A Decentralized Network Coordinate System for Robust Internet Distance Prediction

Li-wei Lehman
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: lilehman@mit.edu

***Abstract*— Network coordinate systems have recently been developed as a scalable mechanism to predict latencies among arbitrary Internet hosts. Our research addresses several design challenges of a large-scale decentralized network coordinate system that were not fully addressed in prior work. In particular, we examine the design issues of a decentralized network coordinate system operating in a peer-to-peer network with high churn, high fractions of faulty or misbehaving peers, and high degrees of network path anomalies. This paper presents a fully decentralized network coordinate system, PCoord, for robust and fault-tolerant Internet distance prediction. Through extensive simulations, we examine the convergence behavior and prediction accuracy of PCoord under a variety of scenarios, and compare its performance with an existing network coordinate system, Vivaldi. Our results indicate that PCoord is robust under high churn, and degrades gracefully even under high fractions of faulty nodes, and high degrees of triangle inequality violations in the underlying network distances. Finally, our results indicate that even under an extremely challenging flash-crowd scenario where 1740 hosts simultaneously join the system, PCoord is able to converge to 12% median relative prediction error within 10 seconds.**

I. Introduction

Network coordinate systems have recently been developed as a scalable mechanism to predict latencies among arbitrary Internet hosts [12], [11], [15], [21], [10], [19], [20], [4], [2], [8], [9]. The idea of a network coordinate system, first proposed by the GNP system [12], is for each host to derive a mapping of itself in some D-dimensional geometric space using a small set of sampled distances so that the actual inter-host network latencies can be estimated as a function of the nodes' geometric distances. GNP relies on a fixed set of landmark nodes in the Internet to provide reference coordinates. Hosts position themselves in the geometric space using measured distances to these landmark nodes.

More recently, several approaches were proposed to construct network coordinates in a completely decentralized fashion [4], [2], [8], [9]. These decentralized approaches do not rely on a fixed set of landmark nodes to construct coordinates. Instead, hosts compute their coordinates based on sampled distances and reference coordinates gathered from other participating peer hosts. Constructing network coordinates in a fully decentralized manner is a challenging task. It is difficult because each host is making parallel, independent, local estimates of its network coordinates based on a small number of potentially noisy and faulty samples. The key challenge is to ensure that locally derived host coordinates converge to a global geometric configuration that accurately captures the actual network distances *using as few network measurements as possible*.

This paper presents a fully decentralized network coordinate system, PCoord, for robust and fault-tolerant Internet distance prediction. PCoord assigns coordinates to hosts on the Internet so that the Euclidean distances between hosts' coordinates accurately predict their network latencies. In PCoord, each host updates its coordinates to minimize a loss function that measures the difference between the actual and the geometric distances between itself and a small set of other hosts.

Our research addresses several design challenges of a large-scale decentralized network coordinate system that were not fully addressed in prior work. In particular, we examine the convergence behavior and performance characteristics of a decentralized network coordinate system operating in a peer-to-peer network with high churn, high fraction of potentially faulty or misbehaving peers, and high degree of triangle inequality violations in the underlying network paths.

This paper extends an earlier description of PCoord [9], and introduces the following stability mechanisms to facilitate fast convergence to low system prediction errors.

- A weighted loss function to distinguish between nodes with high and low errors.
- A "resistance" mechanism that helps to stabilize the convergence and avoid oscillation.
- A "damping" mechanism to avoid instability and oscillation caused by noisy, mis-behaving or faulty information.

Through extensive simulations using real network measurements, we examine the convergence behavior and prediction accuracy of PCoord under a variety of instability factors, and compare its performance with an existing network coordinate system, Vivaldi. Our findings are highlighted as follow.

- Our results suggest that, in a 1740 nodes peer-to-peer system, PCoord can converge to a low prediction error configuration within 10 seconds, where each node performs less than 10 coordinate updates using 10 reference points per update.
- Under an extremely challenging flash-crowd scenario where 1740 nodes join simultaneously, PCoord is able to converge to a 12% system-wide medium relative prediction error using half as many samples as Vivaldi.
- PCoord is robust under high churn, and degrades grace-

To be submitted for publication.

fully even under high fraction of faulty nodes, and high degree of triangle inequality violations in the network distances.
- PCoord's damping and resistance mechanisms reduce the instability caused by network path anomalies, and thus enable PCoord to be more robust than Vivaldi under heavy triangle inequality violations.
- Vivaldi's performance, both in terms of convergence speed and prediction accuracy, is more sensitive to the degree of path anomalies in the underlying network topologies than PCoord. When the number of triangle inequality violations doubles in the underlying network topology, Vivaldi's performance degradation is 60% more than that of PCoord's.

The rest of the paper is organized as follows. Section II describes the PCoord algorithm. In section III, we present our simulation results and compare the performance of PCoord with other existing schemes. In section IV, we describe related work. Finally, we present our conclusions and ideas for future work in Section V.

II. The Algorithm

PCoord is a fully decentralized network coordinate system with each host updating its coordinate iteratively to refine the prediction accuracy of its estimated position. Each host updates its coordinates to minimize a loss function that measures the difference between the actual and the geometric distances between itself and a small set of other hosts.

We previously introduced the term PCoord in [9] to refer to a general set of peer-to-peer based approaches in decentralized network coordinates construction. Under the PCoord framework in [9], we evaluated several proof-of-concept coordinates mapping strategies and explored their performance characteristics under different peer sampling strategies. Our prior work in [9] did not provide any mechanisms to stabilize system convergence, nor did it address issues in dynamic join/leave and fault-tolerance. In this paper, we present the PCoord algorithm which introduces the following mechanisms.

- A weighted loss function that allows sampled coordinates with higher prediction accuracy to have a higher weight in the loss function.
- A weighted "resistance" factor in the loss function that helps to stabilize the convergence process.
- A threshold-based mechanism to dampen the amount a node moves toward new coordinates by a factor that is inversely proportional to the fit error of the current batch of sampled peer nodes' coordinates and distances.

We present the PCoord algorithm in two steps. We first present a Simple algorithm that provides the basic structure and steps taken by each PCoord host to perform coordinate update. Next, we present the actual PCoord algorithm which introduces several mechanisms to ensure convergence to a lower prediction error.

