Strategies for Improving the Trustworthiness and Efficiency of Network Positioning Algorithms

Balasubramaneyam Maniymaran
Dept. of Elect. and Computer Eng.,
McGill University,
Montreal, QC, Canada
bmaniy@cs.mcgill.ca

Muthucumaru Maheswaran
School of Computer Science,
McGill University,
Montreal, QC, Canada
maheswar@cs.mcgill.ca

Yuanyuan Gao
School of Computer Science,
McGill University,
Montreal, QC, Canada
ygao30@cs.mcgill.ca

Abstract

As real-time interactive applications start embracing the service-oriented paradigm, it becomes increasingly important to locate services by proximity. One way to implement this is via network latency estimation using approaches such as network positioning. In this paper, we propose simple and practical strategies to improve the trustworthiness of network positioning schemes. In particular, our strategies make network positioning immune to non-random perturbations such as denial-of-service attacks and localized network congestion. Additionally, we studied the overhead generated by existing network positioning algorithms and propose an algorithm that results in low overhead while retaining very high accuracies. We performed extensive simulations and implementations on PlanetLab to examine the performance trade-offs.

1. Introduction

In a brave new service-oriented world, network latencies are going to be a critical factor in determining the best *point of presence* for a service. This is more so as the services model is embraced by real-time and interactive applications such as online games, interactive live streaming, and *voice over IP* (VoIP). In addition, network latency also affects other TCP based applications because the TCP congestion handling mechanisms makes the throughput of a TCP connection depend on the end-to-end latency.

The network latency issue can be approached in two ways: choosing one end point of a transaction such that end-to-end latency is minimized or predicting a *quality of service* (QoS) deliverable for a given network configuration.

One emerging approach for network latency estimation is *network positioning*, where network hosts are mapped onto a linear space such that the latency and proximity properties along the actual network can be deduced from the coordinates on this linear space. The obvious advantage of this solution is that it can be used both to predict the network delay and to derive the proximity information using spatial algorithms. Further, once the mapping is found, the network delay and proximity information can be readily extracted without creating additional network traffic providing a scalable solution.

One of the key concerns with network positioning schemes is the errors associated with latency measurements such as: (a) *probe noise*: probe packet (usually an application level ping) is subjected to random queuing and processing delays as any other packet, (b) *probing error*: probes in the Internet do not travel in a straight line, thus overestimating distance, (c) *non-random error*: probing is impacted by situations such as network congestion, route instability, and various forms of denial of service attacks. Probe noise and probing error have been addressed by previous proposals for network positioning but non-random errors are yet to be addressed.

With a services oriented computing model, service providers can use proximity to attract customers to their deployment. If we use a network positioning scheme for proximity determination, it should be immune to non-random errors. This paper proposes a simple and practical approach to addressing the non-random errors. Our approach is to divide the network into trusted clusters using offline trust relationships. For example, nodes with a campus network would fall into a single trusted cluster because the nodes can mutually trust each other and can reach each other without crossing untrusted networks. The coordinate mapping algorithms use the coordinate constraints determined from the trusted measurements to check and correct the coordinates yielded by the untrusted global scope measurements.

We studied the clustering strategy using extensive simulations and also implemented and tested the algorithms on PlanetLab. The results indicate that the clustering approach can significantly improve the trustworthiness of the coordinate computation especially for large networks.

Another important issue with regard to network positioning is efficiency. Because these algorithms run periodically as part of the infrastructure, they should be minimally intrusive in terms of message and processing overheads. We examined the overheads of two distinct network positioning algorithms and propose a third one that can outperform other two under certain conditions. Simulations as well as implementations were used to evaluate the performance.

The paper is organized as follows: Section 2 introduces the network positioning scheme used in our research and formulates the problem; Section 3 describes three positioning algorithms used in our system; An cluster-based positioning solution is introduced in Section 4 to address the impact of non-random errors; Section 5 presents the performance study on the cluster-based scheme and the used algorithms; A concise description of related works is given in Section 6.

2. Landmark-Aided Positioning

Network positioning schemes can be categorized into two groups: *infrastructure-based* and *infrastructure-less* solutions. The infrastructure-based schemes employ infrastructure nodes called *landmarks* as the reference points for the other nodes. The infrastructure-less schemes rely on no infrastructure nodes; instead, any node can be a reference point for another node. A node in this type of scheme learn about another node in the system from the source field of an incoming message. Then, the node probes the remote node and adjusts its position accordingly. The infrastructure-less schemes have a number of practical issues: (a) as nodes solely depend on incoming messages, a node might take a long time to receive enough unique messages to find its correct position; Conversely, in infrastructure mode, node positioning is extremely fast, because a node is proactive in determining its position; (b) previous research has established the need for probing distant reference points to produce a good mapping. In infrastructure-less scheme, it is not guaranteed that sufficient incoming messages are from distant nodes. In fact, the motivation behind a network positioning scheme is to confine traffic as much as possible to a locality, which could mean less traffic from distant nodes; (c) a set of malicious nodes can disturb the positioning system by just sending messages to selected nodes in the system. On receiving these messages, the innocent nodes will contact the malicious nodes who lie about their positions which leads to incorrect positioning.

Therefore, an infrastructure-based positioning scheme is considered for this paper, which we call as *landmark-aided positioning* (LAP). The primary objective in LAP design is to maximize scalability and minimize message overhead generated by the algorithms. The LAP scheme consists of two phases: landmark positioning and node positioning. In the *landmark positioning* phase, the landmarks run a distributed algorithm to positioning themselves relative to each other in a Cartesian space. When the landmark positioning phase is complete, the positions of the landmarks provide the reference frame for the other nodes. In the *node positioning* phase, each non-landmark node uses the reference frame to fix its position. The two phases of the LAP scheme are repeated periodically to handle with the changing network conditions.

2.1. Node positioning

The node positioning algorithms try to minimize the objective function defined as:

$$\sum_j \varepsilon(l_{ij}, \hat{l}_{ij}) \tag{1}$$

where l_{ij} is the measured network delay and $\hat{l}_{ij}$ is the predicted network delay between the node i and the landmark j. The predicted delay is the Euclidean distance between the nodes, i.e. $\hat{l}_{ij} = |\vec{x}_i - \vec{x}_j|$ where $\vec{x}_i, \vec{x}_j \in \Re^d$ are the coordinates of the node i and landmark j, respectively. The error function $\varepsilon()$ defines the positioning error as the *delay prediction error*:

$$\varepsilon(l_{ij}, \hat{l}_{ij}) = |l_{ij} - \hat{l}_{ij}| \tag{2}$$

The inputs to a node positioning algorithm are the measured network delay values and the coordinates of the landmarks. It was previously shown that, for a d-dimensional Cartesian space, $d+1$ landmarks are enough to produce reasonably accurate positioning [1]. Therefore, the node positioning algorithm requires only a $O(d)$ number of probe messages causing very low message overhead (the total message complexity is $O(Nd)$, where N is the number of nodes in the system). Optimization algorithms are iterative procedures. Therefore, the computational load on each node is determined by the number of iterations taken by the algorithm to reach convergence.

2.2. Landmark positioning

The landmark positioning problem is formulated as an optimization minimizing the objective function defined in Equation 3, where nodes i and j are landmarks.

$$\sum_i \sum_j \varepsilon(l_{ij}, \hat{l}_{ij}) \tag{3}$$

```
1:  repeat
2:    /* outer loop */
3:    for all node $i$ do
4:      $x_j \leftarrow$ coordinates of other landmarks
5:      $l_{ij} \leftarrow$ ping values to other landmarks
6:      repeat
7:        /* inner loop */
8:        for all landmark $j \neq i$ do
9:          adjusting $\vec{x}_i$ related landmark $j$
10:       end for
11:     until convergence on $x_i$
12:   end for
13: until convergence on system
```

Figure 1: Pseudo code for landmark positioning.

Figure 1 shows the pseudo code of the landmark positioning algorithm. In the inner loop (steps 6–11), each landmark adjusts its position relative to other landmarks and in the outer loop (steps 1–13), the landmarks exchange their adjusted coordinates. The algorithm continues until there is no adjustment in the coordinates. The number of iterations taken by both the inner and outer loops determines the computational load on each landmark node, while the outer loop iterations determines the message overhead created by the algorithm due to inter-landmark coordinate exchanges. Therefore, the message complexity of landmark positioning is $O(I_o L^2)$, where I_o is the number of outer loop iterations and L is the number of landmarks in the system.

3. LAP Algorithms

This section examines two representative algorithms from the literature: Simplex and Spring. Also, it presents a new algorithm we developed called SpringEq.

3.1. Simplex Downhill Algorithm

As network positioning is a *multi-dimensional optimization problem*[1], *Simplex downhill* (SDH) algorithm is an obvious solution. The pseudo code of the Simplex algorithm for node positioning is given in Figure 2. The complexity of step 4 is $O(Ld)$ (from Equations 1 and 2), where L is the total number of landmarks and d is the dimension of the Cartesian space. Further, since the SDH algorithm uses a geometric construct called simplex [2] which is a polygon of $d+1$ vertexes, the complexity of the steps 3–6 becomes $O(Ld^2)$. Hence, the complexity of the SDH algorithm becomes $O(ILd^2)$, where I is the number of iterations taken by the `repeat...until` loop.

