USENIX Association

# Proceedings of the General Track: 2004 USENIX Annual Technical Conference

Boston, MA, USA
June 27–July 2, 2004

# A Network Positioning System for the Internet [*]

T. S. Eugene Ng
*Department of Computer Science*
*Rice University*

Hui Zhang
*Department of Computer Science*
*Carnegie Mellon University*

## Abstract

Network positioning has recently been demonstrated to be a viable concept to represent the network distance relationships among Internet end hosts. Several subsequent studies have examined the potential benefits of using network position in applications, and proposed alternative network positioning algorithms. In this paper, we study the problem of designing and building a network positioning system (NPS). We identify several key system-building issues such as the consistency, adaptivity and stability of host network positions over time. We propose a hierarchical network positioning architecture that maintains consistency while enabling decentralization, a set of adaptive decentralized algorithms to compute and maintain accurate, stable network positions, and finally present a prototype system deployed on PlanetLab nodes that can be used by a variety of applications. We believe our system is a viable first step to provide a network positioning capability in the Internet.

## 1 Introduction

Network distance, i.e. round-trip propagation and transmission delay[1], is a fundamental property of a network path that affects application performance. Our initial Internet measurement study [15] and subsequent measurement-based studies [22][6] have shown that by knowing the network distances from end hosts to a few, say 20, other hosts, it is feasible to characterize the end hosts' *network positions* as points in a low-dimensional Euclidean space model (say 6 dimensions) such that the Euclidean distance between two end hosts' network positions accurately predicts the actual Internet network distance in most cases. In short, network position is a feasible concept and can represent Internet network distance relationships efficiently.

Network position can be very useful for a variety of wide-area network applications. For example, a CAN overlay network [18] can use network positions as logical overlay positions and reduce the average overlay delay compared to an un-optimized structure. A wide-area service location agent can also use network positions of clients and servers to direct clients to nearby servers without measuring the paths between the clients and the servers. For bandwidth demanding applications, network positions can be a highly scalable mechanism to select a small subset of nearby nodes and then bandwidth probing techniques (e.g. [12]) can be applied to only this subset to achieve higher efficiency. Several recent studies have examined the benefits of using network position in wide-area network applications [3][9][24][5]. Alternative network position computation algorithms have also been proposed [13][16][20][22][6][5].

In this paper, we study the problem of building a network positioning system (NPS) to provide a positioning capability in the Internet. To the best of our knowledge, this is the first study to consider system-building issues in network positioning. There are two alternatives to deploy an NPS, the first is to integrate an NPS with a particular application, the second is to deploy an NPS that is shared by many applications. In this paper, we consider the latter alternative. This is similar to the deployment model of the Domain Name System (DNS) which provides a naming capability that is shared by applications in the Internet. Like many other systems work, this paper addresses a broad set of issues including identifying key design issues, system architecture design, algorithms design, implementation details, system tuning, and to a limited extent security. We make the following contributions in this paper:

- Identify key system-building issues: (1) consistency of network positions, (2) adapt positions to network topology changes, (3) maintain position stability when network topology is not changing.
- A hierarchical network positioning architecture that maintains position consistency while enabling decentralization.
- A set of adaptive decentralized algorithms to compute and maintain accurate, stable network posi-

[*] This research was sponsored by DARPA under contract number F30602-99-1-0518, by NSF under grant numbers Career Award NCR-9624979, ANI-9730105, ITR Award ANI-0085920, and ANI-9814929, and by the Texas Advanced Research Program under grant No.003604-0078-2003. Additional support was provided by Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Texas ARP, Intel, or the U.S. government.

[1] Note that network distance is a property of the network topology and routing, and is not dependent on the instantaneous network load, thus it is easier to model and predict.

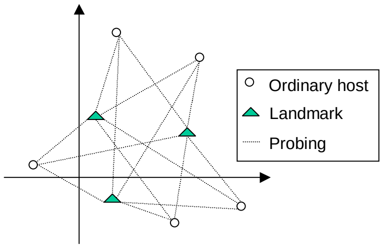

Figure 1: Basic network positioning method.

tions, even when a minority of malicious nodes may lie about their positions.

- A functional prototype deployed on PlanetLab [17] nodes that can be used by a variety of applications.

We believe our system may serve as a first step towards providing a network positioning capability in the Internet and can benefit existing and emerging applications. Our system is in a relatively early stage of development. Operational experience and application experience in the future will help us better select the system parameters and guide the evolution of the design of the system.

In the next section, we will describe the network positioning concept as originally proposed in [15] and outline the basic method for simulating network positioning in an idealized setting. We will also present some application examples. In Section 3, we present the detailed design of the system, including the architecture and algorithms. In Section 4, we describe the implementation of the system. Results from our experience with the system on PlanetLab and controlled experiments performed in a simulator are presented Section 5. We present the related work in Section 6 and finally conclude in Section 7.

## 2 Network Positioning

We begin the discussion by describing the concept of network positioning and the network positioning method proposed in [15]. The accuracy of the network positions computed by an idealized simulation of this method will serve as the target for the NPS to approximate.

Conceptually, network positioning seeks to embed the Internet network distance relationship into a geometric space (e.g. an Euclidean space). That is, each Internet host is assigned a position (a set of coordinates) in a geometric space model such that the Internet network distances among the hosts are as well approximated by the geometric distances in the model as possible.

The network distance between two hosts $\mathcal{H}_1$ and $\mathcal{H}_2$, denoted by $d_{\mathcal{H}_1\mathcal{H}_2}$, is defined as the round-trip propagation delay and transmission delay of the network path between them. Operationally, it is the minimum observable round-trip time (RTT).

A method to compute an embedding is outlined in [15]. This method works as follows. $N$ special hosts called Landmarks, denoted by $\mathcal{L}_1, .., \mathcal{L}_N$, are deployed in the Internet and the inter-Landmark distances are measured. The inter-Landmark distances are transmitted to a central node. The central node then computes a set of Landmark coordinates and return the coordinates to the Landmarks. The Landmark coordinates, denoted by $c_{\mathcal{L}_1}, .., c_{\mathcal{L}_N}$, are the result of minimizing the following objective function:

$$f_{obj1}(c_{\mathcal{L}_1}, .., c_{\mathcal{L}_N}) = \sum_{i,j\in\{1,..,N\} \mid i>j} \mathcal{E}(d_{\mathcal{L}_i\mathcal{L}_j}, \hat{d}_{\mathcal{L}_i\mathcal{L}_j}) \tag{1}$$

where $\hat{d}_{\mathcal{L}_i\mathcal{L}_j}$ is the geometric distance between $c_{\mathcal{L}_i}$ and $c_{\mathcal{L}_j}$ and $\mathcal{E}(\cdot)$ is the error measurement function:

$$\mathcal{E}(d_{\mathcal{H}_1\mathcal{H}_2}, \hat{d}_{\mathcal{H}_1\mathcal{H}_2}) = (\frac{d_{\mathcal{H}_1\mathcal{H}_2} - \hat{d}_{\mathcal{H}_1\mathcal{H}_2}}{d_{\mathcal{H}_1\mathcal{H}_2}})^2 \tag{2}$$

The simplex downhill algorithm [14] is used to solve the optimization problem. The Landmarks' coordinates define the bases for the geometric space. To embed an ordinary host $\mathcal{H}$, $\mathcal{H}$ uses the Landmarks as *reference points* and probes all the Landmarks to obtain the Landmarks' coordinates and the network distances to the Landmarks (see Figure 1). It then computes its coordinates, $c_{\mathcal{H}}$, that minimize the following objective function:

$$f_{obj2}(c_{\mathcal{H}}) = \sum_{i\in\{1,..,N\}} \mathcal{E}(d_{\mathcal{H}\mathcal{L}_i}, \hat{d}_{\mathcal{H}\mathcal{L}_i}) \tag{3}$$

An idealized simulation of this network positioning method is a centralized computation procedure that takes static network distance data as input and solves the optimization problems to determine a set of network positions. Using this method with Internet measurements, it has been shown in [15] that the Internet network distance relationship is accurately embeddable in a low-dimensional Euclidean space. In particular, in a 7-dimensional Euclidean space, 50% of the distance predictions have less than 10% error, 90% of them have less than 50% error, and the network positions can be used to accurately select nearby hosts in the Internet.