A. A Simple Algorithm

In Simple PCoord, each node performs continuous update on its coordinate. Each of the coordinate update consists of three phases: (1) the sampling and information exchange phase in which a node selects M reference points, gathers distance measurements and coordinates, and exchanges peer list information with those M reference points, (2) the coordinate update phase, in which a new coordinate is computed to minimize a loss function defined in terms of those M reference points, and (3) the near peer probing phase in which each host refines its search for its nearby peers by probing other hosts based on their triangulated distances.

In this paper, we focus on phase two only, which is the coordinate update step. A variation of the Simple algorithm was previously described in [9], [7], which gave a detailed description on the peer sampling and probing strategies from phase one and three above. Next, we summarize the notations below and then present the algorithm in more details.

M = Number of reference points for each coordinate update
c_i = Coordinates of host i
d_{ij} = RTT between i and j
c_{origin} = Coordinates at the origin

//i is the node that is running the procedure
SimplePCoord() {
    $c_i = c_{origin}$
    while (in the system) {
        $Samples$ = SamplePeers()
        c_i^{new} = MinimizeError($Samples$, c_i)
        $c_i = c_i^{new}$
        ProbeNearNeighbors()
    } //end while
}

//$Samples$ consist of M sampled peer nodes
//c_{guess} is the initial guess for the coordinates
MinimizeError ($Samples$**,** c_{guess} **)** {
    find c_i^{new} that minimizes E using
    c_{guess} as an initial guess, where
        $E = \sum_{j \in Samples} (d_{ij} - \|c_i^{new} - c_j\|)^2$
    return c_i^{new}
}

At each update iteration, each host i measures its round trip latency to M other peer nodes, and obtains those M nodes' current coordinates. Host i then updates its coordinates to minimize the sum of squared differences between the measured and computed distances with those M peer nodes. The computation of the coordinates for node i then involves finding c_i^{new} that minimizes the following loss function.

$$E = \sum_{j=1}^{j=M} (d_{ij} - \|c_i^{new} - c_j\|)^2$$

B. The PCoord Algorithm

There are several potential problems with the above simple version of the algorithm. One problem is that it does not distinguish between nodes with coordinates that have different

prediction accuracy. To avoid reacting too quickly to bad reference points, we propose a weighted loss function, in which the loss each reference point contributes is weighted by the prediction accuracy of each reference point's coordinates. The weight is computed based on the relative prediction accuracy of each reference point so that the nodes with more accurate coordinates will have more influence on the solution than the less accurate ones.

Another problem is that the algorithm determines the new coordinate entirely based on measurements from the current batch of reference points. There is no mechanism for the coordinates generated using previous samples to cast any vote on the the position of the new coordinate in the current update. The simple scheme thus tends to react too quickly based on the measurements of the current batch of reference points, and leads to potential oscillation.

In order to reduce oscillation, we introduce an additional "resistance" factor into the loss function so that a node with highly accurate coordinates will not overly react to reference points with less accurate coordinates. When computing c_i^{new}, node i adds itself as the $(M+1)$th node in its reference points set, and thus introduces c_i into the loss function as a resistance factor that penalizes movement of c_i^{new} to a new location. This term is weighted by the relative prediction accuracy (relative to other reference points of i) of node i's coordinates, so that the more confident a node is about the accuracy of its own coordinates, the more resistance the term introduces. For a newly joined node, the weight to this resistance term is initialized to zero. Each node continuously updates the confidence index of its own coordinates as a function of the weighted moving average of its current and past prediction error.

1) Weighted Error Function with a Resistance Factor: More precisely, the weighted loss function with the resistance factor is as follows. Let w_i be the weight of node i. The coordinate update procedure now becomes a problem of finding c_i^{new} that minimizes the weighted loss $\mathcal{E}$, where $\mathcal{E}$ is defined as follows.

$$\mathcal{E} = w_i(d_{ii} - \|c_i^{new} - c_i\|)^2 + \sum_{j=1}^{j=M} w_j(d_{ij} - \|c_i^{new} - c_j\|)^2$$

where $d_{ii} = 0$, $0 \leq w_i \leq 1, 0 \leq w_j \leq 1$, and $w_i + \sum_{j=1}^{j=M} w_j = 1$.

The procedure for maintaining the weighted moving average of the relative prediction error is invoked at each coordinate update iteration after the sampling phase.

e_i^p = weighted moving average of relative prediction error at node i
e_{ij}^p = relative prediction error for distance between node i and j
α = weight for computing weighted moving average of prediction error

//i is this node
UpdatePredictionError($Samples$**)** {
  for each j in $Samples$ {
    $e_{ij}^p = \frac{|\|c_i - c_j\| - d_{ij}|}{d_{ij}}$
    $w = \frac{(e_i^p)^2}{(e_i^p)^2 + (e_j^p)^2}$
    $e_i^{newp} = e_{ij}^p * w + e_i^p * (1 - w)$
    $e_i^p = \alpha * e_i^p + (1 - \alpha) * e_i^{newp}$
    $e_i^p = MIN(1, e_i^p)$
  } //end for
}

The following pseudocode fragment describes how the weight is assigned based on the relative prediction error. τ is a small constant added to one to define e_{TOP}^p for weight computation. When $\tau = 0$, nodes with relative prediction error of one have zero weight in the loss function. Setting $\tau \geq 0$ allows nodes with relative error of one to have a non-zero weight. In this study, τ is set to 0.05.

Weight Assignment

  τ = a small constant, $\tau \geq 0$
  $e_{TOP}^p = 1 + \tau$
  //assign weight to each sample in $Samples$, which
  //includes the "resistance" term
  for each node j in $Samples$ {
    $a_j = e_{TOP}^p - e_j^p$
    $w_j = \frac{a_j^2}{\sum_{k \in Samples} a_k^2}$
  }

2) Adjusting Amount of Coordinate Updates Based on Goodness-of-Fit: The weighted loss function described above helps to reduce the negative effect of reference points with high prediction error. However, the prediction error does not necessarily reflect whether a particular pair-wise distance between nodes i and j serves as a good sample to predict the position of the two nodes.

