Playing Vivaldi in Hyperbolic Space

Cristian Lumezanu and Neil Spring
University of Maryland
{lume,nspring}@cs.umd.edu

Abstract

Internet coordinate systems have emerged as an efficient method to estimate the latency between pairs of nodes without any communication between them. They avoid the cost of explicit measurements by placing each node in a finite coordinate space and estimating the latency between two nodes as the distance between their positions in the space. In this paper, we adapt the Vivaldi algorithm to use the Hyperbolic space for embedding. Researchers have found promise in Hyperbolic space due to its mathematical elegance and its ability to model the structure of the Internet. We attempt to combine the elegance of the Hyperbolic space with the practical, decentralized Vivaldi algorithm.

We evaluate both Euclidean and Hyperbolic Vivaldi on three sets of real-world latencies. Contrary to what we expected, we find that the performance of the two versions of Vivaldi varies with each data set. Furthermore, we show that the Hyperbolic coordinates have the tendency to underestimate large latencies ($>$ 100 ms) but behave better when estimating short distances. Finally, we propose two distributed heuristics that help nodes decide whether to choose Euclidean or Hyperbolic coordinates when estimating distances to their peers. This is the first comparison of Euclidean and Hyperbolic embeddings using the same distributed solver and using a data set with more than 200 nodes.

1 Introduction

Internet coordinate systems estimate the latency between pairs of nodes without any communication between them. They associate each node to a position in a finite coordinate space; the latency between two nodes is then estimated as the distance between their positions in the space [10, 12, 20, 7, 9, 15, 2, 3]. Coordinate systems support wide-area applications and protocols that allow the choice of communication peers based on the latency metric. For example, in the construction of structured overlay networks [17, 14, 13], nodes usually select the closest peers to optimize queries and responses. In file-sharing applications [5] or content distribution networks [4], it is more efficient to download the required content from the server with the lowest latency [**?**].

Existing coordinate systems have three important components to their designs: *space selection*, *probing*, and *positioning*. Space selection involves how to calculate the distances between points, whether the space is Euclidean, how many dimensions it has, etc. Probing is the process of

measuring latency to a few peers, or chosen landmarks. Positioning is the optimization process of using probe results to assign a location to every node in the space. These components determine whether the system is accurate, scalable, and adaptive.

We separate these components so that we can reassemble a system that uses Hyperbolic space and the Vivaldi algorithm for probing and positioning. Vivaldi is distributed and adaptive, running without global state and accommodating the dynamics of the network. Our choice for embedding space is based on the works of Shavitt and Tankel [16], Lua *et al.* [8], Subramanian *et al.* [19], and Tauro *et al.* [21]. Shavitt and Tankel use a centralized simulation-based algorithm to embed nodes into a Hyperbolic space. They present good accuracy results later confirmed by the study performed by Lua *et al.*[8]. The Hyperbolic space, in which the distance between two nodes is computed along a curved line bent towards the origin, seems better fit for the jellyfish structure of the Internet (a core in the middle with many tendrils) [21].

We evaluate both Euclidean and Hyperbolic Vivaldi on three sets of latency measurements, two from King [6] between DNS servers and the third between PlanetLab nodes. Contrary to what we expected, we find that the performance of the two versions of Vivaldi varies with each data set. We show that the PlanetLab data set behaves much worse in Hyperbolic space than in Euclidean space, while for one of the King data sets the Hyperbolic embedding is more accurate. Furthermore, our results show that Hyperbolic coordinates underestimate large latencies (> 100 ms) but are more accurate in estimating distances between closer nodes. This is the first comparison between Euclidean and Hyperbolic embeddings in a distributed setting and using a data set with more than 200 nodes.

The rest of the paper is organized as follows. Section 2 gives a short overview on Hyperbolic spaces and describes the changes we made to the Vivaldi algorithm. Section 3 presents the experimental results. We conclude in Section 4.

2 Approach

2.1 Vivaldi algorithm

We use the same positioning algorithm as Dabek *et al.* [3]. We briefly and informally describe the algorithm in this subsection.

