HyperSpring: Accurate and Stable Latency Estimation in the Hyperbolic Space*

Yongquan Fu and Yijie Wang
National Key Laboratory for Parallel and Distributed Processing, School of Computer
National University of Defense Technology, China
yongquanf@nudt.edu.cn, wwyyjj1971@vip.sina.com

Abstract

Predicting network latencies between Internet hosts can efficiently support large-scale Internet applications, e.g., file sharing service and the overlay construction. Several study use the Hyperbolic space to model the Internet dense-core and many-tendril structure. However, existing Hyperbolic space based embedding approaches are not designed for accurate latency estimation in the distributed context. We present HyperSpring, which estimates latency by modelling a mass spring system in the Hyperbolic similar with Vivaldi. HyperSpring adopts coordinate initialization to speed up the convergence of coordinate computation, uses multiple-round symmetric updates to escape from bad local minima, and stabilizes coordinates by compensating RTT measurements to reduce the coordinate drifts. Evaluation results based on a network trace of 226 PlanetLab nodes indicate that, compared to Euclidean-space based Vivaldi, HyperSpring provides performance improvements for most nodes, and incurs slightly higher distortions for a small number of nodes.

1 Introduction

The Network latency (e.g., Round Trip Time, RTT) between Internet hosts has been an important parameter for distributed sapplications, e.g., content distribution networks can direct hosts to nearby replicas based on latency estimates; file sharing applications, e.g., BitTorrent, pick low-network-latency hosts as cooperating swarming peers. Overlays can select closest peers to minimize the interactive latency and increase the throughput of data dissemination.

A scalable network latency estimation technique between Internet hosts is to embed hosts into a geometric space with low errors [1, 5, 4]. Hyperbolic space based Internet latency estimation receives increased study [5, 3], since it can model the Internet AS topology structure more accurately than the Euclidean model, and it can guarantee greedy routing to enable geographic forwarding in a wireless ad-hoc network [2]. However, existing Hyperbolic space based embedding approaches are not designed for accurate latency estimation in the distributed context, either due to the centralized computation process [5] or high-distortion compared to embedding nodes into Euclidean space [3].

The main contribution of this paper is a new latency estimation scheme called HyperSpring that operates on the Hyperbolic space, based on minimizing the energy configuration in a mass-spring system, similar with that of Vivaldi [1]. HyperSpring consists of three main components: the bootstrap process which initializes the participants' positions according to a small set of landmarks, to increase the convergence speed of coordinate computation; the multi-round symmetric update process which refines coordinates in both directions of each communicating node pair, in order to avoid high-erroneous local minima; and the stabilization process that reduce the significant movements of positions for stability and robustness.

Evaluation results with real-world network traces from 226 hosts on the PlanetLab show that HyperSpring provides performance improvements with slack: with low dimension and small-sized landmarks, it is more accurate and stable than Euclidean space based Vivaldi for most nodes, and incurs mildly higher distortions for small fraction of nodes.

2 Related work

Latency estimation based on network coordinates has received great attention since the coordinate model provides a compact and powerful model by exploring the geometric structure of the Internet, e.g., GNP [4] and Vivaldi [1]. However, since the Internet latency space contains asymmetric routing and triangle inequality violations (TIV), which distorts the estimated accuracy. To reduce the

This work was supported in part by the National Natural Science Foundation of China under Grant No.60873215 and No.60621003; the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321801; Specialized Research Fund for the Doctoral Program of Higher Education No.200899980003; A Foundation for the Author of National Excellent Doctoral Dissertation of PR China No.200141.

TIV impacts, Vivaldi uses height vectors to capture the time to traverse the access links from a node to the core of the Internet.

By treating the Internet structure with curvature in a Hyperbolic space, the tendency of Internet distances towards the core can be efficiently represented. Shavitt and Tankel [5] model the Internet distance map with a centralized embedding procedure based on minimizing energy of a set of particles. However, it is not clear to extend their methods to the distributed fashion. Lumezanu and Spring extend the Vivaldi method into the Hyperbolic space, by formulating the virtual forces of springs based on the Hyperboloid model [3]. However, the formulation of directions of virtual forces consists of mistakes in Section 5.2 of [3], which is identified in 3.6 with techniques from Section II.C of [5]. Similar with Hyperbolic space based Vivaldi [3] and Vivaldi [1], HyperSpring uses the mass-spring system to perform the energy minimization. However, there are three distinctions in HyperSpring: firstly, HyperSpring initializes coordinates with a subset of landmark nodes to increase the convergence of coordinates; secondly, HyperSpring symmetrically updates a pair of nodes in multiple rounds to avoid bad local minima; thirdly, HyperSpring remove the degrees of coordinate drifts by compensating the RTT measurements with current embedding errors, which simultaneously preserves the estimating accuracy.