**Example Applications** Network position can be very useful in wide-area network performance optimization. We briefly present two straight-forward applications to illustrate. The first application is the CAN [18] overlay network. We compare the overlay routing efficiency in terms of end-to-end delay of a 1000-host 3-dimensional CAN on a 100-node transit-sub topology [26] when using random overlay positions versus using network positions as overlay positions. The network positions are computed with the method above using 4 randomly chosen Landmarks among the nodes. We find that when using random overlay positions, the overlay delay is on av-

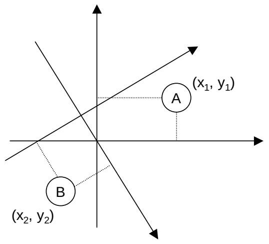

Figure 2: Host position inconsistency.

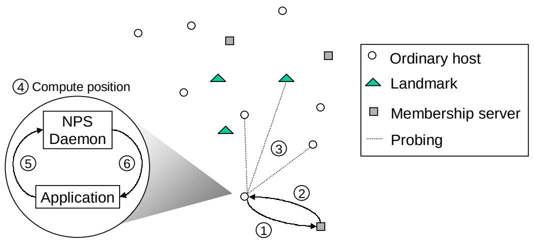

Figure 3: NPS overview.

erage 6.5 times the underlying network delay. In contrast, when using network positions as overlay positions, the overlay delay is reduced to only 2.7 times the underlying network delay on average, i.e. a factor of 2.4 improvement.

The second application is closest server selection. We use the Internet measurement data set collected by [15], which is available on-line. Following the same setup in [15], we compute 7-dimensional network positions for the hosts in the data set, we pick several non-Landmark host as clients. Then we perform 1000 experiments. In each experiment, we randomly pick 10 other non-Landmark hosts as servers and we compute how well the network positions can select the actual closest server for the clients. We find that using network positions, on average the chosen server is only 10% further away than the optimal choice. On the other hand, picking a random server out of 10 is on average 200% further than the optimal choice.

## 3 NPS Design

In this section, we describe the detailed design of the NPS. We begin by presenting a set of design issues.

### 3.1 Design Issues

**Consistency** - An important goal of a NPS is to produce consistent positions for hosts. What do we mean by consistent positions? The positions of hosts, i.e. their numerical coordinates, can be compared meaningfully only if they have the same bases. For example, in Figure 2, it is meaningless to compare the coordinates of $\mathcal{A}$ and $\mathcal{B}$ since they have different bases. Of course if $\mathcal{A}$ and $\mathcal{B}$ are aware of the relationship between their different bases, they can still correct for the difference. Thus, the positions of two hosts are said to be inconsistent if they have different bases and they are unaware of it.

**Adaptivity** - The Internet is a dynamic environment. At small time scales, network distance measurements can be affected by fluctuating queuing delays in the network. At long time scales, the Internet topology may change, thus the positions of hosts may need to be updated accordingly to reflect changes in the intrinsic network distances. A NPS should adapt to these Internet dynamics.

**Stability** - In an adaptive system, a potential problem is that network positions may fluctuate even when there is no change in the network topology, leading to poor accuracy. The stability goal is to minimize the unnecessary drifting of the positions of hosts, while properly reacting to network topology changes.

**Accuracy** - We seek to design a decentralized system that can compute positions with accuracy that closely approximates the accuracy of an idealized simulation of the method outlined in Section 2.

### 3.2 High-Level Overview

Figure 3 illustrates the high-level operations of the system. There are three kinds of nodes in the system: membership servers, Landmarks, and ordinary hosts. Membership servers are easily replicable, they store system configuration parameters and maintain soft state about some participating hosts. There can be many independent membership servers in the system and there is no need to synchronize their membership information. Landmark nodes are infrastructure nodes used to define the bases of the Euclidean space model and can serve as reference points. Landmarks bootstrap their positions using a decentralized Landmark positioning algorithm. Any host in the Internet can run a NPS daemon that maintains the host's position. When the NPS daemon is started, it first contacts a membership server (e.g. picked by DNS-round robin) to obtain a set of system configuration parameters (e.g. the dimensionality of the Euclidean space model) and a list of reference points (steps 1 and 2). Then, the NPS daemon begins probing the reference points. These probes serve the purpose of obtaining the latest positions of the reference points and the network distances to the reference points (step 3). Based on this information, the NPS daemon computes the position for the host (step 4). Step 3 and 4 are repeated until the position of the host is stabilized. At any given time, an application can query the NPS daemon to obtain the host's current position instantaneously (steps 5 and 6). The exception is when the host has just been booted-up

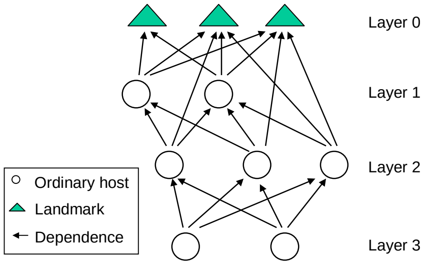

Figure 4: Hierarchical network positioning architecture.

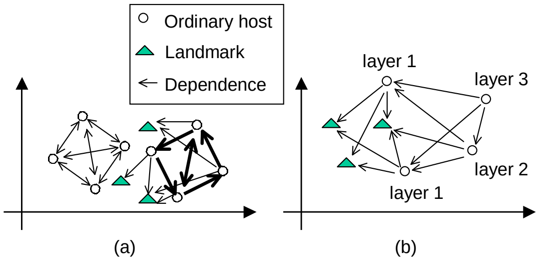

Figure 5: Host dependency in 2D Euclidean embedding. (a) Inconsistent. (b) Hierarchical, consistent.

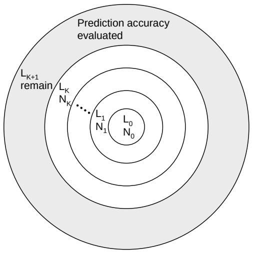

Figure 6: Logical host partitioning in layering experiment.

and there exists no prior cached position information for the host. Landmarks and hosts periodically re-compute their positions to adapt to network topology changes.

## 3.3 Hierarchical Architecture

Position consistency is an important goal. To ensure that all host positions are consistent and have the same bases, a straight-forward solution is to use the positions of a set of Landmark hosts to define the bases and require all hosts to use the Landmarks as their reference points. This is the method outlined in Section 2. Although this approach guarantees consistency, it has some undesirable properties. As illustrated in Figure 1, *every* ordinary host probes *all* Landmarks to compute its coordinates. In such a system, the Landmarks and their network access links become performance bottlenecks, and there is no obvious solution to recover gracefully from Landmark failures.