Vivaldi simulates a physical system of springs where each spring corresponds to a pair of nodes. The rest length of a spring emulates the real distance between two nodes while the actual length is the distance computed by the embedding. The energy of each spring is proportional to its displacement (the difference between the rest length and the current length). The algorithm runs iteratively at each node and simulates the progress of the springs toward a state with minimum energy. At every step, each node will be pushed to a new position that minimizes the displacement of the springs it is connected to. Two factors affect the position of a node after each step of the algorithm: the magnitude (M) and the direction (D) of the movement. The magnitude of the movement is proportional to the displacement of the associated springs and the direction of movement is given by the opposite of the gradient of the energy function with respect to the position of the node. After computing the magnitude and the direction of movement the following rule

updates the coordinates of a node:

$$x = x + \delta \times M \times D$$

where δ is the timestep between two consecutive updates.

We consider an n-dimensional coordinate space. The energy of a spring between nodes $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ is:

$$E_{xy} = \frac{1}{2}k(rtt_{xy} - d(x,y))^2$$

where $rtt_{xy} - d(x,y)$ is the displacement of the spring[1] and k is the elasticity constant.

2.2 Euclidean Vivaldi

Dabek *et al.* [3] use Euclidean distance:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

After computing the gradient of the energy function, they obtain the following expressions for the magnitude and direction of movement:

$$D = u(x - y)$$

$$M = \frac{rtt_{xy} - d(x,y)}{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}}$$

where u is a unit vector.

2.3 Hyperbolic Vivaldi

We now consider that the nodes and the conceptual springs lie instead in a Hyperbolic space.

It is hard for an observer in the three-dimensional Euclidean world to visualize and imagine the Hyperbolic world [1]. To better analyze and understand it, different Euclidean models or maps of the Hyperbolic space have been constructed. As an analogy, people create maps that project points from the Earth sphere to a plane, which yields the familiar Mercator projection among others. Unfortunately, each model of the Hyperbolic geometry depicts a warped version of the Hyperbolic space, just as a two-dimensional map represents the Earth in a distorted way.

To describe the Hyperbolic space, all we need to know is the amount of distortion introduced by the model used to embed the space. The distortion is determined by two parameters: *curvature* and *metric*. Curvature is a characteristic of the space and represents the amount by which an object in the space deviates from being flat. Euclidean spaces have curvature 0 because all lines are flat and Hyperbolic spaces have negative curvature. The metric is a characteristic of the model used to embed the space and represents the distance function between two points in the model.

[1] We use subscripts to differentiate between a measured value (rtt_{xy}) and a computed value ($d(x,y)$)

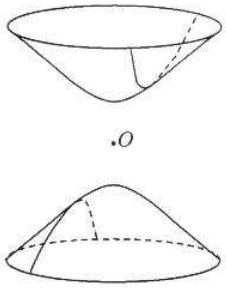


Figure 1: Hyperboloid with two sheets; the points in the Hyperbolic space are represented using only the upper sheet.

Knowing the curvature of a Hyperbolic space and the metric of its model we can determine the distance between any two points in the space as well as the curved line along which the distance is computed.

The spring placed between the points will move along this curved line under a force proportional to the difference between the real distance and the Hyperbolic distance.

There are several equivalent models of the Hyperbolic world. We choose the hyperboloid model, in which all points lie on the upper sheet of a hyperboloid, due to the simplicity of its metric. Figure 1 presents a hyperboloid with two sheets. Although the model stretches to infinity, it is cut for observation. The distance between two points on the hyperboloid is computed along a line formed by the intersection of the hyperboloid with the plane determined by the two points and the origin of the space.

The distance between points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ in a n-dimensional Hyperbolic space of curvature k is:

$$d(x,y) = \operatorname{arccosh}\left(\sqrt{(1+\sum_{i=1}^{n} x_i^2)(1+\sum_{i=1}^{n} y_i^2)} - \sum_{i=1}^{n} x_i y_i\right) \times k$$

This formula is derived from the distance between two points on the unit hyperboloid (the *arccosh* part) multiplied by the curvature of the Hyperbolic space (k).

