Distributed Algorithms for Stable and Secure Network Coordinates[*]

Guohui Wang   T. S. Eugene Ng
Dept. of Computer Science, Rice University
Houston, TX 77005, USA

ABSTRACT

Since its inception, the concept of network coordinates has been proposed to solve a wide variety of problems such as overlay optimization, network routing, network localization, and network modeling. However, two practical problems significantly limit the applications of network coordinates today. First, how can network coordinates be stabilized without losing accuracy so that they can be cached by applications? Second, how can network coordinates be secured such that legitimate nodes' coordinates are not impacted by misbehaving nodes? Although these problems have been discussed extensively, solving them in decentralized network coordinates systems remains an open problem.

This paper presents new distributed algorithms to solve the coordinates stability and security problems. For the stability problem, we propose an error elimination model that can achieve stability without hurting accuracy. A novel algorithm based on this model is presented. For the security problem, we show that recently proposed statistical detection mechanisms cannot achieve an acceptable level of security against even simple attacks. We propose to address the security problem in two parts. First, we show how the computation of coordinates can be protected by a customized Byzantine fault detection algorithm. Second, we adopt a triangle inequality violation detection algorithm to protect delay measurements. These algorithms can be integrated together to provide stable and secure network coordinates.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Algorithms, Security, Performance

[*]This research was sponsored by the NSF under CNS-0721990, CNS-0520280, CNS-0435425 and CAREER Award CNS-0448546. 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 NSF or the U.S. government.

Keywords

Network coordinates, security, stability, distributed algorithms

1. INTRODUCTION

Network coordinates provide a lightweight approximation of Internet delays. Since its inception, it has attracted much attention from the networking research and industrial communities. Szymaniak et al. [31] report the deployment of a large scale network coordinates system for delay estimation in the Google content delivery network and forecasts that Google may offer a public interface to the coordinates system. In addition to the delay estimation, researchers have explored the applications of network coordinates in many other areas. To name only a few examples, in [8], [1] and [13], the authors discuss the application of network coordinates in compact Internet routing. Nandi et al. [20] use network coordinates for overlay construction. Lumezanu et al. [18] propose to use network coordinates to discover fast overlay paths. Zhang et al. [36] demonstrate the application of network coordinates in modeling and synthesizing Internet delays at large scales. Bazzi et al. [2] use network coordinates to defend against the Sybil attack in overlay networks.

However, the applications of network coordinates are still significantly limited by two important problems. The first problem is coordinates stability. Many previous studies, such as [6], [24], [14] and [33], have reported the coordinates instability problem in Vivaldi [5], which is the state of the art decentralized network coordinates system. Their results show that even when the network delays do not change, network coordinates in the Vivaldi system are drifting rapidly. The coordinates instability problem makes it hard for applications to use network coordinates. This is because applications cannot cache changing coordinates. When an application process wants to use the network coordinates of other nodes, it has to query their coordinates very frequently. Although previous studies [14, 12] have attempted to reduce the impact of coordinates instability on applications, achieving coordinates stability without losing accuracy in a decentralized system is still an unsolved problem.

The second problem is coordinates security. Previous studies [11, 35] have explored the performance of network coordinates systems in adversarial environments. The results show that, a small fraction of malicious nodes in the system can dramatically degrade the accuracy of network coordinates. This problem becomes especially important when network coordinates are applied in an open network environment, such as peer-to-peer networks, where there may be no global access control in the system. Recent studies [10, 35] have proposed to use statistical detection mechanisms to differentiate good nodes and malicious nodes. However, all statistical detection mechanisms inevitably suffer from false positive and false negative cases. Facing a large number of potential attacks, it remains unclear how well these statistical detection mechanisms could contain malicious nodes.

This paper presents new distributed algorithms to solve the stability and security problems in decentralized network coordinates systems. For coordinates stability, we discuss two abstract models for stabilizing coordinates in decentralized network coordinates systems: stopping coordinates and error elimination. The analytical framework leads to two insights. First, the intuitive model of stabilizing coordinates by stopping coordinates movement has a fundamental problem that hurts coordinates accuracy. Second, the alternative model of stabilizing coordinates by eliminating embedding error can potentially achieve stability while preserving accuracy. We propose a distributed algorithm to closely approximate the error elimination model for stability in the Vivaldi system. The evaluation results demonstrate that, this method can achieve coordinates stability without sacrificing accuracy in decentralized network coordinates systems.

For coordinates security, we empirically study the behavior of existing statistical detection mechanisms. The results show that none of the existing mechanisms can provide satisfactory protection against even simple attacks. We propose to address the coordinates security problem in two parts: securing coordinates computation and securing delay measurement. We show that coordinates computation can be completely protected by a customized Byzantine fault detection technique. We propose a triangle inequality violation (TIV) detection technique that can effectively protect delay measurement. We demonstrate that the distributed algorithms proposed in this paper can be integrated to provide stable and secure network coordinates.

The rest of this paper is organized as follows. Section 2 introduces the experimental methodology we use to study the key questions in this paper. Section 3 addresses the coordinates stability problem and Section 4 addresses the coordinates security problem. Section 5 illustrates a way to integrate the stability and security algorithms. Section 6 provides an overview of the related work. Finally, we conclude in Section 7.

2. EXPERIMENTAL METHODOLOGY

In this section, we introduce the experimental methodology we use in addressing the questions in this paper.

2.1 Experimental Platform

There are many proposals to build network coordinates systems with different architectures, computation methods and performance optimization techniques. We have implemented a configurable network coordinates platform. In this platform, we can choose different components to realize a specific network coordinates system. By changing the configuration, we are able to evaluate the performance of many existing proposals. The platform has the following configurable features:

Algorithm configuration - The platform can be configured to use different algorithms to compute network coordinates. It can also be configured with different mechanisms to optimize system performance.

Environment configuration - The platform can be configured to run in a simulated environment for controlled experimentation, or in a real network environment such as PlanetLab for realism. In the simulated environment, the delay measurements are loaded from a delay matrix. In the real network environment, the delay measurements are performed by actual network probing. To filter out the impact of queuing delay, the delay measurement takes the minimum delay from multiple probes.

2.2 Experimental Setup

Using the configurable platform, we can set up both PlanetLab experiments and simulation experiments. The PlanetLab experiments are used to evaluate the real deployment performance of the system and the simulation experiments are used to set up well controlled scenarios to analyze the system properties in detail. The PlanetLab experiments are deployed on 306 hosts which are all the reachable PlanetLab nodes during our experiment period. The simulation experiments are based on the p2psim data [5] which is the delay matrix among 1740 DNS servers measured by the King tool [7].

2.3 Performance Metrics

We use the following metrics to evaluate the coordinates' accuracy and stability properties.

Relative Error - The relative error metric is used to evaluate coordinates' accuracy. The definition of relative error is:

$$\frac{|measured\_delay - predicted\_delay|}{measured\_delay}$$

Other metrics such as relative rank loss and closest neighbors loss [17] have been proposed to evaluate the accuracy of network coordinates for server selection application. We do not use these metrics in this paper because we are not focused on the coordinates' accuracy for a particular application. We evaluate coordinates' accuracy generally in term of delay prediction.

Average Coordinates Movement - We use an average coordinates movement metric to evaluate the stability of network coordinates. The average coordinates movement is defined in the following way. In a network coordinates system with N nodes, node i moves its coordinates at time t with the distance of $d_i(t)$. During a time period T, the average coordinates movement in the system is

$$\frac{\sum_{i=1}^{N}\sum_{t\in T} d_i(t)}{N}$$

This metric takes into account all the coordinates movements in the system. We normalize the overall coordinates movement by the number of nodes so that the metric is not impacted by the system scale. In our experiments, we monitor average coordinates movement in 10 second periods.

3. COORDINATES STABILITY IN DECENTRALIZED NETWORK COORDINATES SYSTEMS

3.1 Coordinates Stability Problem

We first define the coordinates stability problem. Two points need to be clarified to understand this problem. First, the coordinates stability problem must be considered together with the accuracy of coordinates. If we only consider the stability of network coordinates, we can simply force all the nodes to stop their coordinates computation at random moments to achieve stable coordinates. But the randomly stopped network coordinates are meaningless because they can have very bad accuracy in predicting network delays. Therefore, when we study coordinates stability, we must consider the accuracy of the stabilized network coordinates. Second, we need to define the best accuracy that can be achieved by stabilized network coordinates. The most accurate coordinates we can get from an unstable network coordinates system are the coordinates from a global snapshot after the coordinates computation has converged. Thus, the problem is, can network coordinates be stabilized while preserving this highest level of accuracy in decentralized network coordinates systems?

The coordinates stability problem is investigated with the following assumptions:

(1) The problem is considered in a fully decentralized system. There is no global coordination and no landmarks in the system.

(2) There is no malicious churn in the system. Here malicious churn means that the nodes join and leave the system so frequently that a node will lose most of its neighbors before its coordinates computation converges. We do not study the coordinates stability problem in this environment because there is no way to stabilize network coordinates while preserving good accuracy.

(3) There is no frequent network routing changes. In network coordinates systems, a node collects multiple delay measurement samples and keeps the minimum value to obtain the stable propagation delay. Network coordinates systems use these propagation delays to compute coordinates. If the underlying network routing is not stable and the propagation delays change frequently, it is not possible to stabilize network coordinates.

3.2 The Difficulties of Stabilizing Coordinates

In this section, we discuss the challenges of the coordinates stability problem. Note that coordinates stability is trivial to achieve in a centralized network coordinates system like GNP. In GNP, a central node collects all the delay measurements among landmarks and computes the coordinates of landmarks by an optimization algorithm. The central node can end the coordinates computation at the coordinates that minimize the prediction error. The landmarks can thus easily get stable and accurate coordinates by this centralized computation. Because all the ordinary nodes compute their own coordinates based on landmarks' stable coordinates, they can also achieve coordinates stability easily.

