

RICE UNIVERSITY

On the Design Principles of Network Coordinates Systems

by

Guohui Wang

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

Dr. T. S. Eugene Ng (Chair),
Assistant Professor,
Computer Science

Dr. Dan Wallach,
Associate Professor,
Computer Science

Dr. Edward Knightly,
Professor,
Electrical and Computer Engineering

HOUSTON, TEXAS

APRIL, 2008

On the Design Principles of Network Coordinates Systems

Guohui Wang

Abstract

Since its inception, the concept of network coordinates has been successfully applied to solve a wide variety of problems such as overlay optimization, network routing, network localization, and network modeling. Despite these successes, several practical problems limit the benefits of network coordinates today. First, the Triangle Inequality Violation(TIV) among the Internet delays degrades the application performance of network coordinates, how to reduce the impact of TIV on network coordinates systems? Second, how can network coordinates be stabilized without losing accuracy in a distributed fashion so that they can be cached by applications? Third, how can network coordinates be secured such that legitimate nodes' coordinates are not impacted by misbehaving nodes? Although these problems have been discussed extensively, the solutions are still unclear. This thesis presents analytical studies for understanding these problems and reveals several new findings: (1) the analysis results from existing Internet delay measurements demonstrate the irregular behaviors of TIVs among the Internet delays, which implies the difficulty of modeling TIVs; (2) a new TIV alert mechanism can identify the edges causing severe TIVs and reduce the impact of TIVs on network coordinates; (3) a new model of coordinates stabilization based on error elimination can achieve stability without hurting accuracy; a novel algorithm based on this model is presented; (4) recently proposed statistical detection mechanisms cannot achieve an acceptable level of security against aggressive attacks. (5) an accountability protocol can completely protect coordinates computation and a TIV alert detection mechanism can effectively protect network coordinates against delay attacks. These findings offer guidelines on the design and applications of network coordinates systems.

Acknowledgments

Contents

Abstract	ii
List of Illustrations	vii
1 Introduction	1
2 Categorizing the Design Space of Network Coordinates Systems	5
2.1 The Theoretical Foundations of Network Coordinates	5
2.2 System Structure	7
2.3 System Performance Targets	8
2.4 Existing Studies on Network Coordinates Systems	9
3 Reducing the Impact of TIVs on Network Coordinates	12
3.1 Coordinates Accuracy Problem Induced by TIVs	12
3.2 Analyzing TIVs in Internet Delays	13
3.2.1 Evaluation Metric for TIV Severity	13
3.2.2 Analysis of TIV Characteristics	14
3.3 Understanding the Problems Caused by TIVs	19
3.4 Strawman Solutions to Deal with TIVs	22
3.4.1 Neighbor Selection Experiment Methodology	22
3.4.2 Existing Models for Accommodating TIV in Network Embedding .	23
3.4.3 Naive avoidance of TIVs	24
3.5 TIV Alert Mechanism	25
3.5.1 Alerting Severe TIVs by Metric Embedding Error	25
3.5.2 Using TIV Alert Mechanism in Vivaldi	28

3.6	Summary	30
4	Coordinates Stability in Decentralized Network Coordinates Systems	31
4.1	Coordinates Stability Problem	31
4.2	The Difficulties of Stabilizing Coordinates	32
4.3	Analytical Framework for Coordinates Stability	34
4.4	A Novel Algorithm for Coordinates Stability	36
4.4.1	The Algorithm	36
4.4.2	Evaluation	39
4.4.3	Adapting to Delay Changes	40
5	Coordinates Security in Decentralized Network Coordinates Systems	42
5.1	Problem Formulation	42
5.2	Modeling Statistical Detection Mechanisms	43
5.3	Empirical Study of the Existing Statistical Detection Mechanisms	46
5.3.1	Basic Principles	46
5.3.2	Malicious Attacks	48
5.3.3	Behavior of Outlier Detection Mechanisms	51
5.3.4	Behavior of the Kalman Filter Detection Mechanism	55
5.3.5	Limitations of Statistical Detection Mechanisms	58
5.4	Securing Network Coordinates in Two Phases	60
5.4.1	Securing Coordinates Computation by Byzantine Fault Detection	60
5.4.2	Securing Delay Measurement by TIV Detection	65
5.4.3	Summary	67
6	Discussion and Conclusion	68

Illustrations

2.1	Network Coordinates System Structure	8
3.1	Illustration of the TIV severity metric	13
3.2	Cumulative distribution of TIV severity	13
3.3	TIV severity by cluster (a white point represents the most severe TIV).	16
3.4	Relation between delay and TIV severity, error bar shows the 10%, median and 90% (a) DS^2 data (b) p2psim data (c) Meridian data (d) PlanetLab data	17
3.5	Shortest path length for edges of DS^2 data at different delays	19
3.6	Proximity property of TIVs	20
3.7	Vivaldi error trace for a simple 3-node network with TIV	20
3.8	Distribution of the oscillation range of all the edges	20
3.9	Neighbor selection performance for IDES	23
3.10	Neighbor selection performance for Vivaldi-LAT	23
3.11	Neighbor selection performance for Vivaldi with TIV severity filter	25
3.12	TIV severity for edges with different prediction ratios	26
3.13	Accuracy of TIV alert mechanism	26
3.14	Recall rate of TIV alert mechanism	26
3.15	TIV severity of Vivaldi neighbor edges	29
3.16	Neighbor selection performance of dynamic neighbor Vivaldi	29
4.1	Coordinates stability and accuracy of Vivaldi with gravity and loss factor (a) Stability (b) Accuracy	32

4.2	Performance of Stabilized Vivaldi in a PlanetLab experiment (a) Stability (b) Accuracy	37
4.3	Performance of Stabilized Vivaldi in a simulation experiment on King data (a) Stability (b) Accuracy	37
5.1	Demonstration of metric distributions for statistical detection	44
5.2	The relationship between α , β and p' ($q = 0.3$)	46
5.3	Impact of attacks on coordinate accuracy (a) Isolation attack (b) Oscillation attack (c) Deflation attack (d) Shifting attack (e) Delay attack (f) Inflation attack	50
5.4	Detection performance of temporal outlier detection (a) Deflation attack (b) Oscillation attack (c) Inflation attack (d) Shifting attack (e) Delay attack (f) Isolation attack	52
5.5	Detection performance of spatial outlier detection (a) Deflation attack (b) Oscillation attack (c) Inflation attack (d) Shifting attack (e) Delay attack (f) Isolation attack	53
5.6	Metric distribution of outlier detection mechanisms for oscillation attack (a) Temporal outlier detection at time 0 second (b) Spatial outlier detection at time 0 second (c) Temporal outlier detection at time 2000 second (d) Spatial outlier detection at time 2000 second	54
5.7	Metric distribution of outlier detection mechanisms for the shifting attack (a) Temporal outlier detection at time 0 second (b) Spatial outlier detection at time 0 second (c) Temporal outlier detection at time 2000 second (d) Spatial outlier detection at time 2000 second	55
5.8	Detection performance of the Kalman filter detection mechanism (a) Isolation attack (b) Oscillation attack (c) Deflation attack (d) Inflation attack (e) Delay attack (f) Shifting attack	56

5.9	Metric distribution of the Kalman filter detection mechanism for different attacks (a) Oscillation attack at time 0 second (b) Shifting attack at time 0 second (c) Inflation attack at time 0 second (d) Oscillation attack at time 2000 second (e) Shifting attack at time 2000 second (f) Inflation attack at time 2000 second	57
5.10	Accuracy of the Vivaldi coordinates with statistical detection mechanisms under shifting attacks	59
5.11	The performance of TIV detection on delay attack (a) Metric distribution (b) Detection performance (c) Coordinates accuracy	65

Chapter 1

Introduction

Since its inception, the concept of network coordinates has attracted much attention from the network research community. Network coordinates provide a lightweight approximation of Internet delays, and they have been used to solve a wide range of problems. In addition to the original proposed application of network latency prediction, researchers have explored the applications of network coordinates in many other areas. To name only a few examples, [GKK⁺04], [AM04] and [LPMS07] discuss the application of network coordinates in compact Internet routing. [NGD⁺07] uses network coordinates for overlay construction. [LLS07] proposes to use network coordinates to discover fast overlay paths. [ZNA⁺06] demonstrates the application of network coordinates in modeling and synthesizing of Internet delays at large scales. [BK05] uses network coordinates to defend against the Sybil attack in overlay networks.

Despite these successes, several practical problems of current network coordinates systems significantly limit the benefits of network coordinates.

The first problem is coordinates accuracy problem induced by TIVs among the Internet delays. Previous studies [LGP⁺05, ZTH⁺06] have evaluated the accuracy of using network coordinates in the neighbor selection application. Their results reveal that network coordinates are not accuracy enough in neighbor selection applications. In [LZSS06], the authors point out that the triangle inequality violation among the Internet delays is an important cause of coordinates accuracy problem. Network triangle inequality violation is the phenomenon that the end to end delay between two nodes is even longer than the delay of them along a detour path going through another node. Numerous studies have reported the existence of TIV among the Internet delays (e.g.

[SCH⁺99, FJP⁺99, NZ02, ZLPG05, LZSS06, ZNA⁺06]). As discussed in [ZLPG05], TIV is a consequence of the Internet's structure and routing policies and thus will remain a property of the Internet for the foreseeable future. Although several previous studies [MS04, LZSS06] have attempted to introduce TIV effects into network coordinates systems, none of them can significantly improve the accuracy of network coordinates. Several questions remain unclear about the TIV phenomena in the Internet delays. What are the properties of TIVs among the Internet delays? How do TIVs impact the performance of network coordinates systems? Are there ways to reduce the impact of TIVs on the application performance of network coordinates?

The second problem is coordinates stability. Many previous studies, such as [dLUB04], [PLS05], [LPS06] and [WZN07], have reported the coordinates instability problem in the Vivaldi system [DCKM04], 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 [LPS06, LGS07] 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 third problem of is coordinates security. Previous studies [KMTD06, ZNR07] 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 is 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 [KMB⁺07, ZNR07] have proposed to use statistical detection mechanisms to differentiate good nodes and malicious nodes. However, it remains unclear how strong the

statistical detection approaches need to be sufficiently contain the malicious nodes, and whether the recently proposed statistical detection mechanisms are strong enough.

This thesis is devoted to understanding these problems by measurement and analytical studies. The analytical results help to reveal several new findings.

The state of the art research on network coordinates accuracy problem motivate our study to analyze the properties of TIVs among the Internet delays, understand the impact of TIVs on network coordinates systems and seek a way to reduce the impact of TIVs. Our analysis results show that TIV does not have strong relationship with delay lengths and cluster effect and even two nearby nodes can have different TIV property. Even though the results highlight complex and irregular characteristics of TIVs that we cannot yet explain completely, we can identify the impact of TIV on the Vivaldi system. Our results show that TIVs can cause high predict error and coordinate oscillation in the Vivaldi systems. We propose a TIV alert mechanism that can identify the edges causing severe TIVs and reduce the impact of TIVs on network coordinates systems. The TIV alert mechanism can also be used to provide TIV awareness in many other distributed systems.

For coordinates stability, we propose two abstract models for stabilizing coordinates in decentralized network coordinates systems: stopping coordinates, and error elimination. This new analytical framework leads to two important observations. First, the intuitive model of stabilizing network 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 coordinates stability while preserving accuracy. We propose a novel algorithm to closely approximate the error elimination model for stability in the Vivaldi system. The evaluation results demonstrate that, coordinates stability can in fact be achieved without sacrificing accuracy in decentralized network coordinates systems.

For coordinates security, we propose an analytical model for understanding the dynamic behavior of statistical detection mechanisms in network coordinates systems. The analytical model exposes several important insights. First, all statistical detection mechanisms

have a performance bottleneck in identifying malicious attacks that depends on the false positive and false negative rates. Second, the false positive and false negative rates of a statistical detection mechanism can evolve, and the properties of this evolution significantly impact the mechanism's performance. We empirically study the evolution properties of recently proposed statistical detection mechanisms. The results show that these statistical detection mechanisms cannot achieve an acceptable level of security against aggressive attacks. These findings motivate our work to rethink the coordinates security problem. We study the coordinates security problem in two parts: security of coordinates computation and security of delay measurement. We find that, the coordinates computation procedure can be modeled as a deterministic state machine, which can be completely secured by an accountability protocol; For delay measurement part, we proposed a TIV alert detection mechanism that can effectively detect faked delays and eliminate the impact of faking delay attacks on network coordinates.

The findings in this thesis provide new guidelines on the design and applications of network coordinates systems.

The rest of this thesis is organized as follows. Chapter 2 introduces the theoretical foundations of network coordinates and overviews the design of existing network coordinates systems. Chapter 3 analyzes the properties of TIVs among the Internet delays, and seeks ways to reduce the impact of TIVs on network coordinates systems. Chapter 4 addresses the coordinates stability problem in decentralized network coordinates systems. Chapter 5 studies the coordinates security problem in decentralized network coordinates systems. Finally, we discuss the implications of our findings and conclude in Chapter 6.

Chapter 2

Categorizing the Design Space of Network Coordinates Systems

In this section, we overview the design space of network coordinates systems and categorize all the existing studies such that we can understand the state of the art research on network coordinates system design.

2.1 The Theoretical Foundations of Network Coordinates

In network coordinates systems, each node chooses a constant number of neighbors. The purpose of network coordinates system is to assign each node with a set of network coordinates, such that the distances among the network coordinates can predict the measured delays accurately. All the neighbor connections form a neighbor graph $G = (V, E)$, where V is the set of nodes in the network, and E is the set of neighbor edges. The coordinates computation can be generally formulated as the graph embedding problem. The problem is to determine the locations x_i of all the vertices in a d dimensional space \mathbb{R}^d such that for each neighbor pair (i, j) , the distance between x_i and x_j can preserve the weight of edge e_{ij} . Several previous studies [EGW⁺04, AEG⁺06] have studied the graph property for network localization problem. Learning from the early literatures in graph theory research community, such as [Hen92], the authors report several important findings about the theoretical foundations of network coordinates.

Theorem 1: A graph is uniquely embeddable if and only if the graph is globally rigid.

Definition 1: A graph G is globally rigid if and only if it is redundantly rigid and 3-connected.

Definition 2: A graph G is redundantly rigid if removal of any single edge results in a graph that is also generically rigid.