The magnitude and direction of movement in the new space are computed using the gradient of the energy function.

$$D = u\left(\frac{\sqrt{1+\sum_{i=1}^{n} y_i}}{\sqrt{1+\sum_{i=1}^{n} x_i}} x - y\right)$$

$$M = \frac{rtt_{xy} - d(x,y)}{\sqrt{\cosh^2 d(x,y) - 1}}$$

3 Results

In this section, we compare the performance of Vivaldi in Euclidean and Hyperbolic spaces using three sets of real-world latencies. First, we present the data sets and the setup of the experiments.

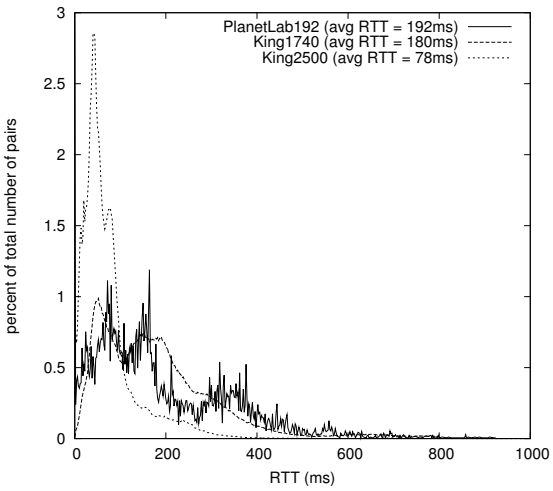


Figure 2: Distribution of RTTs for each data set.

Next, we describe the choice for the optimal Hyperbolic space curvature and we analyze the accuracy of the embeddings using scatter plots and error distributions. Finally, we propose two distributed heuristics that enable a node to choose either Euclidean or Hyperbolic coordinates and distance functions when estimating the latency to a peer.

3.1 Data Sets and Experiment Setup

We use three data sets, two of latencies between DNS servers and the third between PlanetLab nodes. We choose them because they are well-established, already used in several research projects [3, 22, 8, 23]. Further, they are diverse enough to allow us to capture the behavior of the Hyperbolic embedding in different environments.

- **King1740** Dabek *et al.* [3] used King [6] to collect the primary data set for evaluating Vivaldi. It contains RTTs between 1,740 DNS nameservers serving hosts in the Gnutella network. Each RTT is the median of all measurements taken at random intervals over a week. The authors removed approximately 10% of the measured nodes because they participated in many triangle inequality violations.

- **King2500** The Meridian [22] project built a data set for evaluation using King to measure between 2,500 DNS servers associated with sites in the Yahoo and DMOZ directories. Each RTT is the average of 10 measurements. The authors did not explicitly remove any nodes based on triangle inequality violations or high latencies. However, they removed all nodes with connectivity issues.

- **PlanetLab192** The third data set captures latencies between 192 PlanetLab nodes distributed around the globe and was derived from Stribling's all-pairs ping trace [18]. It was used for the evaluation of the Vivaldi algorithm.

We omit the details of the measurements that lead to the three data sets. The average round-trip times are 180 ms, 74 ms and 190 ms, while the number of triples that violate the triangle inequality amounts to 4%, 8%, and 5% of the triples in each data set. Figure 2 shows the distribution of RTTs; we plot the percentage of pairs for each round-trip time in the range from 1 to 1000 ms.

We use the MIT p2psim [11] packet-level network simulator. We modified the Vivaldi code to allow nodes to have two sets of coordinates: Euclidean and Hyperbolic. The Euclidean coordinates are two-dimensional with a height vector, while the Hyperbolic coordinates have three dimensions. In this way, we ensure that a node has the same number of degrees of freedom in both spaces. As the algorithm runs, both coordinates are updated, allowing nodes to position themselves simultaneously in Euclidean and Hyperbolic spaces. We fix the number of neighbors of each node to 32: half are selected as the closest peers in network latency and the rest are selected at random. Similarly to the original Vivaldi algorithm, we use an adaptive timestep to help nodes find their position faster. We initialize both sets of coordinates for a node to the origin of the corresponding space. We stop the algorithm after 10 seconds and collect the coordinates.