However, in decentralized network coordinates systems, achieving coordinates stability remains an open problem. Many previous studies [6], [24], [14], [12], [33] have discussed the serious coordinates instability problem in the Vivaldi system. Their results show that even when there are no delay changes at all, Vivaldi coordinates are still drifting rapidly. In [6], [14], [12], the authors have proposed techniques to alleviate this problem in the Vivaldi system. Ultimately, the key difficulty in solving the stability problem lies in the fact that there is no global coordination in decentralized systems. Each node only knows the local information, such as its own coordinates and error and its direct neighbors' coordinates and errors. Based on this incomplete information, it is hard for a node to decide when and how to stabilize its coordinates without hurting the overall prediction accuracy.

In [6], the authors propose to add a $loss$ factor to stabilize the coordinates in the Vivaldi system. $loss$ is a value in $[0, 1]$. The coordinates movements in the Vivaldi system are always multiplied by the factor $(1 - loss)$. $loss$ is set to 0 at the beginning. While the system is converging, $loss$ is gradually increased to 1 by the function $loss = c_l + (1 - c_l) \times loss$, where c_l is a constant factor empirically set to 0.02. Finally, when $loss$ reaches 1 on all the nodes, all the coordinates will be stabilized. Although this technique can stabilize the coordinates in the Vivaldi system, the problem is, because nodes stabilize their coordinates independently without considering the overall prediction error, the stabilized coordinates can have very bad accuracy.

In [14], the authors propose to add update filters to the Vivaldi system to reduce the impact of coordinates instability on applications. The idea of the update filter is to have two sets of coordinates on each node, a system level coordinates $\vec{c_s}$ and an application-level coordinates $\vec{c_a}$. The system level coordinates $\vec{c_s}$ keep drifting as the original Vivaldi coordinates. The application level coordinates $\vec{c_a}$ is updated only after the update filter detect a significant change in $\vec{c_s}$. The update filters detect coordinates changes by a statistical heuristic. Although these update filters can reduce the impact of coordinates drift on applications, they are not meant to stabilize network coordinates.

In [12], the authors propose to add a gravity force in the Vivaldi system to limit coordinates drifting. The idea is to apply a polynomially increasing gravity to coordinates as they become farther away from the origin coordinates. Gravity $\vec{G}$ is a force vector applied to a node's coordinates $\vec{x_i}$ after each update: $\vec{G} = \left(\frac{\|\vec{x_i}\|}{\rho}\right)^2 \times u(\vec{x_i})$, where $u(\vec{x_i})$ is the unit vector in the opposite direction of $\vec{x_i}$, ρ is a gravity factor, which is empirically set to 256. It is shown in [12] that the gravity technique can reduce the drifting of the coordinates' centroid. However, the problem is that, the gravity technique only prevents the coordinates space from drifting away from the origin, the coordinates of each node are still moving in a chaotic way as the original Vivaldi.

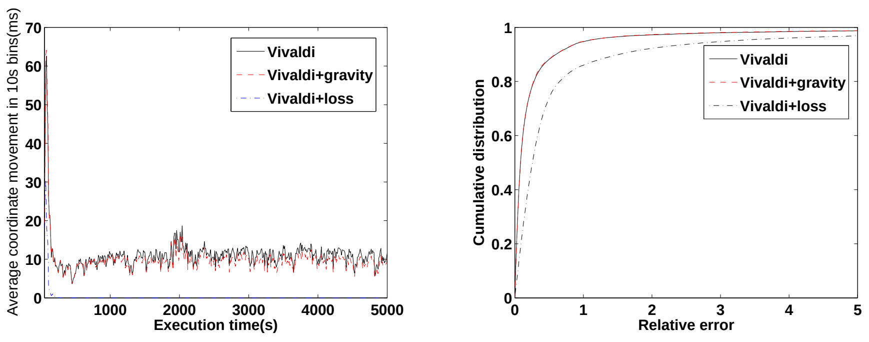
To quantify the performance of the $loss$ factor technique and the gravity technique, we compare these systems to the original Vivaldi system by conducting 5,000 second long PlanetLab experiments. Figure 1 shows the coordinates stability and accuracy of these different techniques. We can see that, although the $loss$ factor technique can stabilize coordinates, it hurts the coordinates' accuracy significantly. On the other hand, the gravity technique does not hurt coordinates' accuracy, but the coordinates' instability is essentially the same as the original Vivaldi system. That is although the gravity technique reduces the drifting of the coordinates' centroid, it does not achieve coordinates stability.

3.3 Analytical Framework for Coordinates Stability

In this section, we discuss two different abstract models for thinking about and understanding coordinates stability.

Stabilizing coordinates by stopping movement - Since the basic goal is to reach a state where the coordinates of all the nodes do not change, the most intuitive model is to stabilize coordinates by stopping movement. The loss factor technique proposed in [6] is an example of this model. Of course, a node should not freeze its coordinates at a random moment. It must do so only when the coordinates' accuracy is high. So, let us for the moment assume an idealized system in which each node has the ability to independently discover that the overall distributed coordinates computation has converged. A node can then choose to freeze its coordinates after it has discovered convergence. Even under this idealized system, the stopping movement model can achieve good accuracy only if all nodes simultaneously freeze their coordinates, which is not possible in decentralized network coordinates systems.

In practice, different nodes' coordinates converge at very different time. All the nodes have to freeze their coordinates based on local information. No matter whether a node freezes its coordinates suddenly or gradually, the problem with the stopping movement model is that a node can only make the decision to freeze its coordinates based on past history. As soon as a node freezes its coordinates, it loses the ability to adapt to future events. Since it is fundamentally not possible for all nodes to simultaneously freeze their coordinates, after a node freezes its coordinates, its prediction errors may be increased because other nodes' coordinates are still moving. For example, consider a node A whose coordinates has been frozen, and one of A's neighbor B is still updating its coordinates to reduce its local error. After some time, B may have drifted away. The prediction error of A's coordinates is thus increased.



(a) (b)

Figure 1: Coordinates stability and accuracy of Vivaldi with gravity and loss factor (a) Stability (b) Accuracy

However, A has stopped updating its coordinates and can no longer react to reduce the error. In summary, stabilizing coordinates by stopping coordinates movement takes away the critical coordinates adaptivity necessary to maintain high accuracy and thus this model cannot solve the stability problem.

Stabilizing coordinates by eliminating error - The coordinates stability problem can be considered from another perspective. All network coordinates systems use some optimization algorithms to compute coordinates that minimize error. An interesting observation is, if the system reaches the state where all prediction errors is 0, the network coordinates will *naturally* be stabilized. If we think about the coordinates stability problem in this way, to achieve stable coordinates, the target is to reach the state where the prediction error is 0. However, because of triangle inequality violations among Internet delays, it is not possible to have network coordinates that can predict Internet delays perfectly.

These observations motivate our second model for achieving stability, that is to stabilize coordinates by artificially eliminating the remaining errors on all the neighbor edges. Let us again assume an idealized system in which each node has the ability to independently discover that the overall distributed coordinates computation has converged. In addition, assume that each node can independently determine the set of coordinates distances D to all its neighbors in a specific snapshot S of the converged coordinates system. In this idealized system, a node can then choose to eliminate the errors on neighbor edges by substituting the set of coordinates distances D for the set of measured delays.

Note that although nodes may not choose to eliminate errors simultaneously, this idealized system will *naturally* stabilize because the artificial distances D are perfectly embeddable. The resulting stable coordinates will have the same accuracy as the specific snapshot S.

The key difference between this model and the previous stopping movement model is that while a node is eliminating its remaining errors, it does not stop updating its coordinates. The node still observes the errors of other nodes and reacts to the coordinates updates of other nodes. Finally when all the nodes remove their error, their coordinates will be stabilized. Therefore this model does not require all the nodes to eliminate error simultaneously. It provides a much more promising way for stabilizing network coordinates in a distributed fashion while preserving overall coordinates accuracy. The obvious remaining question is whether the idealized system can be closely approximated in practice.

3.4 A Novel Algorithm for Coordinates Stability

In this section, we introduce a novel algorithm that applies the error elimination model to achieve coordinates stability.

3.4.1 The Algorithm

Overview - The idea of the algorithm is, during the embedding procedure, each node monitors the local errors to its neighbors to decide whether coordinates computation has converged. When the node decides its coordinates has converged, the node starts to stabilize its coordinates. At this stage, the node knows the remaining errors for all its neighbor edges. When the node tries to stabilize its coordinates, it does not stop the coordinates computation. Rather, the node gradually drives out the remaining errors in the embedding procedure by artificially adjusting the target delays to compensate for the errors. During this procedure, a node still observes the errors to its neighbors. If some of its neighbors have not converged in their coordinates computations, the node can still adapt to the neighbors' coordinates updates and reduce the errors to these neighbors. After all the nodes have eliminated the remaining errors, all the coordinates can be stabilized naturally. Generally, this algorithm can be applied to any decentralized network coordinates systems. In this section, we specifically show how this algorithm can be applied to the Vivaldi system.

