Nearest Neighbor

Cristian Lumezanu
Georgia Tech

Dave Levin, Bo Han, Neil Spring, Bobby Bhattacharjee
University of Maryland

1 Introduction

One of the most flexible features of Internet routing—that service providers have administrative autonomy over their networks—has, over the years, proven to be one of the most restrictive—users have little to no say in how their packets are forwarded, nor what kind of quality of service they receive.

As online services' and users' demands have evolved, the goals of Internet routing have broadened. For example, although ISPs engineer for high throughput, users often benefit from low-latency paths, as well [22]. Also, recent concerns over traffic shaping [10] have led to demands for net-neutral routing mechanisms that subvert ISPs' routing policies [29]. Overlay routing systems [1, 22] address some of these concerns by allowing users to forward traffic through one another. Because they can choose the overlay path, users have some say in what path their packets take.

An important primitive in overlay routing systems is *node location*: the ability to find nodes that satisfy latency constraints. For example, content distribution networks (CDNs) [11] and online games' matchmaking services [7] are more efficient when performed through hosts that decrease delay to receivers [27, 15]. Network coordinates [9, 23] offer a practical and scalable solution to node location by assigning nodes position in a geometric space such that the distance between the positions estimates the real Internet latency.

Viewing hosts as network coordinates, many complex routing queries can be rephrased as *geometry* problems [25]. For instance, the problem of finding a game server that minimizes the average latency to a set of players' machines can be thought of as finding the game server that is the centroid of the players' network coordinates. Of course, it is highly unlikely that there will be a server at the precise centroid; the goal is thus to find the node "closest" to such an ideal point—typically called *nearest neighbor* search.

In this paper, we find that nearest neighbor search often does not result in the *best* result for a given query. Figure 1(top) shows the contour plot of a possible cost function for the centroid computation problem. B and N represent nodes in the Internet and M is the theoretically optimal centroid for a set of nodes (not shown on the plot). Although N is closer to M than B, B is in fact a better choice for the centroid, since it lies on a lower cost contour.

We present the design of *Sherpa*, an overlay routing system in which routing queries take the form of arbitrary, continuous cost functions over node coordinates. Sherpa returns nodes corresponding to local minima of the cost function. Sherpa routing consists of two phases: (1) a modified compass routing algorithm [16] on the network coordinate space

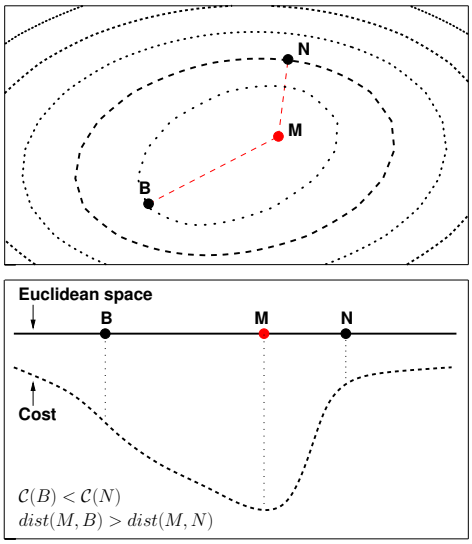

Figure 1: Nearest neighbor is not enough. M is the theoretical lowest cost point, N and B are nodes in the geometric space. Although N is closer to M than B, the cost of B is lower, making it more desirable than N.

to discover the nearest neighbor to a theoretical optimal point, and (2) a gradient descent algorithm based on Voronoi region information to find the lowest cost node. We note that neither of the mechanisms underlying Sherpa's routing are, in and of themselves, new. Rather, the main contribution of Sherpa is the observation that only by combining them can complex, latency-based routing queries be resolved accurately.

Simulations on two real world latency data sets show that Sherpa finds nodes with cost that is significantly lower than the nearest neighbor and close to the optimal, even considering the embedding error of network coordinates.