3.2 Curvature of the Space

We want to select the Hyperbolic space curvature that produces the lowest embedding error. We show how curvature affects the accuracy of the embedding of the PlanetLab192 data set in Figure 3. The horizontal axis represents the value of the curvature and the vertical axis is the median error of the embedding. This is computed after the system has run for 10 seconds and represents the absolute value of the difference between the embedding distance and the real distance. Longer runs than 10 seconds did not improve the median error significantly. Figure 3 indicates that the higher the curvature of the embedding space, the lower the median error. To better analyze this relationship, we separate overestimation error from underestimation error. The overestimation error is the median error of all the pairs for which the embedding distance is greater than the real distance, while the underestimation error is the median error of the pairs whose real distance is underestimated. The overestimation error decreases with the curvature of the space, while the underestimation error is lowest around curvature -17 and grows gradually in both directions starting from this value. The plot also indicates that any curvature in the range -20 to -11 will produce a low embedding error. We choose the curvature for the embedding of the PlanetLab192 data set to be -14. Based on similar results, we set the curvatures for the King1740 and King2500 data sets to be -16 and -7. We believe that the King2500 requires a smaller absolute value for the curvature because, on the average, it contains smaller latencies that need to be modeled. We use different curvatures for the three data sets because we want to obtain the best possible Hyperbolic embeddings.

3.3 Latency estimation

We evaluate the accuracy of Vivaldi in Euclidean and Hyperbolic spaces by capturing the relationship between the real and the estimated distances using scatter plots. Figures 4 and 5 present scatter plots in which the horizontal axis is the measured round-trip time and the vertical axis corresponds to the estimated distances. The axes have a logarithmic scale and each dot represents a

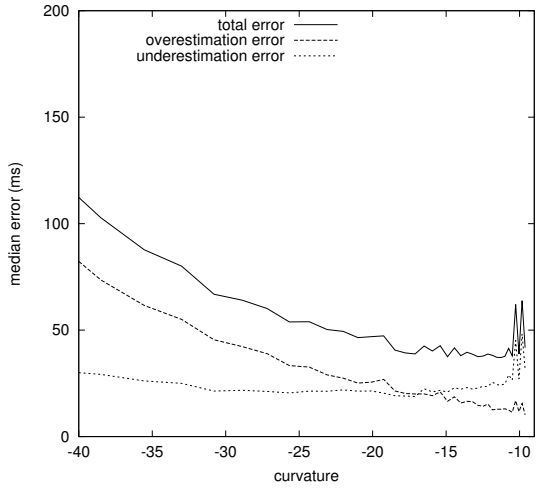
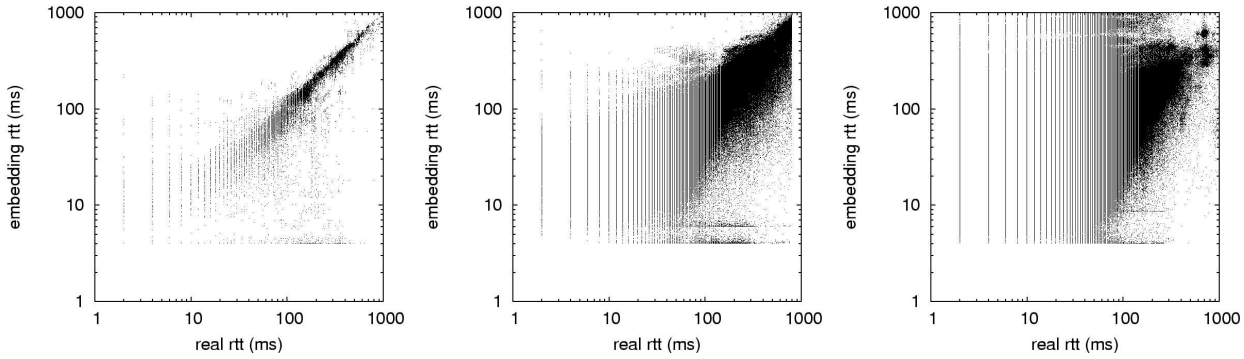


Figure 3: Median error of the embedding versus space curvature.