Local error monitoring - The purpose of local error monitoring is for each node to learn when its coordinates computation has converged, and what is the remaining errors on its neighbor edges. From this information, the node can decide when to stabilize its coordinates and how much error should be eliminated in the stabilizing procedure. For an arbitrary node A in the Vivaldi system, A has N neighbors. During each round of its embedding procedure, node A records four data items to monitor the status of its embedding procedure: (1) The prediction error of the i^{th} neighbor

$$\epsilon_i = predict\_delay(A, i) - measured\_delay(A, i)$$

(2) The weighted average error of the i^{th} neighbor, which is defined to be

$$\bar{\epsilon_i} = \alpha\bar{\epsilon_i} + (1-\alpha)\epsilon_i$$

where α is the weight in computing the weighted average value, which is set to 0.9. The first two data items can be used to learn the current remaining errors for all the neighbor edges in the embedding procedure;

(3) The average error of all the neighbors

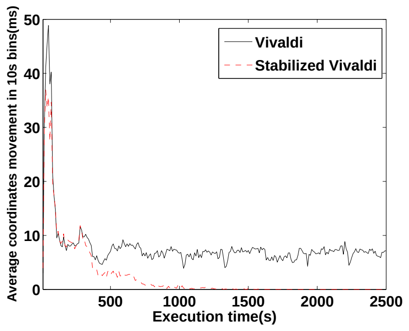
$$\epsilon = \frac{\sum_{i=1}^{N} |\epsilon_i|}{N}$$

(4) The weighted average of the average neighbor error, which is defined to be

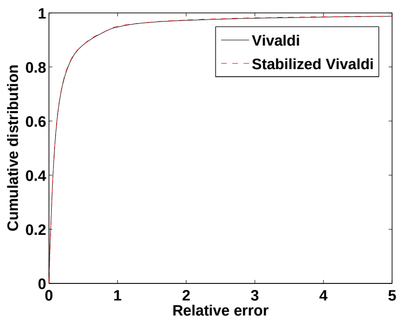
$$\bar{\epsilon} = \alpha\bar{\epsilon} + (1-\alpha)\epsilon$$

These two data items are used to monitor the convergence procedure of A's coordinates.

Stabilizing coordinates - To stabilize the network coordinates, we define two states in the spring algorithm of the Vivaldi system:

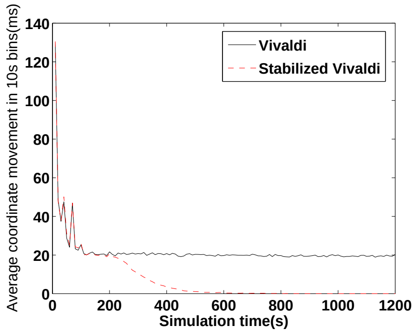


(a)

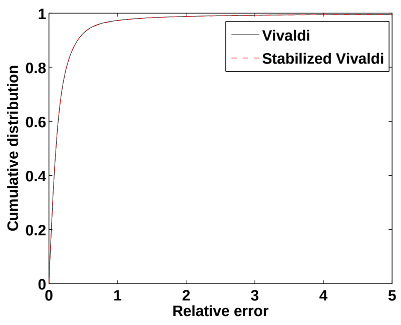


(b)

Figure 2: Performance of Stabilized Vivaldi in a PlanetLab experiment (a) Stability (b) Accuracy



(a)



(b)

Figure 3: Performance of Stabilized Vivaldi in a simulation experiment on p2psim data (a) Stability (b) Accuracy

the normal state and the stabilizing state. In the normal state, node A just computes its coordinates normally as the original spring algorithm. A needs to check its observation data items $\{\epsilon_i, \bar{\epsilon_i}, \epsilon, \bar{\epsilon}\}$ to decide whether its coordinates have converged. Many strategies can be used to make the decision. Here, we just use a simple strategy: if A finds $\bar{\epsilon}$ cannot be decreased for L rounds, it decides the coordinates have converged and enters the stabilizing state to stabilize the coordinates.

In the stabilizing state, node A still updates its coordinates. However, A will compensate for the remaining error of each neighbor edge to compute the coordinates. More specifically, in the stabilizing state, A uses the compensated delay of all the neighbors to compute the coordinates. For the i^{th} neighbor,

$$compensated\_delay(A, i) = measured\_delay(A, i) + \bar{\epsilon_i}$$

Since node A only eliminates the weighted average error of its neighbor edges, if one of its neighbor's error changes rapidly, A can still adapt to the changes. When all its neighbors' errors are stabilized, A will eliminate all these errors and stabilize its coordinates.

3.4.2 Evaluation

We use both PlanetLab and simulation experiments to evaluate the coordinates stability and accuracy after we apply the stabilizing algorithm to the Vivaldi system. The PlanetLab experiment runs on 306 PlanetLab nodes, all nodes join simultaneously. To avoid overloading the PlanetLab by continuous probings, the steps of coordinates update are separated by a random delay of up to 5 seconds. Figure 2 shows the results for the PlanetLab experiment. Figure 2 (a) shows that our stabilizing algorithm can completely stabilize the coordinates within 1500 seconds. This translates to, on average, it takes 154 steps for a node to stabilize its coordinates. The result in Figure 2(b) shows that the stabilized coordinates have the same accuracy as a snapshot of the Vivaldi coordinates in the PlanetLab experiment. To show how the algorithm performs with a larger number of nodes, we run a simulation experiment on the p2psim data, which is a relatively large delay matrix with 1740 nodes. Again, all nodes join simultaneously. To make a comparison possible, we keep the same methodology that adds a random delay of up to 5 seconds between coordinates update steps. Figure 3 shows the coordinates stability and accuracy of the stabilized Vivaldi system in the simulation experiment. From Figure 3(a), we can see that, our stabilizing algorithm can completely stabilize the network coordinates within 600 seconds. On average, it takes 126 steps for a node to stabilize its coordinates. The accuracy of the stabilized coordinates is also the same as that of a snapshot of the Vivaldi coordinates.

Note that the long 600 seconds and 1500 seconds stabilizing times are mostly caused by the random delay (up to 5 seconds) between coordinates update steps. In practice, the update steps can have smaller gaps, and nodes can stabilize their coordinates much faster. Also, in PlanetLab, it takes longer to stabilize the coordinates because some PlanetLab nodes are occasionally unreachable during our experiment (possibly due to high load). Our system reacts to these failures automatically by updating affected Vivaldi nodes' neighbor sets. These changes in neighbor sets lead to changes in prediction errors, resulting in a longer stabilization time. This experiment unexpectedly allows us to show that the stabilizing algorithm is able to maintain strong stability even with some neighbor sets churn.

3.4.3 Adapting to Delay Changes

Two types of delay changes occur in practice: the short-term delay changes caused by queuing delays and the long-term delay changes caused by network routing changes. Network coordinates cannot adapt to the short-term delay changes because they can ex-

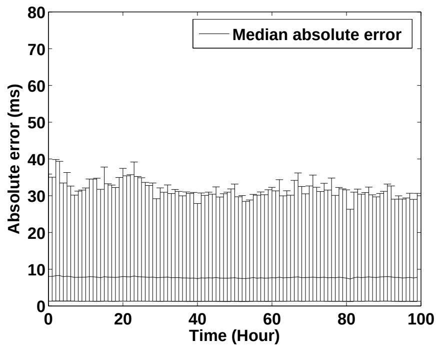


Figure 4: The accuracy of Stabilized Vivaldi coordinates in predicting dynamic delays among PlanetLab nodes. Error bar shows 10%, median and 90% values.

pire quickly even before the network coordinates can converge in adapting to them. Therefore, the strategy to deal with them is taking the minimum RTT from multiple probes to filter out the short-term fluctuations in delay measurements.

Network coordinates should be able to adapt to long-term delay changes caused by network routing changes. The original Vivaldi system can adapt to network changes easily because coordinates are recomputed and changed perpetually, but the coordinates instability brings serious problems to applications. For stabilized Vivaldi system, although the pattern of Internet delay changes is still unclear, our PlanetLab experiment shows that the stabilized network coordinates do not have to adapt to delay changes very frequently. In our experiment, we deploy the stabilized Vivaldi system over PlanetLab nodes, and get a set of stabilized Vivaldi coordinates $\{C_i\}$. After that, we continuously measure the delays among these PlanetLab nodes for 100 hours. Each edge is measured once per hour. Figure 4 shows the 10 percentile, median, and 90 percentile absolute error of $\{C_i\}$ in predicting the dynamic delays among PlanetLab nodes at different time. The absolute error is defined to be $|measured\_delay - predicted\_delay|$. From this graph, we can see that, the stabilized coordinates $\{C_i\}$ have almost the same accuracy in predict dynamic delays within 100 hours. This result shows that, after network coordinates are stabilized, they can be used for a long time to predict dynamic delays accurately. Therefore, stabilized network coordinates do not need to keep updating to adapt to network changes. Different strategies can be used to adapt to delay changes. One simple strategy is to adapt to delay changes by periodical recomputation. The PlanetLab experiment result shows that, the recomputation period can be set to a long time (in the magnitude of days). Stabilized coordinates can also adapt to the delay changes by on-demand triggered re-computation. After the coordinates are stabilized, nodes can continue to observe its delays and errors to other nodes. The system can trigger coordinates recomputation if significant error changes are detected.

However, the coordinates recomputation must be managed as a system level behavior. Nodes cannot trigger the coordinates recomputation independently. The reason is that, different nodes may have different clocks and observe different delay changes. If every node handles coordinates recomputation independently, there will always be some nodes recomputing their coordinates, which will cause inherent instability in the system. A strategy is to use one membership server to manage the members in the system, and control the coordinates recomputation. Since the membership server only handle the registration messages from nodes and maintain a member list, it will not have high overhead. The membership server does not need to trigger the coordinates recomputation on all the

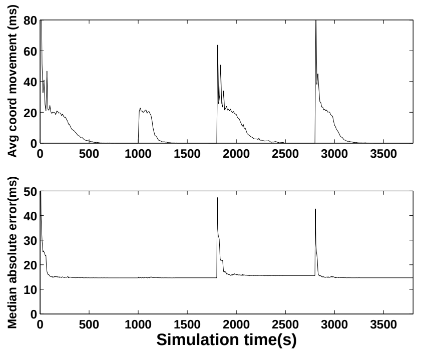


Figure 5: Coordinates stability and accuracy of stabilized Vivaldi under network changes

nodes. It only need to trigger the recomputation on one or a small number nodes. These small number of nodes can trigger the recomputation of their neighbors, and the neighbors trigger the next hop neighbors. The recomputation triggering can be piggybacked on neighbor probing packets. Since each node typically has tens of neighbors, the recomputation can be flooded throughout the whole system in very short time.