[1] an optimization problem with more than one independent variable

```
1: create simplex
2: repeat
3:     for all simplex points do
4:         evaluate objective function
5:         find/refine downhill direction
6:     end for
7:     move down hill
8:     if could not move then
9:         shrink simplex size
10:    end if
11: until converged
```

Figure 2: Simplex downhill algorithm.

SDH method has number of disadvantages. First, it is inherently a centralized algorithm. The distributed implementation for landmark positioning needs to iteratively run the centralized version at every outer loop iteration. Therefore, the complexity of simplex algorithm for landmark positioning is $O(I_o ILd^2)$, where I_o is the number of iterations in the outer loop. Second, the performance of the algorithm is sensitive to various design parameters (for example, the initial simplex size and position).

3.2. Spring Algorithm

Spring algorithm models the network as a *spring system* [3, 4]. The network hosts are considered as massless nodes connected to each other with springs whose *natural lengths*[2] are the measured network delay. In such a system, the springs are almost always under deformation (either shrunken or elongated). The Spring algorithm iteratively adjusts the positions of the nodes so that total deformation in the spring system is minimized. Comparing this procedure with the error function defined in Equation 2, it can be seen that the Spring algorithm effectively solves the optimization problem defined in Equation 3. Spring algorithm is an attractive solution as it is very simple to code and is naturally distributed.

```
1: $\mathrm{limit} \leftarrow 0.05, \mathrm{dec} \leftarrow 0.025$
2: $s \leftarrow 0.5$    /* scale factor */
3: repeat
4:     $x_j \leftarrow$ coordinates of the other landmarks
5:     $l_{ij} \leftarrow$ ping values to the other landmarks
6:     for all landmarks $j$ do
7:         $\vec{u}_{ij} \leftarrow$ unit direction towards landmark $j$
8:         $\hat{l}_{ij} \leftarrow |\vec{x}_i - \vec{x}_j|$
9:         $\vec{x}_i \leftarrow \vec{x}_i + s(\hat{l}_{ij} - l_{ij})\vec{u}_{ij}$
10:    end for
11:    $s \leftarrow \max((s - \mathrm{dec}), \mathrm{limit})$
12: until converged
```

Figure 3: Pseudo code for the Spring algorithm.

For every iteration of the Spring algorithm in Figure 3, a landmark adjusts its position relative to each of the other landmarks. The size of the adjustment is controlled by *scale factor* s. As proposed in [3], the scale factor is decreased by a fixed value at every iteration, but lower bounded by another fixed value (step 11).

The complexity of the Spring algorithm given by $O(I_o Ld)$, because steps 6–10 have complexity of $O(d)$. As the dimension of the Cartesian space is constant, Spring algorithm has complexity linear to the size of the landmark set like the SDH method. Even though the values of I_o and I depend on the data set, the Spring algorithm can be expected to perform better than SDH as it does not really have an inner loop like SDH (or $I = 1$ always in Spring).

Even though the Spring algorithm is actually proposed as a distributed algorithm, by moving steps 4 and 5 out-

[2]length of a spring at rest, i.e. when no force applied upon

side the `repeat...until` loop, it can be used for node positioning. This provides a much simpler solution (coding complexity) than the SDH method.

3.3. SpringEq Algorithm

SpringEq (spring system in equilibrium) is inspired by the spring system-based algorithms [4, 5]. SpringEq algorithm uses the same spring model used by the Spring algorithm; However, instead of considering the individual springs attached to a node, SpringEq algorithm considers the resultant force on a node and formulates the solution by deriving the equilibrium condition of the node as a system of homogeneous equations.

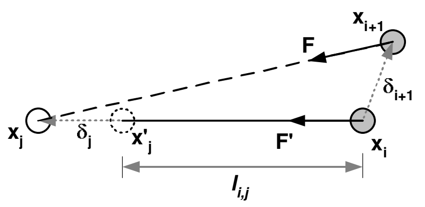

Figure 4: A spring under deformation due to the movement of the end points.

Using the Hooke's Law, the force $\vec{F}$ applied on a spring under deformation $\vec{\delta}$ can be written as $\vec{F} = k\vec{\delta}$, where k is the *spring constant*. Considering the spring under deformation as shown in Figure 4, the force $\vec{F}$ can be written as:

$$\vec{F} = k[(\vec{x}_j - \vec{x}_{i'}) - \vec{l}_{i',j}] \tag{4}$$

where, $\vec{l}_{i',j}$ is the natural length of the spring measured in the direction of the spring layout. i.e. $\vec{l}_{i',j} = l_{i,j}\frac{(\vec{x}_j - \vec{x}_{i'})}{|\vec{x}_j - \vec{x}_{i'}|}$. However, for small displacement of $\vec{\delta}_i$, $\vec{F} = \vec{F}'$ and

$$\begin{aligned} \vec{l}_{i',j} = \vec{l}_{i,j} &= \vec{x}_j - \vec{\delta}_j - \vec{x}_i && (5) \\ \text{From Equations 4 and 5,} \quad \vec{F} &= k[(\vec{x}_j - \vec{x}_{i'}) - (\vec{x}_j - \vec{\delta}_j - \vec{x}_i)] \\ &= k[\vec{\delta}_j - (\vec{x}_{i'} - \vec{x}_i)] \\ \vec{F} &= k(\vec{\delta}_j - \vec{\delta}_i) && (6) \end{aligned}$$

For the equilibrium of the node i connected to $L-1$ springs:

$$\begin{aligned} \sum_{j=1}^{L-1} \vec{F} = \sum_{j=1}^{L-1} [k_{i,j}(\vec{\delta}_j - \vec{\delta}_i)] &= 0 \\ \sum_{j=1}^{L-1} \vec{\delta}_j - (L-1)\vec{\delta}_i &= 0 \quad \forall i \quad \text{assuming} \quad k_{i,j} = 1 && (7) \end{aligned}$$

$\vec{\delta}_j$ can be calculated as $\vec{\delta}_j = (\hat{l}_{ij} - l_{ij})\vec{u}_{ji}$, where $\vec{u}_{ji} = (\vec{x}_j - \vec{x}_i)/\hat{l}_{ij}$. Equation 7 represents a system of homogeneous equation, $A\delta = 0$, where

$$A = \{a_{ij}\}_{L\times L} = \begin{cases} 1 & j \neq i \\ -(L-1) & j = i \end{cases} \quad \text{and} \quad \delta = \{\vec{\delta}_i\}$$

```
1:  $r \leftarrow 0.85$        /* relax factor */
2:  repeat
3:      $x_j \leftarrow$ coordinates of the other landmarks
4:      $l_{ij} \leftarrow$ ping values to the other landmarks
5:      $\vec{\Delta} \leftarrow -\vec{\delta}_i(1-r)(L-1)$
6:      for all landmarks $j$ do
7:          $\hat{l}_{ij} \leftarrow |\vec{x}_i - \vec{x}_j|$
8:          $\vec{u}_{ij} \leftarrow$ unit direction towards landmark $j$
9:          $\vec{\delta}_j \leftarrow (\hat{l}_{ij} - l_{ij})\vec{u}_{ij}$
10:         $\vec{\Delta} \leftarrow \vec{\Delta} - \vec{\delta}_j$
11:     end for
12:     $\vec{\delta}_i \leftarrow -\frac{1}{r(L-1)}\vec{\Delta}$
13:     $\vec{x}_i \leftarrow \vec{x}_i + \vec{\delta}_i$
14: until converged
```

Figure 5: Pseudo code for the SpringEq algorithm.

At equilibrium, the system is at the minimum energy, which means that the deformation in the springs in the system is minimized. Therefore, solving the system for equilibrium effectively solves the landmark positioning problem.

SpringEq algorithm solves this system of equations using an iterative method. An iterative method is chosen because it can be easily converted into a distributed solution, where each node i solves the i^{th} equation of the system. SpringEq uses *successive over relaxation* (SOR) method for its speed and simplicity. Figure 5 shows the pseudo code for the SpringEq algorithm. Similar to the Spring algorithm, the complexity of the SpringEq algorithm is $O(ILd)$.

The position of a node relative to a set of landmarks can be found by re-writing the Equation 7 as:

$$\vec{\delta}_i = \frac{1}{L-1}\sum_{j=1}^{L-1}\vec{\delta}_j \tag{8}$$

The SpringEq algorithm for node positioning runs the Equation 8 iteratively to compensate the error due to the assumption that that $\vec{\delta}_i$ is small.

4. Clustered Landmark-Aided Positioning

To position itself, a node needs to obtain latency measurements from several landmarks along with their coordinates. Here, the node places trust on the coordinates given by the landmarks and the measured latencies to them. To prevent simple spoofing attacks disrupting the network positioning process, the coordinates given by the landmarks must be secured. This involves signing the messages at the sender side and verifying them at the receiver. To determine application level network latencies, we need to subtract the signing and verifying times from the measured times.

The malicious nodes may not be the only source of errors in latency measurements. The Internet itself is a dynamic system with variable network conditions in different network segments which can introduce arbitrary errors that are non-random. Non-random errors are not persistent, therefore, can not be handled by optimization algorithms; at the same time, they cannot be filtered by data smoothing the measurements.