This theorem implied that to set up a network coordinates system, we must provide enough connectivity in the neighbor graph to guarantee that the graph can be embedded in to a unique framework in space \mathbb{R}^d , which requires the neighbor graph to be globally rigid. Then how to test the global rigidity of a graph? Since it is easy to test whether a graph is 3-connected, the major difficult is to test whether a graph is generically rigid. The following theorem provides a way to test the generic rigidity of a graph.

Theorem 2: Laman condition [Lam70] - A graph $G = (V, E)$ with n vertices is generically rigid in \mathbb{R}^2 if and only if E contains a subset E_1 consisting of $2n - 3$ edges with the property that for any nonempty subset $E_2 \subset E_1$, the number of edges in E_2 cannot exceed $2k - 3$ where k is the number of vertices of G which are endpoints of edges in E_2 .

The Laman condition can be used to test the globally rigidity property of a graph in 2D space \mathbb{R}^2 . However, the Laman condition literally leads to an inefficient algorithm to test globally rigidity property because it needs to compute all the subgraphs of the tested graph. Therefore, Jacobs et al. [JH97] proposed an pebble game algorithm to efficiently test the global rigidity property of a given graph in 2D space \mathbb{R}^2 . Using these test methods, we can easily test the global rigidity property of the neighbor graph in 2D Euclidean space. The result shows, if the neighbor graph is 6-connect, it is sufficient to guarantee global rigidity in 2D Euclidean space. In current network coordinates systems, each node often randomly choose tens of neighbors. Therefore, if we are compute network coordinates in 2D Euclidean space, we can easily achieve global rigidity by the current neighbor selection strategy.

However, the test of global rigidity property of a graph in 3D and higher dimensional space is a long time open problem. If we are using 3D or higher dimensional space in network coordinates system, we can not test the global rigidity property of the neighbor graph. Currently, network coordinates systems often use 5D - 8D Euclidean space and each node chooses at least 32 neighbors. In our study, we have to assume this dense connectivity

can provide a globally rigid neighbor graph in network coordinates systems.

2.2 System Structure

All the network coordinates systems can be abstracted into a general system framework. As shown in Figure 2.1, a network coordinates system is composed of four major components: architecture, model, optimization procedure and assistant processes.

Architecture - The architecture is defined to be the neighbor graph used in network coordinates system to collect delay measurement and compute coordinates. Each node can select a constant number of nodes in the network as its neighbors. After the neighbor selection procedure, all the nodes can form a neighbor graph in the network. This neighbor graph, also known as the architecture of the system, provides a basis for the delay measurement and coordinates computation. A network coordinates system can use different architectures. For example, GNP [NZ02] is based on a centralized architecture, in which all the nodes can only choose a fixed set of landmarks as their neighbors; while Vivaldi [DCKM04] is based on a fully decentralized architecture, in which all the nodes can choose any other peers of nodes as their neighbors.

Model - The model is the compact model the network coordinates systems use to approximate the Internet delay space so that the distance between virtual coordinates can predict the Internet delays accurately. Several models can be used to assign virtual coordinates and estimate Internet delays, such as metric spaces model in GNP, Vivaldi and matrix factorization model in IDES [MS04].

Optimization procedures - The optimization procedures are the algorithms that can be used to compute the virtual coordinates based on the delay measurements and a particular model, which can be Downhill Simplex algorithm as in GNP, NPS [NZ04] or Big-bang simulation algorithm proposed in [ST03].

Assistant process - In addition to these indispensable system components, a network coordinates system can also have some assistant processes which are proposed to improve the system performance in some specific aspects. For example, gravity technique is proposed

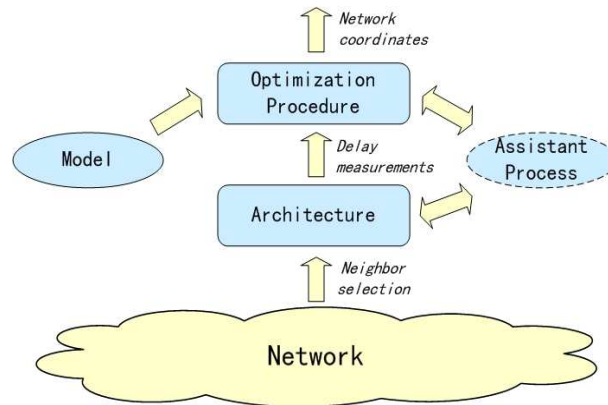


Figure 2.1 : Network Coordinates System Structure

in [LGS07] to reduce the coordinates drifting and Kalman filter technique is proposed in [KMB⁺07] to protect the system from malicious attacks.

2.3 System Performance Targets

In building a network coordinates system, there are several performance targets that people try to achieve from the system design perspective.

Accuracy - One of the major applications of network coordinates systems is the Internet delay estimation, so accuracy is an important performance target. A good network coordinates system should be able to predict the Internet delays accurately by its coordinates distances.

Scalability - Many applications, such as overlay construction and geometric routing, require network coordinates systems to be run in very large scale networks. In these scenarios, scalability is a non-neglectable target for system designers. The network coordinates system should have good scalability so that when it is run over very large scale networks, there is no significant performance degradation.

Stability - Coordinates stability is characterized by the oscillation speed of the network coordinates. The coordinates stability is more relevant to the users of network coordinates

systems. If the network coordinates keep oscillation, this means the coordinates are easily to be staled and they are not cacheable. When a user program wants to use the coordinates of another node in the network, it has to query that node very frequently to get the most recent coordinate. From the system design perspective, a good network coordinates system should be able to provide stable coordinates, so that these coordinates are cacheable and can be easily used by the user programs.

Reliability - System reliability refers to the system performance under unexpected scenarios. The system reliability is especially important for the real deployment of network coordinates systems. When the system is run in the real Internet environment, it is possible that some nodes in the network are malicious. Malicious nodes may provide wrong delays or coordinates to fool other nodes in the network. So the system designers have to consider the reliability of the network coordinates. A good network coordinates system should be able to tolerate malicious attacks.

2.4 Existing Studies on Network Coordinates Systems

Since the concept of network coordinates was first introduced in GNP [NZ02], there has been a significant amount of work on how to build network coordinates systems. These studies focus on different aspects of the problem. In this section, we categorize all the existing studies to get a better understanding on the state of the art research in this area.

Architectures - Because the centralized architecture in GNP suffers from a scalability problem, several papers have proposed different architectures to build scalable coordinates systems. NPS [NZ04] and PIC [CCRK03] extend GNP into a hierarchical architecture. Lighthouse [PCW⁺03], Virtual Landmarks [TC03] and Vivaldi [DCKM04] explore different methods to build fully decentralized network coordinates systems.

Models - In addition to the most popular Euclidean model [NZ02, DCKM04, NZ04, ST03, CCRK03], 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 [LZSS06] and Height [DCKM04].

Other metric models have been examined as well, for example, the Hyperbolic model in [ST04]. 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 [LHC03] and [TC03], and the SVD, NMF models in the IDES system [MS04]. These models however do not dramatically out-perform the Euclidean space model.

Optimization procedures - 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 [NZ02] and NPS [NZ04]. 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 [ST03] and the Spring algorithm [DCKM04].

Assistant processes - Several assistant processes have been proposed to improve the performance of coordinates systems. the gravity technique [LGS07] introduces the gravity force in spring algorithm to reduce the coordinates drifting and improve the coordinates stability in the Vivaldi system. The Kalman filter detector [KMB⁺07] and outlier detector [ZNR07] use statistical models to detect the malicious nodes and therefore to protect network coordinates from malicious attacks. Zhang et al. [ZHLF06] propose hierarchical coordinates to improve the accuracy of network coordinates.

Theoretical analysis - Theoretical studies [SKW04, Sli04, Sli06] 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. [ZTH⁺06] and [LGP⁺05] quantified the imprecision of using network coordinates in several applications, such as nearest neighbor selection

and overlay multicast. [LZSS06] points 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 [NGD⁺07, LLS07], compact Internet routing [AM04, LPMS07, GKK⁺04], network modeling [ZNA⁺06] and security [BK05].

Chapter 3

Reducing the Impact of TIVs on Network Coordinates

3.1 Coordinates Accuracy Problem Induced by TIVs

Previous studies [LGP⁺05, ZTH⁺06] have evaluated the accuracy of using network coordinates in the neighbor selection application. Their results reveal that network coordinates are not accuracy enough in neighbor selection applications. In [LZSS06], the authors point out that the Triangle Inequality Violation(TIV) among the Internet delays is an important cause of coordinates accuracy problem. Network triangle inequality violation is the phenomenon in the Internet that the end to end delay between two nodes is even longer than the delay of them along a detour path going through another node. Formally, given any three nodes A , B and C in the Internet, they form a triangle ABC . Edge AC is considered to cause a triangle inequality violation if $d(A, B) + d(B, C) < d(A, C)$, where $d(X, Y)$ is the measured delay between X and Y . Numerous studies have reported the existence of TIV among the Internet delays (e.g. [SCH⁺99, FJP⁺99, NZ02, ZLPG05, LZSS06, ZNA⁺06]). As discussed in [ZLPG05], TIV is a consequence of the Internet's structure and routing policies and thus will remain a property of the Internet for the foreseeable future. However, several questions that are unclear about the TIVs phenomenon. What are the properties of TIVs among the Internet delays? How do TIVs impact the performance of network coordinates systems? Are there ways to reduce the impact of TIVs on the application performance of network coordinates? In this section, we analyze the properties of TIVs by several Internet delay measurements and identify the impact of TIVs on network coordinates systems. Following the analysis results, we propose a TIV alert mechanism to reduce the impact of TIVs on the application performance of network coordinates.

3.2 Analyzing TIVs in Internet Delays

We begin the discussion by analyzing the characteristics of TIVs in several available Internet delay data sets.

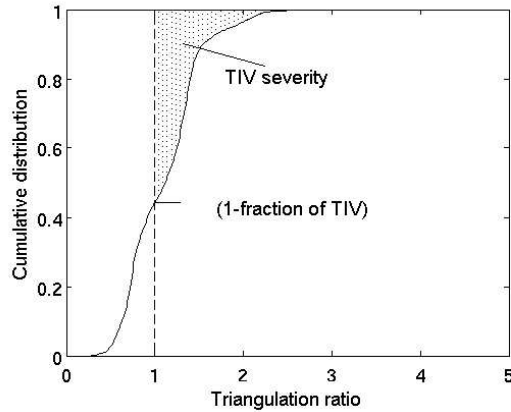


Figure 3.1 : Illustration of the TIV severity metric

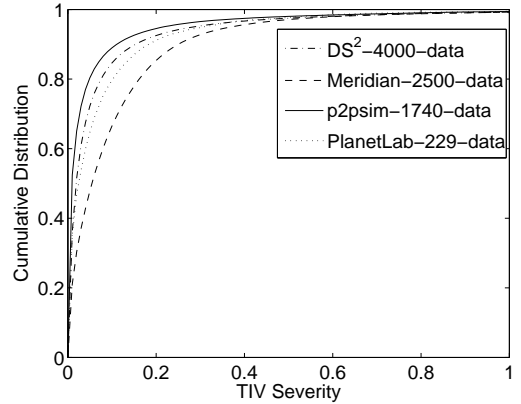


Figure 3.2 : Cumulative distribution of TIV severity

3.2.1 Evaluation Metric for TIV Severity

Given any three nodes A , B and C in the Internet, they form a triangle ABC . Edge AC is considered to cause a triangle inequality violation if $d(A, B) + d(B, C) < d(A, C)$, where $d(X, Y)$ is the measured delay between X and Y . The triangulation ratio of the violation caused by AC in triangle ABC is defined as $d(A, C)/(d(A, B) + d(B, C))$. Previous studies (e.g. [SCH⁺99, FJP⁺99, ZLPG05, LZSS06, ZNA⁺06]) have reported characteristics of TIVs in the Internet delay space by triangulation ratio distribution and the fraction of triangles that suffer from TIV. However, to achieve a better understanding on the TIV properties, we would like to define a numeric metric that captures the *severity* of TIV for any particular edge.

The fraction of triangles that suffer from TIV is not the right metric to use when evaluating the TIV severity of an edge because the triangulation ratios of these violations were

not considered. In the DS2 data, among the top 10% edges causing the highest fraction of triangles suffer from TIV, 16% of them do indeed have the average triangulation ratio belonging to the lowest 10%. Similarly, the average triangulation ratio is not the right metric to use neither, because it does not take the number of TIVs caused by the edge into account. In the DS2 data, among the top 10% edges with the highest average triangulation ratio, 64% of them only cause less than 3 TIVs. Hereby we define the TIV severity metric as following: Given a delay space where the set of all nodes is S , for two nodes $A, C \in S$, the TIV severity of the edge AC is:

$$\frac{\sum d(A, C)/(d(A, B) + d(B, C))}{|S|}$$

where $B \in S$ and $d(A, C) > d(A, B) + d(B, C)$.

To illustrate this metric, Figure 3.1 shows the cumulative distribution of triangulation ratios for an hypothetical edge AC . The TIV severity of the edge AC is then proportional to the area of the shadowed region. Note that the intersection between the dotted vertical line and the curve indicates the fraction of triangles that cause TIV. In the rest of this chapter, we use this TIV severity metric to evaluate the TIVs caused by an edge. A TIV severity value of 0 means the edge does not cause any violation and larger TIV severity means more violations.

3.2.2 Analysis of TIV Characteristics

Figure 3.2 shows the extent of TIVs found in 4 different measured Internet delay data sets: p2psim data (1740 nodes) [p2p], Meridian data (2500 nodes) [WSS05], DS² data (4000 nodes) [ZNA⁺06], and PlanetLab data. Here, the PlanetLab data is the measured delay matrix among 229 PlanetLab nodes we collected. Clearly, TIVs are present in all datasets. For all the data sets, most of the edges only cause slight violations, but a small fraction of edges do cause severe violations and all the curves have long tails.

Our previous study [ZNA⁺06] classified nodes in a delay space into major clusters that correspond to major continents and showed that edges within the same major cluster cause

fewer violations while edges across different clusters cause more violations. We study the TIV severity of the edges using the same clustering method.