(a) PlanetLab192 (b) King1740 (c) King2500

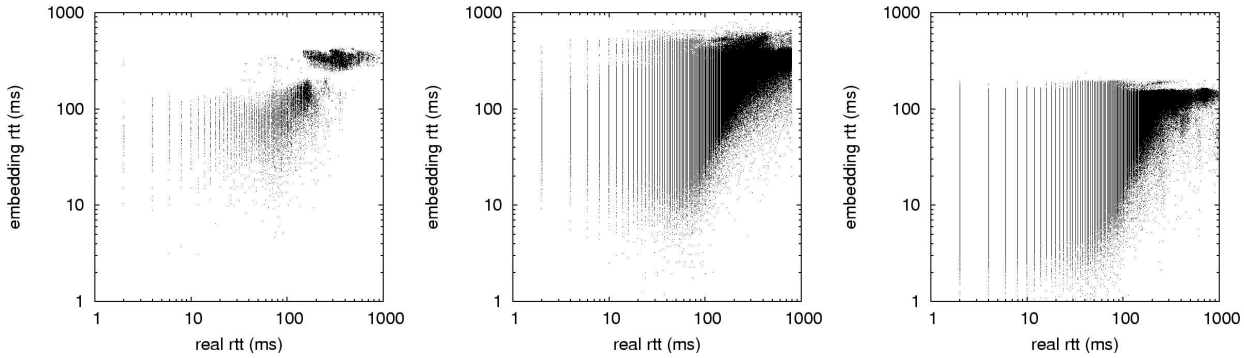
Figure 4: Estimated distance versus real distance for the three data sets in 2-dimensional Euclidean Vivaldi with heights.

pair of nodes. The closer the dots are to the diagonal of the box, the lower the embedding error for the corresponding pairs of nodes is.

We make two observations. First, the two versions of the algorithm perform differently across the three latency matrices. Vivaldi Euclidean outperforms Vivaldi Hyperbolic on the PlanetLab192 and King1740 data sets (Figures 4(a) and 5(a)) but it is less accurate on the King2500 data set (Figures 4(c) and 5(c)). Second, the Hyperbolic version of the algorithm tends to underestimate large round-trip times ($>$100ms) for all data sets, while performing better for smaller distances. In the next subsection, we present error distribution plots that support and explain these observations.

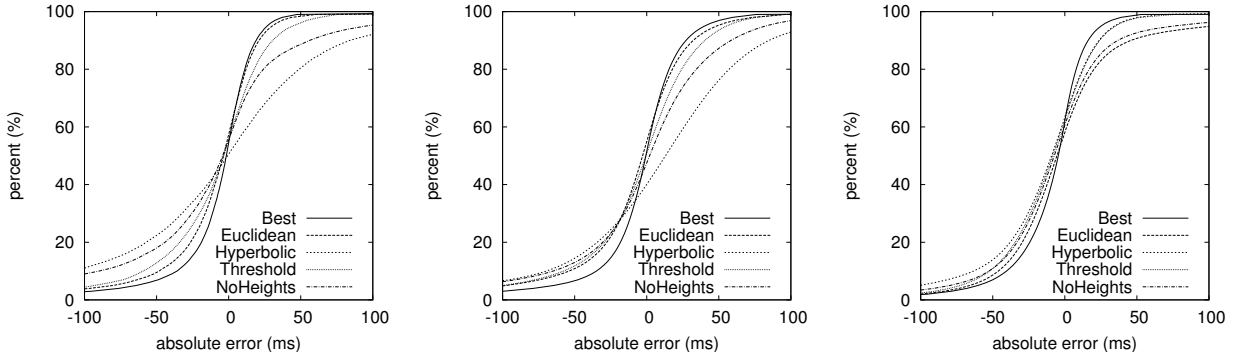
3.4 Error Distributions

We present cumulative distributions over all pairs for both absolute and relative error. Absolute error is defined as the difference between the embedded distance and the real distance while relative



(a) PlanetLab192 (b) King1740 (c) King2500

Figure 5: Estimated distance versus real distance for the three data sets in 3-dimensional Hyperbolic Vivaldi.