One way of reducing the impact of these non-random errors is to increase the number of measurements so that the overall fraction of bad measurements considered for the calculation of the position is low. But, this is not an efficient solution, because it not only produces additional network traffic but the signing and verification process can create significant overhead on the nodes as well.

We introduce a new scheme called *clustered landmark aided positioning* (CLAP) to address these issues. The primary idea behind CLAP is to create *clusters* of trusted neighboring peers and share information within the cluster. Further, the proximity-based clustering reduces the possibility of non-random errors, because the intra-cluster measurements are limited to short distances that hardly cross congested paths. The CLAP uses this trust on local measurements to build a trusted global coordinates.

4.1. Cluster Formation and Maintenance

One of the problems in CLAP is forming the clusters. The simplest approach is to create clusters statically. For example, all nodes in a campus network or in a highly controlled network can be selected as part of a cluster. In highly trusted and stable environments, this approach can be sufficient.

In more dynamic networks, we need to use online schemes to create clusters. One approach is to select a *seed* node around which the cluster should be established. Then, probe for other candidate nodes that lie within a given network latency and include them in the cluster. This approach is suitable for dynamic network environments. Unlike the static case given above, the trust issue is not directly considered here.

Another approach is to define clusters using off-line trust relations very much like the static approach and then refine it dynamically. The cluster defined by the offline trust relations is the total set of nodes that can participate in the cluster. The dynamic scheme measures the inter-node network latencies and temporarily removes some nodes from the cluster. Nodes could be removed because they are computationally overloaded and unable to respond to network positioning requests in a timely manner or lie behind a locally congested network link. The dynamic scheme can be considered as a cluster maintenance routine.

4.2. Cluster Oriented Positioning

The CLAP algorithms is shown in Figure 6. It uses the *simple LAP* (SLAP) in two different ways. We assume that a global set of landmarks are already chosen and a landmark positioning algorithm has already run among them to establish the reference frame.

We run the *cluster formation* process to setup the clusters that will be used by the CLAP scheme. Once the cluster is setup, we do the following operations periodically. In the first step, each node runs the SLAP node positioning process to establish its coordinates with respect to the global reference frame. In the second step, each node runs the SLAP landmark positioning process within its cluster. It should be noted that the landmark positioning process will take its initial values from the outputs of the node positioning scheme. Because the landmark positioning is entirely within the cluster it is trusted. Thus, we can consider this process as a correction on the coordinates obtained by the SLAP node positioning in the previous step.

```
 1: cluster_formation()
 2: loop
 3:    /* CLAP iteration */
 4:    doSLAPnodePosition()
 5:    doSLAPlandmarkPosition([cluster_nodes])
 6:    if maintain_time() then
 7:       cluster_maintain()
 8:    end if
 9:    wait(NODE_POS_WAIT_TIME)
10: end loop
```

Figure 6: CLAP Algorithm

In the third step, the *cluster maintenance* process is run by each node. It should be noted that this step is not run at each iteration (i.e., we don't expect the cluster to require maintenance every iteration). In the last step, we wait for a specified time and then iterate.

In the next iteration, the coordinates corrected by the cluster-based landmark positioning will be fed into the SLAP network positioning. This network positioning process will use the corrected coordinates as the starting point.

The cost of the CLAP scheme is the additional message overhead it produces. It increases the node positioning message complexity to $O(Nd + CI_cN_c^2)$, where C is the number of clusters in the system, N_c is the average number of nodes in a cluster, and I_c is the average number of iterations taken by the cluster-restricted landmark positioning algorithm. N_c can be a relatively small number and I_c is expected to be small as the algorithm is applied on positions that have already converged. Therefore, the CLAP scheme does not create excessive message overhead for very large networks.

5. Performance Analysis

In this section, we discuss the results from simulations and implementations of the different algorithms. It is organized in three subsections. We devote one subsection for each different type of positioning and examine them in the following order: node positioning, landmark positioning, and CLAP. Below we discuss the parameter settings that were common for all the simulations and implementations. Specific settings are discussed in that particular subsection.

Detailed simulations were performed for exploring the three algorithms described in Section 3. The simulations allowed us to examine much larger parameter space than the implementation which were limited by the resources provided by PlanetLab. We used a random network for the simulations, where nodes are assigned with random coordinates that is known only to the simulator (the nodes know only the coordinates found by the positioning algorithms). The value of a ping between two nodes i and j, l_{ij}, is taken as $l'_{ij}(1 + 2\xi\rho)$, where l'_{ij} is the Euclidean distance between *i* and *j* based on the assigned coordinates and ξ and ρ are positive fractions simulating the ping error and ping noise respectively. While ξ is constant for an experiment, ρ is randomly varied for each ping. The results presented here are averaged over many tests using different hypothetical setups.

We implemented the different positioning schemes on PlanetLab [6]. It is an ideal environment for testing our schemes because it presents realistic scenarios like congestion, diverse link behaviors, and client workloads. Although PlanetLab consists of about three hundred nodes, many of them show frequent down-time. Therefore, we probed the PlanetLab nodes for a long interval and chose 127 nodes to run our network positioning schemes. Out of these 127 nodes, 30 nodes are randomly selected as landmarks and the remaining 97 nodes as ordinary hosts. When some of the selected nodes became unresponsive, the algorithms continued to run temporarily not considering them for calculations. In the implementation, the landmarks started landmark positioning 10 minutes prior to the other nodes starting the node positioning. The landmarks updated their positions every 3 hours by re-running the landmark positioning algorithm. Ordinary hosts updated their positions once every hour. The experiments lasted for 5 landmark updating periods.

5.1. Node Positioning Performance

Figures 7 and 8 show the results from the simulations that compared the different node positioning algorithms. Even though the performance difference in terms of accuracy is marginal, Spring outperforms others in terms of number of iterations taken.

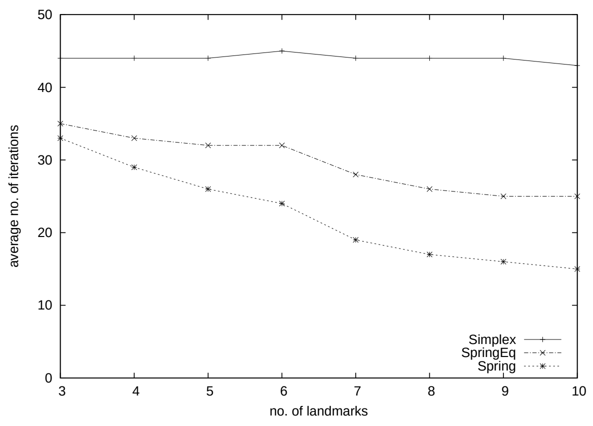

Figure 7: Variation average number of iterations taken with varying number of landmarks per node.

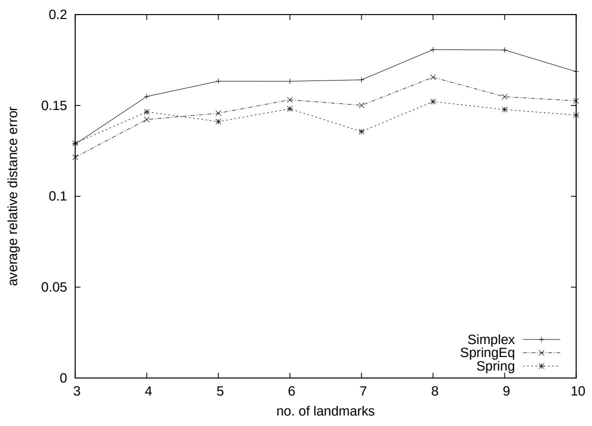

Figure 8: Variation of average relative prediction error with varying number of landmarks per node.

The fast convergence of the Spring algorithm can be attributed to the large scale factor in the initial iterations. This convergence behavior is illustrated in Figure 9. This plot tracks the coordinates of a node for first few iterations of the Spring and SpringEq algorithms for 1000 trials. The node's actual coordinates are $(750, 750)$ and initial coordinates are randomly chosen in the range $([0, 1000], [0, 1000])$ for the trials. It can be seen from the figure that the Spring algorithm converges very close to the correct position in the first iteration itself, while the SpringEq algorithm slowly converges towards the correct position.

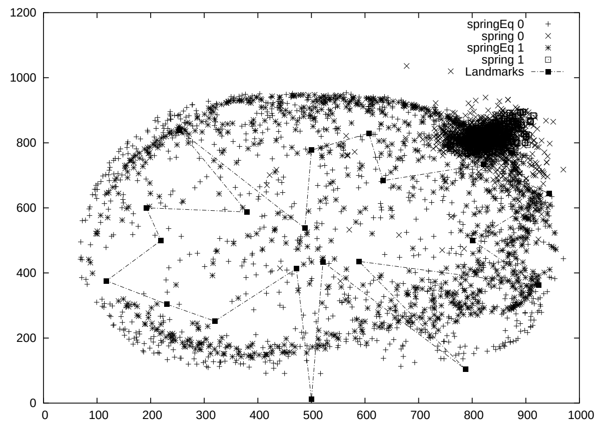

Figure 9: Coordinates after first two iterations.