The experiment is based on the DS^2 data matrix. We use the same clustering algorithm as presented in [ZNA⁺06] to classify nodes into three major clusters and the nodes that did not get classified into any of the three major clusters form the noise cluster. To show how the TIV severities are distributed over the major clusters, we present a matrix in Figure 3.3. To produce this figure, we first reorganize the original matrix by grouping nodes in the same cluster together. The top left corner has index (0,0). The matrix indices of the nodes in the largest cluster are the smallest, the indices for nodes in the second largest cluster are next, then the indices for nodes in the third largest cluster, followed by indices for nodes in the noise cluster. Each point (i, j) in the plot represents the TIV severity of the edge ij as a shade of gray. A black point indicates least severe violation and a white point indicates most severe violation encountered for any edge in the analysis. Missing values in the matrix are drawn as black points. This result confirms that clustering is also useful for classifying TIV severity. It can be seen that edges within the same cluster (i.e. the 3 blocks along the diagonal) tend to have less severe TIVs (darker) than edges that cross clusters (lighter)¹. This is because when restrained in one cluster, most edges are relatively short, and would cause violations mainly with the nodes in the same cluster, thus limits the number of TIVs. While for crossing cluster edges, although they can not cause very high ratio violations since their end nodes are far apart, they can still induce a large number of violations with nodes existed in any clusters, due to the fact that intercontinental routing usually have many alternative paths. This trend could be observed in the DS2 data, the average number of TIVs caused by edges within the same cluster is 80, while the average number of TIVs caused by crossing cluster edges is 206.

In order to understand what kind of edges cause severe TIVs, we first study the relationship between TIV severity and the length of edges. All edges in the delay matrix are first grouped into 10-millisecond bins based on their lengths, then we plot the TIV severity

¹Note that [LZSS06] uses a different definition for TIV and thus the results are different

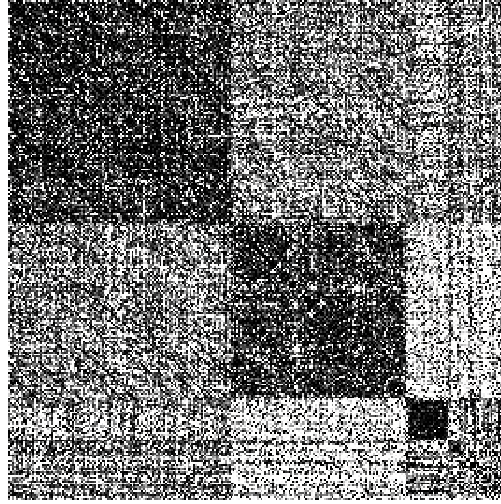


Figure 3.3 : TIV severity by cluster (a white point represents the most severe TIV).

of edges within each bin. Figure 3.4(a) shows the median TIV severity versus lengths of edges based on the DS^2 data. The error bars show the 90th and 10th percentile TIV severity. The general trend is that longer edges cause more severe violations. For example, the edges shorter than 200 ms usually only cause slight violations and edges longer than 300 ms cause increasingly more severe violations. The other observation from Figure 3.4(a) is that edges of very different lengths can cause violations of the same severity level. For example, a 600 ms edge may have violations of the same severity level as both a 300 ms edge and a 800 ms edge. Moreover, the TIV severity of edges has an irregular relationship with their lengths. For example in Figure 3.4(a), the median TIV severity has a peak for the edges around 500-600 ms. Similar irregular behavior can be observed in Figure 3.4(b), (c), (d) that show the relationship between length of edges and TIV severities for p2psim data, Meridian data and PlanetLab data respectively. Since no *traceroute* data is available to completely understand this irregular behavior, our surmise is that it is caused by the irregular routing inefficiency. In Figure 8, The top graph shows in the DS2 data, the fraction of edges that are within the same cluster as the edge length is increased. The bottom graph shows in the DS2 data, the distribution of the shortest path lengths for all edges at different

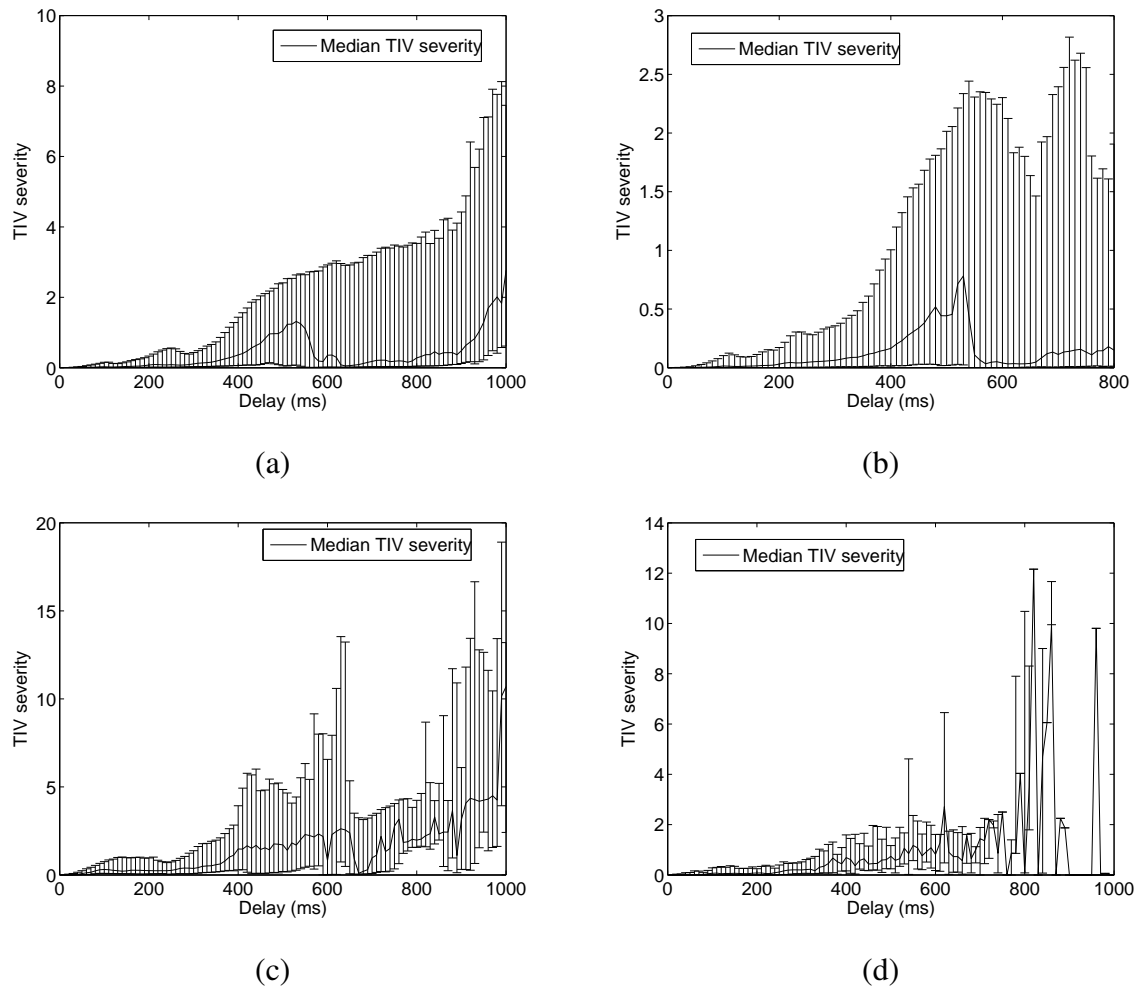


Figure 3.4 : Relation between delay and TIV severity, error bar shows the 10%, median and 90% (a) DS^2 data (b) p2psim data (c) Meridian data (d) PlanetLab data

edge lengths. The error bars represent the 10th and 90th percentile values. As illustrated, most edges longer than 200ms are crossing cluster edges, and generally, longer edges have longer shortest paths. However, when the edge lengths increase from 300ms to 550ms, the lengths of their shortest paths do not reveal a very clear increment, which means most of these edges can find short alternative paths, and they would also cause severe TIVs. When the edges are longer than 550ms, the lengths of their shortest paths make a significant jump, which indicates for many edges in this area, even their shortest path are still very

long, therefore they are not possible to cause severe TIVs. From the above results, we can see that although long edges tend to cause more severe violation, the relationship between TIV severity and edge length is unclear, which indicates that it is very hard to determine whether one edge is causing severe violations only based on its length.

Next, we study whether TIVs can be predicted based on proximity. The hypothesis is that two close-by nodes may have similar TIV characteristics because they are more likely to share similar Internet routes. Obviously, if nodes A and A_l belong to the same local area network, and nodes B and B_l belong to the same local area network, then AB and A_lB_l should have very similar TIV severity. However, we are more interested in whether a more general proximity based relationship exists for nodes that do not belong to the same local area network. To test this hypothesis, for each data set, we randomly choose 10,000 edges. Each edge is assigned with a nearest pair edge by the following method: For an edge AB with end nodes A and B , A_n and B_n are the nearest neighbors of A and B respectively, then the edge A_nB_n is the nearest pair edge of AB . For comparison, each edge is also assigned with a random pair edge. We calculate the TIV severity differences of each edge and its pair edges to evaluate their similarity. Figure 3.6 shows the cumulative distributions of the TIV severity differences of the nearest-pair edges and random-pair edges for four data sets. For all the four data sets, the nearest-pair edges are just slightly more similar to each other than the random-pair edges in terms of TIV severity. This means that close-by nodes do not necessarily have similar TIV severity characteristic. Note that the methodology used to collect the four data sets actually tend to avoid nodes that belong to the same local area network. In these data sets, the nearest neighbor of a node is typically a few milliseconds away and may belong to a different ISP. This result indicates that, in general, it is not possible to predict the TIV severity of edges based on their proximity.

In summary, our results show that TIV is a complex phenomenon in the Internet. Most edges only cause slight TIVs but some edges do cause very severe TIVs. The relationship between TIV severity and edge length is irregular in all data sets, which means we cannot determine whether an edge will cause severe violations only based on its length. In addi-

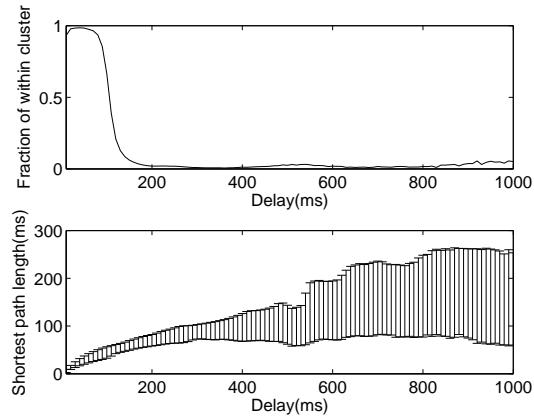


Figure 3.5 : Shortest path length for edges of DS^2 data at different delays

tion, it is hard to predict the TIV severity of an edge by simply considering the TIV severity of some nearby edges because they can have very different TIV severities.

Since the results in this section show that the 4 data sets have similar TIV properties, for simplicity, in the rest of this chapter, the experiments are performed on the DS^2 4000-node delay data set unless otherwise noted.

3.3 Understanding the Problems Caused by TIVs

A recent study [LZSS06] has shown that TIV among the Internet delays causes inaccuracies in network coordinate systems like Vivaldi, however, it remains unclear in what ways does TIV impact Vivaldi. In this section, we try to understand how TIVs impact the performance of Vivaldi. We firstly use concrete scenarios to show how Vivaldi may behave when there are triangle inequality violations.

Suppose we have a network with 3 nodes A , B and C , where the delay of edge AB , $d(A, B)$, is $5ms$, $d(B, C)$ is $5ms$, and $d(C, A)$ is $100ms$ because of inefficient routing or routing policy. Obviously, the triangle inequality is violated because $d(A, B) + d(B, C) < d(C, A)$. We run Vivaldi over this 3-node network, and Figure 3.7 shows the error trace of edge AB , BC , and CA over 100 second simulation time. Here the error is defined to be

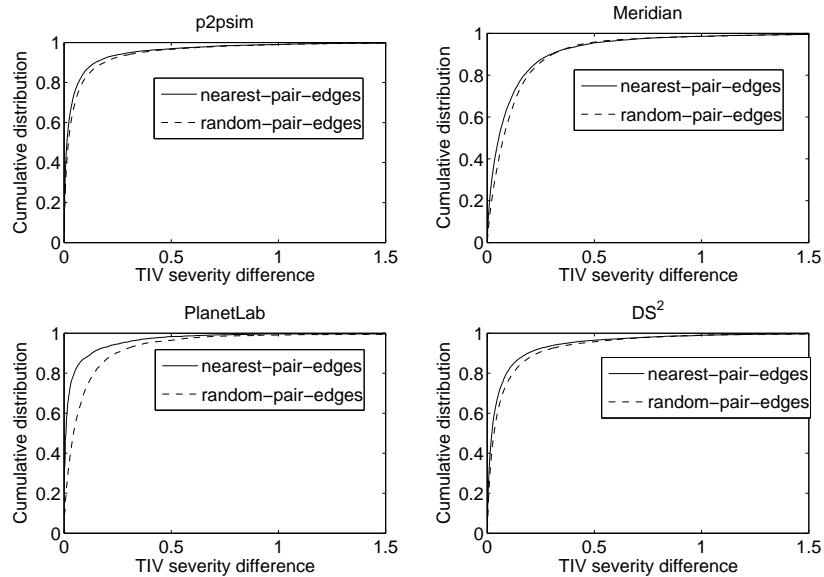


Figure 3.6 : Proximity property of TIVs

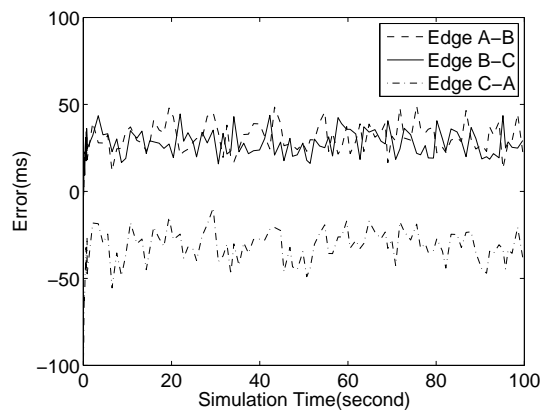


Figure 3.7 : Vivaldi error trace for a simple 3-node network with TIV

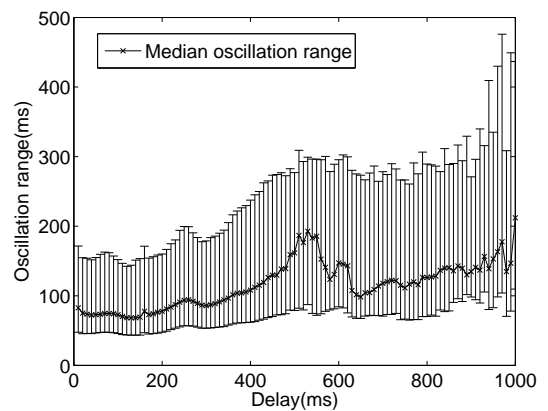


Figure 3.8 : Distribution of the oscillation range of all the edges

(*euclidean_distance - measured_delay*).