We set up a simulation experiment to demonstrate the coordinates recomputation. We run stabilized Vivaldi system on 500 nodes that are randomly sampled from p2psim data. Each node has 32 neighbors. Simple strategies are used for triggering coordinates recomputation. If there is no dramatic delay changes, the coordinates will be recomputed every 1000s. In this experiment, we use small recomputation period in order to shorten the simulation time. For on-demand triggered recomputation, if a node detects a 5-ms median error change, it will send a recomputation request to the membership server. The membership server will trigger coordinates recomputation if it receives requests from more than 30% of nodes. We simulate a scenario of dramatic delay changes. At 1800 second simulation time, the delays among the 500 nodes are changed to another delay matrix that is randomly sampled from p2psim data. At 2800 second, the delays are changed back to original values. In this experiment, we find that, for periodical recomputation, if the membership server triggers the recomputation from one node, it only takes 2.1s for the recomputation to be flooded throughout the whole system. When delays are dramatically changed, it only takes 17s for the nodes to detect the delay changes and trigger the on-demand recomputation. Figure 5 shows the coordinates stability and accuracy, in which the top graph plots the average coordinate movement, and the bottom graph shows the median absolute error of the coordinates during the simulation period. From this graph, we can see that, for all the cases, the recomputation can be triggered quickly, and the coordinates can be stabilized again with good accuracy in a short period of time.

4. COORDINATES SECURITY IN DECENTRALIZED NETWORK COORDINATES SYSTEMS

4.1 Coordinates Security Problem

As network coordinates systems are deployed in open network environment, they must be able to handle many unexpected scenarios. One important problem is how to handle malicious attacks. Malicious nodes may have different incentives to attack network coordinates systems. Since network coordinates can be used to provide network proximity for applications, the disruption of network

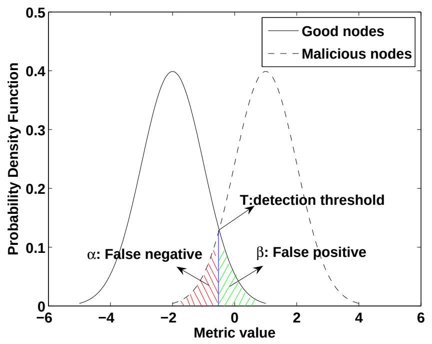


Figure 6: Demonstration of metric distributions for statistical detection

coordinates could result in the malfunctioning of applications that use network coordinates. For example, when network coordinates are used in geometric routing [8, 1, 13], the disrupted network coordinates can totally destroy the routing in the network.

Furthermore, malicious nodes can attack existing decentralized network coordinates systems easily. They can simply provide wrong coordinates and/or delays to mislead other nodes in the system. Previous work [11, 10, 35] has studied the performance of network coordinates systems in adversarial environments. The results show that without protection, a small fraction of malicious nodes can significantly degrade the accuracy of legitimate nodes' coordinates. Thus, the coordinates security problem can be stated as, how can the system be protected such that the accuracy of the good nodes' coordinates with respect to each other is unaffected by the behavior of the malicious nodes?

4.2 Behavior of Statistical Detection Mechanisms

Several previous studies [10, 35] have proposed to use statistical detection mechanisms to identify misbehaving nodes. We study the coordinates security problem by first understanding how well statistical detection can protect network coordinates.

The basic idea of statistical detection mechanisms is to use statistical metrics to differentiate good nodes from malicious nodes. Ideally, if good nodes and malicious nodes present totally different statistical behavior and the statistical metrics distributions for good nodes and malicious nodes have no overlap, then we can set a threshold to perfectly distinguish the good nodes from the malicious nodes. However, in practice, the metric distributions always have an overlapping area. Figure 6 demonstrates an artificial example of metric distributions for good nodes and malicious nodes. If T is the detection threshold used, then the shadowed areas represent the false negative rate α and the false positive rate β.

To apply a statistical detection mechanism to protect a network coordinates system, each node checks its neighbors using the detection mechanism before relying on them as references to compute coordinates. If a node A detects one neighbor B to be malicious, this neighbor B will be discarded. To maintain a fixed size neighbor set, the node A will randomly find another neighbor to replace B. Different detection mechanisms may use different models for detecting malicious neighbors.

Because of the false negative and false positive cases, a statistical detection mechanism cannot detect all the malicious nodes. On one hand, the false negative cases would miss some malicious nodes; on the other hand, the false positive cases would falsely remove good nodes and can potentially introduce more malicious nodes into a node's neighbor set. While the detection procedure is

Attack / Percentage	0%	5%	10%	20%	30%
Shifting	14.76	20.06	23.93	28.72	36.36
Delay	14.76	17.56	22.07	35.03	50.25

Table 1: Median absolute error (ms) of Vivaldi coordinates with different percents of attackers

running, the distribution of statistical metrics can shift. Therefore, the false negative and false positive rate can change. For existing statistical detection mechanisms, it is hard to predict the evolution of their detection performance. Thus, an empirical study is needed to further understand the dynamic behavior of existing statistical detection mechanisms.

4.3 Empirical Study of the Existing Statistical Detection Mechanisms

4.3.1 Basic Principles

We first introduce the basic principles of existing statistical detection mechanisms.

The Kalman filter detection mechanism - The Kalman filter mechanism [10] uses a set of trusted nodes called surveyors as external references to detect malicious nodes. Surveyors can only choose surveyors as their neighbors, which sets up a clean network coordinates system without malicious nodes. Each surveyor trains a Kalman filter to model the error convergence features it observes. The result is a set of Kalman filter parameters. Ordinary nodes may have malicious nodes in their neighbor sets. Every ordinary node retrieves the Kalman filter parameters from its nearest surveyor and uses them to predict its own neighbors' error. If the actual error with respect to a neighbor deviates too much from the predicted value, the neighbor is deemed malicious.

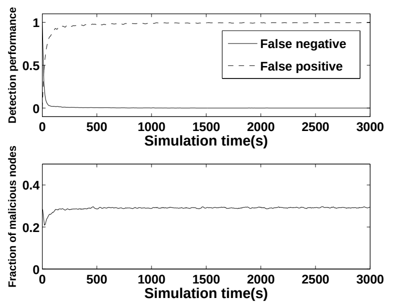
The Kalman filter model is an recursive filter that estimates the state of a linear dynamic system from a series of incomplete and noisy measurements. For a node i, at its n^{th} embedding step, the node uses its neighbor j to update its coordinates. Before the update, the coordinate of i is C_i and the coordinate of j is C_j, the RTT between i and j is RTT_{ij}. The measured relative error at step n is $D_n = \frac{|||C_i - C_j|| - RTT_{ij}|}{RTT_{ij}}$. The nominal relative error at embedding step n is denoted by Δ_n, which is the system state. The Kalman filter uses the following equation to model the measured relative error and nominal relative error.

$D_n = \Delta_n + U_n$

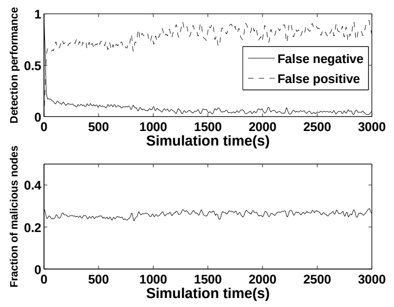
$\Delta_{n+1} = \beta\Delta_n + W_n$

Where β is the recursive factor of the linear dynamic system, U_n is a Gaussian random variable with mean zero and variance v_U, which represents the error measurement noise. W_n is a white Gaussian process with mean $\bar{w}$ and variance v_W, which represents the error fluctuation. This Kalman filter model can be presented by a parameter set $\{\beta, \bar{w}, v_W, v_U, w_0, p_0\}$. Here, w_0 and p_0 are the original states of the system.

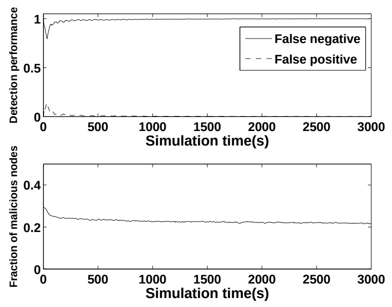
The outlier detection mechanism - Unlike the Kalman filter mechanism, the outlier detection mechanism [35] does not assume any trusted external references. Every node totally relies on its own observations to detect malicious neighbors. The idea is that, at each embedding step, a node records several observed features of its neighbors, such as the neighbor's coordinates movement and error. The node then computes the Mahalanobis distance between the current neighbor's record and the centroid of all neighbors' records. The Mahalanobis distance is a weighted Euclidean distance, which uses the correlations of different dimensions as weights. If a neighbor deviates too much from the centroid, it will be deemed as malicious. Two outlier detection techniques are proposed in [35]: spa-



(a)

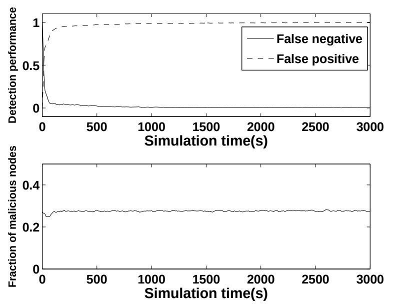


(b)

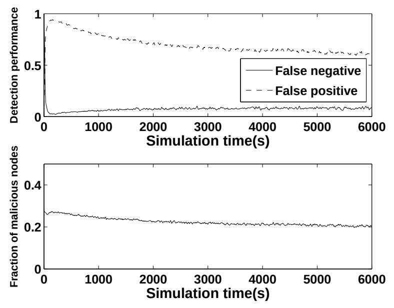


(c)

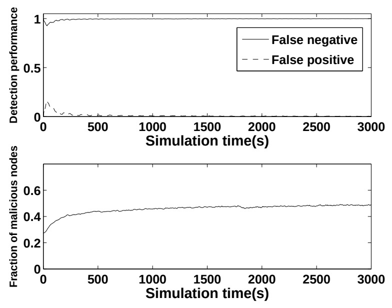
Figure 7: Detection performance for shifting attack (a) Spatial outlier detection (b) Temporal outlier detection (c) Kalman filter detection



(a)



(b)



(c)

Figure 8: Detection performance for delay attack (a) Spatial outlier detection (b) Temporal outlier detection (c) Kalman filter detection

tial outlier detection and temporal outlier detection. In spatial outlier detection, only the most recent neighbor observations are used to compute the centroid; while in temporal outlier detection, all the previous observations are used to compute the centroid.