3 Latency estimation in the Hyperbolic space

3.1 Overview

HyperSpring computes the coordinates in a Hyperbolic space, following the idea of virtual force similar with Vivaldi. Firstly, a small number of randomly selected landmarks initialize their Hyperbolic coordinates based on virtual forces towards other landmarks in the Hyperbolic space, such that the positions of landmarks are accurate enough to make end hosts' coordinates converge quickly. Secondly, each non-landmark node initializes its Hyperbolic coordinate by iteratively applying virtual forces with respect to positions of landmarks to find an accurate starting point. Thirdly, to keep coordinates accurate and stable after the initialization process, a node i compensates the new RTT measurements to neighbors to avoid coordinate oscillation caused by RTT perturbations, if and only if the accuracy of node i's coordinate is not improved; then, node i applies new virtual forces to its current position with a neighbor symmetrically in multiple rounds, such that the resultant coordinates can converge to better local minima.

As a result, although simply implementing a Hyperbolic space based Vivaldi incurs large distortions and high coordinate drifts, with the above components, HyperSpring can provide a scalable, and stable approach for accurately estimating latency.

3.2 Coordinate space selection

The Internet Autonomous System (AS) topology structure is believed to be jellyfish-like, in that there is a core in the middle and many tendrils connected to it [3]. A common model for coordinates is the Euclidean space, where RTTs between pairs of nodes are approximated by their coordinate distances in the Euclidean norm. However, embedding nodes in Euclidean spaces can not reveal the geometric shape of bending routing paths towards the core, due to introducing shortcuts between them. Alternatively, the Hyperbolic space can model the bending characteristics by defining the hyperbolic line as a parametric curve connecting between nodes bending toward the origin point. A hyperbolic space can be determined with two parameters: 1) curvature, which denotes the amount by which a point in the space deviates from being flat, e.g., Euclidean spaces have curvature 0 since distances are represented as the length of flat lines, and Hyperbolic spaces have negative curvature; 2) metric, which describes how to embed the space and to represent the distance function between points in the Hyperbolic space.

To simplify the coordinate computation, we choose the hyperboloid model where all points are on the upper sheet of a hyperboloid (the 'Loid model):

$$S^n = \left\{ x : x_1^2 + x_2^2 + \cdots + x_n^2 - x_{n+1}^2 = -1 \right\}$$
$$x = \left(x_1, \ldots, x_n, \sqrt{1 + \textstyle\sum_{i=1}^{n} x_i^2} \right).$$

We denote $x_{n+1} = \sqrt{1 + \sum_{i=1}^{n} x_i^2}$, similar with that in [3] and [5].

The distance function is defined as a line of the intersection of the hyperboloid with the plane determined by two points and the origin of the space. Specifically, for two points x and y, the approximated network distance $d(x, y)$ embedded in a N-dimensional Hyperbolic space of curvature $|k|$ is

$$d(x, y) = d_{xy}^H \times |k|$$

d_{xy}^H is the pairwise Hyperbolic distance between x and y, which is defined as:

$$d_{xy}^H = \operatorname{arccos} h \left(\sqrt{\left(1 + \sum_{i=1}^{n} x_i^2\right)} \sqrt{\left(1 + \sum_{i=1}^{n} y_i^2\right)} - \sum_{i=1}^{n} x_i y_i \right)$$

3.3 Bootstrap of landmarks

For an end host which joins the system, it can quickly and accurately determine its initial coordinate based on distances towards a set of pre-configured landmarks that

have initialized their coordinates. Since, the landmarks can speedup the coordinate convergence process of end nodes by providing good guesses of end hosts' initial coordinates, which reduce the coordinate movement towards accurate coordinate positions. Moreover, the landmarks are configured as trustable nodes, which can be used to verify malicious nodes which report fake coordinates.