We bring the following contributions. First, we frame the problem of finding nodes that satisfy location constraints as a cost optimization problem over the network coordinate space (Section 2). We then describe the design of Sherpa, a system that finds low cost nodes given arbitrary continuous cost functions on the network coordinate space (Section 3). We evaluate Sherpa through simulations within the context of distributed game server selection and show that finding low cost nodes is accurate (Section 4). Last, we observe that there is a far broader set of Internet routing queries that can be viewed as geometry problems than previously considered (Section 5), and we discuss how Sherpa supports them.

2 Nearest Neighbor Is Not Enough

We express finding nodes under location constraints as a cost optimization problem, where the cost of each node is deter-

mined by a function over node coordinates. Specifically, if we consider $\mathcal{S}$ a finite set of nodes with network coordinates in a d-dimensional space, and a cost function $\mathcal{C} : \mathbb{R}^d \rightarrow \mathbb{R}$, the optimization problem is to find:

$$\arg\min_{n \in \mathcal{S}} \mathcal{C}(n) \tag{1}$$

For simple location problems, such as finding the closest node to a coordinate T, the cost is represented by the distance to the target (*i.e.*, $\mathcal{C}(n) = d(n, T), \forall n \in \mathcal{S}$). The lowest cost node can be easily computed using distributed nearest neighbor geometric routing algorithms over the coordinate space [18].

Finding the nearest neighbor using network coordinates is not always sufficient. Consider the problem of computing the centroid among a set of nodes $\mathcal{P}$: the best node will offer minimal latency to all nodes in $\mathcal{P}$. A simple cost function is:

$$\mathcal{C}(n) = \frac{\sum_{i \in \mathcal{P}} d(i, n)}{|\mathcal{P}|}, \forall n \in \mathcal{S} \tag{2}$$

Figure 1 presents two ways to visualize the cost function. On top, there is a contour plot, where each curve corresponds to the same cost value and the cost is increasing towards the center. Another way to imagine the cost function, depicted on the bottom of Figure 1, is that it defines some surface over the Euclidean space.

In Figure 1, M represents the theoretical minimum cost point, B and N candidate closest nodes. We emphasize the distinction between a node, a physical entity whose coordinates place it in a unique position in space, and a point, a coordinate in space without necessarily a node occupying it. Although, in the Euclidean space, node B is further to M than N, B has a lower cost. A nearest neighbor algorithm using network coordinates would incorrectly choose node N as the best node that approximates the centroid of all nodes in $\mathcal{P}$.

3 Sherpa

Sherpa is a general substrate for searching broad classes of cost surfaces and without necessarily knowing all the nodes in the set that we are querying. It uses information about each node's position in a virtual geometric space to search for the lowest cost node satisfying location constraints. Sherpa is scalable and provides accurate results regardless of the placement of nodes in the geometric space. In this section, we present the design of Sherpa, focusing on components (what are the requirements to join Sherpa) and operation (how participants look for lowest cost nodes).

3.1 Joining Sherpa

To join Sherpa, a node has to compute its network coordinate and its Voronoi region—the set of points in the space which are closer to it than to any other node. As part of the overlay, every node maintains connections with its Delaunay neighbors—those nodes whose Voronoi regions are adjacent.

Network coordinates Before serving and forwarding requests, every Sherpa node must compute its network coordinate. We use Vivaldi for network coordinates because it is distributed and scalable [9]. Every node maintains a set of neighbors, which are other Sherpa nodes, that it probes periodically and uses the round trip time and the network coordinates of these neighbors to update its own coordinate. After each probe, the node computes the coordinate that minimizes the squared estimation error to all of its neighbors.

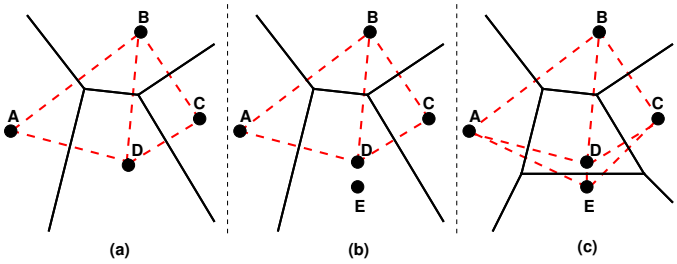

Figure 2: *Computing the Voronoi region for a Sherpa node*: (a) A, B, C and D are Sherpa nodes that have already computed their Voronoi regions; solid lines represent Voronoi edges, dotted lines connect Delaunay neighbors; (b) Node E joins and finds that its coordinate lies in the region owned by D; it advertises its position to D and to D's neighbors, requesting a re-computation of their Voronoi regions; (c) E becomes responsible for its own region, with neighbors A, D and C.