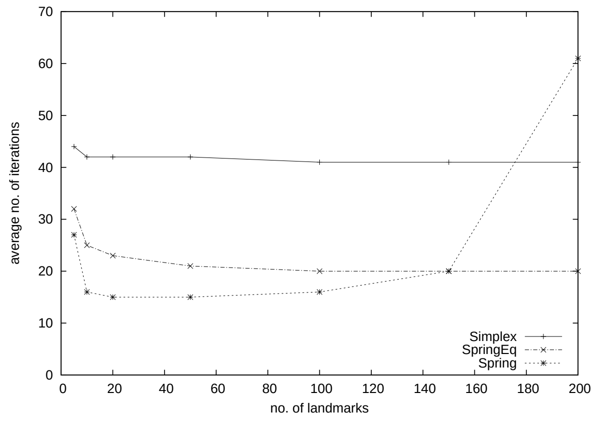

Figure 10: Variation of average number of iterations taken when using large number of landmarks per node.

Figure 10 shows the performance of the algorithms for large number of landmarks. Interestingly, the performance of Spring algorithm quickly deteriorates as the number of landmarks increases[3]. This prompted us to look at the convergence of the algorithms closely to understand this observations. Figure 11 shows the average displacements in the positions of the nodes between iterations at the final few iterations (close to convergence) for different configurations. It shows that the displacement in position of the nodes hardly becomes zero in the Spring algorithm. When using 20 landmarks, the displacements after reaching the maximum allowable iterations are still within the convergence threshold of the algorithm. However, when using larger number of landmarks, the displacements exceeds the threshold making the algorithm not converge. On the other hand, SpringEq is able reach convergence in both cases, because the displacements reach zero irrespective of the number of landmarks used. A careful look at the pseudo code reveals that the reason for the constant displacement (or oscillation) is the non-zero scale factor. Therefore, instead of having a fixed limit on the decreasing scale factor, we modified the scale factor adjustment step (step 11 in Figure 3) as $s = \max((s-0.025), 1/L)$, where L is the number of landmarks used. This idea is supported by the Figure 10, where performance degradation is observed at 20 landmarks and above when the scale factor is limited to $0.05(= 1/20 = 1/L)$. Figures 11 and 12 show that this modification greatly improve the performance of the Spring algorithm.

The implementation results for node positioning scheme using Spring algorithm with varying number of landmarks per node are shown in Figures 13 and 14. Figure 13 shows the average distance correlation of the system. Figure 14 shows the cumulative distribution of relative error for the group of nodes using different amount of landmarks. As

[3]even though few landmarks are enough for node positioning, this observation is important for analyzing the behavior of the algorithm in landmark positioning.

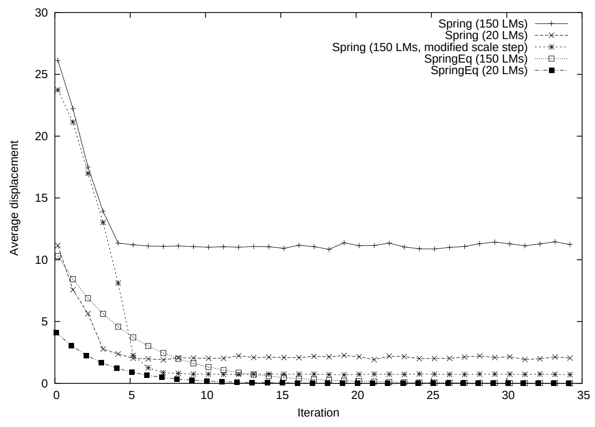

Figure 11: Displacement in position between iterations in final few iterations (landmarks = 20).

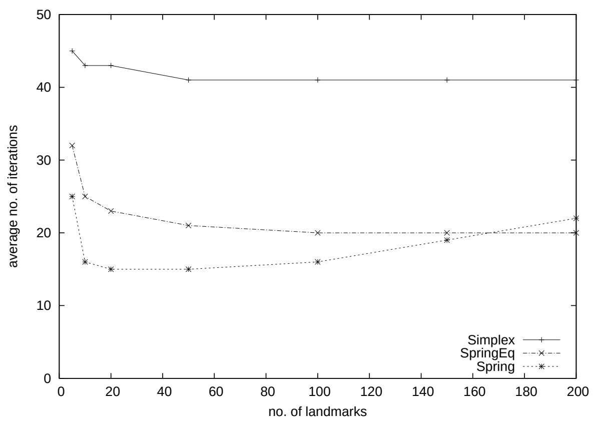

Figure 12: Variation of average number of iterations with modified scale factor.

expected, the accuracy improves with the number of landmarks used per node; but the improvement is not very significant.

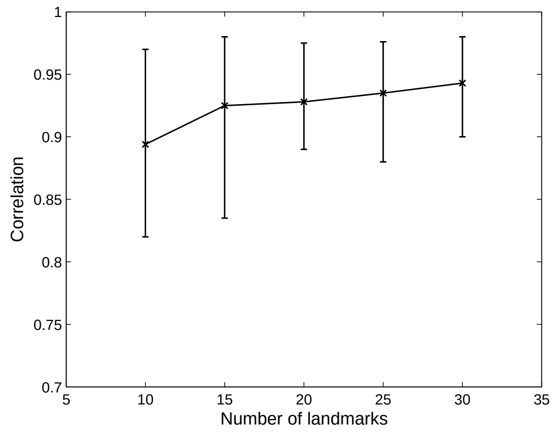

Figure 13: Variation of distance correlation with number of landmarks per node.

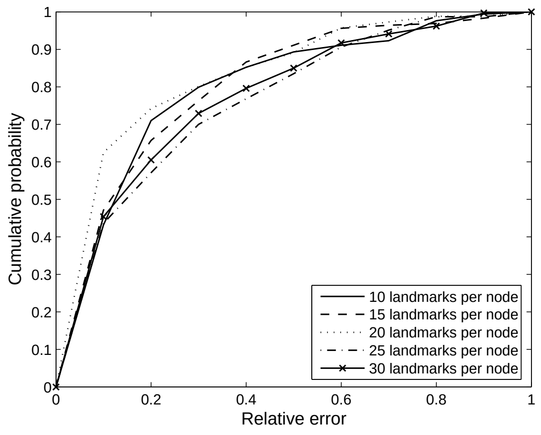

Figure 14: Cumulative distribution of relative distance error.

5.2. Landmark Positioning Performance

This section presents the performance of the algorithms for landmark positioning. The experiments run the landmark positioning algorithms over several time steps (epochs) simulating the periodic re-runs of the algorithm in a practical scenario. The figures show the performance of the algorithms for the first three epochs.

Figure 15 shows the accuracy in positioning using the three algorithms. Even though all the algorithms perform almost equal in terms of accuracy, Figure 16 shows that they significantly differ in terms of number of iterations taken in the outer loop of algorithm (as in Figure 1). As described in Section 2.2, number of outer loop iterations determines the message overhead. Simplex is the worst among the three in this regard (note that iterations are limited to 100 by the experiment). As expected from the node positioning results, Spring algorithm outperforms the SpringEq algorithm for small number of landmarks, but the SpringEq algorithm takes over when the number of landmarks increases. Importantly, despite the number of landmarks, the number of iterations taken by the SpringEq algorithm is much smaller than that taken by the Spring algorithm in the second and third epochs. This shows that when there is no perturbation in the system, SpringEq provides much lower overhead than the Spring algorithm.

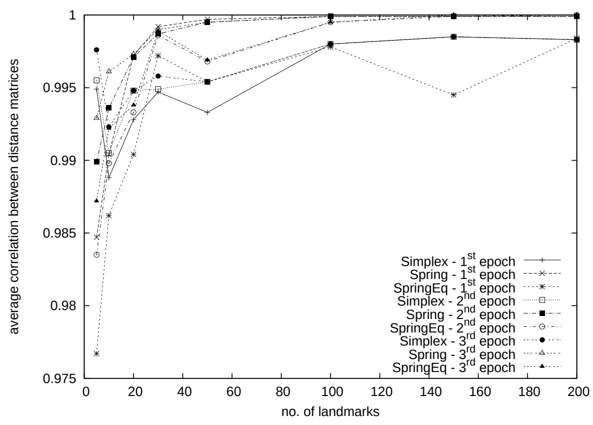

Figure 15: Variation of distance correlation with number of landmarks with (original Spring algorithm).

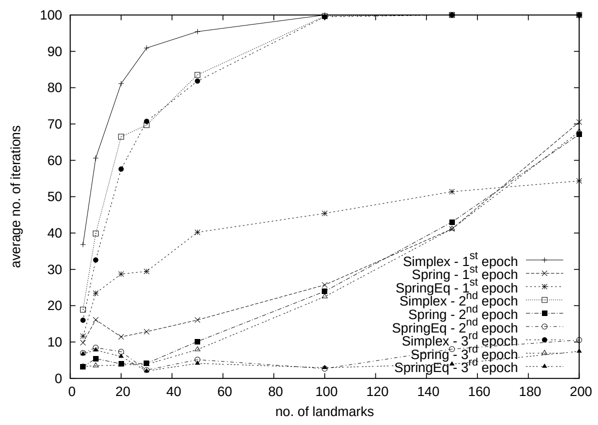

Figure 16: Variation of number of outer loop iterations with number of landmarks (original Spring algorithm).

Figure 17 shows the displacements in the positions of the landmarks between epochs. Note that, these are inter-epoch displacements, not inter-iteration displacements and are required to be zero to produce a *robust* positioning, where two subsequent positions (in time) of a node are same or very close unless perturbed by a network condition. The figure shows that Spring algorithm performs poorly compared to SpringEq algorithm and this justifies the constant workload it requires at every epoch as shown in Figure 16.