Thus, we seek a decentralized solution that can at the same time maintain consistency. Our design is a hierarchical architecture. In this architecture, Landmarks still define the bases and can serve as reference points for hosts in the system. The main departure from the basic approach is that any host that has determined its position can be chosen by the membership server to be a reference point for other hosts. As a result, Landmarks become much less critical, and temporary Landmark failures will not halt the entire system. Currently, the membership server randomly chooses among the eligible hosts as reference points when the Landmarks are too heavily loaded or unavailable. However, to ensure consistency, we impose a hierarchical position dependency among the hosts. Figure 4 illustrates the hierarchical architecture. We say that host $\mathcal{A}$ *depends* on host $\mathcal{B}$ if $\mathcal{A}$ uses $\mathcal{B}$ as one of its reference points. We also define a notion called *layer number*. The layer number of a host is the maximum number of dependency hops separating it from the Landmarks. For convenience, the Landmarks are given layer number 0. We call a system that allows a highest layer number $L$ an "$L+1$ layer system".

By imposing a position dependency hierarchy, we ensure that the positions of all hosts have the same bases defined by the Landmarks. Note that if the position dependency is arbitrary, there can be a danger of losing consistency. In general, any dependency cycle can prevent hosts' positions from converging since position updates are propagated through the cycle. An example is shown in Figure 5(a), where one cluster of hosts is partitioned, and hosts have cyclic dependencies. Thus the positions of the two clusters of hosts do not have the same bases, and some hosts' positions may not converge.

The hierarchical dependency invariant is easy to maintain in the system by checking reference points' layer numbers. Landmarks always have layer number 0. When a NPS daemon starts, it initializes its layer number to be the highest layer number allowed in the system, $L_{max}$ (a static parameter retrieved from the membership server). To maintain the invariant, each host $\mathcal{H}$ with current layer number $L_{\mathcal{H}}$ simply refuses to depend on any reference point $\mathcal{R}$ that has a layer number $L_{\mathcal{R}} \geq L_{\mathcal{H}}$. When $\mathcal{H}$ accepts a set of reference points $\mathcal{R}_i$, $\mathcal{H}$ updates its layer number to be $\max_i(L_{\mathcal{R}_i}) + 1$. Figure 5(b) shows a 2-dimensional example.

### 3.3.1 How Many Layers?

An important question to ask is, how many layers are needed and how is accuracy affected? To shed light on these questions, we have performed idealized simulations on Internet-like synthetic topologies. Figure 6 illustrates the methodology. Given a set of hosts, we partition them into $K+2$ different layers. Layer $L_0$ contains $N_0$

| $N_{1..K} = 500$ | $K = 0$ | $K = 1$ | $K = 2$ | $K = 3$ | $K = 4$ |
|---|---|---|---|---|---|
| Inet 3050 | .42 | .46 | .87 | .76 | 1.01 |
| Inet 6000 | .35 | .41 | .69 | .66 | .83 |
| PLRG 3000 | .32 | .44 | .57 | .61 | .68 |
| PLRG 6000 | .35 | .48 | .81 | .72 | .97 |

Table 1: 90 percentile relative error ($N_0 = 40, N_{1..K} = 500$, varying $K$).

| $K = 1$ | $N_1 = 500$ | $= 1000$ | $= 1500$ | $= 2000$ |
|---|---|---|---|---|
| Inet 3050 | .46 | .45 | .44 | .44 |
| Inet 6000 | .41 | .41 | .42 | .41 |
| PLRG 3000 | .44 | .44 | .43 | .43 |
| PLRG 6000 | .48 | .47 | .46 | .46 |

Table 2: 90 percentile relative error ($N_0 = 40, K = 1$, varying $N_1$).

Landmarks. For $i = 1, .., K$, $N_i$ randomly chosen hosts are placed into layer $L_i$. The remaining hosts are placed in layer $L_{K+1}$. In each experiment, for $i = 1, .., K + 1$, each host in layer $L_i$ randomly picks $N_0$ hosts in layer $L_i - 1$ as its reference points. After every host has been embedded, we evaluate the relative error of the distance predictions among the hosts in layer $L_{K+1}$. The relative error of the distance prediction between a pair of hosts is defined as:

$$relative\ error = \frac{|actual\ -\ predicted|}{min(actual, predicted)} \qquad (4)$$

By varying $K$ and $N_i$, we can study the impact of adding layers or changing the layer size on embedding accuracy. Note that these experiments are centrally simulated under static conditions, no network dynamics are introduced. We conduct these experiments on 4 different synthetic power-law networks. Two are generated by the Inet 3.0 generator [23] (3050 and 6000 nodes), two are generated based on the PLRG model [2] (using the largest connected component, roughly 3000 and 6000 nodes). As usual, nodes are randomly placed in a square region, and the delay of a link is the Euclidean distance between the end points of that link. End-to-end delay is the shortest path delay. For each network, we choose 5 random sets of 40 Landmarks that are well-separated. For each configuration, we run 5 experiments, one per each set of Landmarks. For embedding, an 8-dimensional Euclidean space is used. The resulting relative error among hosts in the outer-most layer is computed and the 90 percentile is reported.

Table 1 and Table 2 show the results. Although we present only the 90 percentile relative errors, note that the the 50 percentile relative error is less than 0.2 for all experiments. From Table 1, we see that with $K = 1$, the relative error is only slightly higher than $K = 0$ when every host uses the Landmarks as reference points. However, allowing more layers inflate the relative error, a sign of error accumulation. Concentrating on $K = 1$, from Table 2, we see that the relative error appears to be independent of the size of layer 1.

Interestingly, we observe that the relative error of the distance predictions between hosts in layer 1 and layer 2 is better than that among layer 2 hosts. For $K = 1$ and $N_1 = 2000$, the 90 percentile relative error of the distance predictions between layer 1 and layer 2 hosts ranges from only 0.23 to 0.28. Thus, prediction accuracy across layers is still maintained.

#### 3.3.2 The 3-Layer System

Based on our findings, we believe a 3-layer system ($K = 1$) is very promising. First of all, accuracy is only slightly affected, and secondly under reasonable assumptions and based on our practical experience with the system, a 3-layer system can already reach an enormous membership size. It is important to note that only a few RTT measurements are needed to perform a position computation, thus the bandwidth overhead is very low. Assuming a 12-hour re-computation cycle, each host in our system conservatively generates less than 0.5bps of measurement traffic to each reference point. Assuming layer 0 Landmarks have 1Mbps of dedicated bandwidth to support layer 1 hosts, layer 1 hosts dedicate 10kbps to support layer 2 hosts, and 20 reference nodes are used by each node, layer 1 can reach 2 million nodes, and layer 2 can already reach 2 billion nodes!

## 3.4 Ordinary Host Positioning

The ordinary host positioning method at a high level follows the one described in Section 2 except of course the reference points used by a host are not necessarily the Landmarks. However, several types of dynamics need to be considered when positioning a host in the system: (1) Variable queuing delay in a network path makes it unclear how many RTT samples are needed to decide the network distance (i.e. minimum RTT) has been measured. (2) When a host is updating its position, its reference points could also be simultaneously updating their positions. (3) Network topology change can increase the network distance of a path, so always keeping the minimum RTT will not detect this kind of topology change.

To automatically adapt to (1) and (2), we employ an iterative positioning procedure that terminates when a stability heuristic condition is met. More precisely, in each iteration, a host only requires one probe exchange with each of its reference points. This provides the host with one additional RTT sample for each reference point and the latest position of each reference point. The newest minimum RTT samples are used to update the network distances. Based on the latest network distances and reference point positions, a new embedding position is computed by minimizing Eq. 3. A random wait time of up to one second is introduced between each iteration. The heuristic is that if in three consecutive iterations the position of the host has moved by less than one millisecond in the Euclidean space, then the procedure is terminated and stability is declared. In our experience, by initially setting the position of a host at the origin, stability can typically be reached in under 15 seconds, including time spent in pacing network probes. To adapt to topology changes, re-positioning of a host starting from its current position will be performed periodically.