As we can see from Figure 3.7, Vivaldi cannot find perfect positions for the nodes and it is stuck in endless oscillations. If we look into the detailed behavior of a node A , what is happening here is, every time A probes a neighbor B , node A will adjust its coordinates

based on this probe to decrease the prediction error of the edge AB . However, because of triangle inequality violation among ABC , there does not exist good positions for nodes A, B, C in the Euclidean space to preserve the delays AB, AC and BC perfectly. So, the result is, every movement A makes to decrease the error of edge AB will increase the error of edge AC and the overall error of node A remains high. For the whole network, the effect is, all the nodes are adjusting their coordinates to decrease the error for the currently probed edges, but it does not help to increase the overall prediction accuracy. All the nodes in the system are wandering rapidly and the error of edges fluctuates. This result shows that, the existence of TIV can hurt the embedding of the whole network and introduce large prediction errors on edges.

Let's extend our analysis from the simple scenario to real Internet measurements. Among all the triangles constructed by any 3 nodes in the DS^2 data set, around 12% of them violate triangle inequality. It turns out that, these violations in the measured data have a significant impact on Vivaldi's performance. When Vivaldi is run on the DS^2 data, the median absolute error is 20ms and the 90th percentile absolute error is 140ms. Moreover, the nodes are moving rapidly. The median movement speed is 1.61 ms per step and the 90th percentile movement speed is 6.18 ms per step. To get a sense of the ranges the predicted distances are oscillating in, we define the oscillation range of an edge to be $(\max(\text{prediction_distance}) - \min(\text{prediction_distance}))$ and collect the oscillation range for all the edges during a 500s simulation period. In Figure 3.8, we divide all the edges into 100 bins with the width of 10ms, and use the error bar to plot the distribution of the oscillation range of the edges in each bin. The ceiling of the error bar is the 90th percentile, the bottom is the 10th percentile, and the marked line is the median. As can be seen, the prediction values are oscillating over large ranges, and the range is large even for edges that are very short.

3.4 Strawman Solutions to Deal with TIVs

Recent studies have reported the inaccuracy of network embedding caused by TIVs and several techniques have been proposed to accommodate TIVs [MS04, LZSS06]. Thus, we would like to first determine how much these techniques can help improve the application performance of network coordinates. We use neighbor selection application to evaluate the application performance in our experiments. The reason is, neighbor selection is one of the primary applications of network coordinates and it has been reported in many previous studies that the coordinates accuracy problem induced by TIVs significantly degrade the neighbor selection performance of network coordinates. Furthermore, we would also like to determine whether removing edges that have large TIV severity can help reduce the impact of TIVs.

3.4.1 Neighbor Selection Experiment Methodology

For the rest of this chapter, unless otherwise noted, the methodology for the closest neighbor selection experiments is as follows. All experiments are performed using the DS^2 4000-node measured Internet delay data set [ZNA⁺06].

For Vivaldi, each node picks 32 random nodes as Vivaldi probing neighbors and iteratively perform Vivaldi embedding computations for 100 seconds (simulation time). The metric space used is a 5-dimensional Euclidean space. A random subset of 200 nodes are selected as candidates for the closest neighbor selection experiment, the remaining 3800 nodes act as clients. This is done so that the chance that a candidate is a Vivaldi probing neighbor of a client is small. One closest neighbor selection test is performed for each client based on the delay predictions given by Vivaldi coordinates. We record the percentage penalty for each test, where percentage penalty is defined as

$$\frac{(\text{delay_to_selected} - \text{delay_to_optimal}) \times 100}{\text{delay_to_optimal}}$$

We run the experiment 5 times using 5 different random subsets of 200 nodes as candidates. Results reported are cumulative over the 5 runs.

3.4.2 Existing Models for Accommodating TIV in Network Embedding

Several existing proposals for improving network embedding systems attempt to accommodate TIVs. In this section, we focus on these existing proposals. IDES [MS04] is a network coordinates system designed to allow for triangle inequality violations and delay asymmetry in the delay space. It is not based on embedding into a metric space. Instead, in IDES, each node is assigned an incoming and an outgoing vector by matrix factorization techniques, such as Singular Value Decomposition (SVD) or Non-negative Matrix Factorization (NMF). The distance between node i and j is estimated by the inner product of i 's outgoing vector and j 's incoming vector.

On the other hand, Lee et al. [LZSS06] proposed to add a localized adjustment term (LAT) to Euclidean coordinates to account for TIVs. In this method, each node x has a d dimension Euclidean coordinate c_x and a non-Euclidean adjustment e_x , and $(c_x; e_x)$ is used to denote the final coordinates of node x . The distance d_{xy} between two nodes x and y is estimated by $\hat{d}_{xy} = d(c_x, c_y) + e_x + e_y$, where $d(c_x, c_y)$ is the Euclidean distance between c_x and c_y . The e_x is set to half of the average error for all the measurements from node x to a set of sampled nodes. Let S denote the set of randomly sampled nodes measured from node x , then $e_x = \frac{\sum_{y \in S} (d_{xy} - \hat{d}_{xy})}{2|S|}$.

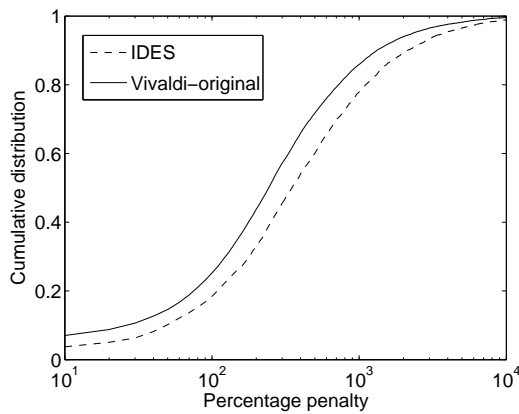


Figure 3.9 : Neighbor selection performance for IDES

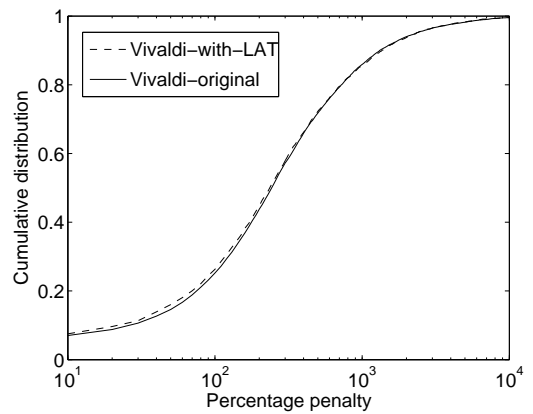


Figure 3.10 : Neighbor selection performance for Vivaldi-LAT

Both IDES and LAT have been shown to provide better aggregate prediction accuracy for Internet delays than the basic Euclidean network embedding approach. However, it remains to be seen whether they can successfully improve performance with respect to the neighbor selection problem. Figure 3.9 and Figure 3.10 show the neighbor selection results of IDES and Vivaldi with LAT on the DS^2 data set. We can see that, neighbor selection performance of IDES is actually worse than that of Vivaldi, and Vivaldi with LAT is only slightly better than the original Vivaldi. For IDES, although it does not constrain predicted delays to satisfy the triangle inequality as in an Euclidean model, it is hard to find vectors that simultaneously approximate TIVs and estimate network delays accurately for all nodes. For the LAT technique, although the localized adjustment term can introduce the non-Euclidean effects with respect to a set of sampled nodes, it is still very hard to predict the triangle inequality violations over the entire network accurately. Ultimately, increase in aggregate prediction accuracy does not always translate into increased neighbor selection performance.

3.4.3 Naive avoidance of TIVs

In this section, we consider another high level strategy based on removing edges that cause TIVs. If we assume we have global information about the delay space, then all the TIVs can be easily calculated and identified. In this case, a straight forward strategy is to clean up the delay matrix by removing the edges that cause severe TIVs.

To test this strategy, we identify 20% of the edges in the delay matrix that have the largest TIV severity. These edges are simply not used by Vivaldi probing neighbors. Neighbor selection performance of this approach is shown in Figure 3.11.

From Figure 3.11, we can see that simply excluding some high violation edges only marginally improve the neighbor selection performance of Vivaldi. The reason for this result is that TIV is a wide spread feature of Internet delays. Thus, naively removing some outliers in the delay matrix cannot remove the fundamental problems caused by TIVs in Vivaldi.

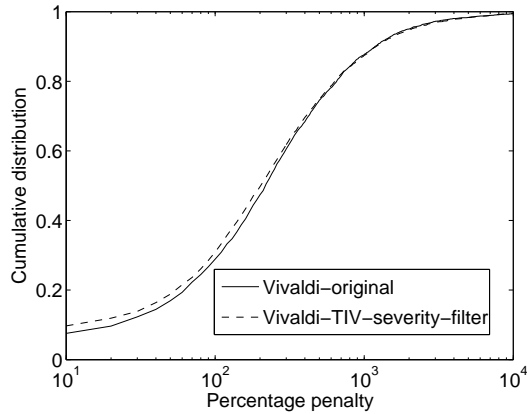


Figure 3.11 : Neighbor selection performance for Vivaldi with TIV severity filter

In summary, given the global information, simply excluding some TIV outliers to clean the delay matrix does not help to improve the performance of network coordinates. Hence even with full knowledge of TIVs, we still need specifically refined strategies for different applications to avoid the impact of TIVs.

3.5 TIV Alert Mechanism

In this section, we propose a TIV alert mechanism and show that it can be used to introduce TIV awareness into network coordinates systems.

3.5.1 Alerting Severe TIVs by Metric Embedding Error

Based on our findings so far, it seems difficult to derive a simple model that can predict the TIV severity of an edge accurately. Instead, we ask, is it possible to at least identify edges that are likely to cause severe TIV based on a small random sample of delays from the network? In other words, if we measure a small number of random edges in the network, can we infer information about whether a given edge causes severe TIVs? An interesting observation of network embedding mechanisms can help in this case.

In network embedding mechanisms, each node can measure the delays to a small num-

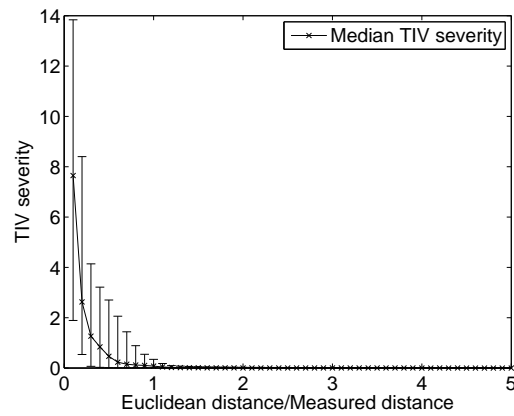


Figure 3.12 : TIV severity for edges with different prediction ratios

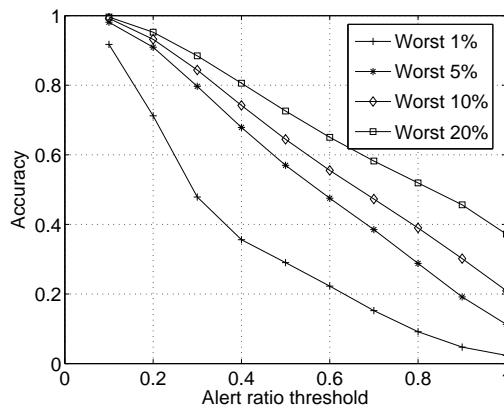


Figure 3.13 : Accuracy of TIV alert mechanism

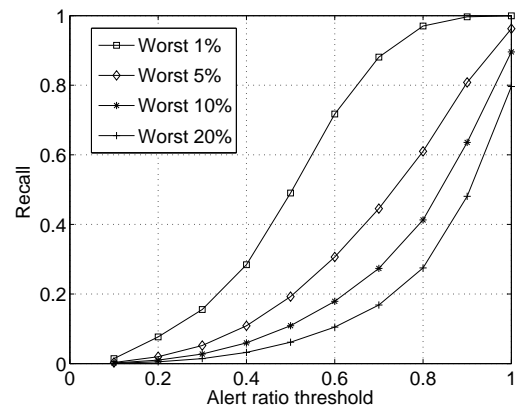


Figure 3.14 : Recall rate of TIV alert mechanism

ber of random nodes and the measured delays are embedded into a metric space. Because of the TIVs among the network delays, it is impossible to predict all the delays accurately by fitting all the nodes into a metric space. However, an interesting observation is, if an edge causes severe TIVs with other edges, it is highly likely that this edge will be shrunk significantly in the metric space. The reason behind this is that, if an edge between node A and B causes a lot of TIVs with other edges, there must be many alternate paths between A and B that are shorter than the measured delay. Thus, the optimization procedures in net-

work embedding mechanisms will tend to sacrifice the accuracy of the edge AB in order to preserve the many other short edges to minimize the overall prediction error.

To demonstrate this observation, we embed the DS^2 data into a 5D Euclidean space using the Vivaldi algorithm, take a snapshot of the produced steady state coordinates, and study the relationship between the prediction error of edges and the TIV severity caused by them. We defined the prediction ratio $\frac{\text{euclidean_distance}}{\text{measured_distance}}$ to measure the Vivaldi prediction error. For the prediction ratio between 0 and 5, we set up 50 bins each with the range of 0.1. For each bin, we collect all the edges whose prediction ratio falling in this bin. We use an error bar to demonstrate the distribution of the TIV severity of all the edges in this bin. The ceiling of the error bar indicates the 90th percentile, the bottom is the 10th percentile, and the marked line shows the median value. Figure 3.12 shows the TIV severity of the edges with different prediction ratios. For those edges whose prediction ratio is very small, i.e. those edges that are shrunk a lot in the Euclidean space, their TIV severity tends to be very high. As the prediction ratios of edges increase, their TIV severities decrease. For those edges whose prediction ratios are larger than 2, their TIV severity is almost 0. Although the TIV severity is highly variable within each prediction ratio bin, there is a clear trend that as the prediction ratio becomes smaller, the distribution of TIV severity shifts towards higher values. This trend is consistently observed for any snapshot of Vivaldi's steady state coordinates.