Voronoi regions Given a discrete set $\mathcal{S}$ of nodes in Euclidean space, the Voronoi diagram is the subdivision of the space into $|\mathcal{S}|$ regions, one for each node, such that any point in the region corresponding to a node is closer to that node than to any other node. We denote the Voronoi region of a node p as $\mathcal{V}(p)$. Two nodes are Delaunay neighbors if their Voronoi regions share an edge. Figure 2(a) illustrates the diagram formed by four nodes. The solid lines are Voronoi edges and the dotted lines connect Delaunay neighbors.

When joining Sherpa, a node E must compute its Voronoi region. Starting from a known overlay node, it finds the closest Sherpa node to its own coordinate (node D in Figure 2) using the algorithm described in the next section. Then it contacts D and its Delaunay neighbors, A, B and C, advertising its position and asking for local Voronoi region recomputation.

Each node needs to know only of its Delaunay neighbors to compute its region [4], ensuring only local operations. Pathological cases may arise, when a region borders all other regions. The node owning the region is a Delaunay neighbor for all other nodes in the system and a re-computation of the whole tessellation would be necessary. However, due to the placement of nodes based on network coordinates, which reflect the nodes' geographical position [17], we do not expect such cases to occur often.

The stability of network coordinates or node churn may influence the computation of Voronoi regions. There are several practical solutions to stabilize network coordinates [26, 19] or to construct and maintain Voronoi regions in dynamic environments [20, 21]. We leave their study and interaction with Sherpa for future work.

3.2 Querying in Sherpa

Querying in Sherpa consists of two stages: nearest neighbor search and gradient descent. Given a cost function $\mathcal{C}$ and a set of nodes $\mathcal{P}$ with known network coordinates, the source (or the query initiator) first calculates the theoretical point c that corresponds to the local (or global) minimum of $\mathcal{C}$. It then uses compass routing [16] to find the *nearest neighbor* to c in the coordinate space. Because the nearest neighbor N may not be the node that minimizes $\mathcal{C}$, we continue with a gradient descent algorithm using the Delaunay neighbors of N.

Nearest neighbor search We use compass routing [16] on the graph formed by Delaunay neighbors (the Delaunay graph) to discover the nearest Sherpa node to a coordinate. Compass routing uses geometric information to advance to the destination. Each node chooses the next hop such that the angle formed by the line segment between itself and the potential hop and the line segment to the destination is minimized. Compass routing is guaranteed to find the destination when running on a convexly embedded geometric graph such as a Delaunay graph. Because Sherpa is looking for a coordinate c and not an actual node in the graph, it risks entering an infinite loop where there is always a Delaunay neighbor that makes a smaller angle with c than the owner of the region where c lies. To eliminate this possibility, we simply require that each node verify whether c belongs to any of its neighbors' regions before computing the next hop.

Descent Finding the nearest neighbor is not enough for generalized node location problems. Certainly, for trivial cost functions such as the Euclidean distance itself, nearest neighbor precisely corresponds to the minimum cost node. However, for more complex surfaces such as that induced by finding the centroid among a set of nodes, proximity in the Euclidean space does not necessarily translate into proximity on the cost surface. We use a deterministic *descent* algorithm to efficiently locate the proper minimum cost node.

The algorithm begins at the nearest neighbor p to a theoretical coordinate c and recursively explores all its Delaunay neighbors that contain points that lower the cost. This is necessary because, even though our optimization is on a discrete set of nodes—the centers of each Voronoi region—the cost is defined on the coordinate space and any point can lead towards the minimum. In practice, due to the difficulty of computing the cost for every point in a Voronoi region, we only compute the cost for the vertices. We show in Section 4 that this is enough to obtain good results.