To address (3) during re-positioning, the strategy is to reconsider the current known minimum RTT of a path based on a set of new RTT measurements. When a host begins the iterative procedure to re-compute its position, it has the previously known minimum RTT $d_{old}$ to each reference point. The host first obtains 10 new RTT samples to each reference point. Supposed the minimum of these 10 new samples is $d_{new}$. If $d_{new}$ differs from $d_{old}$ by less than one percent, then we use $d_{old}$ as the initial network distance and begin the iterative procedure. Otherwise, we use $d_{new}$. As a result, if the network distance is increased by less than one percent, it will be ignored. A decrease in network distance can be discovered eventually in the iterative procedure.

## 3.5 Decentralized Landmark Positioning

In the basic network positioning method described in Section 2, Landmarks measure their inter-Landmark distances and ship all the data to a central node to compute Landmarks' positions in an Euclidean space that minimize $f_{obj1}(\cdot)$ (Eq.(1)). In contrast, NPS implements a decentralized method. Observe that $2 \times f_{obj1}(\cdot)$ can be expanded as:

$$\begin{aligned}2 \times f_{obj1}(c_{\mathcal{L}_1}, .., c_{\mathcal{L}_N}) \quad = \quad & \sum_{i \neq 1} \mathcal{E}(d_{\mathcal{L}_i\mathcal{L}_1}, \hat{d}_{\mathcal{L}_i\mathcal{L}_1}) + \\ & \sum_{i \neq 2} \mathcal{E}(d_{\mathcal{L}_i\mathcal{L}_2}, \hat{d}_{\mathcal{L}_i\mathcal{L}_2}) + \\ & \vdots \\ & \sum_{i \neq N} \mathcal{E}(d_{\mathcal{L}_i\mathcal{L}_N}, \hat{d}_{\mathcal{L}_i\mathcal{L}_N})\end{aligned}$$

Thus, the objective function can be decomposed into $N$ terms, each term corresponds to how one Landmark relate to the others. We implement a decentralized strategy which in essence randomly chooses one term from the above equation, for example the term for $\mathcal{L}_2$, and computes the corresponding host's position that locally minimizes that chosen term, and then repeats until no further progress can be made.

More specifically, all Landmarks upon boot-up are initially positioned at the origin of the Euclidean space. Each Landmark then uses all the other Landmarks as its reference points, probes the other Landmarks, retrieves their latest positions and obtains network distance samples, and computes a new position for itself. These steps are in fact very similar to what an ordinary host does when computing its position. A random pause time of up to one second is introduced after each computation step. Then the steps repeat until a convergence heuristic criterion is met. The criterion again is that convergence is achieved when in 3 consecutive iterations a Landmark's position has not moved by more than one millisecond in the Euclidean space. Note that all Landmarks need to roughly simultaneously begin re-considering their positions. We do this by triggering a Landmark to begin re-considering its position when it is probed by another Landmark, a signal that the other Landmark is reconsidering its position. To make it harder for a spoofed Landmark to trigger computation, Landmarks exchange random 32-bit authorization IDs when they probe each other and a node must send the correct authorization ID to a Landmark to trigger it. In our experience, this approach can embed 20 Landmarks starting from their origin positions in approximately one minute and the resulting positions are just as accurate as the centralized approach. Again, to adapt to topology changes, the Landmarks also need to re-compute their positions periodically. If the network change is minimal, convergence can be achieved in a small number of iterations.

### 3.5.1 Maintaining Stability

The Landmarks' positions define the bases of the Euclidean space which all other hosts share. If the network topology is changed, then the Landmarks' position should be updated accordingly to reflect the change. However, if the network topology has not changed, it is important to minimize unnecessary drifting of the Landmarks' positions when they are re-computed. To increase the stability of the Landmarks' positions, we apply the following mechanisms.

First, to get good measures of the current network distances, before any position re-computation, a Landmark probes every other Landmark 50 times. In addition, piggybacked in these probes, the Landmarks conduct bi-directional exchanges of their current known minimum RTT to synchronize. This is important because two Landmarks must agree on the same network distance between them to reduce oscillations in positions. Once the probings are done, the new minimum RTTs are compared to the previous known minimum RTTs. Again if the change is less than one percent, the old value is kept as the initial network distance. Finally, if the network distances from a Landmark to all other Landmarks have not been changed by more than one percent, the Landmark keeps its old position.

### 3.6 Congestion Control

Probing from ordinary hosts are congestion controlled in the system. This is to prevent overwhelming a reference point if too many hosts are simultaneously probing it. To implement congestion control, each probe packet carries a sequence number. By observing the sequence numbers in the probe reply packets, we can infer packet losses in the same manner as TCP. The initial probing rate is one probe per second in the implementation. An additive-increase-multiplicative-decrease (AIMD) [25] rate adjustment procedure is applied when probe reply packets are received or lost. The increase and decrease factors can be tuned to be less aggressive than TCP if desired, but this tuning is out of the scope of this paper. Probing rate is also upper bounded by a constant (the implementation uses 10 probes per second as maximum rate).

Note that probing between Landmarks are sent at a constant rate (10 probes per second) and no congestion control is applied. This is to ensure no Landmark will fall far behind in the position computation process.

### 3.7 Triggered Re-Positioning

Although hosts periodically re-compute their positions, the setting of the re-computation interval represents a trade-off between the load on the system and the accuracy of the hosts' positions. To be conservative, by default a host re-computes its position only once every 12 hours in the current system. However, we use triggered re-positioning to improve the accuracy and consistency of the system when network topology changes. When a reference point has undergone a drastic change in position indicating a large change in network topology, the reference point can trigger dependent hosts to re-compute their positions. The aggressiveness in triggering dependent hosts is controlled by the reference point based on the load it can tolerate. To ensure only a reference point of a host can trigger the host to re-compute its position, we do the following. Before beginning computing a new position, a host randomly chooses a 32-bit authorization ID. This ID is included in probe packets sent to the reference points. A reference point records the authorization ID for each dependent host. In the current system, when a reference point's position has moved by over 10 milliseconds in the Euclidean space, it triggers a dependent host to re-compute by sending the corresponding authorization ID to the dependent host. The dependent host re-computes its position if the authorization ID is valid.

Note that the Landmarks are never triggered by ordinary hosts in the system. They periodically re-compute their positions. The current period used in the implementation for Landmarks is 3 hours.

### 3.8 Detecting Malicious Reference Points

In our system, Landmark nodes are trusted entities, but ordinary hosts in the system that serve as reference points for other hosts could potentially lie to their dependents if they are malicious. They could lie about their actual positions and/or inflate the network distances by holding onto probe packets. We separately discuss two types of lies: (1) continuously changing lies, (2) fixed lies.

First, the damage of continuously changing lies is limited. This is because a reference point that continuously changes its position can easily be noticed and be eliminated by the dependent. We do this by putting a time limit on each reference point within which the reference point's position must stabilize or else it will be removed. A time limit is also put on the stabilization of the network distance (i.e. minimum RTT) to a reference point.