Original Vivaldi [1] does not include landmarks for simplicity; instead, it updates individual coordinates from the origin point. In HyperSpring, to provide accurate initial coordinates for end hosts, the position of each landmark is iteratively refined based on virtual forces towards other landmarks. The landmarks are chosen at random, since in a mass spring field, each node acts the same role by pushing other nodes parts.

Specifically, each node sets its position as the origin point, then, for a node A which has obtained a new RTT measurement to a node B, A updates its coordinate based on the following rules:

$$\begin{gathered} w_s \leftarrow w_A / \left(w_A + w_B\right) \\ \varepsilon \leftarrow \left|\left(d(A,B) - d_{AB}\right)\right| / d_{AB} \\ w_A \leftarrow c_e w_s \varepsilon + \left(1 - c_e w_s\right) w_A \\ \vec{x}_A \leftarrow \vec{x}_A + c_c w_s \Delta u \left(\vec{x}_A - \vec{x}_B\right) \end{gathered}$$

where $\overrightarrow{x}_i$ is the hyperbolic coordinate of node i, w_i is the uncertainty of node i's coordinate, c_e and w_s are algorithmic constant parameters, Δ is the magnitude of the movement, and $u(\overrightarrow{x}_y)$ is the unit vector in the direction of $\overrightarrow{x}_y$. We introduce the formulation of magnitudes and directions of the virtual forces in the Hyperbolic space in Section 3.6.

Furthermore, to maintain the uncertainty of the coordinates, each node uses a weighted moving average of the absolute errors observed during the coordinate update process. The uncertainty is for the purpose of adjusting the magnitude of the coordinate movement, in that the greater of a machine's uncertainty, or the lower the other node's uncertainty, the wider the movement will be. Meanwhile, since the RTT measurements are prone to errors or oscillations, we use a percentile filter as in Pyxida [7]. Specifically, we use the RTT values of 50 percentile, and the length of the filter is 10.

3.4 Bootstrap of non-landmark end hosts

For each new end host, it contacts with the list of landmarks when joining the system and initializes its own Hyperbolic coordinate by iteratively updating its coordinate with respect to landmarks, which is also based on virtual forces described in Section 3.3. However, to avoid distortions caused by the inaccuracy of non-landmarks' initial coordinates, landmarks do not refine their coordinates according to measurements from end hosts.

After the initialization process, end hosts update their coordinates with other machines, without the need of landmarks.

3.5 Coordinate Refinement

After successfully completing the initialization process, to keep Hyperbolic coordinates accurate and stable, a node will compensate the RTT measurements towards other neighbors, if and only if its coordinate's accuracy can not be improved; then it updates its coordinate by applying virtual forces symmetrically with the corresponding neighbor in multiple rounds, based on the coordinate computation process in Section 3.3.

3.5.1 Stablization based on RTT compensation

The coordinate computation process in Section 3.3 does not consider the drift of coordinates, which causes caching machines' coordinates inappropriate since the positions soon become obsolete. An improved coordinate update policy is to keep coordinates stable, where positions are refined in small steps after the accuracy of positions can not be significantly promoted.

When a machine A updates its coordinate by communicating with a neighbor B, it determines the estimated error with all neighbors firstly, by computing its weighted absolute error E:

$$E_{t+1} = w \times \frac{\sum_i \left|d(A,i) - d_{A,i}\right|}{N} + \left(1 - w\right) E_t$$

where w is a weighted constant, N is the size of neighbors, $d(A,i)$ is the estimated distance between node A and i, $d_{A,i}$ is the real RTT value. If E does not become lower, then node A's coordinate is marked as "stable", and we refine A's coordinate based on a stabilized process, similar to stable process proposed in [8]. otherwise, the coordinate is marked as "normal", and we update A's coordinate based on the process in Section 3.3.

In the stabilized process, to reduce movement magnitude of A's coordinate which is computed based on new RTT d_{AB}, we compensate the RTT between A and B as $d_{AB}+e_i$ with the estimated absolute error e_i, which is defined as

$$e_{i+1} = e_i \times w + (1 - w) \times \left(d(A,B) - d_{AB}\right)$$

. Then A's position is updated as those in the "normal" update process (w is 0.9 by default).

3.5.2 Multi-round symmetric update

If a node A refines its coordinate in a small step after measuring new RTT to a node B, but they do not cooperatively update their coordinates according to new RTT values, the

gained performance improvement is easily weakened or totally neutralized by B's large-step movement that contradicts A's refinement according to a third node. As a result, their coordinates can converge to local minima with low accuracy globally.