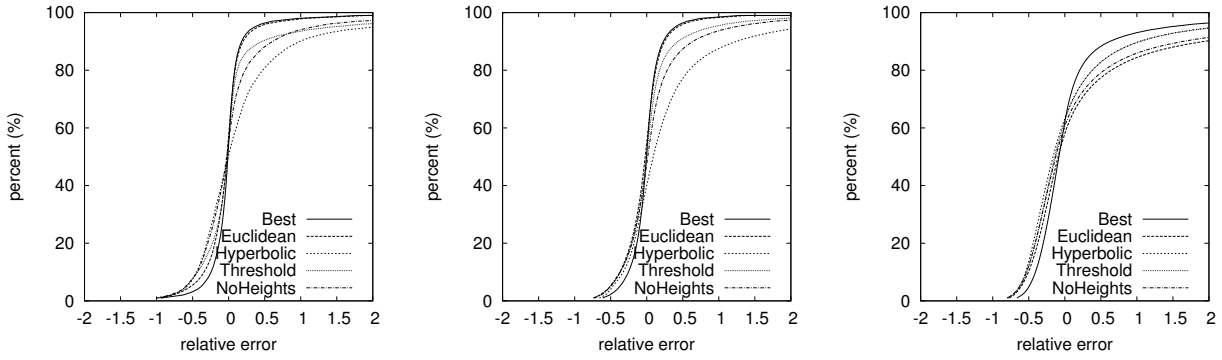


(a) PlanetLab192 (b) King1740 (c) King2500

Figure 6: Cumulative distribution of absolute errors for the three data-sets for different embeddings.

error is absolute error divided by the real distance. We do not use absolute values for these two measures (as in other work) because we want to separate underestimation and overestimation. As mentioned in Section 3.3, Hyperbolic Vivaldi tends to underestimate long distances and we would like to be able to quantify the effect of this underestimation on the accuracy. Recent accuracy metrics, like relative rank loss [8] capture better the quality of the embedding from the point of view of the users. However, by relying only on the relative distance to nodes, they tend to inflate the importance of small errors. We plan to evaluate the relative rank loss for the three data sets in future work.

Figure 6 presents the distribution of the absolute error for the three data sets. Ignore the lines labeled *Best*, *Threshold* and *NoHeights* for now; we will describe them in the next subsection. Each point corresponds to one pair of nodes. For the PlanetLab192 latencies, both versions of the algorithm have the same ratio of underestimated distances to overestimated distances (around 50% of the pairs are underestimated). However, Hyperbolic Vivaldi has a higher error in both underestimation and overestimation. 10% of the pairs are estimated to have a distance with at least 100ms less than the real distance when embedded into Hyperbolic space, while less that 5% of the pairs exhibit the same behavior in the Euclidean space. The same difference in accuracy between



(a) PlanetLab192 (b) King1740 (c) King2500

Figure 7: Cumulative distribution of relative errors for the three data-sets for different embeddings.

Euclidean and Hyperbolic Vivaldi appears in the King1740 data set (Figure 6(b)), though fewer pairs are underestimated when nodes are embedded in a Hyperbolic space. On the other hand, the accuracy of Hyperbolic Vivaldi improves visibly when simulated on the King2500 data set, which is consistent with our findings from Section 3.3.

To better quantify the importance of the errors we plot the distribution of relative errors in Figure 7. A relative error of 1 between a pair of nodes means that the embedded distance is twice the real distance, whereas a relative error of -0.5 signifies that the embedded distance is half the real distance. There can be no relative error less than -1 since the embedded distance cannot have negative values. The plots show almost no difference in underestimation between the two algorithms, as opposed to the absolute underestimation error. We explain this as follows. Hyperbolic Vivaldi underestimates more distances than Euclidean Vivaldi but most of these are large distances, therefore the large absolute errors. On the other hand, Euclidean Vivaldi underestimates smaller distances and although the size of the error is much smaller, it is significant when compared to the estimated distance.

3.5 Improving the Hyperbolic Embedding

We want to obtain an embedding algorithm that benefits from both the Euclidean and Hyperbolic coordinates. Ideally, when it estimates the latencies to other nodes, a node in our system would select to use the coordinates and the distance function that yield the smallest error. Unfortunately, this approach is infeasible in a distributed setting where nodes make decisions without a global knowledge of the system and where the number of measurements is limited. To address this, we propose two heuristics that allow a node to choose the best coordinates based only on local information.