In this section, we introduce a mechanism that allows a node to adjust how much it should move its coordinates in response to a particular batch of samples based on the goodness-of-fit index. The goodness-of-fit is a confidence measurement associated with an entire batch of samples. The idea is that a batch of samples containing "un-representative" distances will likely yield higher residual error than good batches of samples. To avoid reacting to a batch of samples with bad fit, each PCoord node maintains a weighted moving average of the fit error over time. A node assigns a weight to each batch of samples as a function of the ratio between the average and current fit error, and then decides how much it should react to the current batch of samples based on the weight. More precisely, if the fit error of the current batch of samples exceeds the average fit error, then the node dampens the amount it moves toward the new coordinate by a factor ρ which is the ratio between the average and current fit error.

The following pseudocode presents the above procedure in two steps: (1) a procedure to update fit error, which is invoked

at the end of each coordinate update phase, and (2) a code fragment that computes fraction of coordinate movement.

e_i^f = Weighted average of fit error of node i
e_i^{newf} = The fit error of the new batch of data
β = Weight for computing weighted moving average of e_i^f
ρ = Fraction of movement toward the new coordinates

//i is this node
UpdateFitError(e_i^{newf} **)** {
  //update the weighted moving average of fit error
  $e_i^f = \beta * e_i^f + (1 - \beta) * e_i^{newf}$
}

Compute Fraction of Coordinate Movement

  //e_i^{newf} is the residual error
  $e_i^{newf} = \sum_{k \in Samples} w_k (d_{ik} - \|c_i^{new} - c_k\|)^2$
  $\rho = MIN(\frac{e_i^f}{e_i^{newf}}, 1)$
  //move ρ fraction of the way toward new solution
  $c_i^{new} = c_i + (\rho * (c_i^{new} - c_i))$

3) Summary: the PCoord Algorithm: In this section, we pull the above pieces together and summarize the PCoord pseudocode below.

//i is this node
PCoord() {
  $c_i = c_{origin}$
  while (in the system) {
    $Samples$ = SamplePeers()
    UpdatePredictionError($Samples$)
    //add node i's own coordinates to the samples
    $Samples$ = $Samples$.add(i)
    (c_i^{new}, e_i^{newf}) = MinimizeWeightedError($Samples$, c_i)
    $c_i = c_i^{new}$
    UpdateFitError(e_i^{newf})
    ProbeNearNeighbors()
  }
}

MinimizeWeightedError ($Samples$**,** c_{guess} **)** {
  $e_{TOP}^p = 1 + \tau$
  for each node k in $Samples$ {
    //assign weight to each node in $Samples$
    $a_k = e_{TOP}^p - e_k^p$
    $w_k = \frac{a_k^2}{\sum_{j \in Samples} a_j^2}$
  } //end for

  //now find new coordinate
  find c_i^{new} that minimizes
    $\sum_{k \in Samples} w_k (d_{ik} - \|c_i^{new} - c_k\|)^2$

  //e_i^{newf} is the residual error after minimization
  $e_i^{newf} = \sum_{k \in Samples} w_k (d_{ik} - \|c_i^{new} - c_k\|)^2$
  //move toward new coordinates
  $\rho = MIN(\frac{e_i^f}{e_i^{newf}}, 1)$
  //move ρ fraction of the way toward the new solution
  $c_i^{new} = c_i + (\rho * (c_i^{new} - c_i))$
  return (c_i^{new}, e_i^{newf})
} //end MinimizeWeightedError

The SamplePeer() and ProbeNearNeighbor() procedures are described in [9], [7]. In order to compare with Vivaldi, we have turned off the PCoord probe near peer option. The results generated in this paper use a random peer sampling strategy.

III. Evaluation of PCoord

A. Evaluation Methodology

We evaluate the PCoord approach extensively through simulations using both real network measurements and simulated topologies. We compare the performance of PCoord with Vivaldi, and the original GNP scheme (referred to as the FixedLM scheme from now on) in terms of pairwise distance prediction accuracy.

1) Performance Metrics: We define the *prediction error (PE)*, or simply error, of a link as the absolute difference between the predicted RTT and the actual RTT. Following the conventions in Vivaldi [4], we define the error of a node as the median of the link errors for links involving that node. The error of the system is defined as the median of the node errors for all nodes in the system.

We use the absolute relative error (RE) defined in [12] as our performance metric when comparing with the GNP scheme. For each pair of nodes, their absolute relative error is defined as $\frac{|\|c_i - c_j\| - d_{ij}|}{MIN(d_{ij}, \|c_i - c_j\|)}$.

2) Data Collection: We evaluate our scheme using the following network measurements.

- The King data set from Vivaldi [4], which involves the round-trip latency among 1740 Internet DNS servers.
- The PlanetLab [16] all-pairs-ping data set among 127 nodes collected on May 10, 2004.
- The AMP [6] data set January 30, 2003 which measure the round-trip ping time among 104 nodes.
- The RON2 data set [17], [1], which measures the RTTs among 15 Internet hosts.

We only present the King results here due to space constraints. The results from other data sets are qualitative similar and can be found in [7].

3) Simulation Setup: We have simulated the execution of PCoord using p2psim [13], an event-driven, packet-level network simulator. We use the Simplex Downhill algorithm to minimize the loss function at each coordinate update. We measure the processing cost of the Simplex Downhill operation on a Sun UltraSparc (with 150 MHz CPU and 4096 Megabytes of memory), and use the measured median processing time in our PCoord simulation. The median processing time is 10 ms per coordinate update when 10 reference points are used. Our results are consistent with the measured Simplex algorithm processing time in GNP [12].

Each node proceeds in its coordinate update independent of other nodes' update progress. Asynchronous communication is used when gathering samples from M peer nodes in each coordinate update step.

4) PCoord Parameter Setting: We ran the PCoord simulations with various sample batch size M to explore its effect on convergence and prediction accuracy. Our results suggest that using 10 reference points at each update step ($M = 10$) yields quick convergence to low error and achieves good tradeoff between communication and computation overhead. The rest of the parameters are set as follow. $\alpha = 0.95$, $\tau = 0.05$, and $\beta = 0.6$.

Simplex Downhill Settings: In order to obtain high quality solutions, the Simplex algorithm usually restarts the minimization routine after it claims to have found a solution. Within each restart, the terminating criteria are usually specified by some tolerance and some threshold $NMAX$, which denotes the maximum allowed function evaluations.