The more damaging lie is the fixed type in which a malicious reference point consistently reports the same false position and/or inflated network distance to each dependent. Such a malicious reference point would appear to be normal to a dependent and it would be difficult to find a mechanism that could 100% accurately identify such a lying reference point. One approach to coping with this type of lie is to eliminate a reference point if it fits poorly in the Euclidean space compared to the other reference points. The specific procedure in the system is as follows. Assume there are $N$ reference points $\mathcal{R}_i$ with positions $P_{\mathcal{R}_i}$, and the network distances from a host $\mathcal{H}$ to them are $D_{\mathcal{R}_i}$. After $\mathcal{H}$ computes a position $P_{\mathcal{H}}$ based on these $N$ reference points, for each $\mathcal{R}_i$, it computes the fitting error $E_{\mathcal{R}_i}$ as $\frac{|distance(P_{\mathcal{H}},P_{\mathcal{R}_i})-D_{\mathcal{R}_i}|}{D_{\mathcal{R}_i}}$. Then we decide whether to eliminate the reference point with the largest $E_{\mathcal{R}_i}$. The current criterion is that if $max_i(E_{\mathcal{R}_i}) > 0.01$ and $max_i(E_{\mathcal{R}_i}) > C \times median_i(E_{\mathcal{R}_i})$, where $C$ is a constant, then the reference point with $max_i(E_{\mathcal{R}_i})$ is eliminated. That is, a reference node is rejected if it has fitting error significantly larger than the median error among all reference nodes. We use median error as the criterion since it is more robust than the average error and cannot be easily affected by a minority of malicious nodes. In Section 5, we explore the choice of the constant $C$.

## 4 NPS Implementation

The system is implemented on the Linux platform. It includes two components: (1) the NPS membership server, (2) the NPS daemon. The membership server's main tasks are to provide basic system configuration information and to serve as a rendezvous point for NPS daemon coordination. Since the membership server does not need to synchronize dynamically changing information, it can be replicated easily. All network communications in the system are implemented with UDP datagrams.

### 4.1 NPS Membership Server

The key functions of the membership server are to (1) identify the Landmarks, (2) provide initial configuration parameters to NPS daemons (i.e. whether the daemon is a Landmark, the number of reference points a daemon should use, the geometric space used for embedding, and the maximum number of layers allowed in the system), (3) maintain a subset of the current hosts that can potentially serve as reference points, (4) hand out potential reference points when requested, and (5) control the number of hosts in each layer. The membership server is an event-driven process. The global parameters are read from a text configuration file.

When a NPS daemon is started, it first asks the membership server for a set of initial configuration parameters (i.e. item (2) above). If the NPS daemon is identified as a Landmark, the number of reference points to use is one less than the number of Landmarks since the other Landmarks are its reference points. Otherwise, the number of reference points to use is the number of Landmarks.[2].

When a NPS daemon needs reference points to perform an embedding computation, it contacts the membership server. If the NPS daemon is a Landmark, the other Landmarks are always returned as the reference points. Otherwise, the membership server has some flexibility in giving out reference points. If the NPS daemon reports it is currently in layer $L$, the membership server will avoid giving out any reference points in layer $L$ or larger because those reference points will be rejected. Note that a NPS daemon assumes it is in the highest layer allowed in the system when it starts. Note also that a NPS daemon will verify the layer number of its reference points during probing and thus the layer number information maintained by the membership server does not need to be precise. The membership server can also select reference points to roughly adjust the number of hosts in each layer. For example, it can maintain a limited number of hosts in layer 1 to ensure that the Landmarks are not overloaded by measurement traffic. Once that threshold is reached, it can direct other hosts to layer 2 or higher by giving them reference points in layer 1 or above.

When a NPS daemon has successfully computed a stable embedding position, it reports to the membership server its stable status and its current layer number. This information is kept as soft state at the membership server. Stale entries are periodically flushed from memory. The network load on the membership server is quite minimal. In the experiments, a newly started NPS daemon altogether sends and receives less than 600 bytes of data between itself and the membership server to boot up, locate reference points, and report its status. If necessary, multiple membership servers can be deployed.

### 4.2 NPS Daemon

The purpose of the NPS daemon is to compute and maintain the embedding position of the host which may be a Landmark or an ordinary host, and cope with the dynamics in the network or the system. The NPS daemon consists of two processes. The parent process implements the NPS logics and protocols, and the child process is a computation engine that solves the optimization problems in embedding computations using the simplex downhill algorithm [14]. The two processes are both event-driven and communicate via a full duplex pipe. The concurrency is desirable because we want the parent NPS process to remain responsive to various events while an optimization problem is being solved (which could last for a few milliseconds).

Once a NPS daemon has configured itself with parameters obtained from the membership server, it begins to compute and maintain its embedding position. At the beginning of an embedding computation cycle, a NPS daemon first declares itself unstable (the precise definition of stability is discussed in Section 3.4). It then makes sure it knows a sufficient number of valid reference points, if not, it asks the membership server for more. The network distance between itself and each reference point is measured by timing the delay between the sending of a probe message and the receiving of a reply from the reference point. When a reference point replies to a probe message, it piggybacks its current coordinates and its current layer number onto the message. As soon as a network distance estimate is obtained for each valid reference point, the NPS daemon performs the embedding computation. This process is repeated until the stability criterion is met. The stable coordinates are stored on disk so that they can be retrieved after a daemon restart. The current layer number is then reported to the membership server, and a re-computation is scheduled. This completes one embedding computation cycle. If 12 probes are sent from an ordinary host to a reference point without any reply, the reference point is removed and a new one is obtained from the membership server. Sequences of probes to different reference points are started at least 10ms apart. The selection of these constants are not significant, they just seem reasonable.

[2] It is possible to use fewer reference points, but this variation is not considered in this study.

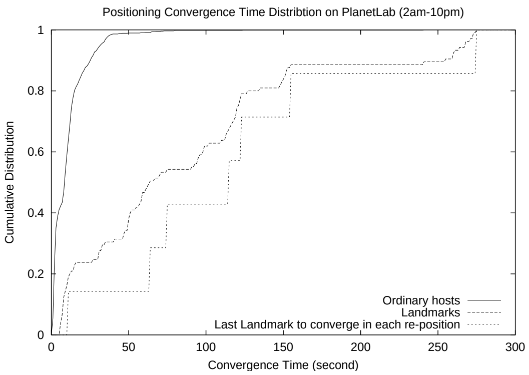

Figure 7: Positioning convergence time distributions.

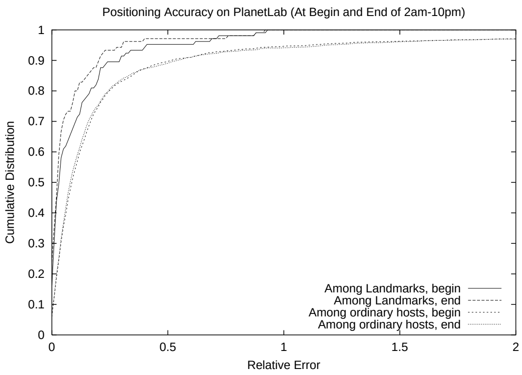

Figure 8: Positioning accuracy on PlanetLab.

### 4.2.1 NPS Application Programming Interface

The NPS daemon currently supports a simple application programming interface for network position queries. An application can query the NPS daemon by sending it a UDP query message. The NPS daemon replies with the dimensionality of the Euclidean space and current numerical coordinates of the host in a UDP reply message. Currently the coordinates are represented as 32-bit integers in unit of microsecond.

### 4.2.2 Working with NAT Hosts

Network Address Translation (NAT) [21] is becoming an increasingly ubiquitous solution to incrementally scale up the size of the Internet. However, a problem with NAT is that it can prevent in-bound data connections. The system has been carefully designed to work with NAT. First, the system uses only UDP datagrams for communication, and a NPS daemon is identified at a membership server by the IP address and UDP port number carried in the messages it sends. Thus, NPS daemons behind NAT can be uniquely identified by the membership server. In addition, the messages between a NPS daemon behind NAT and the membership server would establish forwarding state in the NPS daemon host's NAT gateway. Under typical implementations of NAT, another NPS daemon can communicate with the NPS daemon behind NAT using this same NAT forwarding state. Thus the system can work around the in-bound connection problem.