4.3.2 Malicious Attacks

In a network coordinates system, a malicious node can lie about its coordinates and delays to other nodes. It can generate faked coordinates in any arbitrary way. It can also inflate its delay to other nodes. Since it is not possible to exhaust all attacks, we systematically design a set of attacks that stresses different aspects of the system. The attack set includes six attacks: Shifting attack, Isolation attack, Inflation attack, Deflation attack, Oscillation attack and Delay attack. We evaluate the performance of existing statistical detection mechanisms on all these attacks. Due to space limit, in this paper, we only discuss the most interesting results for two simple attacks. A complete set of results is provided in a technical report [34].

Shifting attack - In the shifting attack, a malicious node always shifts its own coordinates by a random distance in a fixed direction and reports the shifted coordinates to other nodes. The shifting attack can mimic the coordinates movement of normal nodes. In our experiments, malicious nodes shift their coordinates by up to 300ms.

Delay attack - In the delay attack, malicious nodes focus on attacking the delay measurement. A delay attacker randomly inflates the delays to victims for up to 300 ms. The delay attackers compute their coordinates honestly based on the inflated delays.

We apply these attacks to the Vivaldi system. Table 1 shows the median absolute error of legitimate nodes' coordinates with different percents of malicious nodes in simulation experiments. We can see that the simple attacks can significantly degrade the accuracy of network coordinates.

We use simulation experiments to study the behavior of the Kalman filter and the outlier detection mechanisms. Each node randomly selects 32 neighbors and uses a 5D Euclidean space to compute coordinates. We always inject 30% of malicious nodes into the system. 30% may seem to be a large fraction, but a successful solution must be able to tolerate a large-scale attack. For the outlier detection mechanisms, we use the recommended detection thresholds, which are 1.5 for spatial outlier detection, and 4.0 for temporal outlier detection. For the Kalman filter detection mechanism, the detection threshold is computed by the Kalman filter parameters, which is different for different nodes. In the Kalman filter experiments, we randomly select 8% of the nodes as surveyors, which is recommended in [10]. To monitor the evolution of the statistical detection procedure, we set up an observation pool that always records the results of the most recent 10000 detections. From this observation pool, we can monitor the dynamic evolution of false negative, false positive rate and the fraction of malicious nodes in neighbor set.

4.3.3 Experimental results

Figure 7 shows the performance of all the detection mechanisms on shifting attack. In these figures, the top graph plots the evolution of false negative and false positive rates, and the bottom graph shows how the fraction of malicious nodes in neighbor sets changes during the experiment. From these graphs, we can see that none of the statistical detection mechanisms can effectively detect shifting attackers. To understand the reason for these results, Figure 9 plots the metric distributions of all the detection mechanisms on good nodes and shifting attackers. We can make two interesting observations.

First, since the Mahalanobis distance metric in outlier detection

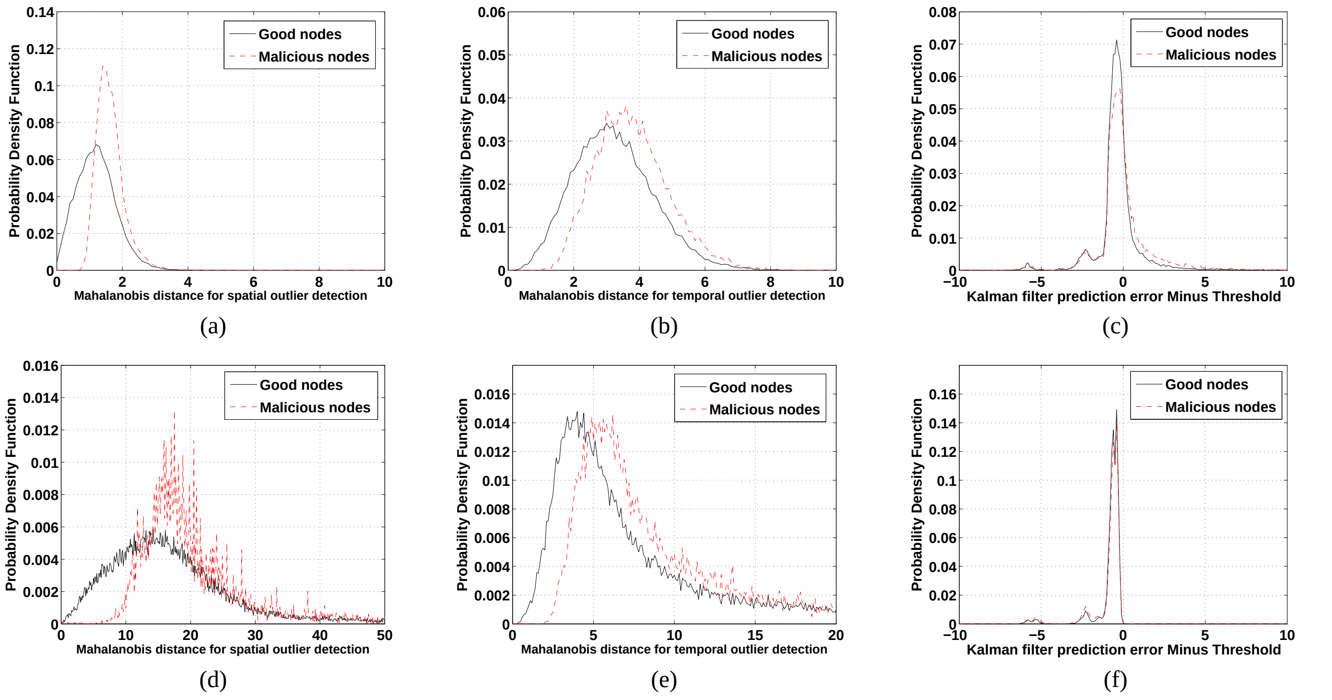


Figure 9: Metric distribution of detection mechanisms for shift attack (a) Spatial outlier detection at time 0 second (b) Temporal outlier detection at time 0 second (c) Kalman filter detection at time 0 second (d) Spatial outlier detection at time 2000 second (e) Temporal outlier detection at time 2000 second (f) Kalman filter detection at time 2000 second

is extracted from the network coordinates that are being attacked, when malicious nodes attack the system for a long time, the metric distribution can be significantly impacted. We can see from Figure 9 (d) and (e) that, after the malicious nodes have attacked the system for 2000 seconds, the Mahalanobis distance metric for spatial outlier detection and temporal outlier detection are significantly shifted to very large values (note the x-axis scales are different). The shifting of metric distribution significantly impacts the performance of outlier detection. Since the outlier detection mechanisms use fixed threshold to detect malicious nodes, while the Mahalanobis distances for good nodes and malicious nodes are all shifted to large values, the false positive rate becomes larger and larger. Finally, the outlier detection mechanisms become so aggressive that they falsely identifies most good nodes as malicious and introduces more malicious nodes into nodes' neighbor set during neighbor replacement procedure. On the contrary, since the Kalman filter detection mechanism uses an external reference model learned from trusted nodes, the metric distribution is not dramatically impacted by malicious attacks.

Second, we can see from Figure 9 that, from the beginning of the experiment to 2000 second, all the metric distributions have large overlapped areas between good nodes and shifting attackers. For outlier detection mechanisms, these results reveal that, the metric distribution shifting is only one problem. Another more important problem is that the Mahalanobis distance metric cannot differentiate good nodes and shifting attackers. The reason why the outlier detection metrics fail on the shifting attack is that, both spatial outlier detection and temporal outlier detection mechanisms rely on coordinates movements to detect malicious nodes. If the malicious nodes' coordinates move wildly, they are more likely to be detected. However, in the shifting attack, the malicious nodes can mimic normal coordinates movements. This makes it hard for outlier detection mechanisms to identify these attackers.

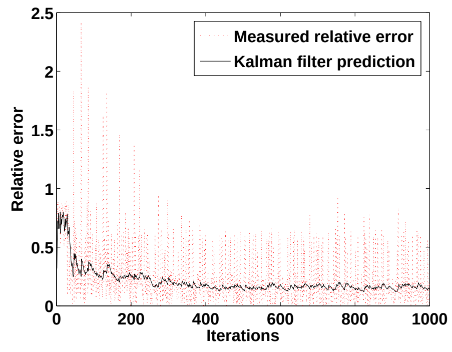


Figure 10: The prediction error of Kalman filter model on a surveyor

For the Kalman filter detection mechanism, we can see from Figure 9 (c) and (f) that, the metric distributions are always very similar on good nodes and shifting attackers. After removing a small fraction of malicious nodes that are beyond detection threshold, the Kalman filter prediction error metric loses its power to differentiate good nodes and malicious nodes. Therefore, the detection procedure ends up with very high false negative rate and a large fraction of malicious nodes remaining in neighbor set. The reason why the Kalman filter detection mechanism does not work well is that the Kalman filter model itself yields large error in predicting the relative error of neighbor edges. The error mainly comes from two sources. First, we observe that, the relative error of neighbor edges has a large variance. The Kalman filter is an adaptive weighted averaging filter. When the relative error has a large variance, even when a node learns the Kalman filter model from its own experienced error, it is still hard to predict its future error. To demonstrate this observation, Figure 10 shows prediction error of the Kalman filter model on a surveyor. The surveyor trains the Kalman filter parameter from its recent neighbors' error, and uses this filter to predict its current neighbor's error. From this graph, we can see that, because the measured relative error has a large variance, the Kalman filter model can only predict the general trend of error curve, but it cannot predict particular neighbor error values accurately. Second, the median delay among ordinary nodes and their surveyors is 21 ms. When a node uses the Kalman filter model learned by a remote surveyor node to predict its own neighbors' error, the prediction error is even higher.