Figure 18 shows the same performance metric with the scale factor in Spring algorithm modified as in Section 5.1. Even though this modification improved the performance of the algorithm in node positioning, it degrades the performance in landmark positioning. It is because the scale factor is brought down too fast so that the adjustment of landmarks' positions are much restricted well before any of them converge to the low error positions. Figure 19 uses the scale factor adjustment as described in the refined Vivaldi algorithm [4] ($s = \frac{\Delta_{\text{local}}}{\Delta_{\text{local}}+\Delta_{\text{remote}}}$). This actually perform worse, because, when the relative error in the positions are same everywhere in the system, the lower limit of the scale factor becomes 0.5.

Another interesting characteristics of the algorithm is the change in performance for different convergence criteria. The above experiments considered a node to have converged to its when the relative distance errors in the links to the landmarks are less than 30%. Figure 20 shows the performance of the algorithm when the node is considered converged to its position when its displacement is within a design threshold. This improves the accuracy a little bit (results are not

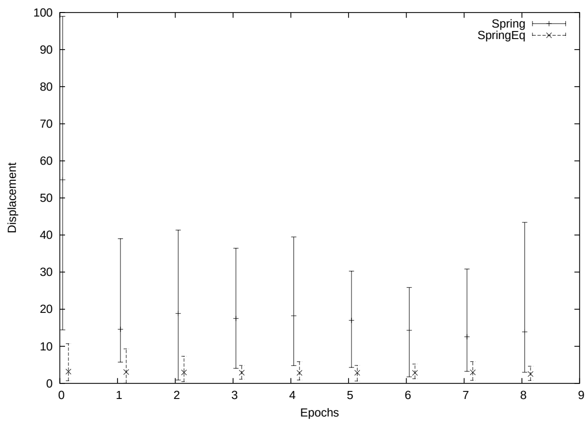

Figure 17: Displacement in landmark positions in subsequent epochs (original Spring algorithm).

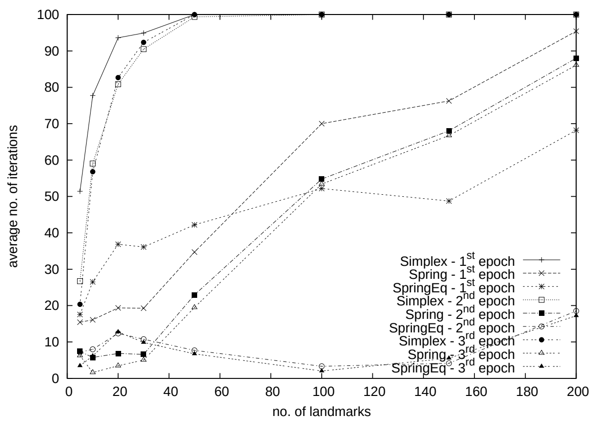

Figure 18: Variation number of outer loop iterations with number of landmarks (modified Spring algorithm).

shown here). However, as the figure shows, performance of the Spring algorithm worsen with modification, whereas the performance of the SpringEq improves. This again proves that displacement in position is the worst behavior in Spring equilibrium while the best in SpringEq. Therefore, in addition to producing lower message overhead, SpringEq algorithm provides a more robust positioning scheme compared to the Spring algorithm. That is, SpringEq provides a positioning scheme that smoothly varies

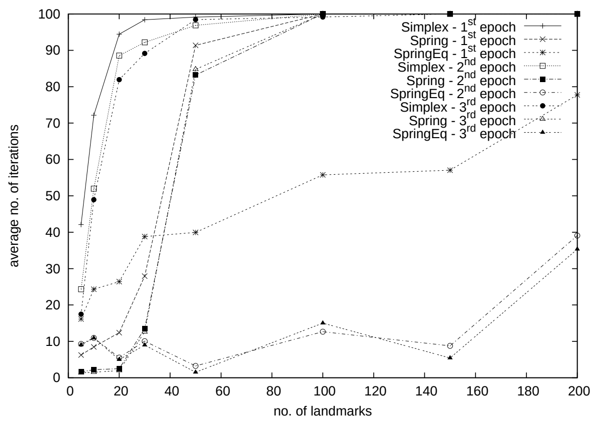

Figure 19: Variation of number of outer loop iterations (modified Spring algorithm according to [4]).

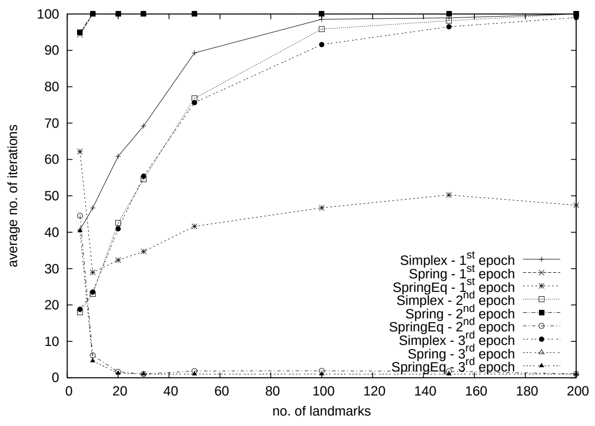

Figure 20: Variation of number of outer loop iterations (original Spring algorithm, convergence on position).

The rest of this section shows the implementation results for landmark positioning. The implementation based experiments were restricted to a maximum of 30 landmarks. In order to reduce the impact of landmark selection on the observed performance, the experiments are repeated for 5 times for each number of landmarks, with different set of landmarks for every experiment.

Figure 21 shows the variation of average distance correlation with number of landmarks. Both algorithms (Spring and SpringEq) show good performance. However, the performance slightly reduces when more nodes are involved.

It is probably because large number of landmarks increases the probability of non-random errors encountered in the measurement which the SLAP algorithms do not handle. Still, comparing the two algorithms with the error bars, SpringEq algorithm provides more stable performance than the Spring algorithm.

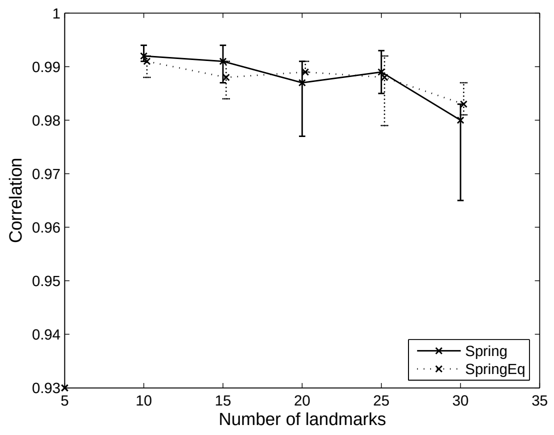

Figure 21: Variation of distance correlation with number of landmarks.

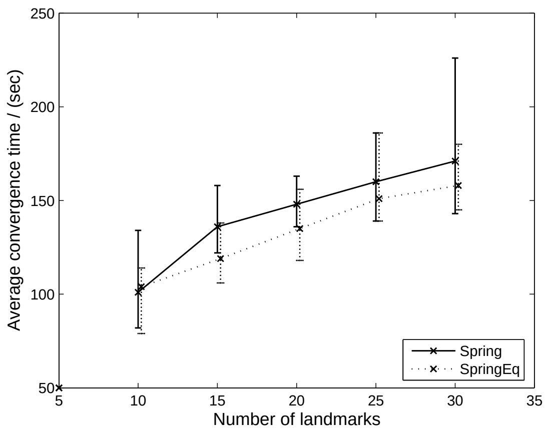

Figure 22: Variation of convergence time with number of landmarks.

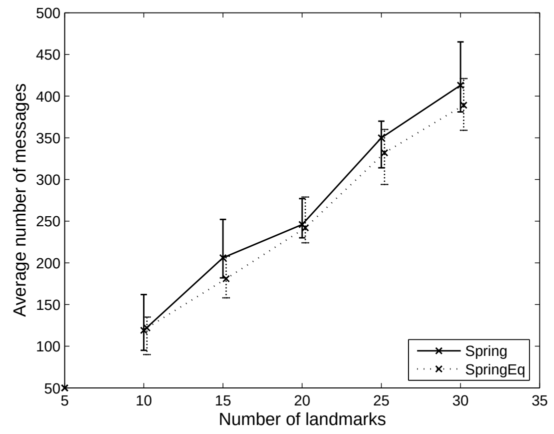

Figure 23: Variation of message overhead with number of landmarks.

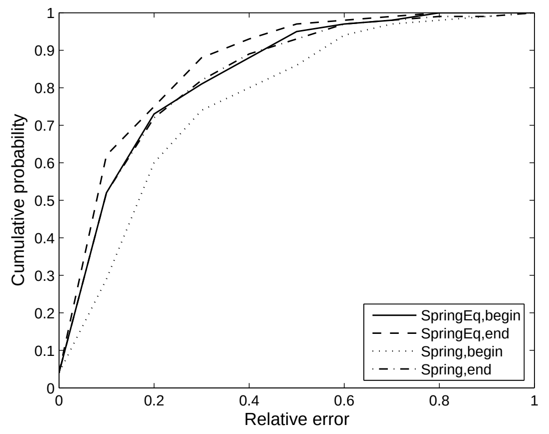

Figure 24: Cumulative distribution of relative Error.