Based on the result in Figure 3.12, it is not possible to exactly predict the TIV severity of an edge based on its prediction ratio. But the result inspires our idea to use the prediction ratio in network embedding as a heuristic indicator for the TIV severity of a given edge. The question is, how effective can the prediction ratio be used as a TIV alert mechanism? To answer this question, we evaluate the accuracy of using different prediction ratio thresholds to alert the worst 1%, 5%, 10% and 20% of edges with the highest TIV severity. The accuracy and recall rate are shown in Figure 3.13 and Figure 3.14. As can be seen, if we use a tight threshold to raise alerts, the alerting accuracy is very high. For example, if we use a 0.1 threshold, we can report the top 1% worst edges with the accuracy of 92%,

and report the top 5% worst edges with the accuracy of 98%. However, the problem of a tight threshold is that the recall rate is very low, which means we can only report few edges among all the ones with severe TIV. For example, as shown in Figure 3.14, if we use a 0.1 threshold, we can only report 1% of the worst 10% edges. As we relax the alert threshold, the recall rate is increased but the accuracy is decreased. Thus, there is a trade-off between the recall rate and the alert accuracy. To use this TIV alert mechanism in practice, we can choose a threshold to provide enough number of alerts while preserving reasonable accuracy. For example, with a 0.6 threshold, the TIV alert mechanism raises alert on around 4% of the edges. Among those edges, 70% of the worst 1% edges are reported, and 65% of them belong to the worst 20% edges. The recall rate for the worst 20% edges is relatively low, this is simply because the TIV alert mechanism raises alert on only 4% of the edges. What is more important is that the edges identified are highly probable to cause severe TIVs.

In summary, we have shown that the prediction ratio of an edge in network embedding has a useful relationship with its TIV severity. The prediction ratio can thus be used to provide a TIV alert mechanism. This makes it possible to introduce TIV awareness into the design of distributed systems. In the following sections, we demonstrate how the TIV alert mechanism can be used in the Vivaldi system.

3.5.2 Using TIV Alert Mechanism in Vivaldi

Since Vivaldi is itself a distributed network embedding mechanism, it is easy to determine the prediction ratio for the edges that have been measured. So it does not require any additional overhead to use the TIV alert mechanism in Vivaldi. A convenient way to use the TIV alert mechanism in Vivaldi is to use the prediction ratio to identify those edges with high TIV severities, and refine the neighbor set for each node.

In particular, the enhanced system we call *dynamic neighbor Vivaldi* can be explained as follows: Vivaldi is started normally, with each node having 32 random neighbors. After Vivaldi runs for a period T , all the nodes begin to update their neighbors. To update the

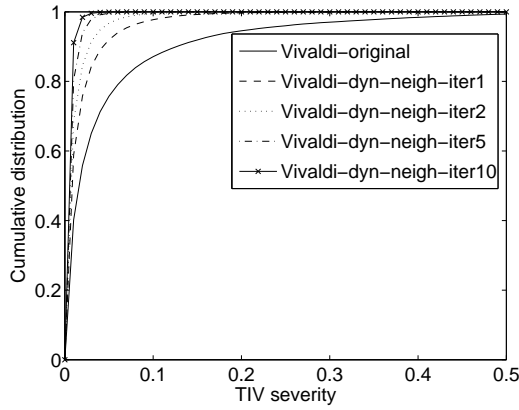


Figure 3.15 : TIV severity of Vivaldi neighbor edges

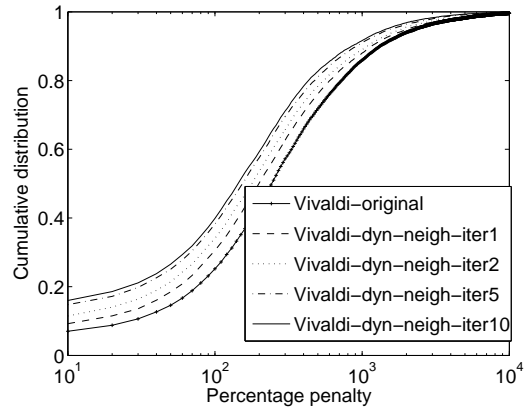


Figure 3.16 : Neighbor selection performance of dynamic neighbor Vivaldi

neighbor set, each node samples another 32 random neighbors. Combined with the original 32 neighbors, each node now has 64 neighbor candidates. The 64 neighbor candidates are ranked by their prediction ratio based on the current Vivaldi coordinates. The prediction ratio here is defined to be $\frac{\text{euclidean_distance}}{\text{measured_delay}}$. If the prediction ratio of an edge is very small, it means this edge is shrunk a lot and it is more likely to cause severe TIV. So we remove the 32 nodes with smallest prediction ratios among the 64 neighbor candidates, and the remaining 32 nodes are used as the neighbors in the next iteration. This procedure is performed iteratively, and the neighbor set is updated every T time. Currently, T is set to 100 second simulation time to make sure that Vivaldi coordinates are converged in each iteration.

To evaluate dynamic neighbor Vivaldi, we first want to show how effectively the TIV alert mechanism can remove edges that cause severe TIV in the neighbor update procedure. Figure 3.15 shows the TIV severity of all the neighbor edges when we update neighbor set from iteration 0 (original random 32 neighbors) to iteration 10. From this figure, we can clearly see that the TIV severity of neighbor edges become smaller and smaller when we iteratively update neighbor set for each node, which means we effectively remove those edges with high TIV severities.

Figure 3.16 shows the neighbor selection performance of dynamic neighbor Vivaldi. We can see that, our technique can effectively improve the neighbor selection performance of Vivaldi when we iteratively update Vivaldi neighbor sets. After only 10 iterations, the performance is clearly better than that of original Vivaldi. In previous sections, we have shown that just removing TIV outliers does not help to improve Vivaldi's performance. The reason why dynamic neighbor Vivaldi can perform better is that, instead of trying to remove outliers, dynamic neighbor Vivaldi refines the neighbor set to eliminate TIVs among Vivaldi neighbors. Furthermore, the dynamic neighbor technique does not add much overhead. The TIV alert mechanism is effective at making Vivaldi TIV aware.

3.6 Summary

TIV in Internet delays can degrade the performance of distributed systems that neglect TIV when choosing overlay neighbors. We have investigated the severity of TIV in several delay data sets and highlighted the irregular behavior of TIV. We have also investigated the problems caused by TIV a representative network coordinates system (Vivaldi) and the feasibility of several strawman solutions. Finally, we have proposed a TIV alert mechanism that can help identify edges with severe TIVs and shown that it can enhance Vivaldi to become TIV-aware. The application of TIV alert mechanism is not limited to network coordinates systems. The idea of TIV alert can be used to provide TIV awareness in many other distributed systems. For example, in paper [WZN07], we have demonstrated that the TIV alert mechanism can be used to provide TIV awareness in another neighbor selection mechanism, the Meridian system [WSS05], and reduce the impact of TIVs on Meridian. In [LLS07], the authors have shown that the TIV alert mechanism can provide TIV awareness in overlay networks and can be used to find shorter overlay paths. We believe these findings serve as a first step towards building robust TIV-aware distributed systems.

Chapter 4

Coordinates Stability in Decentralized Network Coordinates Systems

4.1 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

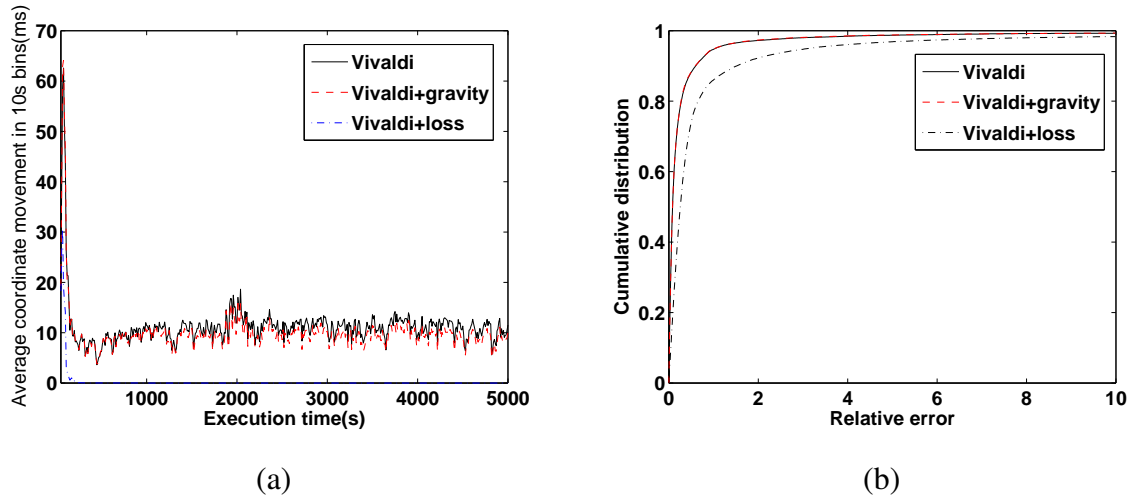


Figure 4.1 : Coordinates stability and accuracy of Vivaldi with gravity and loss factor (a) Stability (b) Accuracy

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.

4.2 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 [dLUB04], [PLS05], [LPS06], [LGS07], [WZN07] 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. [dLUB04], [LPS06], [LGS07] 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 [dLUB04], 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 [LPS06], 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 [LGS07], 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 [LGS07] 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 4.1 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.

4.3 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 [dLUB04] 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. 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.

4.4 A Novel Algorithm for Coordinates Stability

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

4.4.1 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

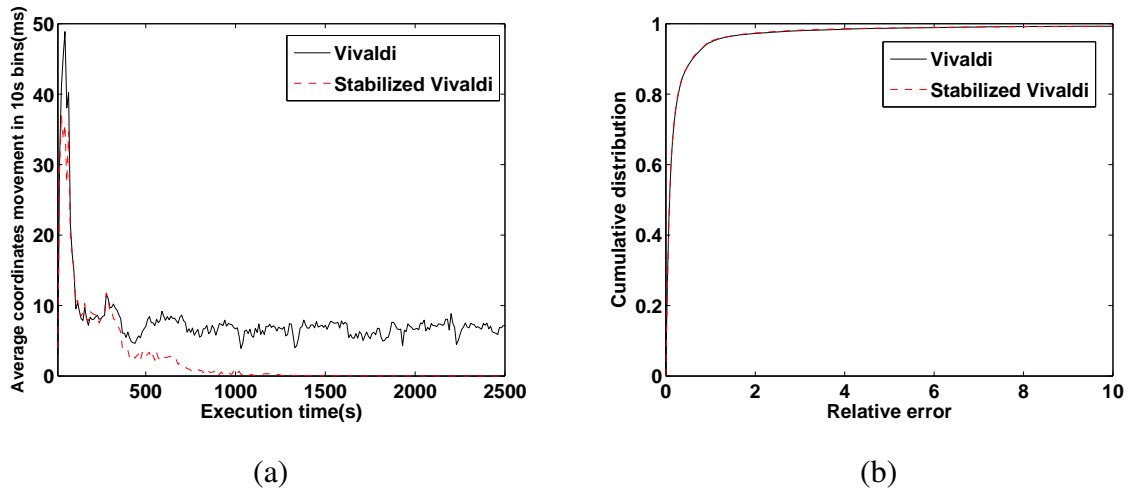


Figure 4.2 : Performance of Stabilized Vivaldi in a PlanetLab experiment (a) Stability (b) Accuracy

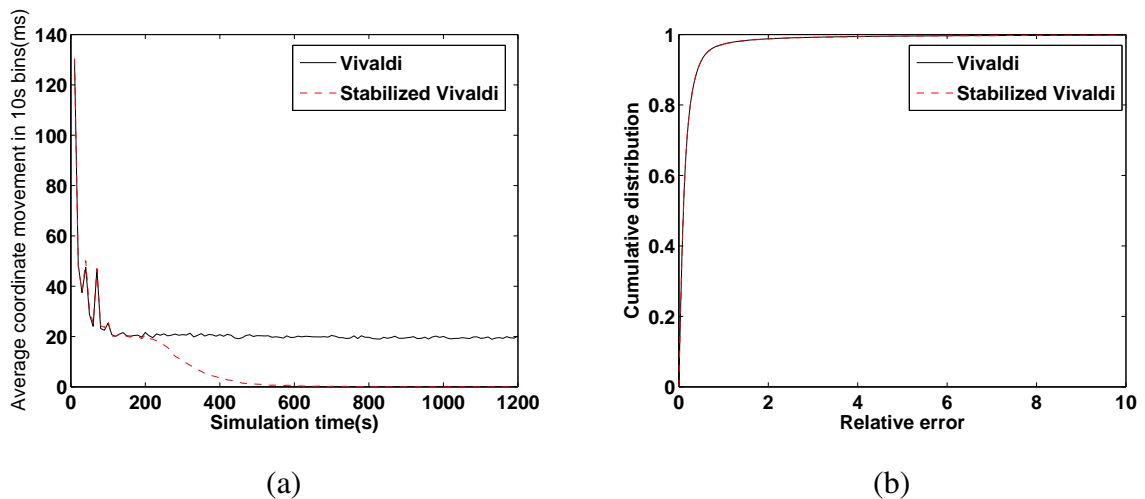


Figure 4.3 : Performance of Stabilized Vivaldi in a simulation experiment on King data (a) Stability (b) Accuracy

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 neigh-

bors' 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

$$\epsilon_i = \text{predict_delay}(A, i) - \text{measured_delay}(A, i)$$

(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: 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.

4.4.2 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 4.2 shows the results for the PlanetLab experiment. Figure 4.2 (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 4.2(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 4.3 shows the coordinates stability and accuracy of the stabilized Vivaldi system in the simulation experiment. From Figure 4.3(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 be closer apart, 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.

4.4.3 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 expire quickly even before the network coordinates can converge in adapting to them. Therefore, the strategy to deal with them is 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. In our algorithm, after the coordinates are

stabilized, a node can continue to observe its delays and errors to other nodes. If it observes a large increase in error (what is considered to be large should be defined by the designer of the system), it can adapt to the change by triggering coordinates re-computation.

Chapter 5

Coordinates Security in Decentralized Network Coordinates Systems

In this section, we study the coordinates security problem in decentralized network coordinates systems.