In PCoord, the Simplex algorithm is run in a "light-weight" mode. More specifically, within each coordinate update step, at most 1000 evaluations of the objective function are performed, and we do not restart the minimization procedure within each coordinate update step. Our results indicate that the light-weight setting performs as well as a more computationally expensive setting, such as 500,000 objective function evaluations with 3 restarts. We believe PCoord is able to perform well using the light-weight Simplex procedure because PCoord itself maps the coordinates iteratively; at each coordinate update, PCoord uses the "optimal" solution found in the previous iteration as its initial guess for the minimization procedure in the next iteration.

5) Vivaldi Parameter Setting: All Vivaldi simulations presented in this work use the adaptive time step mechanism described in [4] and implemented in the p2psim [13]. In Vivaldi, a constant C_c ($0 \leq C_c \leq 1$) is used to control how much a node reacts to each new sample. We set C_c to 0.25, which is reported to yield the best overall convergence to low prediction error [4].

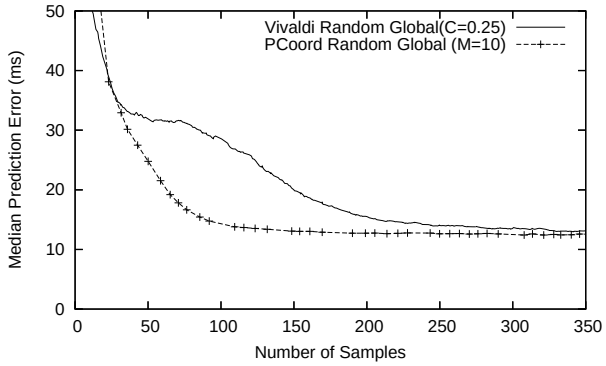
B. Convergence Behavior of PCoord and Vivaldi

In this section, we compare PCoord and Vivaldi in terms of number of samples required for convergence. We compare PCoord and Vivaldi using random sampling strategy, in which each peer randomly samples other peers drawn from the global population of the system.

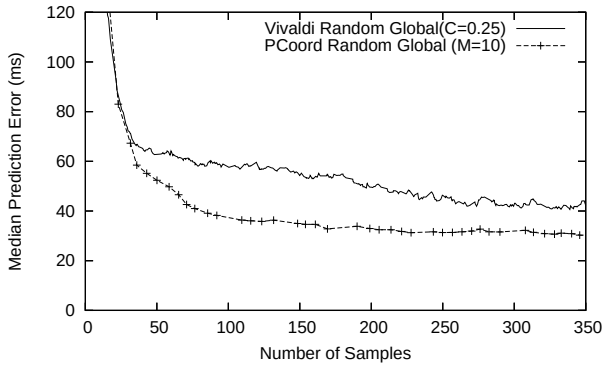
Figure 1 plots the median, 95th and 5th percentile error of PCoord and Vivaldi using the King data set. Our results indicate that PCoord is able to converge faster than Vivald: it takes PCoord approximately 170 samples to converge to the 12 ms error range, and Vivaldi requires approximately 350 samples to reach the same error range.

C. Comparison of PCoord, Vivaldi and FixedLM

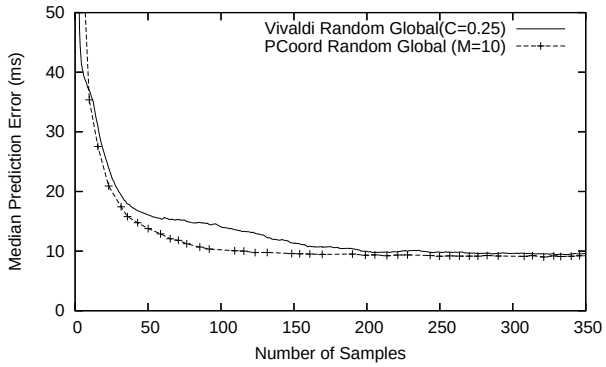
Figure 2 shows the cumulative distribution of relative error of PCoord, Vivaldi, and the FixedLM scheme after PCoord and Vivaldi hosts have updated their coordinates using an average of 100, 150, and 300 samples. With the FixedLM scheme,



(a) Median error



(b) 95th Percentile Error



(c) 5th Percentile Error

Fig. 1. Convergence of PCoord (10 reference points) and Vivaldi (C_c=0.25) using random peer sampling. King data.

each host uses 30 fixed landmarks; the results shown have the lowest median error of all 20 different randomly-generated landmark configurations.

We note that PCoord's prediction accuracy is comparable to that of the FixedLM scheme after 100 samples. After 100 samples, PCoord's median relative error is approximately 12% which is only 2% more than that of the FixedLM scheme. Vivaldi ($C_c = 0.25$) takes about 300 samples to achieve 12% median relative error. After 100 samples, Vivaldi's median relative error is 21%, which is about twice as much as that of PCoord's using the same number of samples.

D. PCoord Performance under Incremental Join

We have also simulated PCoord in an incremental join scenario, where a node joins one at a time after the rest of the system has converged. On average, the median prediction error of a newly joined node can decrease to the 12 ms range within two coordinate updates using 10 reference points per update (i.e., within 20 samples).

E. PCoord Performance under High Churn

In this section, we examine PCoord's performance under churn, i.e., when the system experiences continuous membership changes as a result of nodes joining and leaving. We examine the following questions. How robust is PCoord under high churn? At what point do we begin to observe significant performance degradation as the join/leave rate increases?

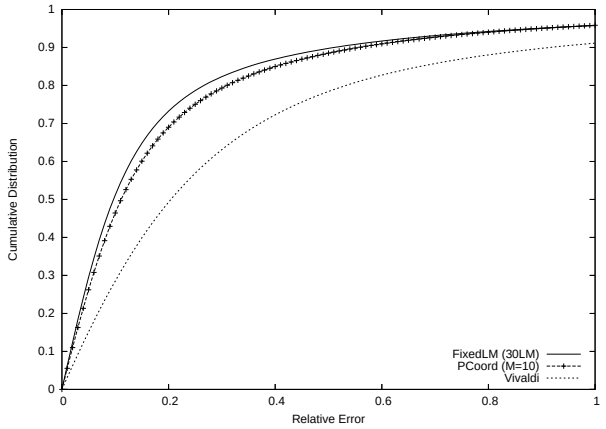
Under the dynamic join and leave mode, each node alternately leaves and re-joins the system. The time interval a node stays in and out of the system is exponentially distributed with a mean t. Recent studies suggest that the median session duration of hosts in peer-to-peer systems is approximately one hour [18]. We have chosen to use shorter time intervals, and thus higher churn rates, in our simulations in order to examine PCoord's performance under extreme conditions. We have experimented with t equal to 2, 5, 10, 20, 30 and 40 seconds, with a total simulated time of 300 seconds. When a node re-joins the system, its coordinates are re-initialized to the origin.