# 5 Experiments

In this section, we present experience with the system on PlanetLab. We also present results from controlled experiments using a simulator.

## 5.1 PlanetLab Experience

The NPS system is operational on PlanetLab. Here, we report our experience from a particular 20-hour period of operation of the system on a weekday from 2am to 10pm. Note that the PlanetLab is a shared test-bed.

We use 127 PlanetLab nodes. We first collect the 127 × 127 network distances using ping (100 RTT samples per path), then we apply clustering to select 15 well-distributed nodes as Landmarks. The remaining 112 hosts serve as ordinary hosts. A membership server is set up at our institution. The Euclidean space is configured to have 8 dimensions. All the ordinary hosts use the Landmarks as reference points unless Landmarks are down. The NPS daemons on Landmarks are started 5 minutes prior to the starts of ordinary hosts. Landmarks update their positions once every 3 hours. Ordinary hosts by default update their positions once every 12 hours, but can also be triggered by a Landmark to update if the Landmark has moved its position by more than 10 milliseconds. Probing congestion control is also enabled.

### 5.1.1 Convergence Time

Figure 7 shows the positioning convergence time distributions during this 20-hour period. The convergence time is measured starting from the first probe to any reference point is sent by a host until the host's position has not changed by more than one millisecond over 3 consecutive computation iterations. Note that most of this time is spent in pacing network probes to reference points. First, consider the distribution for ordinary hosts, we can see that updating the position of an ordinary host takes less than 15 seconds in 80% of the cases. As expected, however, the convergence times for Landmarks are generally longer since Landmarks simultaneously updates their positions and need more time to agree on their positions distributedly. We have also computed the time it takes the last Landmark to converge in each of the 7 update rounds during the 20-hour period. We can see that in most rounds, all Landmarks converge in less than 160 seconds. In a better provisioned environment, we expect the convergence times to be reduced.

The convergence times we observe are far shorter than the time scale at which the Internet topology changes, which is typically on the order of a day [27], thus the system is sufficiently agile. When the NPS daemon is run as a background process, the only occasion an application asking for the position of a host needs to wait for the NPS daemon to converge is when the NPS daemon is initially started without any prior position information. In most cases, applications can obtain the position information instantaneously.

### 5.1.2 Position Accuracy Over Time

Figure 8 summarizes the accuracy of the system on PlanetLab. The accuracy metric is relative error (Eq.(4)). We have plotted the cumulative distribution of relative error for different classes of hosts near the beginning of the 20-hour period and near the end of the 20-hour period. First, since Landmarks directly use the inter-Landmark distances in computing positions, not surprisingly, the resulting Landmark positions approximate the distances among Landmarks well, achieving a 90 percentile relative error of roughly 0.25. Comparing the accuracy of the Landmarks' positions at the beginning and end of the 20-hour period, we see that the level of accuracy is maintained by the system without degradation. This is an important validation of the system's ability to maintain position accuracy over time in a dynamic environment.

For the ordinary hosts, we consider only the position accuracy for the inter-ordinary-host distances. Note that these distances are not used in computing the ordinary hosts' positions. The level of accuracy achieved by the system is very good with a 50 percentile relative error of 0.08 and a 90 percentile relative error of 0.52. This accuracy distribution is very similar to previous network measurement-based results reported in [15] and [22]. The point that we wish to again emphasize is that the level of accuracy is maintained by the system over time without degradation.

## 5.2 Controlled Experiments

To explore specific aspects of the system, we perform controlled experiments of the system in a simulator. This allows us to quantify the impact of various designs of the system under different scenarios, and to allow us to compare how close the system's accuracy is to the target accuracy of an idealized simulation.

Our network simulator is event-driven. It implements a set of system interfaces to interact with NPS components. The important interfaces are *get_current_time()*, *set_timer()*, *unset_timer()*, *send_message()*, and *solve_for_coordinates()* to compute the coordinates for the requester. Only one computation engine exists in the simulator, the real computation time is recorded and is used to simulate the computation delay. The NPS daemon and membership server provide an *event_handler()* interface to receive either timer events, messages, or computation results.

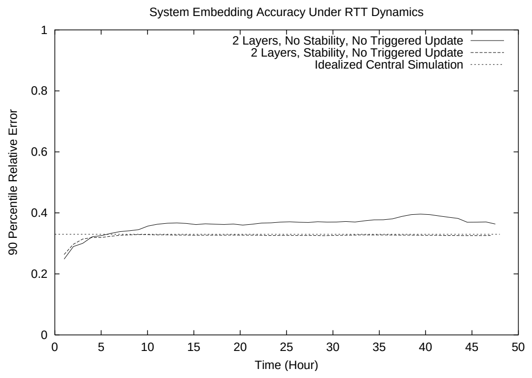

Figure 9: 2 layer system embedding accuracy under RTT dynamics.

The simulator is capable of simulating system dynamics. The underlying network topology can be changed during simulation. It can also simulate variations in the network round-trip times. We do this by adding to a network delay $d$ some exponentially distributed noise with a mean $m$, and we choose $0 < m \leq d$. Our goal is only to observe how different aspects of the system function under noise, and we do not claim that noise generated by this method is representative of the noise on the Internet. It has been observed that Internet RTT measurements are often fairly close to the minimum RTT [1]. In the simulator, 12% of the paths are given the worst case variation where we set $m = d$. Intuitively, in this case only one in 100 RTT measurements is within 1% of the true minimum. For 50% of the paths, $m > 0.05d$, and for 10% of the paths, $m < 0.005d$.

We use a 1044-node PLRG [2] topology for the simulations. As usual, nodes are randomly placed in a square region, and the delay of a link is the Euclidean distance between the end points of that link. End-to-end delay is the shortest path delay. First, a set of 20 well-separated Landmarks are randomly chosen. We use an 8-dimensional Euclidean space for the embedding. To generate an ideal accuracy target, we conduct an idealized simulation of networking positioning. We use the 90 percentile relative error as a point to compare between different experiments. We find that using the 20 Landmarks as reference points for the entire system, the 90 percentile relative error achieved is 0.33. This can be considered a rough lower bound for the 90 percentile relative error for the system. The goal will be to achieve an accuracy as close to this possible. In the system simulations, the Landmarks recompute their positions once every 3 hours.

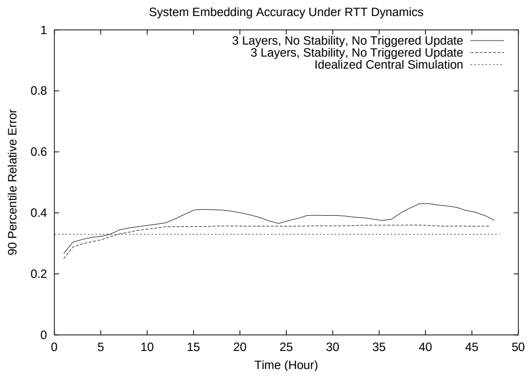

Figure 10: 3 layer system embedding accuracy under RTT dynamics.

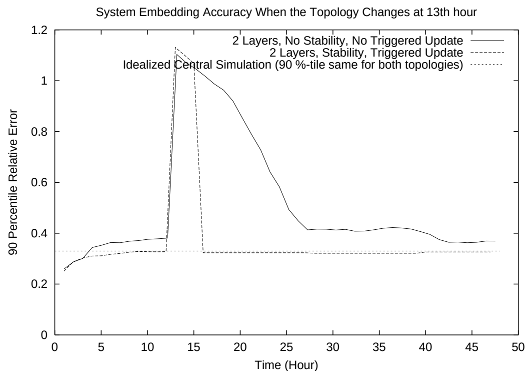

Figure 11: 2 layer system embedding accuracy under drastic topology change.