Figure 8 shows the results for the delay attack. We can make the same observation that, all the statistical detection mechanisms fail to mitigate the delay attack. In the delay attack, malicious nodes are honest in computing their coordinates. Therefore the delay attackers exhibit normal behavior in terms of coordinates movement and thus the outlier detection mechanisms fail. The Kalman filter detection mechanism works especially poorly on the delay attack. It actually increases the fraction of malicious nodes in neighbor sets. The reason is that Kalman filter detection is based on the relative error of neighbor edges. However, in the delay attack, malicious nodes attack the system by inflating their delays to other nodes, which, because of how relative error is defined, in most cases, makes the malicious nodes exhibit smaller relative error than good nodes.

4.4 Securing Network Coordinates in Two Phases

The existing statistical detection mechanisms try to detect all types of attacks by a single metric. However, our evaluation results reveal that they cannot provide satisfactory protection against even simple attacks. This indicates the difficulty of detecting arbitrarily behaving malicious nodes by a small number of statistical features. In this section, we address the coordinates security problem by a different approach. Instead of defending all types of attacks as a whole, we decouple the coordinates security problem in two parts. In each round of the coordinates embedding procedure, a node takes two steps to update its coordinates. The first step is delay measurement, in which the node probes its neighbors to measure the propagation delay. The second step is coordinates computation, in which the node uses an optimization algorithm to compute its own coordinates based on the neighbors' delays and coordinates. To secure network coordinates, both steps must be secured. Therefore, the coordinates security problem has two parts: securing coordinates computation and securing delay measurement.

4.4.1 Securing Coordinates Computation by Byzantine Fault Detection

Let us at this moment assume that malicious nodes are honest in delay measurements. They attack the network coordinates by only reporting faked coordinates to other nodes.

Coordinates computation can be formalized as follows: for a node A, at each embedding step i, its coordinates can be computed by $c_i = f(c_{i-1}, \{c_i^k, d_i^k\}, s_i)$, where c_i is A's coordinates at the i^{th} embedding step, $\{c_i^k, d_i^k\}$ are node A's neighbors' coordinates and delays, and s_i is the random seed used in the optimization algorithm $f(\cdot)$. At the beginning of the embedding procedure, c_0 is initialized to the origin of the metric space. Note that, the spring algorithm in Vivaldi requires the neighbors to reply an estimated error for the purpose of computation convergence. This error value can be treated as an extra dimension of the neighbors' coordinates. The optimization algorithm $f(\cdot)$ used in coordinates computation is deterministic. Therefore, *fundamentally, the coordinates computation can be protected by well-known Byzantine Fault Tolerance (BFT) [3] or Byzantine Fault Detection (BFD) [9] techniques*. BFT and BFD techniques represent different design points in defending against Byzantine faulty nodes in distributed systems. BFT techniques can tolerate f Byzantine faulty nodes out of $3f+1$ nodes and prevent them from attacking the system. But BFT techniques have large communication overhead and scale poorly. BFD techniques detect Byzantine faulty nodes after they have attacked the system, which is insufficient to deal with faults that have irreversible effects. But BFD techniques offer better efficiency and scalability. In the case of coordinates computation, BFD techniques are suitable since network coordinates can be recomputed after a faulty node is detected. In the rest of this subsection, we show that one BFD technique, PeerReview [9], can be applied to protect coordinates computation.

PeerReview - To apply PeerReview to a certain distributed system, the following general requirements must be satisfied:

(1) Any node n can be modeled as a deterministic state machine S_n. Each node has access to a reference implementation of all S_n. The implementation can create a snapshot of its state, and its state can be initialized according to a given snapshot.

(2) Each node is associated with a set of witnesses. For a node n, the witness set is denoted $w(n)$. The set $n \cup w(n)$ must contain at least one correct node.

(3) A message sent from one correct node to another is eventually received, if retransmitted sufficiently often. Each node has a public/private key-pair bound to a unique node ID. Nodes can sign messages, and faulty nodes cannot forge the signature of a correct node.

PeerReview uses the following mechanisms to verify the behavior of a node:

(1) Tamper-evident log: Each node keeps a secured log of its own behavior. The log is secured by a hash chain. A log commitment protocol is used to ensure that a node cannot add or hide messages it sent and received in its log. Every time when a node communicates with another node, it has to send the node the corresponding log entry signed by its private key, which is called an authenticator.

(2) Auditing and consistency verification: Suppose a node n is associated with a set of witnesses $w(n)$. Each witness w of n will periodically challenge n to return all the log entries since the last audit. Then w can create a local copy of n's log. w can audit n's behavior by replaying the reference implementation of n's state machine with the same input in n's log. If n does not follow the state machine correctly, it will be detected as faulty. The witnesses uses n's signed log entries as verifiable evidence for faulty behavior. Witnesses use a consistency protocol to verify the information in n's log entries. Suppose node n has communicated with a set of nodes $p(n)$. The witnesses collect all the authenticators n has sent to $p(n)$, and thus know whether n has lied about other nodes' information or sent faked information to other nodes. To prevent node n from colluding with nodes in $p(n)$, each witness w will also contact the witnesses of every node in $p(n)$.

A node is called *detectably faulty* if it breaks the state machine in a way that affects a correct node; a node is called *detectably ignorant* if it never acknowledges that it received a message sent by a correct node. It has been proven in [9] that PeerReview can achieve the following security guarantees:

(1) Completeness: Eventually, every detectably ignorant node is suspected forever by every correct node and every detectably faulty node is detected or forever suspected by every correct node.

(2) Accuracy: No correct node is ever mistaken as a faulty node by a correct node.

Suitability of PeerReview - Let us now show that a network coordinates system satisfies all the PeerReview requirements.

(1) The coordinates computation is performed by a deterministic state machine $f(\cdot)$. This state machine is available to all nodes since all nodes use the same algorithm to compute their coordinates. The coordinates are computed in a step by step fashion, therefore it is easy to create a snapshot and initialize the state machine according to a given snapshot.

(2) PeerReview requires that for any node n, the set $n \cup w(n)$ contains at least one correct node. By using a membership server to randomly assign witnesses, this requirement can be achieved with very high probability. For example, suppose 30% of the nodes are malicious, even with just 5 randomly assigned witnesses for n, the probability that $n \cup w(n)$ contains at least one correct node is 0.999.

(3) The third requirement of PeerReview is a general assumption for distributed systems. A network coordinates system can meet this requirement.

Customizing PeerReview - To use PeerReview to secure coordinates computation, we only need to customize what should be recorded in the secured log and what should be verified in the auditing procedure:

(1) Log entries: The secured log on each node contains all the inputs and outputs it uses to compute coordinates. Every time when a node sends or receives a message or computes its coordinates, the node adds one entry to its log. Each log entry is a 3-tuple $\{seq, type, data\}$, where seq is a sequence number; $type$ is the entry type, and $data$ contains the data associated with the specific type of entry. There are three types of entry: *PROBE*, *PROBE_REPLY* and *COMPUTE*. The *PROBE* type records the sending or receipt of a probe message, the corresponding $data$ contains either the source (for receipt) or the destination (for sending) address of the message. The *PROBE_REPLY* type records the sending or receipt of a reply message, the corresponding $data$ contains the source (for receipt) or the destination (for sending) address of the message and the replied coordinates c_i^k (for receipt). The *COMPUTE* type records coordinates computation, the corresponding $data$ contains information about the computation, i.e. $\{c_i, c_{i-1}, \{d_i^k\}, s_i\}$ (note that $\{c_i^k\}$ is already recorded in *PROBE_REPLY*).

(2) Coordinates auditing: When a witness w audits a node n, it simply recompute n's coordinates with the information in n's log. If n's coordinates are inconsistent with w's computation, n is detected as faulty, and w uses n's log as a public proof.

Overhead evaluation - Suppose in a network coordinates system, each node has N neighbors, and M witnesses. Every time when a witness w audits a node n, it will contact node n, n's neighbors and the neighbors' witnesses. This step has $O(MN)$ message complexity. Therefore, $O(M^2N)$ messages are required to fully audit a node. Since M and N are constant numbers, the communication overhead to audit a node does not increase when the system scales up. Thus PeerReview has good scalability. However, $O(M^2N)$ messages per auditing is not ignorable overhead in the system. Fortunately, because our stabilization algorithm makes computing stable coordinates possible, the audit needs only to be performed once after a node's coordinates have stabilized. Moreover, PeerReview guarantees to detect faked coordinates; once a node is deemed malicious, it can be forever banned from the system. Thus, there is no more incentives for malicious nodes to fake coordinates.

To quantify the communication incurred in auditing a node, consider a network coordinates system using 5D Euclidean coordinates. Assume each coordinate, delay, or random seed is 2 bytes, sequence number or address is 4 bytes and the type field is 1 byte. Therefore, each *PROBE* entry takes 9 bytes; each *PROBE_REPLY* entry takes 19 bytes, and each *COMPUTE* entry takes $2N + 27$ bytes. In one embedding step, a node probes all its neighbors and computes its coordinates. This will add N *PROBE* entries, N *PROBE_REPLY* entries, and one *COMPUTE* entry in its log, i.e. $30N + 27$ bytes. When a witness w audits one computation of a node n, w will retrieve the log entries from n, which is $30N + 27$ bytes. For the consistency verification protocol, w also needs to collect the *PROBE* and *PROBE_REPLY* entries from all of n's neighbors and send them to n's neighbors' witnesses to verify the information. This step transmits $28 \times (M + 1) \times N$ bytes. Thus, altogether, when a witness w audits one computation of n, $28 \times (M + 1) \times N + 30N + 27$ bytes of data is transmitted. Of course, a witness in reality audits all the computations of a node before it stabilizes in one batch, thus the data is transmitted in bulk. By applying a compression tool such as gzip, the data can be compressed to 30% of its original size.