Figure 3 plots the median prediction accuracy of PCoord as a function of time when t is 2, 5 and 20 seconds. As a comparison, we also plot prediction accuracy of PCoord when there is no churn, i.e., when all nodes join simultaneously in the beginning and none subsequently leave. Figure 4 plots the average median prediction accuracy (averaged over time) as a function of the mean host session life time, t. The average median prediction accuracy represents the steady-state prediction error of the system averaged over time using statistics gathered after 60 seconds of simulated time.

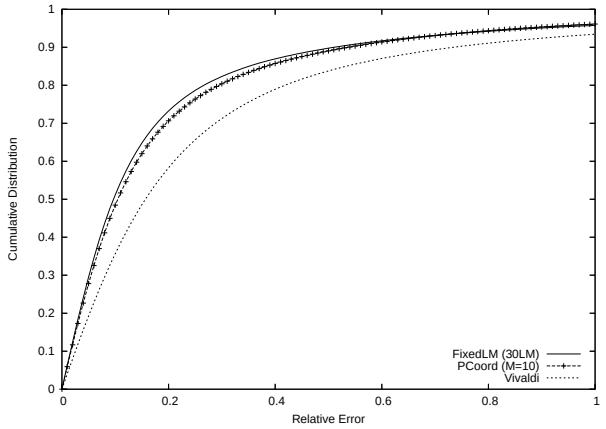
Figure 4 shows that when join/leave interval t is 10 seconds or greater, the median prediction error stays in the range of 12 ms, indicating that the churn has very little effect on the prediction accuracy of PCoord.

F. Effects of Different PCoord Mechanisms

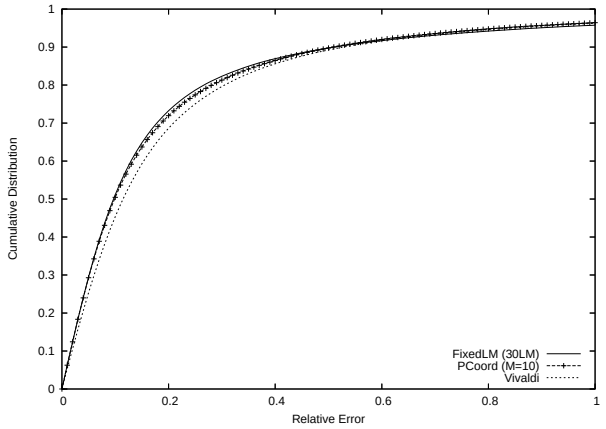
In this section, we examine the effects of different PCoord mechanisms under churn. In particular, we would like to



(a) After 100 Samples



(b) After 150 Samples



(c) After 300 Samples

Fig. 2. Comparing PCoord, Vivaldi and FixedLM in terms of relative error using King data set. We show results of PCoord and Vivaldi after using approximately 100, 150, and 300 samples per host on average, and compare with the FixedLM results where each host uses 30 fixed landmark nodes. King, N = 1740, D = 5, M = 10, C_c = 0.25.

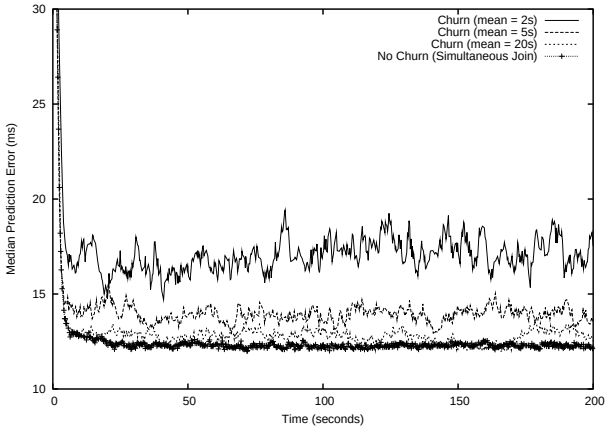


Fig. 3. PCoord performance under dynamic join and leave. King, N = 1740, M = 10, and D = 5.

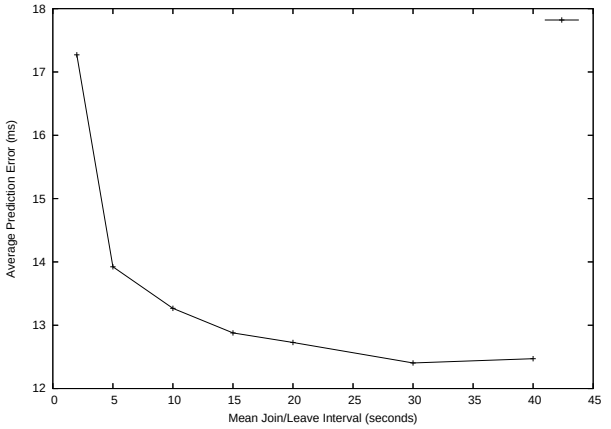


Fig. 4. PCoord prediction accuracy as a function of mean join/leave intervals. King, N = 1740, M = 10, and D = 5.

answer the following questions. How much better is PCoord in comparison to the Simple algorithm? How much added benefit does damping provide? Is the damping mechanism alone sufficient to yield good prediction accuracy? Under what scenario is each PCoord mechanism useful?

To answer the above questions, we have simulated PCoord's performance with different combinations of the mechanisms:

- Simple: this is the Simple algorithm without weighted loss, resistance, or damping.
- WLoss + Resistance: this is a version of PCoord that implements weighted loss function and the resistance mechanism. Damping is turned off in this version.
- Damp: this version only implements the damping mechanism without the weighted loss and resistance mechanisms.
- PCoord (WLoss + Resistance + Damp): this is the default PCoord algorithm with all three mechanisms turned on.