The ordinary hosts, on the other hand recompute their positions once every 12 hours. Triggered re-positioning are turned on in some experiments. The Landmarks recompute more frequently so that any topology change can be fairly promptly detected by the Landmarks. The experiments simulate the life of the system for 48 hours. The Landmarks begin the process of computing their positions at time zero. Unless otherwise stated, the 1024 ordinary hosts join the system in random order, starting at the 10 minute mark, with an exponentially distributed inter-arrival time chosen such that the last host joins the system at roughly the 12 hour mark. This spreading is done to expose the embedding accuracy of the system under asynchronous host computations. To summarize the performance of the system over time, at every hour in the simulation, we compute the 90 percentile relative error for all the network distances among hosts that are in the system at the moment.

### 5.2.1 System Accuracy and Effectiveness of Stability Control

We first present the result of an experiment under RTT dynamics (no real topology change) where all hosts use the Landmarks as their reference points (i.e. a 2 layer system). In this experiment, the triggered re-positioning mechanism is turned off. The stability control mechanisms for Landmarks is experimented with. The results are shown in Figure 9. The accuracy of the idealized simulation is also plotted as a baseline for comparison.

First, note that without the stability control mechanisms described in Section 3.5.1, we can clearly see that the accuracy of the system is adversely affected by the drifting of Landmarks' positions. The problem is that as the Landmarks' positions drift every 3 hours, different ordinary hosts are computing their positions based on different sets of Landmark positions without knowing it, which leads to a noticeably higher position error. With the stability control mechanisms, the system's accuracy is stable throughout the 48 hour period. Second, observe that the system's accuracy is matching that of the idealized simulation. The lower error at the beginning is because hosts are still joining in the dynamic system. Thus, the system achieves the basic objective of matching the performance of an idealized simulation.

In the simulated environment, an ordinary host on average takes 14.8 seconds to converge to a position. The last Landmark to converge in each round takes on average 41.2 seconds. These figures are not so different from those we see on PlanetLab.

Figure 10 shows the result when only 300 hosts are allowed to be at layer 1 and the other 724 hosts are forced to be at layer 2 and use layer 1 hosts as reference points (i.e. a 3 layer system). As expected, there is a small accuracy penalty over the 2 layer case. However, what is gained here is a much larger potential system size and robustness since hosts do not have to use Landmarks as reference points. We also observe that the impact of not having the stability mechanisms is magnified by the decentralization (compare to Figure 9). With the stability mechanisms, the accuracy of the system remains stable.

### 5.2.2 Effectiveness of Triggered Re-Positioning

In the next set of experiments, we keep the settings the same as before, except that at the 13th hour into the simulation, we replace the underlying network topology with a different PLRG. This is done to simulate a drastic and instantaneous change in the network topology. What we want to show is that the NPS system can gracefully adapt to such catastrophic network change and eventually converge to a consistent global embedding. Coincidentally, the 90 percentile relative error achievable in an idealized simulation for the second topology is also 0.33. Figure 11 shows the result for a 2 layer system.

First, notice the sharp jump in relative error at the 13th

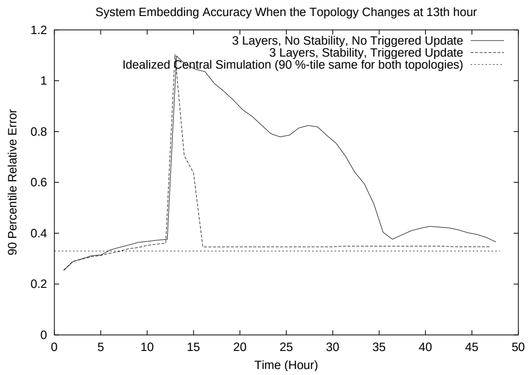

Figure 12: 3 layer system embedding accuracy under drastic topology change.

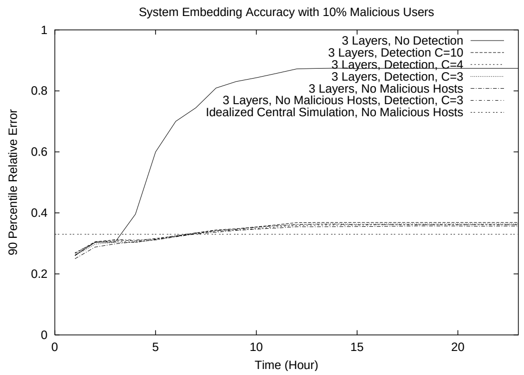

Figure 13: 3 layer system accuracy with 10% lying hosts.

hour due to the topology change. Over time, the system is able to detect the topology change and adapt. We can also see the benefits of triggered re-positioning and the stability mechanisms. Without either of these features, the system adapts but at a slow pace since hosts do not recompute their positions for 12 hours, and the system accuracy still fluctuates after 12 hours due to instability. With both features, we see that after the 15th hour when Landmarks have recomputed their positions, other hosts are triggered to update their positions. By the 16th hour, all hosts have reached their new positions, the accuracy of the system is stable and matches that of the ideal case.

Figure 12 presents the result for the same network topology change scenario but with a 3 layer system where 300 hosts are in layer 1. First, consider the case with no triggered re-positioning and stability mechanisms. Because in the 3 layer system there is one level of indirection between the majority of the hosts and the Landmarks, it takes roughly one 12 hour period for all the layer 1 hosts to all move to their new positions, and roughly another 12 hour period for all the layer 2 hosts to move to their new positions relative to the layer 1 hosts. In contrast, with the stability and triggered update features, convergence to good performance happens quickly even with a layer of indirection. The layer 1 hosts are able to inform layer 2 hosts of the network topology change and within 1 hour after the Landmarks detect the topology change, all hosts have converged to their new positions with high and stable accuracy.

### 5.2.3 Effectiveness of Malicious Reference Node Detection

To see the effects of lying reference points, we conduct several experiments with a 3 layer system with 300 hosts in layer 1. First 10% of the ordinary hosts are designated to be malicious. Each of them randomly chooses a number between 0 and 1000. It then consistently lies about its position and inflate network distances by that amount (in unit of millisecond). In Figure 13, we show the effect of these malicious hosts to the system in a 24 hour simulation period. When the malicious reference node detection mechanism is off, notice the sharp climb in relative error when layer 2 hosts begin to join the system. Without any malicious reference node detection mechanism, the accuracy of the system is destroyed by the lying layer 1 hosts. Fortunately, the malicious reference node detection mechanism is able to dramatically restore performance to nearly the same level as the "3 layers, no malicious hosts" case. That is, the mechanism is highly effective at detecting malicious hosts. We have experimented with several different sensitivity constant $C$ values to tune the aggressiveness of the mechanism and found that a conservative value of 4 is sufficient. Moreover, when there is no malicious hosts in the system, the malicious reference node detection mechanism only slightly affects the system's overall accuracy.

## 6 Related Work

This work is aimed at building a networking positioning system that can be used by Internet applications. As such, this work deals broadly with issues including system architecture, algorithms design, implementation details, system tuning, and to a limited extent security. To the best of our knowledge, this is the first study to consider system-building issues in network positioning. We have identified several key system issues such as position consistency, stability, and adaptivity that are important in a practical system. While this system is being developed, many parallel work has explored a variety of issues in network positioning.