Suppose $N = 32$, $M = 5$, and we make a pessimistic assumption that a node stabilizes after 200 computation steps (this is pessimistic because it represents a scenario where a large number of nodes join simultaneously), the audit of a node transmits a total of 1.9MB of data. In the common case, the number of computation steps should be much smaller and the overhead will be proportionally reduced. The power of this mechanism is that each node only needs to be audited once after its coordinates stabilize to decide whether its coordinates are faulty or correct. The audit result can be recorded by the membership server for all nodes to see. If a node refuses to stabilize, it must be detectably faulty and can be caught easily.

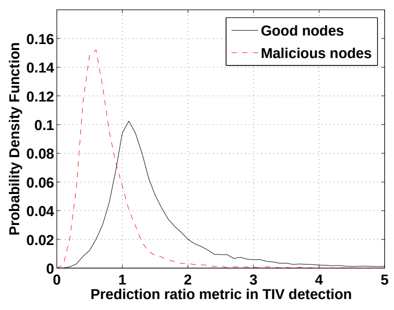
4.4.2 Securing Delay Measurement by TIV Detection

The difficulty of securing delay measurement is that delay measurement relies on the honesty of end hosts. For two nodes A and B, the only way for A to know the propagation delay between them is to measure against B by sending probing packets. If node B is honest and replies to a probing packet immediately after it is received, after collecting enough samples, A can obtain the true propagation delay between A and B. However, if node B is malicious, it can manipulate the delay by not replying to a probing packet immediately. There is no foolproof way for node A to differentiate the real network propagation delay from the artificial delay added by B.

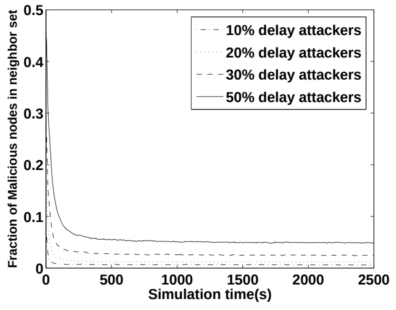
Note that a malicious node cannot shorten the propagation delay. Node A can simply add a random number in each probing packet and require B to include the random number in the reply packet. Thus, B cannot shorten network delay because it cannot generate a reply before receiving the probe. With this observation, we can design a strategy to protect delay measurement.

Statistical detection of faked delays - Since PeerReview can guarantee the security of coordinates computation, the only remaining way a malicious node can impact good nodes' coordinates is to inflate the delays to them and mislead their coordinates. This is exactly the behavior of the delay attack. In the previous section, we have already seen that all existing statistical detection mechanisms fail to mitigate the delay attack. However, now that we have narrowed down the coordinates security problem to a specific kind of attacks, delay attacks we can design a statistical detection mechanism to protect good nodes' coordinates against faked delays.

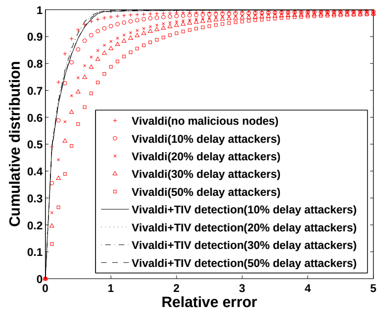
The way delay attackers impact good nodes' coordinates is that the artificially inflated delays can cause more triangle inequality violations (TIVs) in the delays among neighbors. If good nodes use these faked delays to compute their coordinates, the coordinates will have very poor accuracy. However, a point we want to clarify is that, the inflated delays do not always cause more TIVs in the



(a)



(b)



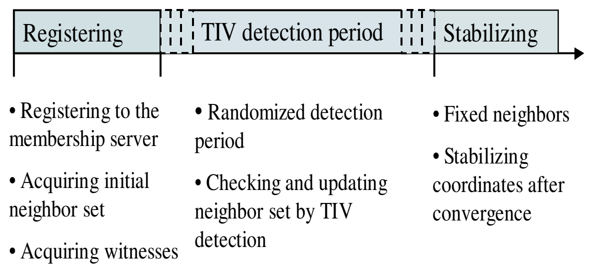
(c)

Figure 11: The performance of TIV detection on delay attack (a) Metric distribution (b) Detection performance (c) Coordinates accuracy

network. If a powerful delay attacker knows all the delays among other nodes, it can manipulate its delays to other nodes to reduce the TIVs in the network. But in this case, these faked delays do not significantly hurt good nodes' coordinates. Previous study [15] has shown that, when there are fewer TIVs in the network, the nodes' coordinates have better accuracy. Therefore, when we consider the delay attacks, what we care most are those faked delays that cause severe TIVs in the network and hurt good nodes' coordinates. This inspires our idea to protect good nodes' coordinates against faked delays using the TIV alert technique proposed in [33]. The TIV alert technique uses the prediction ratio ($\frac{predicted}{measured}$) of coordinates as a heuristic indicator to detect the edges causing severe TIVs in the network. As discussed in [33], the edges that cause severe TIVs are highly like to have a low prediction ratio. Since those harmful faked delays cause severe TIVs, we should be able to use the prediction ratio metric to detect them.

Figure 11(a) shows the prediction ratio distributions of good nodes and delay attackers, where the delay attackers randomly inflates their delays for up to 300ms (30% of the nodes are delay attackers). From this graph, we can see that the prediction ratio metric has a reasonable ability to differentiate good nodes and delay attackers. The faked delays in this attack cause much more severe TIVs.

Using the prediction ratio metric, we can design a simple TIV detection technique to defend against the delay attack as follows. At each embedding step, a node A checks one of its neighbors B. If the prediction ratio of edge AB is lower than a threshold, this neighbor is deemed malicious. We empirically set the detection threshold to be 0.9. To evaluate the performance of TIV detection, we use it to defend against 10%, 20%, 30% and 50% of malicious nodes performing the delay attack in the system. Figure 11(b) shows the fraction of malicious nodes in the neighbor sets during the experiment. As can be seen, for all the cases, the TIV detection mechanism can effectively identify 90% of the malicious nodes and remove them from nodes' neighbor sets. 10% of the malicious nodes are still left in nodes' neighbor sets. These small number of malicious nodes do not cause severe TIVs, therefore the TIV detection mechanism cannot detect them. That is why the TIV detection mechanism has a large false negative rate in the end. However, our coordinates accuracy result shows that these small number of remaining malicious nodes do not impact coordinates accuracy very much. Figure 11(c) shows the accuracy of good nodes' coordinates. From this graph, we can see that, without any protection mechanism, the delay attackers can significantly degrade the accuracy of good nodes' coordinates. The TIV detection mechanism can effectively protect good nodes' coordinates by removing most of the malicious nodes. Even when there are 50% of malicious nodes in the Vivaldi system, the good nodes' coordinates still have good accuracy that is close to the Vivaldi coordinates when there is no malicious node.



Registering
TIV detection period
Stabilizing
• Registering to the membership server
• Acquiring initial neighbor set
• Acquiring witnesses
• Randomized detection period
• Checking and updating neighbor set by TIV detection
• Fixed neighbors
• Stabilizing coordinates after convergence

Figure 12: Integrating the security and stability mechanisms

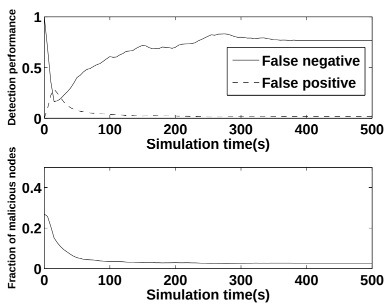
5. INTEGRATING STABILITY AND SECURITY ALGORITHMS

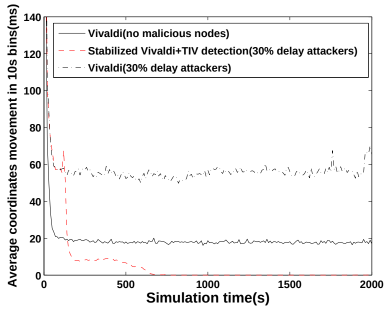
In previous sections, we introduced several algorithms to solve the stability and security problems of network coordinates. In this section, we illustrate a simple way to integrate stability and security algorithms.

We consider a decentralized network coordinates system. As illustrated in Figure 12, when a node joins the system, it first registers with a membership server. The membership server assigns the node with a set of neighbors to compute coordinates with, and a set of witnesses to audit the behavior of this node. PeerReview is used to protect coordinates computation. With its protection, any faked coordinates are guaranteed to be detected. When a node is detected or suspected by a witness with evidence, this node will be removed from the system.

The TIV detection mechanism is used to protect good nodes against faked delays. After a node starts computing its coordinates, it chooses a random TIV detection period. During this period, the node focuses on checking its neighbors and detecting faked delays that can hurt their coordinates accuracy. During the TIV detection procedure, the node's neighbor set is dynamically changing, therefore it cannot stabilize the coordinates in this period. After the detection period, the node can start to stabilize its coordinates by the error elimination algorithm. During the coordinates stabilization procedure, nodes cannot change their neighbor sets. Finally, the node's coordinates are stabilized.