The descent algorithm discovers the lowest cost node using only network coordinate information. Due to inherent inaccuracies in embedding Internet latencies into geometric spaces, the best node in coordinate space (B') may not be the best in latency space (B). To alleviate this problem, B' also computes its cost using latencies—by sending a constant number of latency probes to all nodes on which the cost depends—and asks its Delaunay neighbors to the same. Sherpa returns the node with the lowest cost, computed with the latency probe results. In Section 4, we show that, by sending latency probes, we improve significantly the accuracy of Sherpa.

To conclude, we use the nearest neighbor algorithm to approach a theoretical optimal point in coordinate space. We then use the descent algorithm to exhaustively explore the geographic region around the nearest neighbor, biasing our search towards lower cost regions.

4 Evaluation

Our general formulation for node location captures a diversity of problems and extends the applicability of network coordinates beyond nearest neighbor searches. To evaluate Sherpa, we consider the distributed game server selection (centroid computation) in multi-player online games.

4.1 Distributed Game Server Selection

Many online game communities allow their players to select which game server to use for their online play. The server is responsible for all player-to-player communication and to act as a trusted generator of random events. Selecting a server that offers good performance and fairness among players is essential for the gameplay experience.

Studies have shown that player performance suffers when latency to the game server is high [5, 24]. When a geographically distributed group of friends wants to play together, it is paramount that they choose a server that minimizes latency to *all* players. For fairness, the relative disparities between player latencies (*e.g.*, between the maximum and minimum latency of all players using the server) should also be low.

Selecting a game server for a group of nodes can be expressed as a centroid computation problem. We consider N players $p_1, p_2, \cdots, p_N$, who all know each other and who wish to choose one of M game servers such that both game performance and fairness among players are preserved. Each game server is assigned a cost value:

$$\begin{aligned} \mathcal{C}(m) &= \frac{\sum_{i=1}^{N} dist(m, p_i)}{N} \\ &+ (\max_i(dist(m, p_i)) - \min_i(dist(m, p_i)))^2 \end{aligned} \quad (3)$$

where $dist(m, p_i)$ represents the distance between server m and player p_i. The cost captures the average latency from the game server to each of the players as well as the relative disparities between latencies.

To find a centrally-located game server, a group of players needs to compute its network coordinates. Then one of the players or a designated third party submits a query request to a Sherpa node, including the coordinates of all players and the cost function. Sherpa computes the best node that minimizes the cost and returns it to the requester.

4.2 Methodology

We build a prototype of Sherpa to study how well it finds the low cost neighbors. The prototype is written in approximately 2,500 lines of Ruby and simulates the computation of Voronoi regions, compass routing to find the nearest neighbor and the descent algorithm to find the lowest cost node.

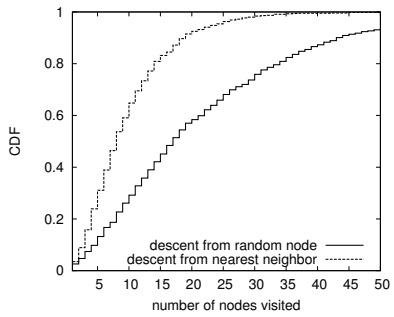

Figure 3: Nodes visited by descent when starting at random and from nearest neighbor of a theoretically optimal centroid for PL-213.

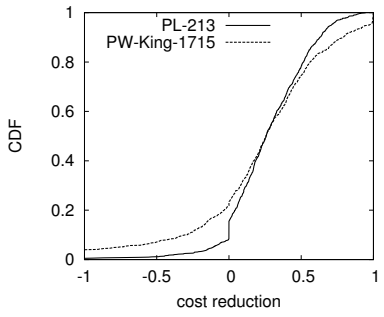

Figure 4: Distribution of cost reduction. Positive values indicate that the node returned by Sherpa has a lower cost than the nearest neighbor.