In [20], an optimization method called Big-Bang-Simulation (BBS) is proposed. This method simulates the error in an embedding as force fields. It uses a multi-phase procedure to reduce the error in the embedding iteratively. It has been shown that this method is computationally efficient and has slightly better relative error performance than the downhill simplex minimization algorithm used in our system. A distributed version of BBS can potentially be developed and used in NPS.

In [16], a distributed network positioning architecture called Lighthouse and a different positioning mathematical framework based on the Gram-Schmidt process rather than multi-dimensional optimization are proposed. In the Lighthouse architecture random hosts with computed positions are used to serve as reference points in the system to enhance scalability, and hosts that use different reference points as bases correct for the difference by using a position translation matrix. The PIC [5] study in parallel proposes an architecture similar to that of Lighthouse, but uses a multi-dimensional optimization framework for its robustness. In addition, the PIC study explores reference point selection algorithms, as well as proposes a triangle inequality based criterion for detecting malicious reference nodes. Interestingly, the PIC malicious reference nodes detection mechanism works somewhat similarly to our embedding error based approach. Comparing these distributed architectures to the hierarchical architecture in NPS, the position dependency structure is not controlled in Lighthouse or PIC, thus consistency becomes a concern in these designs. Our work focuses on system issues and does not consider reference node selection algorithms.

In [6], a highly symmetric, distributed network positioning architecture called Vivaldi is proposed. In this architecture, hosts do not need to have computed positions before serving as reference nodes. Instead, every node starts at the origin position and continuously measures network distances against a small set of random reference nodes and update its position to minimize the locally observed embedding error. The accuracy of this approach turns out to be very promising compared to GNP. A significant property of this approach is that in the steady state, the positions of all hosts are continuously evolving and consistent dependency is not ensured. It remains to be shown how quickly positions change in this architecture, how frequent are re-computations required to maintain accuracy, and how quickly this approach adapts to network topology changes.

A recent study [22] has shown the potential of using Lipschitz embedding with dimensionality reduction based on principal component analysis (PCA) as a method for network positioning. Moreover, in this method, it is easy to introduce multiple Landmark sets for hosts to use to increase scalability and use the translation matrices between the different sets of Landmarks to correct for the difference in bases accordingly. This study also empirically examines the intrinsic dimensionality in large network distance data sets and found the dimensionality to be low. Compared to Euclidean embedding, Lipschitz embedding is much more efficient to compute and the accuracy is comparable to Euclidean embedding. A parallel study [13] has also suggested the use of PCA of Lipschitz embedding to compute network positions. This study analyzes the difficulty in the nonlinear optimization of Euclidean embedding, and show that a PCA based scheme is more computationally efficient. This study also explores methods to reduce the number of Landmarks that need to be probed without adversely affecting the accuracy. Distributed versions of these methods however remain to be developed.

Besides network positioning, an alternative way to provide network topology information to applications is to supply them with network distance estimates directly. In particular, the IDMaps [7] service provides network distance information. IDMaps builds a simplified overlay topology map of the Internet based on network measurements performed by Tracer nodes in the network. Distance predictions are then computed by performing shortest path routing on this topology map model. IDMaps supports a general distance query interface such that an application can query IDMaps servers to find out the network distance between two hosts. Comparing to the IDMaps service, NPS is different in that the infrastructure nodes (Landmarks and membership servers) merely enables end hosts to use their own resources to compute their positions in the Internet and does not directly interact with any applications. It is up to the applications running on end hosts to decide how to use the computed locations. Distance prediction for example can be computed by end hosts and is a by-product of the position information. The Isobar [4] project proposes an efficient overlay network delay monitoring mechanism that uses clustering techniques to group together hosts that have similar network distance observations and thus reduce the amount of monitoring traffic significantly.

There are also other indirect methods for determining nearby neighbors in the Internet. For example, nearby Internet hosts can be clustered implicitly based on Internet routing table information [11]. In [19], a Landmark distance vector based scheme called Binning is proposed to estimate the proximity between hosts. With such a scheme, the location of a host would be represented by the Landmark distance vector. This scheme however is not aimed at producing actual network distance estimates. Triangulated heuristic [8] is another solution, where Landmark vectors are also used as locations, and upper and lower bounds on the network distance between two locations are estimated. In [10], a scheme called Beaconing is proposed to find nearby network hosts. The idea is to use the distance from a host to a Beacon to determine the subset of hosts that lie within a similar distance from the Beacon. By intersecting the subsets of hosts provided by multiple Beacons, a set of nearby hosts can be found.

## 7 Conclusion

In a very short time, network positioning has developed into a fascinating research area. This paper, to the best of our knowledge, is the first to study the system-building issues in network positioning. We have identified key issues such as consistency, adaptivity, and stability in building a network positioning system, and found that with a carefully designed system, these issues can be addressed effectively in practice. There has been a lot of interest in the research community in having access to a publicly available network positioning system, and we believe our prototype can be a first step in providing such a capability. The operational experience will guide the future evolution of the system. Ultimately, our goal is to provide a network positioning capability to all hosts in the Internet.

## Acknowledgement

We thank our shepherd Honesty Young and the anonymous reviewers for their constructive feedbacks on this work. We also thank Sylvia Ratnasamy for providing us with the content-addressable network (CAN) software for experimentations.

## References

[1] A. Acharya and J. Saltz. A Study of Internet Round-Trip Delay. Technical Report CS-TR-3736, University of Maryland, College Park, 1996.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proccedings of ACM Symposium on Theory of Computing*, pages 171–180, 2000.

[3] S. Banerjee, Z. Xu, S.-J. Lee, and C. Tang. Service multicast for media distribution networks. In *IEEE Workshop on Internet Applications (WIAPP)*, San Jose, CA, June 2003.

[4] Y. Chen, C. Overton, and R. H. Katz. Internet Iso-bar: A scalable overlay distance monitoring system. *Journal of Computer Resource Management, Computer Measurement Group, Spring Edition*, 2002.

[5] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. Technical Report MSR-TR-2003-53, Microsoft Research, September 2003.

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

[7] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniewicz, and Y. Jin. An architecture for a global Internet host distance estimation service. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[8] S.M. Hotz. Routing information organization to support scalable interdomain routing with heterogeneous path requirements, 1994. Ph.D. Thesis (draft), University of Southern Californi a.

[9] A.-C. Huang and P. Steenkiste. Network-sensitive service discovery. In *Proceedings of USENIX - USITS*, 2003.

[10] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding close friends on the Internet. In *Proceedings of IEEE ICNP*, Riverside, CA, November 2001.

[11] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[12] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[13] H. Lim, J. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proceedings of Internet Measurement Conference*, Miami, FL, October 2003.

[14] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[15] T. S. E. Ng and H. Zhang. Predicting Internet networking distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM*, June 2002.

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

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

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[19] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM*, New York, NY, 2002.

[20] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, March 2003.

[21] P. Srisuresh and K. Egevang. Traditional IP network address translator (Traditional NAT), January 2001. RFC-3022.

[22] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of Internet Measurement Conference*, Miami, FL, October 2003.

[23] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, University of Michigan, 2002.

[24] Z. Xu, C. Tang, S. Banerjee, and S.-J. Lee. Receiver initiated just-in-time tree adaptation for rich media distribution. In *Proceedings of NOSSDAV*, Monterey, CA, June 2003.

[25] Y. Yang and S. Lam. General AIMD congestion control. Technical Report TR-200009, Dept. of Computer Science, University of Texas at Austin, May 2000.

[26] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM*, March 1996.

[27] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of internet path properties: Routing, loss, and throughp ut. Technical report, ACIRI, May 2000.