In the first heuristic, called *NoHeightsHyperbolic*, the estimated distance between each pair of nodes is computed using the Euclidean metric only when neither node has height. When one of the nodes has a height greater than 1, we use the Hyperbolic distance. The intuition behind this heuristic is drawn from the initial motivation for height vectors by Dabek *et al.* [3]. The Euclidean distance models the Internet core where latencies are proportional to geographic distances, while the height accounts for the time taken to reach the core from a node behind an access link. By

using Hyperbolic distance estimation whenever a node has height, we offer an alternative to the Vivaldi's height model.

The second heuristic, *ThresholdHyperbolic* is partly based on the observations made in Section 3.3. Since Hyperbolic space seems to underestimate larger latencies, we choose Euclidean distance estimation for every pair whose Hyperbolic distance is computed to be more than a certain threshold. In the experiments we used a threshold value of 100ms.

The absolute and relative error distributions of the embeddings using the two heuristics are labeled *NoHeights* and *Threshold* in Figures 6 and 7. We denote by *Best* the embedding in which, for every pair, the distance with the smallest estimation error is chosen. The *ThresholdHyperbolic* embedding improves the accuracy of the pure Hyperbolic embedding and, as expected, eliminates almost completely the difference in underestimation when compared to the Euclidean embedding. It proves to be a more general alternative than Euclidean Vivaldi due to the good accuracy it achieves for **all** the data sets.

4 Conclusions and Future Work

We have successfully adapted the Vivaldi algorithm so that nodes position themselves in a Hyperbolic space. We compared the performance of Euclidean and Hyperbolic embeddings using three different data sets, two derived from the King measurement tool, the other composed of latency measurements between PlanetLab nodes. Our results show the that accuracy of the two versions of Vivaldi varies with each data set and that the Hyperbolic coordinates have the tendency to underestimate large latencies. Based on these observations, we present a distributed heuristic, *ThresholdHyperbolic*, that uses both Euclidean and Hyperbolic coordinates and achieves *good* accuracy for **all** data sets.

In the future, we plan to extend our comparison to include the BBS Hyperbolic [16], a centralized simulation-based network coordinate system that also uses Hyperbolic embedding. In particular, we are interested in evaluating how, if at all, the performance of the Hyperbolic embedding is influenced by a distributed deployment. We also intend to understand more completely the behavior of Hyperbolic Vivaldi and of the two heuristics by evaluating their accuracy based on how the embedding modifies the relative distance among nodes [8].

Acknowledgments

We thank Eng Keong Lua for our discussions and for his comments on an initial draft of this paper.

References

[1] E. A. Abbott. *Flatland: A Romance of Many Dimensions*. publisher unknown, 1880.

[2] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. In *ICDCS*, 2004.

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

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

[5] Gnutella. http://www.gnutella.com.

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

[7] H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *IMC*, 2003.

[8] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of the embeddings for Internet coordinate systems. In *IMC*, 2005.

[9] Y. Mao and L. K. Saul. Modeling distances in large-scale networks by matrix factorization. In *IMC*, 2004.

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

[11] P2PSim. http://pdos.csail.mit.edu/p2psim.

[12] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *IPTPS*, 2003.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems*, 2001.

[15] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. In *INFOCOM*, 2003.

[16] Y. Shavitt and T. Tankel. On the curvature of the Internet and its usage for overlay construction and distance estimation. In *INFOCOM*, 2004.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM*, 2001.

[18] J. Stribling. Planetlab all pairs ping. http://www.pdos.lcs.mit.edu/~strib/pl_app/.

[19] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proceedings of INFOCOM*, 2002.

[20] L. Tang and M. Crovella. Virtual landmarks for the internet. In *IMC*, 2003.

[21] S. Tauro, C. Palmer, G. Siganos, and M. Faloutsos. A simple conceptual model for the internet topology. In *Global Internet*, 2001.

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

[23] R. Zhang, Y. C. Hu, Z. Lin, and S. Fahmy. A hierarchical approach to internet distance prediction. In *IEEE ICDCS*, 2006.