5.1 Problem Formulation

As network coordinates systems have been applied in a wide variety of applications, they are more likely to become targets of malicious attacks. Malicious nodes have different incentives to attack network coordinates systems. Here we just list several examples. One incentive is to create chaos in network coordinates as a form of denial of service(DoS) attack. Since network coordinates can be set up as a navigation service in the Internet and provide network proximity in many applications, the disruption of network coordinates could result in the mis-functioning of many applications that rely on network coordinates. Therefore, network coordinates become a primary target of hackers to perform DoS attacks. Another incentive is, malicious nodes can get potential benefits by disrupting network coordinates. One of the primary applications of network coordinates is overlay construction. Malicious nodes can pretend to be far away from all the other nodes by disrupting network coordinates, such that they will not be chosen as neighbors. By doing this, malicious nodes can avoid system obligations and alleviate their resource consumption. The third example is that, malicious nodes can break other security mechanisms by disrupting network coordinates. For example, previous studies, e.g. [BK05], have proposed to use network coordinates to defend against Sybil attacks. Malicious nodes can easily break the Sybil attack defense by disrupting network coordinates.

Malicious nodes can attack network coordinates systems easily. They can simply provide wrong information to mislead other nodes in the system. A malicious node can arbitrarily lie about its current coordinates. It can also arbitrarily inflate its network delays to other nodes. When other nodes use the wrong information to compute their coordinates, the resulting coordinates can be totally skewed.

Previous work [KMTD06, KMB⁺07, ZNR07] 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 network coordinates. Thus, the coordinates security problem is, 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?

Several previous studies [KMB⁺07, ZNR07] have proposed to use statistical detection mechanisms to identify misbehaving nodes. The first question we ask is how well the statistical detection mechanisms can contain the misbehaving nodes? To answer this question, we introduce an analytical model to understand the behavior of statistical detection in network coordinates systems and study the performance of statistical detection in protecting network coordinates.

5.2 Modeling Statistical Detection Mechanisms

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 5.1 demonstrates an 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 systems,

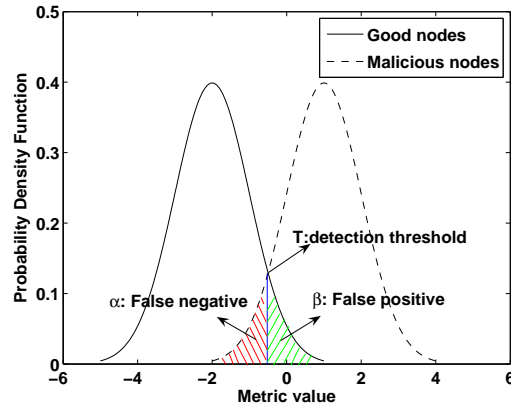


Figure 5.1 : Demonstration of metric distributions for statistical detection

each node checks its neighbors using the detection mechanism before relying on them as references to compute coordinates. A node has to check its neighbors at every coordinates computation step because a malicious node can perform an attack at any step. 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 to detect malicious neighbors.

We can use the following model to analyze the performance of statistical detection mechanisms. Suppose in a network coordinates system with N nodes, each node has M neighbors and the fraction of malicious nodes in the whole system is q . For an arbitrary node A , at step i of its coordinates computation procedure, the fraction of malicious nodes in A 's neighbor set is p_i . The node A will check the current neighbor B to protect the system from malicious attacks. Let us consider the fraction of malicious nodes in A 's neighbor set at step $i + 1$ (i.e. p_{i+1}). After A checks the neighbor B , the number of malicious nodes in its neighbor set can be decreased by 1, or can be increased by 1, or remains unchanged. If B is a malicious node, and A successfully detects B as malicious, and A also fortunately replaces B with a good node, the number of malicious nodes in A 's neighbor set will be decreased by 1. On the contrary, if B is a good node, and A falsely treats B as malicious,

and A unfortunately replaces B with a malicious node, the number of malicious nodes in A 's neighbor set will be increased by 1. For all the other cases, the number of malicious nodes in A 's neighbor set will remain unchanged. Correspondingly, the possible values for p_{i+1} are $p_i + \frac{1}{M}$, $p_i - \frac{1}{M}$ and p_i . Let the false negative rate of the detection mechanism be α , and the false positive rate be β . When the detection procedure proceeds, the metric distributions for good nodes and bad nodes can be changed, therefore the false negative and false positive rates vary as functions of p_i , denoted as $\alpha(p_i)$ and $\beta(p_i)$. The probability to accurately detect a malicious nodes is $(1 - \alpha(p_i))$, and the probability of treating a good node as malicious is $\beta(p_i)$. Therefore, the probabilities of all the potential values of p_{i+1} can be computed as following:

$$p_{i+1} = \begin{cases} p_i - \frac{1}{M} & \text{prob. : } P_a = p_i \times (1 - \alpha(p_i)) \times (1 - q) \\ p_i + \frac{1}{M} & \text{prob. : } P_b = (1 - p_i) \times \beta(p_i) \times q \\ p_i & \text{prob. : } P_c = 1 - P_a - P_b \end{cases}$$

If $P_a \geq P_b$, the detection mechanism can make progress to reduce the fraction of malicious nodes in A 's neighbor set. When $P_a = P_b$, the detection mechanism runs into an equilibrium state where it cannot effectively reduce the fraction of malicious nodes in A 's neighbor set any further. This equilibrium state represents the performance bottleneck of the detection mechanism.

In a simple case where $\alpha(p_i)$ and $\beta(p_i)$ are constant numbers α and β , we can easily compute the fraction of malicious nodes in the neighbor sets when the detection mechanism is in its equilibrium state. Let us denote this fraction as p' , then $p' \times (1 - \alpha) \times (1 - q) = (1 - p') \times \beta \times q$, rearranging terms yield

$$p' = \frac{1}{1 + \frac{(1-\alpha)}{\beta}(\frac{1}{q} - 1)}$$

Figure 5.2 shows the relationship between α , β and p' with $q = 0.3$. This graph demonstrates two interesting observations. First, keeping the false positive rate β small is critical to the performance of detection mechanisms. As long as β is small, even if α is moderately

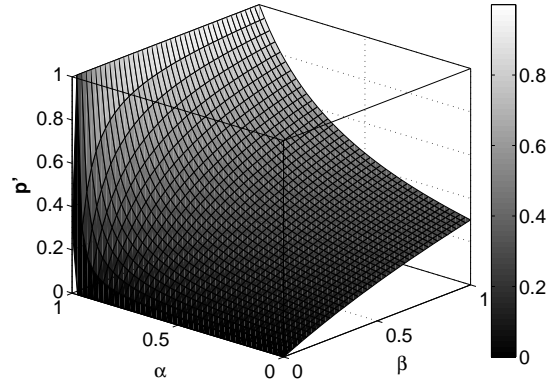


Figure 5.2 : The relationship between α , β and p' ($q = 0.3$)

large, p' is still very small. Second, a high false negative rate α is especially harmful to the detection performance. If α is too large, even if β is small, p' is always high.

In practice, however, $\alpha(p_i)$ and $\beta(p_i)$ can be any strange functions. The evolution of functions $\alpha(p_i)$ and $\beta(p_i)$ actually has a significant impact on the performance of a detection mechanism. If both $\alpha(p_i)$ and $\beta(p_i)$ are decreasing when p_i is getting smaller, the detection mechanism can decrease its performance bottleneck and continue to make progress. However, if $\frac{(1-\alpha(p_i))}{\beta(p_i)}$ is decreasing or constant when p_i is getting smaller, the detection mechanism will reach its performance bottleneck. In the following sections, we will demonstrate how $\alpha(p_i)$ and $\beta(p_i)$ evolve for existing detection mechanisms by empirical experiments.

5.3 Empirical Study of the Existing Statistical Detection Mechanisms

5.3.1 Basic Principles

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

The Kalman filter detection mechanism - Kaafar et al. [KMB⁺07] propose a Kalman filter mechanism to detect malicious nodes in network coordinates systems.

The Kalman filter mechanism assumes a set of trusted nodes called surveyors. Each

surveyor only chooses other surveyors as its neighbors when computing coordinates. The trusted surveyors are meant to provide a reference for the “normal” error convergence features of network coordinates. Each surveyor trains a Kalman filter to model the error convergence features it observes. The result is a set of Kalman filter parameters. These surveyor Kalman filter parameters are then used by ordinary nodes to detect malicious neighbors. The assumption is that, nodes that are nearby may have more similar error convergence features. Thus, an ordinary node uses the Kalman filter parameters from its closest surveyor. An ordinary node uses a Kalman filter with these parameters to predict the errors with respect to its neighbors. If the actual error with respect to a neighbor deviates too much from the predicted value, the neighbor is deemed malicious.

Internet, the closest surveyor can still be far from an ordinary node. Even in a network coordinates system without any malicious nodes, it is not clear that the errors observed by a node can be predicted accurately by the Kalman filter model learned by a remote surveyor. Second, when malicious nodes are present, the coordinates of a good neighbor can be impacted by the behavior of malicious nodes. With the polluted input data, it becomes more difficult for the Kalman filter to correctly differentiate good neighbors and malicious neighbors.

The outlier detection mechanism - Unlike the Kalman filter mechanism, the outlier detection mechanism [ZNR07] does not assume any trusted infrastructure. Each node in the system relies on its own observations to detect misbehaving neighbors. Two outlier detection techniques are proposed in [ZNR07]: spatial outlier detection and temporal outlier detection.

In spatial outlier detection, a node records an observation tuple $\{neighbor\ error, change\ in\ neighbor's\ coordinates\}$ for each coordinates computation step. The node saves the most recent u observation tuples. To perform spatial outlier detection, the node computes the Mahalanobis distance between the current observation tuple and the centroid of u most recent tuples. If the Mahalanobis distance is too large, the current neighbor is deemed malicious.

In temporal outlier detection, a node records a tuple $\{\textit{neighbor error}, \textit{local error}, \textit{change in neighbor's coordinates}, \textit{change in its own coordinates}\}$ for each coordinates computation step. To perform temporal outlier detection, the node updates the temporal centroid, and standard deviation of all the data in the tuple based on all the historical observations. The node then computes the simplified Mahalanobis distance between the current tuple and the temporal centroid. If the current tuple deviates too much from the temporal centroid, the current neighbor is deemed malicious.

Outlier detection mechanisms rely on the introspective statistical metrics to detect malicious nodes. When there are too many malicious nodes in the system, malicious nodes are no longer outliers. Therefore we cannot expect outlier detection mechanisms to work very well if a large fraction of nodes are malicious.

5.3.2 Malicious Attacks

To evaluate the effectiveness of statistical detection mechanisms systematically, we use a set of malicious attacks. In a network coordinates system, a malicious node can lie about its coordinates and inflate its delays to other nodes. It can generate both fixed and dynamic coordinates when it is lying to another node. However, it can only lie about its delay to another node consistently because the victim only uses the minimum measured delay in coordinates computation. Although it is not possible to exhaust all attacks, we choose a set of attacks in which each represents one category of attacks that stresses a certain aspect of the system.

Isolation attack - The isolation attack is described in [KMTD06]. In this attack, the malicious nodes agree on a large exclusion zone and randomly set their own coordinates outside of this zone so as to attract other nodes out of the exclusion zone and isolate some target node inside the zone. In this attack, a malicious node always reports a fixed coordinates. In our experiment, each malicious node chooses a set of random coordinates outside the zone $[-100 \text{ ms}, 100 \text{ ms}]$ but within $[-300 \text{ ms}, 300 \text{ ms}]$ on each dimension of the coordinate space.

Oscillation attack - The oscillation attack is described in [ZNR07]. In this attack, the malicious nodes send victim nodes erroneous random coordinates selected over the coordinate space. This attack generates randomly moving coordinates. In our experiment, the malicious nodes randomly generate their coordinates within range [-100 ms, 100ms] on each dimension.

Deflation attack - The deflation attack is described in [ZNR07]. An attacker sends the victim node coordinates that minimize the difference between the actual RTT and estimated RTT. The victims will stay at its unconverged coordinates and believe that it has low prediction error. In the deflation attack, malicious nodes always exhibit good prediction error to victims. This attack cannot be easily detected if only the prediction error is considered.

In addition to these attacks that have been described in previous studies, we also introduce three new attacks.

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. Both oscillation attack and shifting attack can report erroneous and dynamic coordinates. The difference between these two attacks is, the shifting attack can mimic the coordinate movement of normal nodes. In our experiments, malicious nodes shift their coordinates for up to 300ms.

Delay attack - In the delay attack, malicious nodes focus on attacking the delay measurement. A delay attacker randomly inflates the delay to a victim up to 300 ms. The delay attacks are honest in coordinates computation. Once a delay attacker has reported faked delays to its neighbors, it also use that faked delays to compute its own coordinates.

Inflation attack - This is a more aggressive form of the inflation attack described in [ZNR07]. In [ZNR07], an inflation attacker artificially inflates the delay to a victim. In our aggressive version, an attacker sends the victim node shifted coordinates as well as an artificially high delay by randomly inflating the delay up to 300ms. In all the previous attacks, malicious nodes only lie about their coordinates. In the inflation attack, malicious nodes lie about both their coordinates and delays to other nodes.