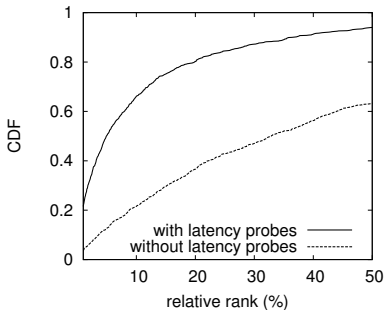

Figure 5: Relative rank of best node returned by Sherpa for centroid computation, with and without latency probes, for PW-King-1715.

We use two real world latency data sets, PW-King-1715 and PL-213. PW-King-1715 was collected by Lumezanu *et al.* [22] between 1,715 DNS servers using the King method [12] in February 2008. We collected the second data set, PL-213, between 213 PlanetLab nodes in January 2009. We considered only one node for each PlanetLab site around the world. To compute the network coordinates of each node, we use the implementation of Vivaldi by Ledlie *et al.* [19].

We consider the cost function in Equation 3 and simulate Sherpa for 1,000 rounds. At each round, we select at most 30 nodes and issue a query to discover their centroid. We choose the nodes such that they form two geographically separate groups: one of at most five, the other of at most 25 nodes. The asymmetry of the group populations makes the distributed centroid population more challenging than when nodes are chosen completely at random.

We first show that nearest neighbor is necessary, but it is not sufficient. Then, we show that nodes chosen by Sherpa as best matches for centroid are ranked high even when real latencies are considered.

4.3 Nearest neighbor is necessary

Is searching for the nearest neighbor of a theoretically minimum even necessary, considering that it can have an arbitrarily high cost? Indeed, one could imagine starting the descent algorithm from any node in the system and eventually converging to the solution.

Finding the nearest neighbor first reduces significantly the overhead incurred to find the lowest cost node. We compute the number of nodes visited by the descent algorithm, first from a random node and then, from the nearest neighbor of a theoretically optimal centroid. Figure 3 shows the distribution of visited nodes. Starting descent from nearest neighbor reduces the number of nodes visited by more than half.

4.4 Nearest neighbor is not enough

The node closest to a theoretically optimal centroid is not necessarily the best choice for the centroid (*i.e.*, the lowest cost node). To understand the difference, we define the cost reduction of a query as the difference between the nearest neighbor cost and the cost of the optimal node divided by the nearest neighbor cost. Although the optimization is done using embedded distances, here we express the cost of a node in terms of real latencies, since it reflects what an application using Sherpa would obtain. Positive cost reductions indicate that the optimal node is a better choice than the nearest neighbor. For example, when the cost reduction is 0.5, the optimal node reduces the cost of the nearest neighbor by half. Queries finding optimal nodes that are worse in latency space than the nearest neighbor have negative cost reductions.

Figure 4 shows the cost reduction for the two data sets. For around 80% of the queries, Sherpa is able to find a node with cost lower than the nearest neighbor. The cost reduction is more than 50% for a fifth of all queries. This proves that finding the nearest neighbor is not sufficient most of the time.

4.5 Relative ranking

Finding the lowest cost node in coordinate space does not imply also finding the lowest cost node when real latencies are considered. For each query, we order all nodes in each data set by their cost expressed in terms of real latency and identify the rank of the optimal node returned by Sherpa. We present the relative rank—the rank normalized to the number of nodes. A relative rank of 10% means that the selected node is among the 10% lowest cost nodes when using real latencies. To understand the importance of issuing latency probes from the neighbors of the node returned by the descent algorithm (see Section 3.2), we consider the situations when they are used and when they are not. Figure 5 shows that latency probes matter: they improve the relative rank of the selected node by at least twice, for both data sets.

5 Applications of Sherpa

The leitmotif of this paper is the power of network coordinates in solving distributed systems problems. Our generalized formulation for node location is applicable to many applications, some whose performance does not depend directly

on latency. We enumerate them below; applying Sherpa to them is the main focus of our future work.

Improving throughput with split-TCP Split-TCP protocols [14] establish a relay between the two endpoints of a TPC connection. Jain and Ott [13] demonstrated that choosing a relay that has a shorter RTT between both source and destination than the two endpoints have to one another results in an overall increase in end-to-end throughput. Sherpa could easily find such relays by specifying a cost function that minimizes the distance from the relay to the endpoints.