To resolve the problem, We present a symmetric update process which update a node pair's coordinates interactively. Each participant of a node pair involving a coordinate update process, updates their own coordinates in a symmetric manner, in multiple k rounds (15 by default). This has two benefits: 1) the RTT values are symmetric, so it is cost effective to notify each node in the pairs to improve their coordinates; 2) multiple rounds of coordinate updates can avoid the local minima caused by a single coordinate movement, thus increasing the robustness against measurement outliers.

3.6 Formulating virtual forces in the Hyperbolic space

In the Hyperbolic space, the magnitude and the direction of virtual force have to be reformulated, due to the changes of distance function, which is described in Section 3.2.

First, the magnitude is the difference between RTT measurements and estimated distances based on the Hyperbolic model. It can be written as $\Delta \leftarrow d_{AB} - d(A,B)$. Here d_{AB} is the RTT measurement, and $d(A,B)$ represents the estimated distance, which is the pairwise hyperbolic distance between A and B in 'Loid model multiplied the curvature k, as shown in Section 3.2.

Second, the direction of the virtual force is an unit vector in the Hyperbolic space (for a vector x, $u(x) \leftarrow x/\left\|x\right\|^H$, where $\left\|\bullet\right\|^H$ is the norm of Hyperboloid model). We find that there exists mistakes in formulating directions of virtual forces in [3], which can be easily verified based on the upper sheet of hyperboloid as follows. Since for any node A involved in a node pair (A,B), the direction of the virtual force is the same with that of the gradient of the hyperbolic distance $d(A,B)$, as shown in [5](from section II.C).

The gradient $\partial d(A,B)/\partial \vec{x}_A$ of the approximited network distance $d(A,B)$ between A ($\vec{x}_A$) and B ($\vec{x}_B$), with respect to $\vec{x}_A$ is:

$$\begin{aligned}\frac{\partial d(A,B)}{\partial \vec{x}_A} &= \left(\frac{\partial}{\partial x_i}\left(d_{AB}^H \times k\right)\right) \\ &= k\left(\frac{\partial}{\partial x_i} d_{AB}^H\right) \\ &= k\left(x_i \frac{(x_B)_{n+1}}{(x_A)_{n+1}} - (x_B)_i\right) \div \sinh d_{AB}^H \\ &= \left(\frac{\sqrt{1+\sum_{i=1}^n (\vec{x}_B)_i^2}}{\sqrt{1+\sum_{i=1}^n (\vec{x}_A)_i^2}}\vec{x}_A - \vec{x}_B\right) \times \frac{k}{\sinh d_{AB}^H}\end{aligned}$$

. Therefore, based on the direction of the gradient function $\partial d(A,B)/\partial \vec{x}_A$, we are ready to formulate the direction $u^H(\vec{x}_A - \vec{x}_B)$ as follows:

$$u^H(\vec{x}_A - \vec{x}_B) \leftarrow u\left(\left(\frac{\sqrt{1+\sum_{i=1}^n (\vec{x}_B)_i^2}}{\sqrt{1+\sum_{i=1}^n (\vec{x}_A)_i^2}}\vec{x}_A - \vec{x}_B\right) \times \frac{k}{\sinh d_{AB}^H}\right)$$

4 Performance Evaluation

4.1 Experiment setup

We have implemented HyperSpring based on Pyxida [7]. There are three additional techniques to increase the stability and accuracy of coordinates. First, the RTT latency is smoothed based on combining new RTT measurements with percentile filters to reduce the RTT oscillation. Second, we only update coordinates with neighbors that currently send new RTT samples, to reduce the misguidance of neighbors' obsolete coordinates. The maximum size of neighbors each node contacts with is 32. Third, large RTT values which surpass a threshold (2s by default) are skipped, since network coordinates can only model long-term network latencies, and large RTTS are usually caused by short-term network congestions instead. The dimension of coordinates is 5.

To guarantee a realistic comparison, we use a four-hour PlanetLab Ping trace between 226 machines on the PlanetLab [6] to evaluate the optimization techniques for Vivaldi in Hyperbolic and Euclidean spaces. The trace records RTTs between node pairs as well as the time-stamps. So we can replay the communication process at a simulation environment.