We examine performance of the above PCoord options using the King data set with a dynamic join/leave interval of 20 seconds. Each node uses random peer sampling with a default sample batch size of ten samples per coordinate update. The results are presented in Figure 5. We observe that the default PCoord mechanism has the best prediction accuracy,

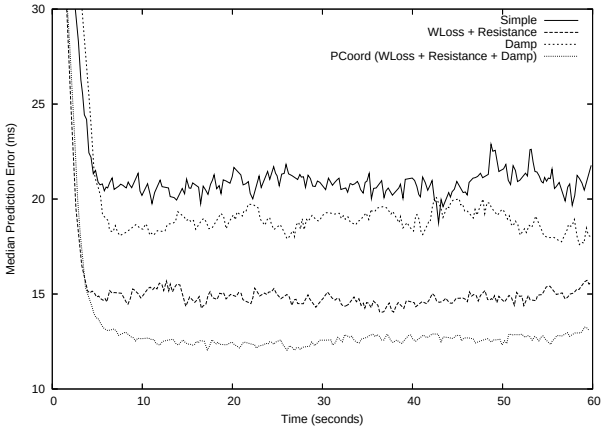


Fig. 5. PCoord performance under dynamic join and leave. Join/Leave mean interval is 20 seconds. King data set. N = 1740, M = 10, D = 5.

with median system prediction error in the 12 ms range. The prediction error of the Simple algorithm is approximately 60% higher than that of PCoord.

Damping alone decreases the prediction error the Simple algorithm by about 10 to 15%. A combination of weighted loss and resistance out-performs the Simple algorithm by almost 30%. Adding damping further decreases the prediction error by 20%.

Our results from the RON2 data set suggest that different PCoord mechanisms tend to be useful under different scenarios. For example, nodes with paths that heavily violate the triangle inequality property tend to oscillate in the geometric space; for these nodes, the damping mechanism is able to significantly reduce the amount of oscillation of those nodes' coordinates, whereas the resistance mechanism provides little help. This is due to the fact that nodes with paths that have high degree of triangle inequality violations tend to have low confidence on its own coordinates; as a result, the resistance factor has a minimal effect on stabilizing the coordinates. On the other hand, for nodes that have mostly well-behaved latencies to other nodes, the friction mechanism provides substantial benefits in improving the accuracy of the node's coordinate mapping, whereas damping provides little added benefit.

G. Robustness of PCoord against Faulty Information

In this section, we examine PCoord's robustness against faulty information. We are interested in understanding how PCoord behaves under increasing amount of faulty or corrupted information.

We study the effects of faults by randomly selecting some fraction of nodes as "faulty" nodes that provide incorrect information to the other nodes. We then measure the prediction accuracy among non-faulty nodes as a function of the fraction of faulty nodes in the system. We model two types of faulty information: (1) corrupted coordinates and (2) corrupted coordinates with a falsely high confidence index.

- Corrupted coordinates. Nodes with corrupted coordinates will generally report low confidence on their coordinates.

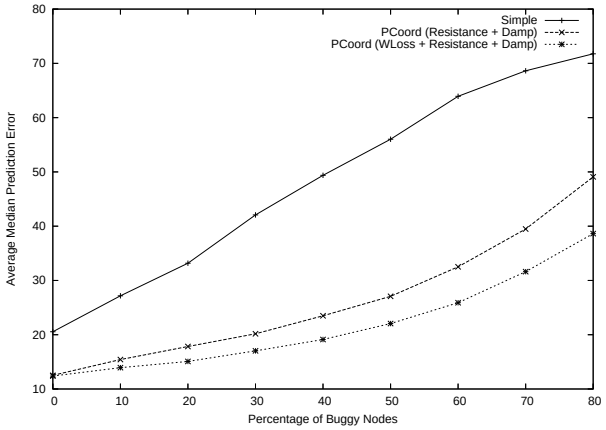


Fig. 6. Compare different PCoord's mechanisms under various fractions of buggy nodes. King data set. N = 1740, M = 10, D = 5.

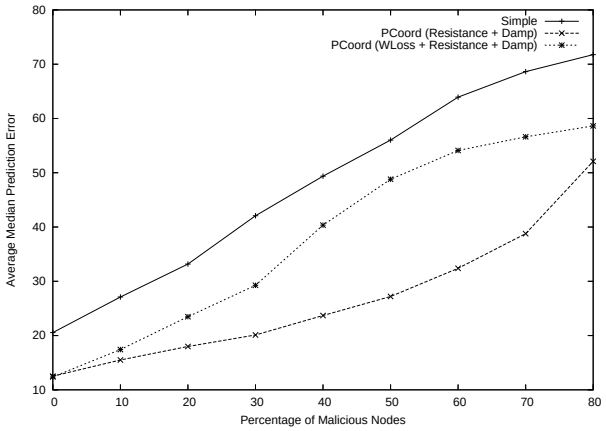


Fig. 7. Compare different PCoord's mechanisms under various fractions of malicious nodes. King data set. N = 1740, M = 10, D = 5.

We call nodes that generates corrupted coordinates the "buggy" nodes.

- Corrupted coordinates with a falsely high confidence index. For this category of faults, we model a rather naive form of "malicious" attack, in which case a node reports a randomly generated coordinates and claims that the coordinates have low prediction error (i.e., high confidence index). This will cause the other nodes to put high weights on the faulty coordinates.

We vary the fraction of faulty nodes in the system from 0 up to 80%. Figures 6 and 7 present PCoord's prediction accuracy as a function of the fraction of buggy and malicious nodes respectively. The average median prediction accuracy represents the steady-state prediction error of the system.

In order to understand how different PCoord mechanisms are affected by faulty information, we present PCoord's performance with different combinations of the mechanisms:

- Simple: this is the Simple algorithm without weighted loss, resistance, or damping.
- Resistance + Damp: this is a version of PCoord that implements the resistance and damping mechanisms.
- PCoord (WLoss + Resistance + Damp): this is the default PCoord algorithm with all three mechanisms turned on.

Since the Simple algorithm does not associate weights with samples, Simple's performance is the same under buggy and malicious modes. The same is true for PCoord when the weighted loss function is turned off.

We observe that the Simple algorithm's prediction error rises rapidly as the percentage of faulty nodes increases. Both versions of PCoord (with or without the weighted loss function) are significantly more robust than the Simple algorithm in the face of high percentage of buggy nodes.

Under the malicious model, the prediction accuracy of PCoord with the weighted loss can degrade quickly under heavy malicious attacks. In general, the combination of damping and resistance provides a fairly robust mechanism in coping with both buggy and malicious nodes. This suggests that, in a real-world deployment, a good engineering choice may be to turn off the weighted loss function if a large number of malicious nodes is expected.