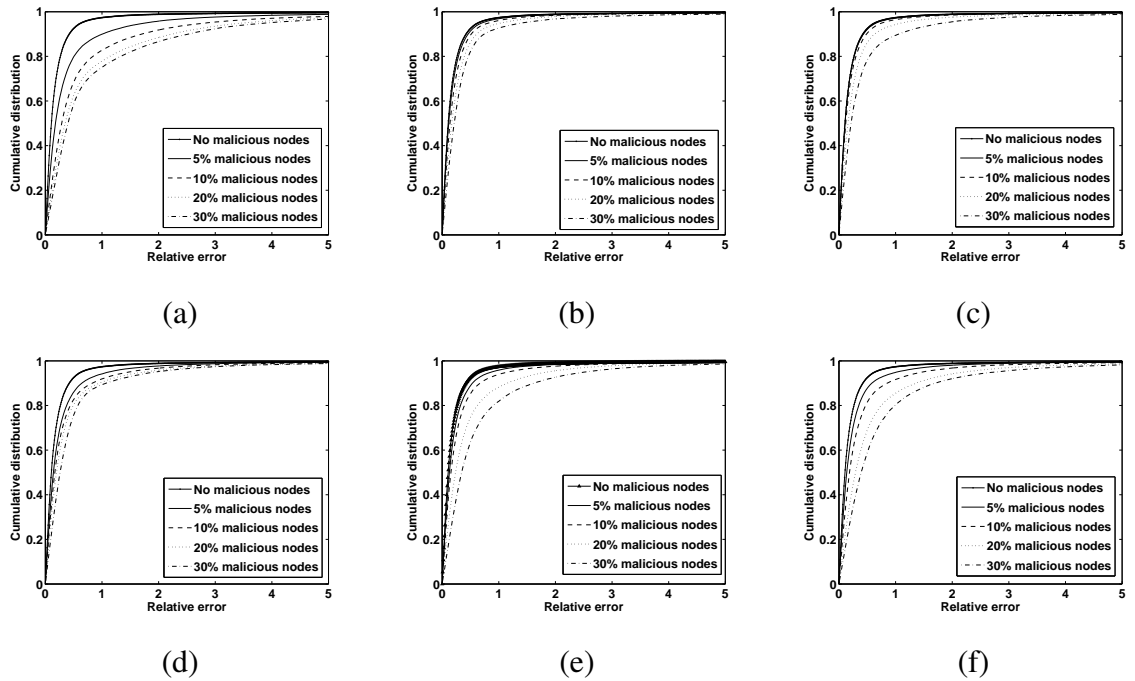


Figure 5.3 : Impact of attacks on coordinate accuracy (a) Isolation attack (b) Oscillation attack (c) Deflation attack (d) Shifting attack (e) Delay attack (f) Inflation attack

In our empirical study of statistical detection mechanisms, we follow the same experimental methodology as introduced in Section ???. We apply these attacks to the Vivaldi system. In the Vivaldi system, all the nodes need to reply probes with their local prediction error. When malicious nodes perform these attacks, they always report the forged delays and coordinates with low prediction error to magnify the impact of the attacks. Figure 5.3 shows the impact of these attacks on the accuracy of legitimate nodes' coordinates in simulation experiments. We can see that all 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. The simulations are still based on the Vivaldi system. 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. For the outlier detection mechanisms, we use the recommended detection thresholds, which are 1.5 for spatial outlier de-

tection, 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 our Kalman filter experiments, we randomly select 8% of the nodes as surveyors, which is recommended in [KMB⁺07].

In our simulation experiments, we always record the most recent 10000 tests in the system to monitor the detection performance and the fraction of malicious nodes in neighbor sets during the detection procedure. By doing this, we can observe the evolution properties of the statistical detection mechanisms.

5.3.3 Behavior of Outlier Detection Mechanisms

Figure 5.4 and Figure 5.5 show the detection performance of temporal and spatial outlier detection mechanisms against all the attacks. In each graph of these figures, the top graph shows the evolution of the false positive and false negative rates and the bottom graph shows how the fraction of malicious nodes in the neighbor sets changes as the detection mechanism proceeds. From these graphs, we can make several interesting observations.

First let us look at the detection performance of outlier detection mechanisms for the oscillation attack. From Figure 5.5 (b) and Figure 5.4 (b), we can see that, outlier detection mechanisms can identify oscillation attackers at the early stage of detection procedure. However, as the detection procedure is proceeding, outlier detection mechanisms gradually lose their power.

Figure 5.6 shows the probability distribution function of statistical metrics of all the detection mechanisms for oscillation attack. For outlier detection mechanisms, we can see from Figure 5.6(a) and (b) that, since malicious nodes just randomly generates faked coordinates in oscillation attack, the Mahalanobis distance metric does present different distributions on good nodes and malicious nodes. That is why at the early stage of detection procedure, outlier detection mechanisms can effectively identify oscillation attackers. However, the problem of outlier detection mechanisms is that, the Mahalanobis distance

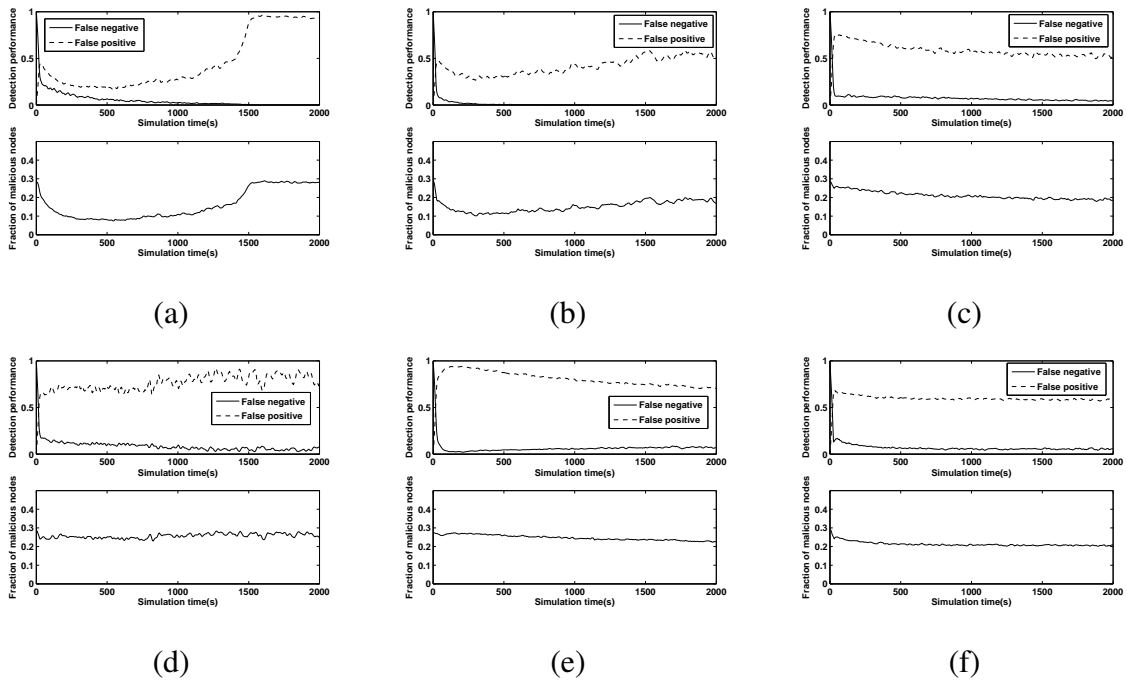


Figure 5.4 : Detection performance of temporal outlier detection (a) Deflation attack (b) Oscillation attack (c) Inflation attack (d) Shifting attack (e) Delay attack (f) Isolation attack

metric is extracted from the network coordinates that are being attacked. When malicious nodes attack the system for a long time, this metric can be significantly impacted. We can see from Figure ??(c) and (d) 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. Although we can still observe the differentiation power from the metric distributions, the detection performance is significantly degraded because outlier detection mechanisms use fixed thresholds in the detection procedure. In this case, there is a hope that we can improve the performance of outlier detection mechanisms by following the shift of metric distribution and always choosing the optimal threshold in detection procedure. However, this is difficult in practice because we cannot get metric distribution for good nodes and malicious nodes.

We can make similar observations from the detection performance for the deflation attack.

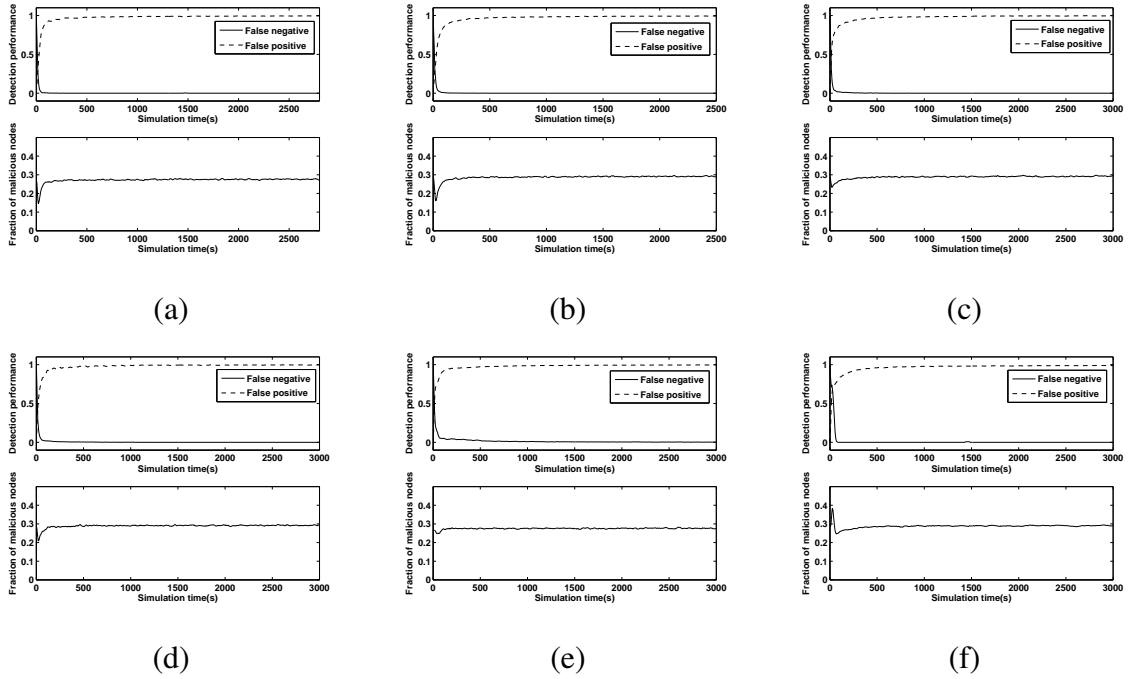


Figure 5.5 : Detection performance of spatial outlier detection (a) Deflation attack (b) Oscillation attack (c) Inflation attack (d) Shifting attack (e) Delay attack (f) Isolation attack

The detection performance results for shifting attack reveal another interesting observation. As illustrated in Figure 5.4(d) and Figure 5.5(d), none of the outlier detection mechanism can effectively detect shifting attack. All the mechanisms have high false negative and high false positive rates, and reach their performance bottlenecks with large fractions of malicious nodes in the neighbor sets. Figure 5.7 plots the metric distribution of all the outlier detection mechanisms at 0 second and 2000 second for the shifting attack. From these graphs, we can see that, the metric distributions of good nodes and malicious nodes cannot be differentiated by a simple threshold. The means that detection mechanisms fail on shifting attack is not just because the shifting of metric distribution. It is because the statistical metrics totally lose their power in differentiating good nodes and shifting attackers.

The reason why outlier detection fails on these attacks is that, both spatial outlier detection and temporal outlier detection mechanisms use neighbor's error and their coordinates

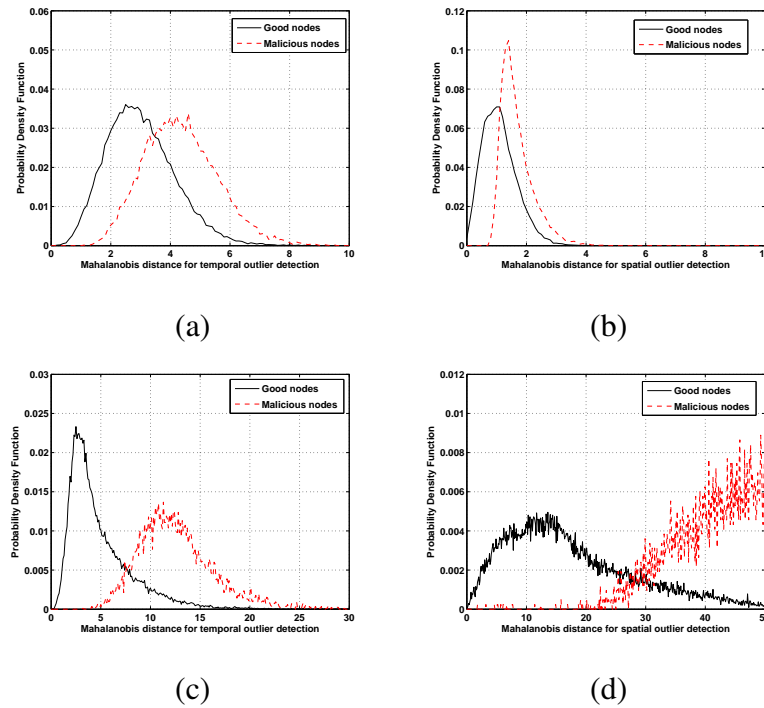


Figure 5.6 : Metric distribution of outlier detection mechanisms for oscillation attack (a) Temporal outlier detection at time 0 second (b) Spatial outlier detection at time 0 second (c) Temporal outlier detection at time 2000 second (d) Spatial outlier detection at time 2000 second

movement information to compute statistical metrics. Malicious nodes always report small errors to victim nodes and the good nodes' errors are always limited in range $(0,1)$ in the Vivaldi system. The neighbors' errors do not have strong power to differentiate good nodes and malicious nodes. Therefore, the outlier detection mechanisms mainly rely on the coordinates movement to detect malicious nodes. If the malicious nodes' coordinates move rapidly, 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 attacks.

We can make the similar observation from the experiment results for inflation attack, delay attack and isolation attack. The reason why outlier detection mechanisms fail on these attacks is similar to shifting attack. In inflation attack and delay attack, malicious

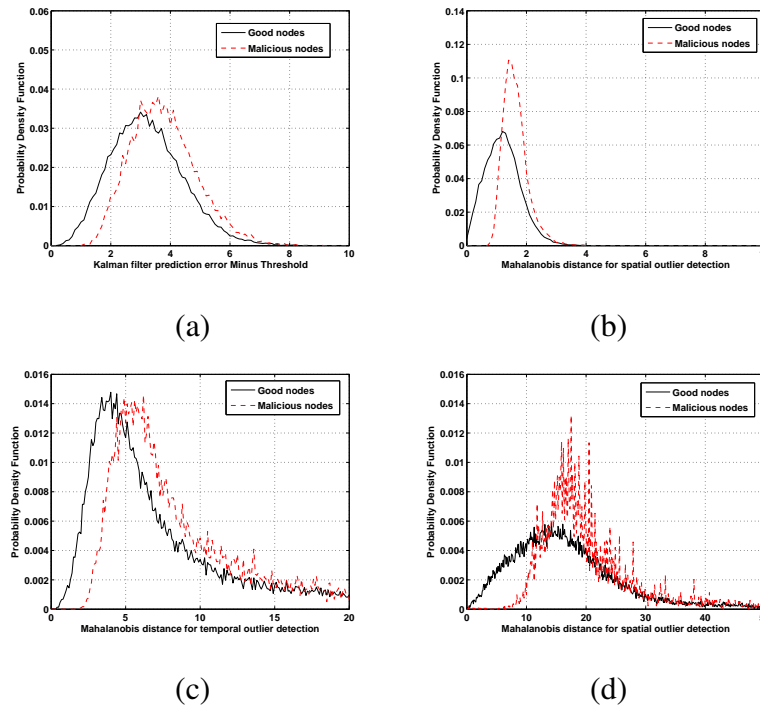


Figure 5.7 : Metric distribution of outlier detection mechanisms for the shifting attack (a) Temporal outlier detection at time 0 second (b) Spatial outlier detection at time 0 second (c) Temporal outlier detection at time 2000 second (d) Spatial outlier detection at time 2000 second

nodes can mimic the normal coordinates movements and in isolation attack, malicious nodes consistently report fixed coordinates, which makes hard to identify these attackers by outlier detection.