The performance metrics involved in our experiments include: 1) relative error: for each pair of nodes A and B, it is defined as $\left|RTT_{AB} - d(A,B)\right| / RTT_{AB}$, where $d(A,B)$ is the estimated distances between A and B; 2) coordinate drift: it is the length of the virtual force added to the present coordinate computed by the coordinate update process in Section 3.6.

4.2 Parameter configuration of HyperSpring

A. Hyperbolic space configuration

First, we examine the sensitivity of curvature and dimension of the Hyperbolic space without the influences of the bootstrap process, the symmetric update process and the stabilization process.

Fig 1 plot the performance of the experiment using varying curvature with dimension as 5. The error-bars in the figures represents the 50th, 80th and 95th percentiles of the corresponding performance metrics(the meanings of error-bars in the following paragraphs are the same). Small values of curvature, such as 5 cause slow convergence and high coordinate drifts; increasing curvature to 10 and 15 causes

faster convergence and slow coordinate drifts, and the variance of performance is low; but increasing curvature to 30 leads to high-error coordinates. The reason for the high error is that too small or large curvature cause estimated distances bend apart from the best situation.

Furthermore, by setting the curvature as 15, we determine the sensitivity of dimensionality according to the curvature. Fig 2 plots the progress of simulation as the increment of dimensionality. We found that performance approximately keeps identical along the increment of the dimensionality. Therefore, The accuracy is insensitive to changes of dimensionality. So we set the dimension as 5 and the curvature as 15 throughout the rest of the evaluation.

B. Parameter selection

We investigate the sensitivity of parameters for bootstrap and the symmetric update process, by combining them independently with the basic coordinate computation process in Section 3.3, so that we can remove the possible correlations between the bootstrap process, the symmetric update process and the stabilization process.

a) Bootstrap process: We vary the number of landmarks which are chosen uniformly at random to determine the variety of estimation accuracy. Fig 3 plots the progress of averaged relative errors as the increment of landmarks. At first, the performance increase rapidly when the number of landmarks are small (less than 20); thereafter, it does not improve much. We also test the neighbor selection which maximizes the inter-distances, and find that the performance is not sensitive to varying neighbor placement. So any node in the system can become the landmarks for the bootstrap process.

b) Symmetric update process: We change the update rounds between each pair of communicating nodes during the symmetric update process. Fig 4 depicts the changes of performance with different updating rounds. The results show that as the number of updating rounds increase from 5 to 15, symmetric updates can increase the accuracy of estimated distances, by escaping from bad local minima during the coordinate update process. However, as the increment of updating rounds (e.g., 20), the performance of a subset of nodes becomes worse, since the updates can cause coordinates more apart from the best positions.

4.3 Comparision results

We compare HyperSpring with two kinds of latency estimation methods: 1) basic Hyperbolic-space based Vivaldi [3]; 2) Euclidean-space based Vivaldi [1] that is implemented in [7]. Fig 5 and 6 depict the performance dynamics as the time-stamp progresses.

Firstly, Fig 5 shows that, HyperSpring is significantly accurate than 1) and 2),where the median and the 80th percentile relative errors of HyperSpring are less than 0.05

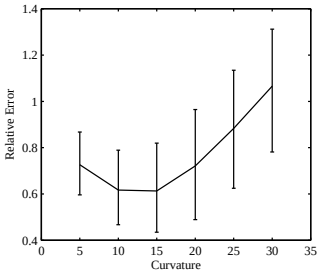

Figure 1. The relative errors as the increment of the curvature.

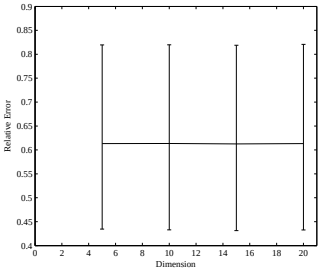

Figure 2. The relative errors along with different dimension.

and around 0.15, compared to the median value 0.5 of basic Hyperbolic-space based Vivaldi, and the median value 0.16 of Euclidean space based Vivaldi. Meanwhile, most of the 95th percentile of relative errors of HyperSpring are between 0.2 and 0.4, which are more than two times smaller than those of basic Hyperbolic-space based Vivaldi, and are about half times smaller than those of Euclidean space based Vivaldi. However, there are also quite a few of samples of HyperSpring having large relative errors which reach 0.8 or event to 1, which indicates that HyperSpring provide estimation guarantees with small slack: it produces embeddings with small dimension and distortion for most of nodes, while it allows a small fraction of RTTs to be arbitrarily distorted.