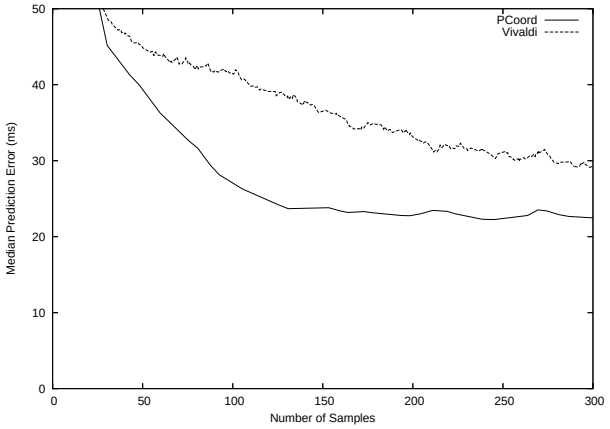
H. Effects of Triangle Inequality Violations

In this section, we examine the effects of triangle inequality violations of the measured end-to-end paths on PCoord and Vivaldi. In particular, we would like to know the performance characteristics of PCoord and Vivaldi when increasing numbers of the measured paths violate the triangle inequality. In order to examine the effects of increasing path anomalies, we randomly perturb 2% - 10% of the total links in the King data set by some arbitrary amount, thereby generating network latencies with varying degrees of triangle inequality violations.

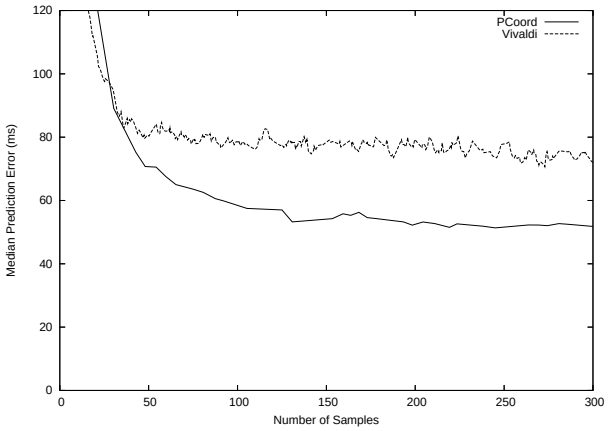
Figure 8 plots the median, 95th and 5th percentile error of PCoord and Vivaldi when 5% of the RTTs in the King data set are perturbed, which causes the number of triples that violate triangle inequality to double in comparison to the original King data set. In comparing Figure 1 and Figure 8, we observe that the performance gap between PCoord and Vivaldi has widened under heavier triangle inequality violations. This suggests that Vivaldi is more sensitive to the increase in path anomalies. While PCoord's medium prediction error worsens by 10 ms when 5% of the paths are perturbed, Vivaldi's performance degradation under the same scenario is approximately 60% higher than that of PCoord's.

Comparing the prediction accuracies of the two schemes when 150 samples are used, Vivaldi's median prediction error is 6 ms higher than that of PCoord's under the original King data set. Under the perturbed data set, Vivaldi's median prediction error is 12 ms higher than that of PCoord's – i.e., the performance gap between the two schemes doubles when the number of triangle inequality violations doubles.

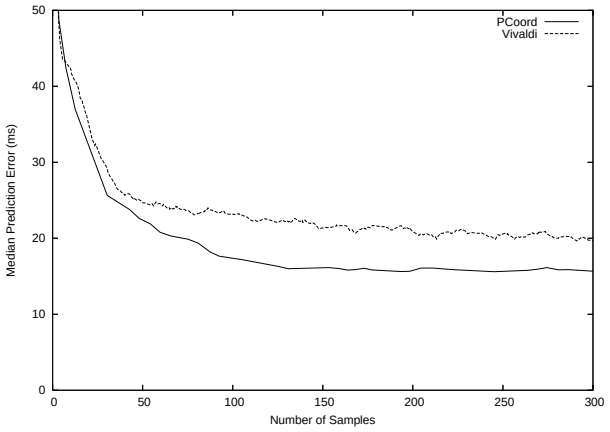
Figure 9 further shows that the performance gap between PCoord and Vivaldi increases as the percentage of perturbed links increases. The plot essentially shows how much worse Vivaldi performs relative to PCoord as the percentage of perturbed links increases. More specifically, the Y-axis shows the amount by which Vivaldi's 95th, 50th, 5th percentile prediction errors are higher than those of PCoord's under the same topology. For both schemes, the error statistics shown is taken after on average 300 samples per host. The plot shows that under the original King data set, Vivaldi's median



(a) Median error



(b) 95th Percentile Error



(c) 5th Percentile Error

Fig. 8. Convergence of PCoord and Vivaldi when randomly selected 5% of the King paths are perturbed. King, N = 1740, M = 10, C_c = 0.25, and D = 5.

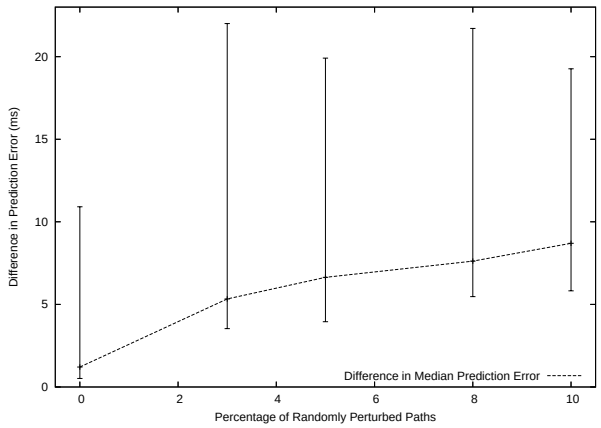


Fig. 9. Difference between Vivaldi and PCoord prediction error under increasing fraction of randomly perturbed links. The Y-axis shows the amount by which Vivaldi's 95th, 50th, 5th percentile prediction errors are higher than those of PCoord's under the same topology. For both schemes, the error statistics shown is taken after on average 300 samples per host. King, N = 1740, M = 10, C_c = 0.25, and D = 5.

prediction error is very close to that of PCoord's (when both schemes use 300 samples per host on average); however, when 10% of the links are randomly perturbed, Vivaldi's median prediction error is close to 9 ms worse than that of PCoord's.