Figures 22 and 23 show the average convergence time and the average message complexity of the two algorithms for different number of landmarks. The convergence time increases for both algorithms with the number of landmarks. This increase in converge time is attributed to both the additional inter-landmark communication delay and the increasing number of iterations. From Figure 23, it can be seen that the increase in convergence time is mainly due to the increase in number of iterations, because the number of messages increases with the number of iterations taken. The figures show that SpringEq performs a little better than Spring algorithm. However, we know from the simulation results that the real performance improvement in SpringEq can be seen only when the number of landmarks is large which is not the case here. Despite this fact, we observed that the Spring algorithm occasionally failed to converge when the number of landmarks is more than 25 supporting our observation on convergence from Section 5.1.

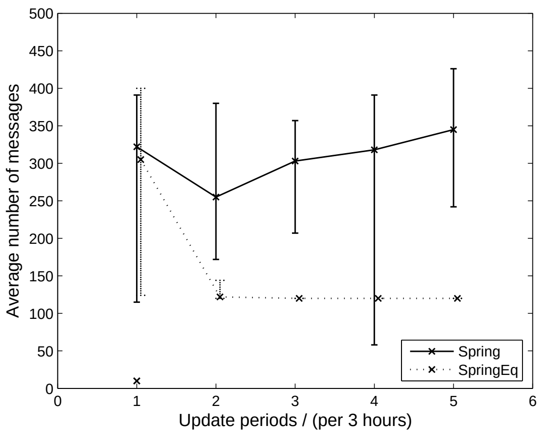

Figure 25: Variation of message overhead with time.

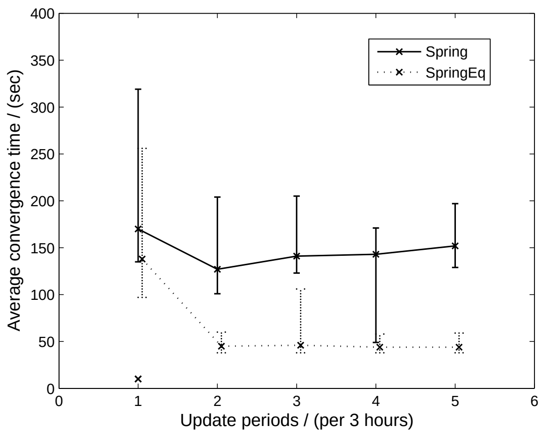

Figure 26: Variation of convergence time with time.

Figure 24 shows the cumulative distribution of the relative distance error at the beginning and the end of the experiment. The level of accuracy for both algorithms improves at the end with the SpringEq algorithm providing slightly better results than the Spring algorithm.

Figures 25 and 26 show the message overhead produced and convergence time required by the algorithms in the periodic re-runs. As expected, SpringEq algorithm show a considerable improvement over Spring algorithm in the re-runs. The first run always costs more due to the random initial coordinates. However, the SpringEq algorithm shows a big improvement in the performance from the second update. While the SpringEq shows a steady, low overhead after the second update, the overhead in Spring algorithm is constantly high and unstable.

5.3. CLAP Performance

This section compares the performance of SLAP and CLAP schemes first using simulations and then using implementations. The simulations run the landmark positioning algorithm once at the beginning and then the node positioning algorithms for 20 times (we call them as "epochs"), with and without CLAP adjustment. Each node maintains SLAP and CLAP coordinates separately. The results presented here are average performance over 10 tests. We assumed that no nodes are malicious and the non-random errors are caused by the congestion at the borders of the clusters. That is, the traffic contained within a cluster are not subjected to non-random errors. In the simulation, the borders of a specific fraction of clusters are considered congested and the values of the pings that crosses those borders are increased by a fixed percentage.

Figure 27 shows the variation of *distance correlation* in the SLAP and CLAP schemes for varying network sizes when 30% of the cluster borders are considered congested. Distance correlation is the correlation between the measured ping and predicted distance (Euclidean distances) values across all the $N \times N$ pairs. The figure shows that CLAP provides much better and more stable performance than SLAP. More interestingly, CLAP provides better performance even when there is no non-random error. It is due to the fact that a given absolute error in predicting short distance has more impact than the same error in predicting long distances. Figure 28 shows the cumulative distribution of relative distance error for the congested and non-congested parts of the network for a 250-nodes system. The performance is better in CLAP and, more importantly, it is same for both congested and non-congested parts of the network. Because all the pings originated from the congested clusters are subjected to non-random error despite their choice of landmarks, the error in the positions of the nodes in those clusters are high. However, the CLAP scheme prioritize the local distances, the effect of the congestion in the borders are minimized to produce a better prediction accuracy.

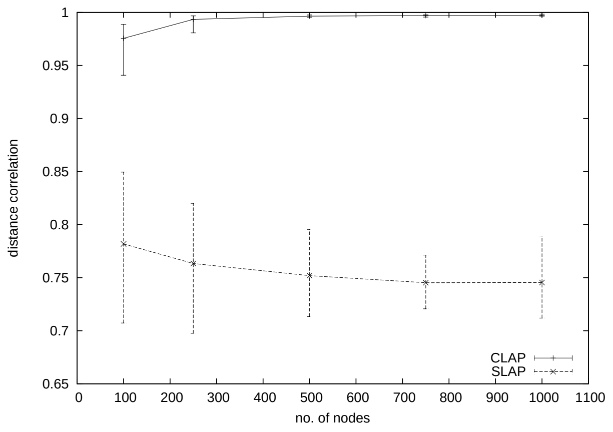

Figure 27: Variation of distance correlation with varying system size.

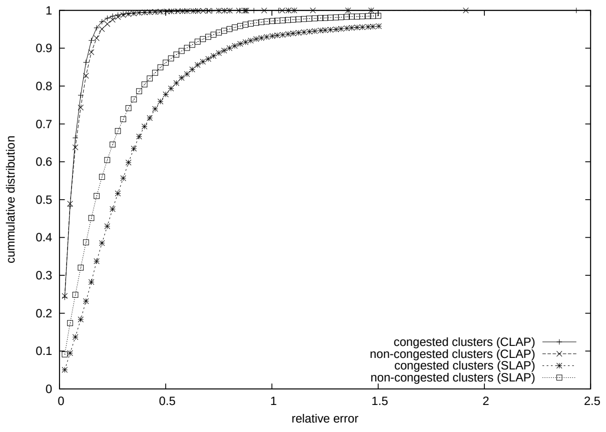

Figure 28: Cumulative distribution of relative prediction error in the congested and non-congested clusters in 250 node system.

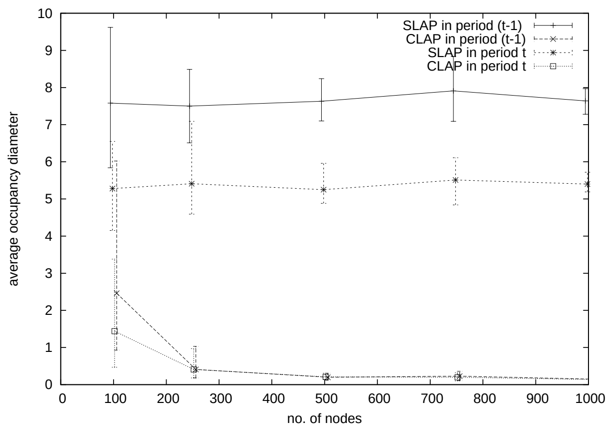

Figure 29: Variation of maximum occupancy diameter with varying system size.

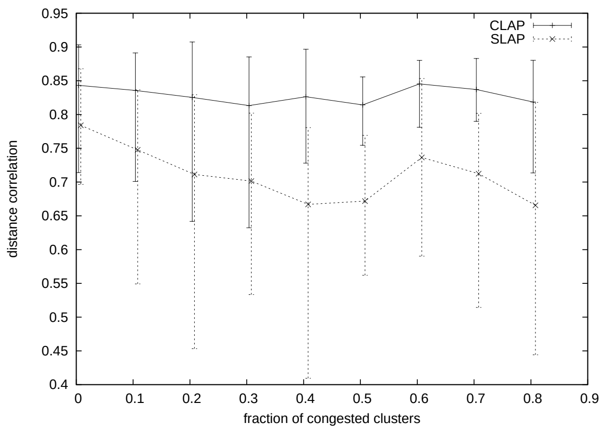

Figure 30: Variation of distance correlation with varying amount of non-random error in the network.

Figure 29 shows the performance of the schemes in terms of maximum occupancy diameter. *Maximum occupancy diameter* of a node is the diameter of the area in which the coordinates of the node fell in the last T epochs, where T is a system defined parameter (it is 4 in our experiment). This metric is a measure to evaluate the robustness of the coordinates. Robustness of the coordinates is important in the network positioning schemes, because the overlay applications using these schemes will be forced to make and break overlay connections based on the coordinates. Unstable coordinates will cause too many unwanted operations causing large application level overhead. Figure 29 shows the maximum occupancy diameters in the system for the last two calculations. It shows that the robustness increases with time for both the algorithms, but CLAP performs better than SLAP in terms of absolute robustness.