Secondly, as shown in Fig 6, the coordinate drifts of HyperSpring are significantly decreased with respect to those of basic Hyperbolic-space based Vivaldi and Euclidean-space based Vivaldi. This is due to the dynamic adjustment of movements of HyperSpring.

5 Conclusion and future work

We consider the problem of estimating latencies in the Hyperbolic space which is believed to model the Internet structure more naturally than the Euclidean space. We present a scalable latency estimation method HyperSpring

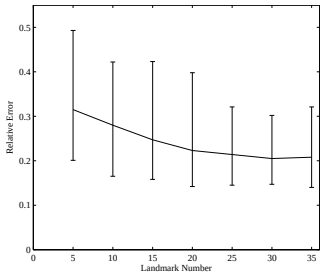

Figure 3. Relative errors as the increment of the number of landmarks.

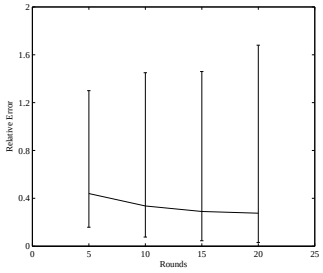

Figure 4. Relative errors with varying rounds of symmetric updates.

which derives from Vivaldi, but consists of three distinct optimization components that increase the accuracy and stability of latency estimation. We evaluated the performance of HyerVivaldi with real-world network traces that can model the network dynamics. The relative errors of HyperSpring are smaller than those of Euclidean space based Vivaldi in most cases. And the coordinate drifts of HyperSpring is smaller than 2, which is about two times smaller than those of Vivaldi. An interesting phenomenon we found from evaluation is that by applying the optimization techniques of HyperSpring into the Euclidean space based Vivaldi, there are significant performance improvements compared to those of HyperSpring. We are exploring the reasons behind this currently.

References

[1] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM*, pages 15–26, 2004.

[2] R. Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM*, 2008.

[3] C. Lumezanu and N. Spring. Measurement manipulation and space selection in network coordinates. In *The 28th International Conference on Distributed Computing Systems*, pages 361–368, 2008.

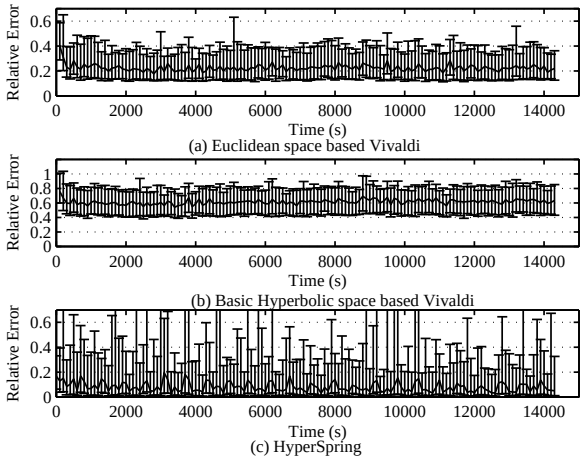

Figure 5. Comparision of relative errors.

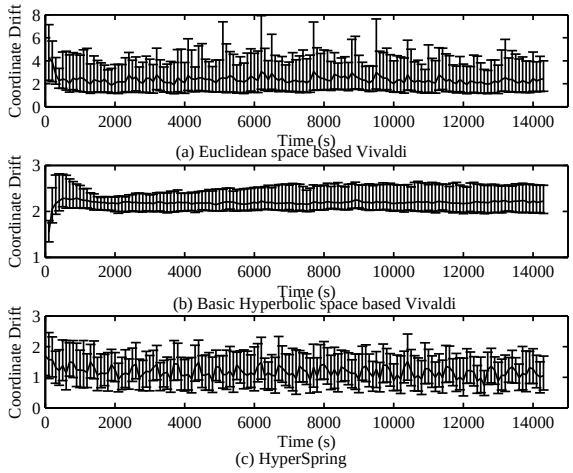

Figure 6. Comparison of coordinate drifts.

[4] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, pages 170–179, 2002.

[5] Y. Shavitt and T. Tankel. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Trans. Netw.*, 16(1):25–36, 2008.

[6] Univ. of Harvard. four hour ping trace. http://www.eecs.harvard.edu/ syrah/nc.

[7] Univ. of Harvard. Pyxida library. http://pyxida.sourceforge.net.

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