There is a potential attack that some malicious nodes may pretend to frequently experience propagation delay changes to cause


(a)


(b)

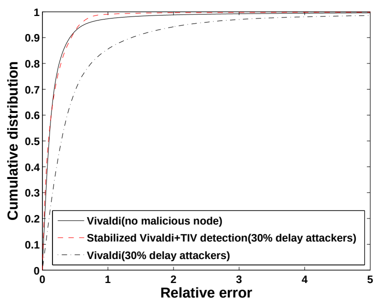

(c)

Figure 13: The performance of integrating TIV detection and coordinates stabilization (a) Detection performance (b) Coordinates stability (c) Coordinates accuracy

unnecessary coordinates recomputation. However, since true propagation delay changes require routing changes, and routing changes do not happen frequently, if a node observes that a neighbor's delay is changing rapidly and frequently, it will remove this neighbor.

Figure 13 shows the experimental results for using this integrated algorithm to protect and stabilize Vivaldi coordinates with 30% delay attackers in the system. In this experiment, each node randomly chooses a detection period from $[200sec, 500sec]$. From Figure 13(a), we can see that, during the detection period, nodes can effectively remove most of the malicious nodes from their neighbor sets. As illustrated in Figure 13 (b) and (c), after the detection procedure is done, the error elimination algorithm can stabilize the coordinates. The stabilized network coordinates has good accuracy which is close to the dynamic Vivaldi coordinates with no malicious nodes.

6. RELATED WORK

Since the concept of network coordinates was first introduce in GNP [21], there has been a significant amount of work on how to build network coordinates systems. These studies focus on different aspects of the problem.

Architectures - Because the centralized architecture in GNP suffers from a scalability problem, several papers have proposed different architectures to build scalable coordinates systems. NPS [22] and PIC [4] extend GNP into a hierarchical architecture. Lighthouse [23], Virtual Landmarks [32] and Vivaldi [5] explore different methods to build fully decentralized network coordinates systems.

Models - In addition to the most popular Euclidean model [21, 5, 22, 25, 4], many other models for approximating Internet delays have been explored. Some Euclidean space extension models slightly modify the Euclidean space to incorporate Internet delay features. The representative proposals are LAT [15] and Height [5]. Other metric models have been examined as well, for example, the Hyperbolic model in [26]. However, all the metric space models suffer from the triangle inequality violation effect of Internet delays. Therefore several numerical optimization models are also proposed to model Internet delays. The representative models are PCA in [16] and [32], and the SVD, NMF models in the IDES system [19]. These models however do not dramatically out-perform the Euclidean space model.

Optimization algorithms - Two kinds of optimization algorithms have been proposed to compute the coordinates in network coordinates systems: multi-dimensional scaling algorithms and simulation based algorithms. An example of a multi-dimensional scaling algorithm is the Downhill Simplex algorithm used in GNP [21] and NPS [22]. The simulation based algorithms model the delay prediction errors by a force system and seek to reduce the forces among all pairs of nodes. The representative algorithms are the Big-Bang Simulation algorithm [25] and the Spring algorithm [5].

System optimization - Several mechanisms have been proposed to improve the performance of coordinates systems. In addition to the techniques we have mentioned for improving coordinates stability and security, a more recent study [27] proposes another security mechanism, Veracity, in which each node is associate with a set of verifying nodes and the node's coordinates are tested based on its error to the verifying nodes. If a node's coordinates have large error in predicting the delays to most of the verifying nodes, the node is deemed malicious and its coordinates will not be used by other nodes. Fundamentally, Veracity is another outlier detection mechanism, which still uses the statistical detection technique to handle all kinds of faked coordinates and faked delays together. Zhang et. al. [37] propose hierarchical coordinates to improve the accuracy of network coordinates. Wang et. al. [33] introduce a TIV alert mechanism to reduce the impact of TIV on network coordinates systems.

Theoretical analysis - Theoretical studies [30, 28, 29] have addressed the reason why network coordinates systems can succeed at predicting Internet delays. The results demonstrate that without triangle inequality violation, network coordinates systems based on a small number of landmarks or a small number of distributed neighbors can have good accuracy.

Performance analysis - Previous work also studies the performance of network coordinates system at the application level. In [38] and [17], the authors quantified the imprecision of using network coordinates in several applications, such as nearest neighbor selection and overlay multicast. Lee et al. [15] point out that triangle inequality violation is an important cause of coordinates inaccuracy.

Applications - As we have mentioned, many previous studies have applied network coordinates to solve a wide range of problems, such as overlay construction [20, 18], compact Internet routing [1, 13, 8], network modeling [36] and security [2].

7. CONCLUSIONS

In this paper, we have presented new algorithms to solve the stability and security problems in decentralized network coordinates systems. We have found a new error elimination model that can achieve coordinates stability without hurting accuracy, and have designed an effective algorithm based on this model. We have also found that solving the security problem with a single statistical mechanism is hard. We recognize that coordinates computation can be separately secured by proven BFD techniques, and we have shown that a customized BFD algorithm can be applied. Furthermore, we have shown that the remaining delay measurement part can be sufficiently secured by TIV detection. Integrating these new algorithms, decentralized network coordinates systems can finally be scalable, stable, and secure simultaneously.

Acknowledgment

We would like to thank our shepherd Jia Wang and the anonymous reviewers for their valuable feedback on earlier versions of this paper.

8. REFERENCES

[1] I. Abraham and D. Malkhi. Compact routing on euclidean metrics. In *Proceedings of PODC 2004*, July 2004.

[2] R.A. Bazzi and G. Konjevod. On the establishment of of distinct identities in overlay networks. In *Proceedings of ACM PODC'05*, July 2005.

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of ACM OSDI*, February 1999.

[4] 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.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceeding of ACM SIGCOMM*, August 2004.

[6] C. de Launois, S. Uhlig, and O. Bonaventure. A stable and distributed network coordinate system. In *Technical report, University Catholique de Louvain*, December 2004.

[7] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the SIGCOMM IMW*, Marseille, France, November 2002.

[8] R. Gummadi, N. Kothari, Y.J. Kim, R. Govindan, B. Karp, and S. Shenker. Reduced state routing in the internet. In *Proceedings of HotNets III*, November 2004.

[9] Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Oct 2007.

[10] M.A. Kaafar, L. Mathy, C. Barakat, K. Salamatian, T. Turletti, and W. Dabbous. Securing internet coordinate embedding systems. In *Proceeding of SIGCOMM'07*, August 2007.

[11] M.A. Kaafar, L. Mathy, T. Turletti, and W. Dabbous. Virtual networks under attack: Disrupting internet coordinate systems. In *Proceeding of CoNext'06*, December 2006.

[12] J. Ledlie, P. Gardner, , and M. Seltzer. Network coordinates in the wild. In *Proceeding of USENIX NSDI'07*, April 2007.

[13] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer. Wired geometric routing. In *Proceedings of IPTPS 2007*, February 2007.

[14] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *Proceeding of International Conference on Distributed Computing Systems*, July 2006.

[15] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of Euclidean embedding of Internet hosts. In *Proc. SIGMETRICS 2006*, June 2006.

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

[17] Eng Keong Lua, Timothy Griffin, Marcelo Pias, Han Zheng, and Jon Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *Proceedings of IMC*, Berkeley, CA, October 2005.

[18] C. Lumezanu, D. Levin, and N. Spring. Peer wise discovery and negotiation of faster path. In *Proceedings of HotNets-VI*, November 2007.

[19] Y. Mao and L. K. Saul. Modeling distances in large-scale networks by matrix factorization. In *Proceedings of Internet Measurement Conference*, Sicily, Italy, October 2004.

[20] Animesh Nandi, Aditya Ganjam, Peter Druschel, T. S. Eugene Ng, Ion Stoica, and Hui Zhang. A reusable control plane for overlay multicast. In *Proceedings of USENIX NSDI'07*, April 2007.

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

[22] T. S. E. Ng and H. Zhang. A network positioning system for the internet. In *Proceedings of USENIX Annual Technical Conference*, June 2004.

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

[24] P. Pietzuch, J. Ledlie, and M. Seltzer. Supporting network coordinates on planetlab. In *Proceeding of the Second Workshop on Real Large Distributed Systems (WORLDS'05)*, December 2005.

[25] 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.

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

[27] M. Sherr, B. T. Loo, and M. Blaze. Veracity: A fully decentralized service for securing network coordinate systems. In *Proceedings of IPTPS'08*, 2008.

[28] A. Slivkins. Distributed Approaches to Triangulation and Embedding. In *Proceedings 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.

[29] A. Slivkins. Network Distance Estimation with Guarantees for All Node Pairs. Technical report, Cornell University, 2006.

[30] A. Slivkins, J. Kleinberg, and T. Wexler. Triangulation and Embedding using Small Sets of Beacons. In *Proceedings of FOCS*, 2004.

[31] M. Szymaniak, D. Presotto, G. Pierre, and M. van Steen. Practical large-scale latency estimation. *Elsevier's Computer Networks*, 52(7), 2008.

[32] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of IMC*, Miami, FL, October 2003.

[33] G. Wang, B. Zhang, and E. Ng. Towards network triangle inequality violations aware distributed systems. In *Proceedings of the ACM/USENIX Internet Measurement Conference (IMC'07)*, Oct 2007.

[34] Guohui Wang. On the design principles of network coordinates systems. April 2008. Master's thesis, Rice University.

[35] D. Zage and C. Nita-Rotaru. On the accuracy of decentralized network coordinate systems in adversarial networks. In *Proceeding of ACM CCS'07*, October 2007.

[36] B. Zhang, T.S.Eugene Ng, A.Nandi, R.Riedi, P.Druschel, and G.Wang. Measurement-based analysis, modeling, and synthesis of the internet delay space. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC)*, October 2006.

[37] R. Zhang, Y. Hu, X. Lin, and S. Fahmy. A hierarchical approach to internet distance prediction. In *Proceedings of IEEE ICDCS*, Lisboa, Portugal, 2006.

[38] R. Zhang, C. Tang, Y. Hu, S. Fahmy, and X. Lin. Impact of the inaccuracy of distance prediction algorithms on internet applications: an analytical and comparative study. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.