Content distribution Multicast and streaming [3] are more efficient when performed through users that minimize latency to receivers. Previous research [3] shows that placing centrally-located nodes higher in the multicast tree can improve transmission latencies. Finding these nodes is similar to the problem of finding a game server for a set of players.

Off-site backup Essentially a resource replication problem [8, 6], off-site backup attempts to add a new replica to an object. The location of the replica should ensure that the original object and the replica would not fail at the same time. To do that, one may choose the location to be outside some radius around the original's position. A potential cost function would try to minimize the distance between the replica and the original (for faster backups) while satisfying the radius constraints.

Node avoidance Network accountability systems seek to prove who originated and forwarded traffic, predominately with the goal of blocking unwanted traffic close to the source [2, 28]. However, they cannot prove that a particular host h was *not* on the path from source to destination. This feature can be achieved with latency constraints by choosing a relay whose latency to the destination is less than the latency that would be incurred were the relay to forward through the host h. This insight can be realized in Sherpa with a cost function that *penalizes* nodes who are too close to hosts which the users wishes to avoid.

6 Conclusions

In this paper, we have considered generalizations of node location problems in distributed systems using network coordinates. Finding nodes that satisfy location constrains can be naturally expressed as an optimization problem over node coordinates. We designed a new system, Sherpa, that finds the lowest cost node in distributed applications, for a wide range of continuous cost functions and regardless of the placement of nodes.

Acknowledgments

Cristian Lumezanu is supported by the National Science Foundation under Grant #0937060 to the Computing Research Association for the CIFellows Project. Dave Levin is supported by a Microsoft Live Labs Fellowship.

References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *SIGCOMM*, 2008.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM*, 2002.

[4] B. A. Bash and P. J. Desnoyers. Exact distributed Voronoi cell computation in sensor networks. In *IPSN*, 2007.

[5] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *NetGames*, 2004.

[6] R. Bhagwan, D. Moore, S. Savage, and G. M. Voelker. Replication strategies for highly available peer-to-peer storage. In *FuDiCO*, 2002.

[7] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM*, 2008.

[8] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of ACM SIGCOMM*, 2002.

[9] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, 2004.

[10] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting BitTorrent blocking. In *IMC*, 2008.

[11] M. J. Freedman, E. Freudenthal, and D. Maziéres. Democratizing content publication with Coral. In *NSDI*, 2004.

[12] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *IMW*, 2002.

[13] R. Jan and T. J. Ott. Design and implementation of split TCP in the Linux kernel. In *Globecom*, 2006.

[14] P. Karbhari, M. Ammar, and E. Zegura. Optimizing end-to-end throughput for data transfers on an Overlay-TCP path. In *NETWORKING*, 2005.

[15] C. Kommareddy and B. Bhattacharjee. Finding close friends on the Internet. In *ICNP*, 2001.

[16] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *CCCG*, 1999.

[17] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *NSDI*, 2007.

[18] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer. Wired geometric routing. In *IPTPS*, 2007.

[19] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *ICDCS*, 2006.

[20] D.-Y. Lee and S. S. Lam. Protocol design for dynamic delaunay triangulation. In *ICDCS*, 2007.

[21] D.-Y. Lee and S. S. Lam. Efficient and accurate protocols for distributed Delaunay triangulation under churn. In *ICNP*, 2008.

[22] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in Internet routing overlays. In *NSDI*, 2009.

[23] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, 2002.

[24] L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV*, 2002.

[25] P. Pietzuch, J. Ledlie, M. Mitzenmacher, and M. Seltzer. Network-aware overlays with network coordinates. In *IWDDS*, 2006.

[26] G. Wang and T. S. E. Ng. Distributed algorithms for stable and secure network coordinates. In *IMC*, 2008.

[27] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *SIGCOMM*, 2005.

[28] X. Yang and X. Liu. Internet protocol made accountable. In *HotNets*, 2009.

[29] X. Yang, G. Tsudik, and X. Liu. A technical approach to net neutrality. In *HotNets*, 2006.