In addition to the random network we used for the above results, we also consider a hypothetical network created from the all-to-all PlanetLab node ping values from Stribling's project [7]. The all-to-all ping data for a day is pre-processed to extract the average node-to-node ping values for the nodes that are active throughout the day. Ping values in the synthetic network is assumed to have these average values with manually added probe noise. The AS numbers of the networks in which the active nodes are located are found using the Route View project [8]. The nodes in a single AS are belongs to the same cluster in CLAP. When the number of nodes in a cluster is too small, nodes from adjacent AS's are aggregated into a single cluster. The topology of the network is derived by extracting AS-level connectivity of these networks from the Route View database. In order to simulate the Internet, shortest path algorithm is used to find the route between all the node pairs. When simulating network congestions, not only the congested end clusters, but a congested cluster in the middle of the path of a ping also increases the value of the ping.

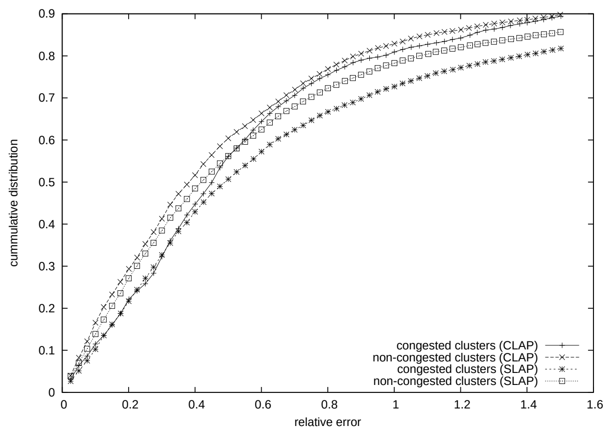

Figure 31: Cumulative distribution of relative prediction error in the congested and non-congested clusters.

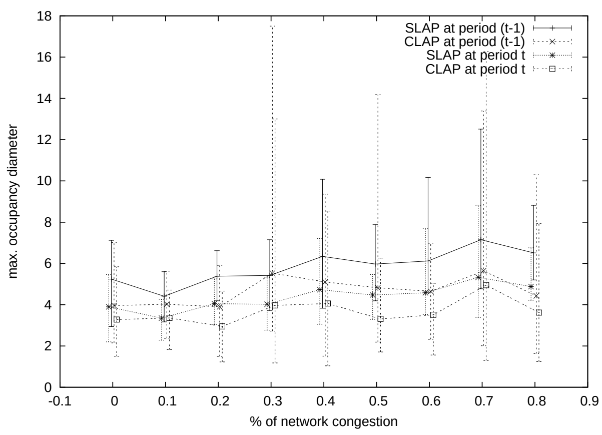

Figure 32: Variation of maximum occupancy diameter with varying amount of non-random error in the network.

Figures 30 – 32 present the performance of the SLAP and CLAP schemes for varying amount of network congestion using this hypothetical network. Even though the performance difference between CLAP and SLAP is not that sharp as in the simulations with random network, the CLAP scheme still outperforms the SLAP scheme. Figure 30 shows the CLAP performance is better than that of the SLAP. Further, the performance range (denoted by the error

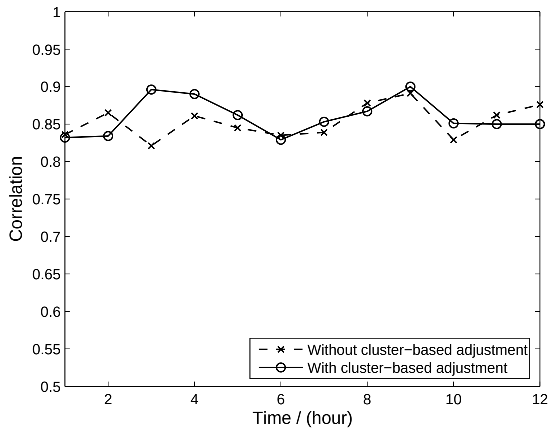

Figure 33: Variation of distance correlation.

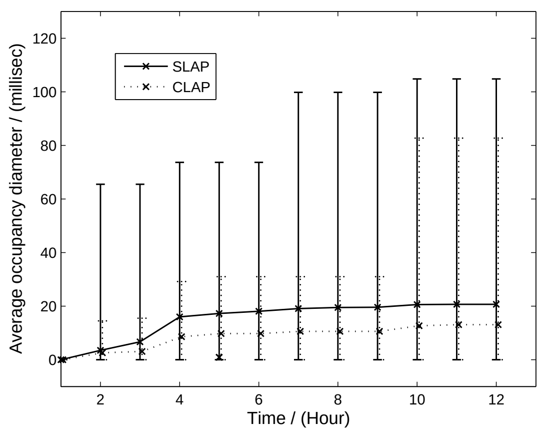

Figure 34: Variation of average occupancy diameter.

bars) shows that CLAP provides a more robust solution.

Figures 33 and 34 show the performance of the SLAP and CLAP schemes in the implementation on the PlanetLab testbed. Even though the performance difference of the schemes in terms of distance correlation is not discernible, the CLAP scheme shows more robust node positions compared to the SLAP scheme. Here the occupancy diameter is calculated in an increasing time window (starting from the beginning of the experiment) instead of moving time window as in the simulation. This is the reason for the the increase in the occupancy diameter after every landmark positioning re-runs. This shows that changing the reference frame has a big impact on the stability of the non-landmark nodes.

6. Related Works

6.1. Predicting Network Delay and Proximity

Internet distance map service relies on a set of infrastructure nodes called *tracers* placed well distributed on the Internet for predicting network delays. Each tracer measures the network delay to all other tracers and every node in the system measures its network delay to the closest tracer. The delay between a node pair is predicted using triangulation via the nearest tracers to the two nodes. IDMaps approach was shown to produce inaccurate predictions especially for short delays ([9]).

Meridian [10] is a *peer-to-peer* (P2P) proximity-based resource discovery mechanism. Each meridian node maintains a fixed number of pointers to a set of other meridian nodes discovered using a gossip protocol. These routing pointers are organized into multiple levels with pointers at each increasing level pointing to nodes at increasing network distance. Another gossip protocol is used for proximity-based resource discovery using these pointers. Meridian's novelty is the real-time proximity detection, however this procedure is costlier than network positioning-based solution in terms of message overhead.

6.2. Network Positioning Schemes

There are two types of network positioning schemes: *binning schemes* and *beaconing schemes*. Even though both of them use landmarks, a binning scheme [11, 12]does not require an explicit landmark positioning for them to create a reference frame. However, binning scheme imposes an explicit order among the landmarks. A normal node pings the landmarks and uses the array of ping distances (in the defined order) as its coordinates. The advantage of a binning scheme is that it is very simple: the landmarks can be completely passive. Also the normal nodes do not require any complex algorithm to position themselves. However, this simple scheme has a number practical concerns: the mapping based on binning can be used to deduce only the proximity information. Additional triangulation-based calculations are required to predict the network delay. Further, binning schemes are more sensitive to landmarks in number of ways: (a) the number of landmarks determines the dimension of the mapping space and this number should be pretty large enough to avoid false clustering of nodes; (b) changing the set of landmarks (members or their order) results in a new coordinate basis requiring the whole system to be reconfigured, which is undesirable in a dynamic environment; (c) the binning scheme is vulnerable for landmark failures.

Binning schemes uses large number of landmarks to avoid the false clustering which leads to high processing costs. *Virtual landmarks* project [13] uses mathematical routines to map the coordinates from high dimensional coordinate space onto another manageable low dimensional space. The new coordinates can be considered as the binned coordinates created using a set of “virtual landmarks.” However, virtual landmarking requires a centralized procedure to find the mapping matrix and a large set of data must be collected to produce a good mapping matrix.

Global network positioning (GNP) [1] is the pioneering work on beaconing-based network positioning schemes. GNP uses Simplex downhill optimization for both landmark and node positioning phases. Landmark positioning is centrally performed in a representative node. GNP is shown to perform better than binning schemes and IDMaps in predicting delays. GNP research establishes many fundamental concepts of network positioning: (a) the node pairs in the Internet that do not satisfy the triangular inequality are the fundamental source of inaccuracy in a beaconing scheme. However, only a little percentage of the node pairs are like that and, thus, a beaconing scheme is generally applicable; (b) choosing a well distributed set of landmarks is better than randomly selecting them; (c) the accuracy increases with number of landmarks used per node, but improvement in accuracy diminishes with the increasing number of landmarks; (d) cardinality of the Cartesian space has the similar effect. A coordinate space with more than four dimension does not show much improvement. The design of GNP has two issues: centralized implementation of landmark positioning algorithm and fixed set of landmarks for all the nodes.

Network positioning system (NPS) [14] extends the GNP design as a distributed scheme using a hierarchical relation among the nodes with a fixed set of "root" landmarks. This hierarchical landmarking prevents nodes forming disjoint clusters and thus multiple disjoint coordinate spaces. NPS design also concentrates on practical issues such as damping the noise in the ping values, periodic updates of the coordinates, maintaining the stability of the coordinates against the topology change, preventing landmarks from getting congested, and handling malicious reference points. One of the primary issue in NPS is that the prediction error is amplified down the hierarchy.