In summary, Vivaldi's performance, both in terms of convergence speed and prediction accuracy, is sensitive to the amount of path anomalies in the underlying network topologies. PCoord's damping and resistance mechanisms reduce the instability caused by path anomalies, and thus are more robust under heavy triangle inequality violations.

IV. Related Work

The IDMaps [5], GNP [12], NPS [11], and King [14] are all architectures for a global distance estimation service. In contrast, PCoord's goal is for peer nodes in an overlay network to estimate their locations relative to other nodes in the same overlay using purely peer-to-peer measurements.

To avoid the fixed landmark problem in GNP, several schemes [15], [21], [10] have been proposed that allow hosts to use different subsets of landmarks to construct a local coordinate system, which are then transformed to a global coordinate system. These schemes, however, are not fully decentralized. Several other works focus on the geometric models and coordinates computation that yield low embedding error assuming global distance measurements are available [19], [20]. Their work did not address issues in decentralized coordinates constructions.

In [22], an approach that builds *network distance maps* based on hierarchical clustering is proposed. The Mithos [23] system embeds the network into a multi-dimensional space. The focus of their work is more on overlay construction and efficient lookup forwarding and less on network distance prediction.

Similar to our work, Vivaldi [3], [4] is a fully decentralized coordinate system. One major difference between PCoord and Vivaldi is that in PCoord a node computes its coordinate by optimizing a loss function over a *batch* of samples, whereas in

Vivaldi a node adjusts its position to minimize the error one sample at a time.

One of the main differences between PIC [2] and PCoord is that the former uses a set of peer nodes to compute the bootstrap coordinates. In contrast, the PCoord algorithm does not require a set of peer nodes to carry out the bootstrap process. Additionally, in PIC, coordinates update at a node is completely determined by current batch of sampled distances; it does not provide mechanism to retain information learned from previous iterations. This could result in a system that reacts too quickly to current measurements.

V. Conclusions

In this paper, we have designed and evaluated a fully-decentralized coordinate system called PCoord. Our simulation results suggest that, in a 1740 nodes peer-to-peer system, PCoord can converge to a low prediction error configuration within 10 seconds, where each node performs less than ten coordinates updates using 10 reference points per update. Under a simultaneous join scenario, PCoord can converge to a low medium prediction error using half the number of samples required by Vivaldi. Additionally, we have demonstrated the following performance advantages of PCoord.

- PCoord is resilient to high fractions of corrupted samples. Our results suggest that even when 30% of the node population maliciously lie about their coordinates, PCoord is able to maintain a median system prediction error below 20 ms, which is half the prediction error of the Simple algorithm under the same fraction of malicious nodes. In general, PCoord performs close to 100% better than the Simple algorithm when the fraction of malicious nodes is less than 60% of the total system population.
- PCoord's damping and resistance mechanisms reduce the instability caused by network path anomalies, and thus enables PCoord to be more robust than Vivaldi under heavy triangle inequality violations. PCoord is able to converge to a median system prediction error below 25 ms even when the number of triangle inequality violations in the underlying network topology doubles.
- Vivaldi's performance, both in terms of convergence speed and prediction accuracy, is more sensitive to the degree of path anomalies in the underlying network topologies than PCoord. When the number of triangle inequality violations doubles in the King data set, Vivaldi's performance degradation is 60% more than that of PCoord's.
- PCoord is robust under high churn. We have shown that dynamic join and leave has little effect on PCoord's prediction accuracy when the host's mean session life time is 10 seconds or longer. This suggests that PCoord is likely to do well under the dynamic membership changes of existing peer-to-peer systems, which were reported to have a median session duration on the order of 60 minutes [18].

As part of the future work, we plan to explore PCoord's performance under dynamic network route changes, and explore possible extensions of the PCoord framework to model "distance" measurements other than Internet latencies.

References

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symp. on Operating Systems Principles (SOSP)*, Banff, Canada, October 2001.

[2] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, March 2004.

[3] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. In *Proceedings of HotNets-II*, November 2003.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of SIGCOMM'04*, August 2004.

[5] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[6] T. Hansen, J. Otero, T. Mcgregor, and H.-W. Braun. Active measurement data analysis techniques. http://amp.nlanr.net/, 2002.

[7] L. Lehman. *PCoord: A Decentralized Network Coordinate System for Internet Distance Prediction*. PhD thesis, Massachusetts Institute of Technology, 2005.

[8] L. Lehman and S. Lerman. PALM: Predicting Internet network distances using peer-to-peer measurements. Technical report, appeared in Annual Singapore-MIT Alliance Symposium, January 2004.

[9] L. Lehman and S. Lerman. PCoord: Network position estimation using peer-to-peer measurements. In *Proceedings of IEEE International Symposium on Network Computing and Applications (NCA'04)*, pages 15–24, Boston, MA, August 2004.

[10] H. Lim, J. Hou, and C.-H. Choi. Constructing Internet coordinate system based on delay measurement. In *Proceedings of Internet Measurement Conference(IMC'03)*, October 2003.

[11] T. Ng and H. Zhang. A network positioning system for the Internet. In *Proceedings of USENIX 2004 Annual Technical Conference*, pages 141–154, Boston, MA, June 2004.

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

[13] p2psim. http://www.pdos.lcs.mit.edu/p2psim/.

[14] K. P.Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop(IMW'02)*, November 2002.

[15] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, February 2003.

[16] PlanetLab. http://www.planet-lab.org.

[17] Resilient overlay networks. http://nms.lcs.mit.edu/ron/.

[18] S. Saroiu, K. P. Gummadi, and S. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems Journal*, 9(2):170–184, August 2003.

[19] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *Proceedings of IEEE INFOCOM'03*, April 2003.

[20] Y. Shavitt and T. Tankel. On the curvature of the Internet and its usage for overlay construction and distance estimation. In *Proceedings of IEEE INFOCOM'04*, April 2004.

[21] L. Tang and M. Crovella. Virtual landmarks for the Internet. In *Proceedings of Internet Measurement Conference(IMC'03)*, October 2003.

[22] W. Theilmann and K. Rothermel. Dynamic distance maps of the Internet. In *Proceedings of IEEE INFOCOM'00*, New York, June 2000.

[23] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. In *Proceedings of the First Workshop on Hot Topics in Networks (Hotnets-I)*, Princeton, NJ, October 2002.