Vivaldi [3] is inspired by GNP, but, uses a spring system model as described Section 3.2. It is a fully distributed algorithm where with no landmarks. Vivaldi shows comparable performance with GNP. However, the convergence of the Vivaldi system depends on the scale factor of the algorithm. Further, introducing a new set of nodes into an already converged system disturbs the whole system. Improved algorithms are presented in [4] to address these problems. However, the best scale factor proposed in the work is only empirical.

Big-bang simulation (BBS) [9] is similar to Vivaldi, but it uses molecular dynamics instead of spring analogy. BBS is more complex than Vivaldi due to the complex potential and kinetic energy calculations and highly likely to produce large message overhead. *Lighthouse* [15] is another distributed network positioning scheme. It does not use any optimization algorithm. Each subset of landmarks chosen by a node form the basis for a new Cartesian space. As this basis is not orthogonal, the node can be positioned in the "local" space using positioning procedure purely algebraic. The coordinates from different local spaces are translated into a global space using a *transition vector* that is calculated by the initial nodes. Lighthouse scheme shows a comparative performance to GNP and its algebraic algorithm looks more attractive than the optimization based approaches. But, how the algebraic solution produces a solution under imperfect conditions is not clear and the same for how well the transition matrix can be transfered across the system.

There are number of schemes mainly addressing the best way to chose the landmarks. *Practical Internet coordinates* (PIC) [16] proposes three landmark selection algorithms, one selecting random nodes as landmarks, the next selecting neighborhood nodes, and the last a hybrid of the other two. PIC shows that the hybrid selection provides an better overall performance. The *PCoord* scheme [17] proposes another set of landmark selection algorithms. Its best landmark selection algorithm uses gossip protocol to get informed about other nodes so that it can select a well-dispersed set of landmarks. PCoord algorithms achieve GNP-level accuracy either with high storage cost (for large amount of route pointers) or increased message overhead.

These existing works on network positioning fail to address message overhead produced by the schemes. Further, they fail to completely analyze the effect of errors in ping values, especially the effect of non-random errors.

6.3. Network Monitoring

Network monitoring services try to predict the behavior of a network segment by collecting information from accessible points of the segment. *Network weather service* (NWS) [18] employs distributed sensor—collector mechanism to forecast the deliverable performance of computing resources and link throughput. *Network tomography* [19] is another type of monitoring service which tries to characterize the internal attributes of a network segment using end-to-end measurements (packet loss, delay, etc.) that crosses the segment. It uses compute-intensive statistical inference techniques. Different traffic measurement and statistical models have to be develop to predict different attributes. Even though network monitoring services can be used to predict network latencies, it is not meant for it and the network positioning scheme can do it with much lower overhead (in terms of computation and message complexities).

6.4. Geo-location

Geo-location services map IP addresses to geographical locations. It is useful for scenarios like targeted advertising, geographical location-based web presentation, or geographical location-based policy modification.

Many commercially used geo-location services like Quova [20] and NetGeo [21] use a huge mapping database extracted with the collaboration of ISPs. The advantages of this type of services are that they are accurate to the zip code level. However, they are not scalable, because the size of the database can grow unmanageably large. Alternative approach is to use in-network queries and measurements. One example is IP2Geo [22]. The common tools used for this approach are DNS names, *whois*, traceroute, and queries on BGP routers. This type of geo-location service generally provides only coarse grain location and always associated with some error.

The *constraint-based geo-location* (CBG) [23] operates similar to network positioning schemes using the ping delays to extract geographical locations. The key problem in CBG is to find the conversion factor between network ping values and the geographical distances. CBG uses a set of landmarks, but here instead of nodes pinging the landmarks, the landmarks pings the node. The geographical location of the host is taken as the centroid of the area covered by the intersection of all the predicted distance circles with landmarks as the centers.

Geo-location services can not be used for predicting network delay or proximity as there is rarely a correlation between geographical and network proximity.

7. Conclusion

In this paper, we proposed two new algorithms for network positioning. We evaluated their performance using simulations and implementation against two most significant algorithms from previous research.

The SpringEq algorithm we proposed for landmark positioning is shown to incur significantly lower message overhead than the previously proposed Spring algorithm. Further, the SpringEq algorithm was found to produce landmark and node positionings that are robust. That is, the subsequent coordinates (in time) computed for a node did not change unless the network conditions have changed between the computations. This property is significant because the coordinates produced by the network positioning schemes will be used by some application level overlays. If the network positioning mechanism changes the coordinates without need, it can lead to very inefficient deployments.

The Clustered LAP algorithm outperformed the simple LAP

without had minimal changes positions computed by Sprin

Our results indicate that the proposed algorithms significantly outperform In this paper, we studied the important topic of network positioning using simulations and implementations.

This paper addressed two important issues in network positioning schemes that is generally overlooked by other schemes: message complexity and abnormal non-random errors in pings (non-random errors). In addressing these issues, this paper makes the following contributions:

- Developed a new algorithm called SpringEq which provides lower message overhead than the existing popular algorithms for landmark positioning.
- Proposed a two step network positioning scheme called CLAP which can (a) greatly reduce the impact of measurement errors using shared knowledge; (b) increase accuracy even in normal cases by giving emphasize on short hop distances.
- Carried out a detailed analysis of three network positioning algorithms both theoretically and using simulation to derive the desired properties of network positioning algorithms: (a) The coordinates of the nodes should change only if there is a permanent changes in the network conditions, but should be robust against the temporary changes; (b) They should always accept the fact that it is impossible to produce a perfect mapping and the convergence should not be affected by the non-zero prediction error in the final position; (c) Producing low error mapping is not enough, but those coordinates have to be robust, i.e. the coordinates of a node must not change for the re-run of the algorithms if the network condition is not changed; (d) When the network conditions are not changed, the amount of overhead should be much lower for the re-runs of the algorithms than the initial run; (e) Convergence criteria can make a big difference so that they have to be chosen wisely.

We chose an infrastructure-based network positioning scheme that a distributed scheme in this research. Our experimental results shows that positioning an ordinary node in an infrastructure-based scheme is very fast. Further, as only a small fraction of total landmark nodes are used by each ordinary node for positioning, the bottle neck and single point failure problems are not that serious as previously argued.

We are building a location-based P2P discovery system on the scheme proposed in this paper to evaluate its applicability.

References

[1] T. S. E. Ng and H. Zhang, “Predicting internet network distance with coordinates-based approaches,” in *INFOCOM 2002*, June 2002.

[2] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical recipes in C : the art of scientific computing*, Cambridge University Press, 2nd edition, 1992.

[3] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, “Practical, distributed network coordinates,” in *The 2nd Workshop on Hot Topics in Networks (HotNets-II)*, Nov. 2003.

[4] F. Dabek, R. Cox, F. Kaahoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *SIGCOMM 2004*, Aug. 2004.

[5] F. J. Blom, “Considerations on spring analogy,” *Interanational Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 647–668, March 2000.

[6] “Planetlab project,” http://www.planet-lab.org/.

[7] J. Stribling, “All pairs pings data for Planetlab,” http://pdos.csail.mit.edu/˜strib/pl_app/.

[8] “Route Views project,” http://routeviews.org/, University of Oregon.

[9] Y. Shavitt and T. Tankel, “Big-Bang simulation for embedding network distance in Euclidean space,” in *INFOCOMM 2003*, April 2003.

[10] B. Wong, A. Slivkins, and E. G. Sirer, “Meridian: A lightweight network location service without virtual coordinates,” in *SIGCOMM 2005*, Aug. 2005.

[11] Z. Xu, M. Mahalingam, and M. Karlsson, “Turning heterogenity to an advantage in overlay routing,” in *INFOCOM*, April 2003.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Topologically-aware overlay construction and server selection,” in *INFOCOM 2002*, June 2002.

[13] L. Tang and M. Crovella, “Virtual landmarks for the internet,” in *Internet Measurement Conference*, Oct. 2003.

[14] T. S. E. Ng and H. Zhang, “A network positioning system for the internet,” in *USENIX*, June 2004.

[15] M. P. J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, “Lighthouses for scalable distributed location,” in *IPTPS 2003*, Feb. 2003.

[16] M. Costa, M. Castro, A. Rowstron, and P. Key, “PIC: Practical internet coordinates for distance estimation,” Techincal Report MSR-TR-2003-53, Microsoft Research, Cambridge, UK, Sept. 2003.

[17] L. Lehman and S. Lerman, “PCoord: Network position estimation using peer-to-peer measurements,” in *Third IEEE International Symposium on Network Computing and Applications (NCA’04)*, Aug. 2004, pp. 15–24.

[18] R. Wolski, N.T. Spring, and J. Hayes, “Network weather service: A distributed resource performance forecasting service for metacomputing,” *Future Generation Computing Systems*, vol. 15, no. 5–6, pp. 757–768, Oct. 1999.

[19] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, “Network tomography: Recent developments,” *Statistical Science*, vol. 19, no. 3, pp. 499–517, 2004.

[20] Quova, “Geolocation,” http://www.quova.com/.

[21] Verifia, “Netgeo,” http://www.netgeo.com/.

[22] V. N. Padmanabhan and L. Subramanian, “An investigation of geographic mapping techniques for internet hosts,” in *SIGCOMM 2001*, Aug 2001, pp. 173–185.

[23] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, “Constraint-based geolocation of internet hosts,” in *SIGCOMM 2004*, Oct. 2004, pp. 288–293.