5.3.4 Behavior of the Kalman Filter Detection Mechanism

Figure 5.8 shows the detection performance of the Kalman filter mechanism for all the attacks. As illustrated in Figure 5.8(a) and (b), the Kalman filter detection mechanism evolves reasonably well in detecting the isolation attack and the oscillation attack. The Kalman filter detection mechanism can identify a large fraction of isolation attackers and oscillation attackers, and reaches its performance bottleneck with around 10% malicious nodes in neighbor sets. However, the Kalman filter mechanism does not work well for

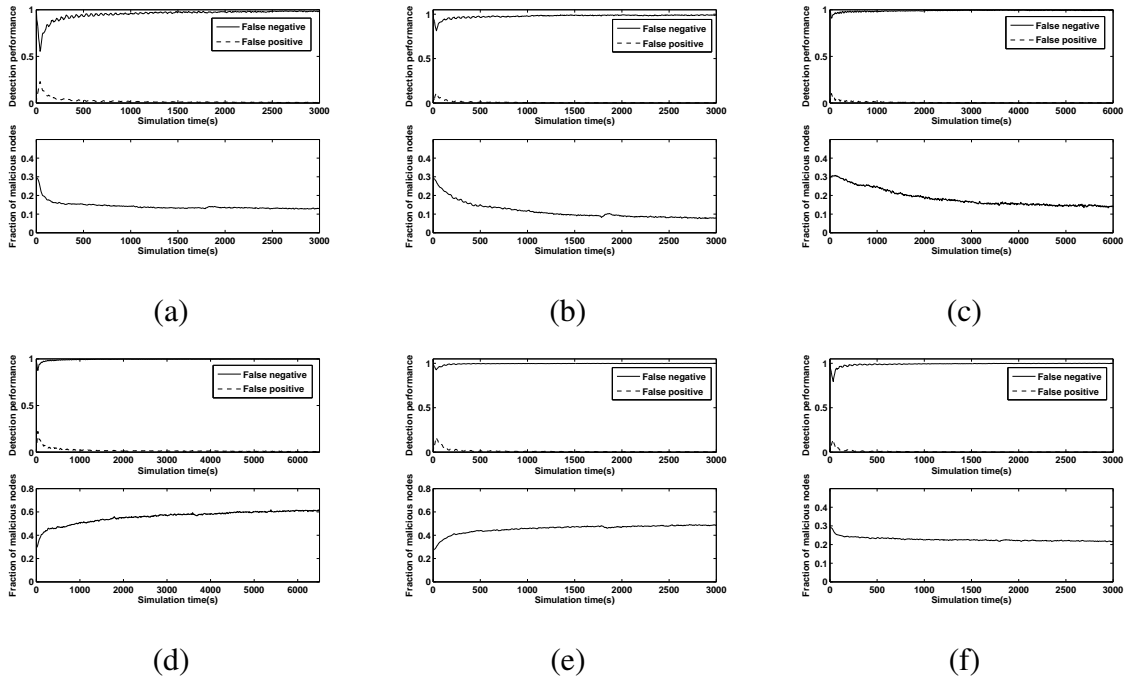


Figure 5.8 : Detection performance of the Kalman filter detection mechanism (a) Isolation attack (b) Oscillation attack (c) Deflation attack (d) Inflation attack (e) Delay attack (f) Shifting attack

deflation attack, shift attack, delay attack and inflation attack. Especially for delay attack and inflation attack, the Kalman filter detection mechanism actually brings more malicious nodes into the neighbor sets.

Figure 5.9 shows the metric distributions of the Kalman filter detection for oscillation attack, shift attack and inflation attack. From these results, we can see that, the metric distribution of the Kalman filter detection does not demonstrate strong power in differentiating good nodes and malicious nodes. The Kalman filter mechanism has very high false negative and very low false positive in detecting malicious nodes. This reveals that the Kalman filter mechanism is a conservative detection procedure in the sense that, the Kalman filter mechanism might ignore many of the malicious nodes already in the neighbor set, but it always tries to avoid treating good nodes as malicious so that to reduce the possibility of

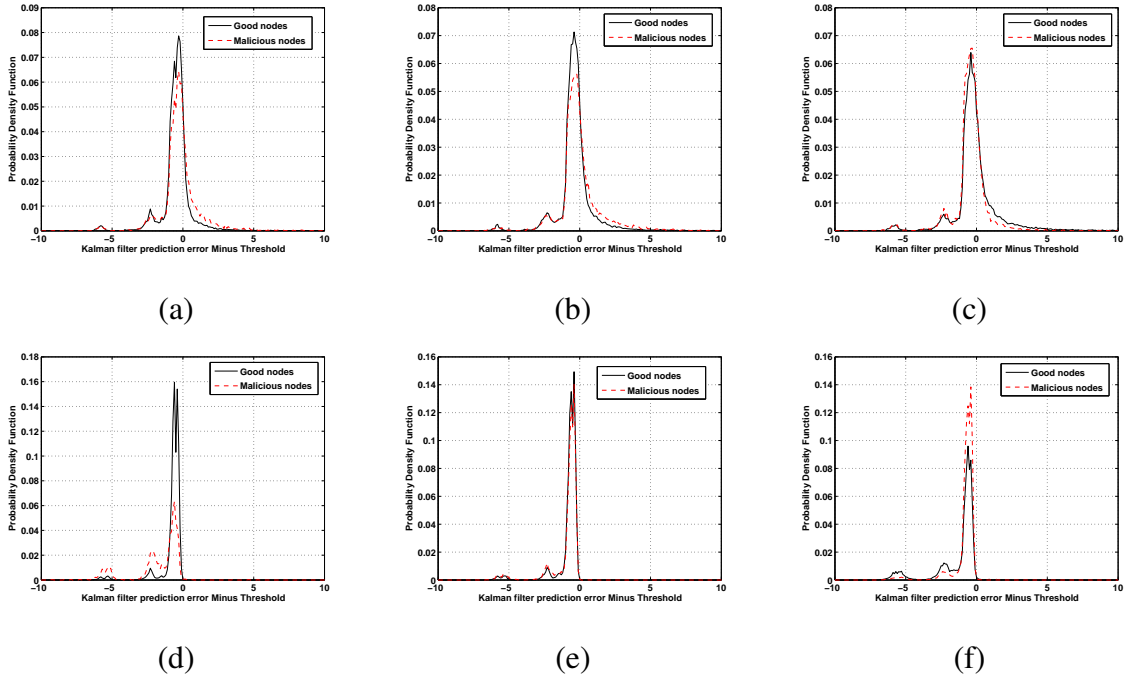


Figure 5.9 : Metric distribution of the Kalman filter detection mechanism for different attacks (a) Oscillation attack at time 0 second (b) Shifting attack at time 0 second (c) Inflation attack at time 0 second (d) Oscillation attack at time 2000 second (e) Shifting attack at time 2000 second (f) Inflation attack at time 2000 second

introducing new malicious nodes in neighbor sets. As implied in our analytical model, the performance bottleneck of a detection mechanism is decided by $\frac{1-\alpha}{\beta}$, α is the false negative rate and β is the false positive rate of the detection mechanism. In the case of the Kalman filter detection, since α is very large, and β is very small, slightly different false positive rate can cause very different detection performance. We can read from Figure 5.9 that, the false positive rates of the Kalman filter mechanism are just slightly different for oscillation attack, shift attack and inflation attack, but the final detection performance are not even similar for these attacks.

The reason why the Kalman filter detection does not work well on shifting attack and deflation attack is the Kalman filter model yield large error in predicting the relative error of neighbor edges. We believe the error comes from two parts. First, we observe that, the

relative error of neighbor edges has very large variances. The Kalman filter is an adaptive weighted averaging filter. When the random variables have large variances, even each node uses the Kalman filter parameter learned from its own historical error, it still cannot predict the future error very accurately. The second part is, we observe that, the average delay among ordinary nodes and their surveyors is 44 ms. It brings more error by predicting the error of neighbor edges using the Kalman filter parameters learned from remote nodes. When the Kalman filter model always yields large error on both good nodes and malicious nodes, it does not have a strong power to differentiate them.

As observed in Figure 5.8, the Kalman filter mechanism works especially poorly in detecting delay attack and inflation attack. The reason is that, in these attacks, the attackers inflate their delays to other nodes. The Kalman filter mechanism relies on the relative error of neighbor edges to detect malicious neighbors. However, when malicious neighbors inflate the delays, the relative error of these neighbor edges tend to be small. This makes the malicious neighbors behave as a normal and accurate neighbor. Therefore the Kalman filter cannot identify these malicious nodes. This reveals a fundamental limitation of the Kalman filter mechanism, which is when malicious nodes lie about their delays, they can easily fool the Kalman filter detector by generating good errors that are very similar to good nodes.

5.3.5 Limitations of Statistical Detection Mechanisms

Our analysis and empirical evaluation reveal the limitations of statistical detection mechanisms. Our analytical model demonstrates that any statistical detection mechanism has a performance bottleneck which depends on its false negative rate and false positive rate and the evolution of a detection mechanism has a significant impact on its performance. All statistical mechanisms inevitably suffer from some amount of false negatives and false positives. Thus, they could never completely protect a coordinates system.

Moreover, empirically, we have found that existing statistical mechanisms have difficulty in detecting aggressive attacks. First, all the existing detection mechanisms have

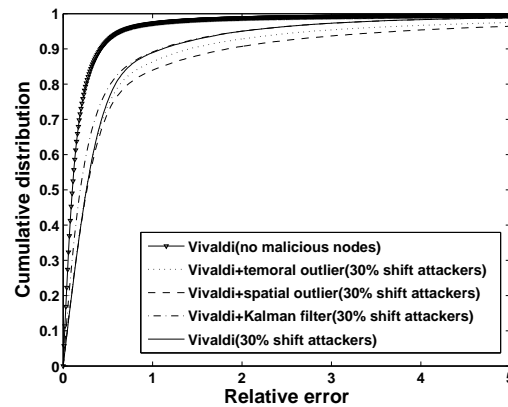


Figure 5.10 : Accuracy of the Vivaldi coordinates with statistical detection mechanisms under shifting attacks

their limitations in detecting some kinds of attacks. For example, outlier detection mechanisms cannot effectively detect the attacks when malicious nodes can mimic the coordinate movements of good nodes; The Kalman filter mechanism cannot detect the attacks when malicious nodes lie about delays to mimic the error of good nodes.

Second, there are some kinds of attacks that none of the existing statistical detection mechanisms can effectively detect. For example, with 30% malicious nodes performing shifting attack and delay attack, none of the statistical detection mechanisms can identify these attacks effectively. The cost of not being able to identify the attackers is high. We can see from Figure 5.10 that, with 30% malicious nodes performing the shifting attack, most of the benefits of the coordinates system are gone. Notice that, in our empirical study, we only test the detection performance of existing detection mechanisms by five attacks. In practical environments, the malicious nodes can perform many other arbitrary behaviors to attack the system. These results show that, all the existing statistical detection mechanisms cannot achieve an acceptable level of security in protecting network coordinates systems.

Although all our results indicate the difficulties of protecting network coordinates systems by statistical detection mechanisms, we do not rule out the possibility that better statistical detection mechanisms may exist. Our analytical model provides some guidelines

on how to design a better detection mechanism to protect network coordinates systems. To design a good detection mechanism, it is important to reduce the false positive rate. As shown in our analytical model, if a detection mechanism has low false positive rate, even the false negative rate is high, it can still reach its performance bottleneck with a low fraction of malicious nodes in neighbor sets. In some sense, the false positive rate decides the performance bottleneck of a detection mechanism, and the false negative rate decides the efficiency of detection procedure.

5.4 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.

5.4.1 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. 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) [CL99] or Byzantine Fault Detection (BFD) [HKD07] 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 [HKD07], 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 use 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 [HKD07] 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 con-

tact 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. 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. Consequently, 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

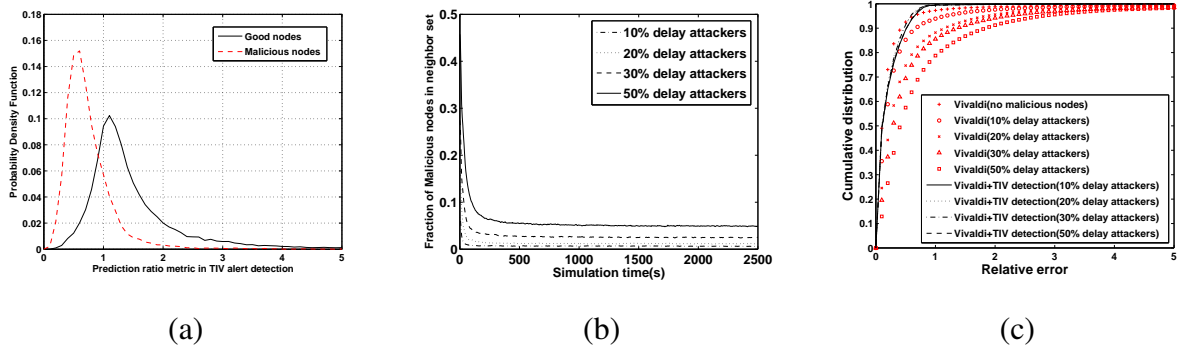


Figure 5.11 : The performance of TIV detection on delay attack (a) Metric distribution (b) Detection performance (c) Coordinates accuracy

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.

5.4.2 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 attack, the delay attack, we can design a statistical detection mechanism to specifically detect faked delays.

The way delay attackers impact good nodes' coordinates is that the artificially inflated delays 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. This inspires our idea to detect faked delays using the TIV alert technique proposed in [WZN07]. The TIV alert technique uses the prediction ratio ($\frac{\text{predicted}}{\text{measured}}$) of coordinates as a heuristic indicator to detect the edges causing severe TIVs in the network. As discussed in [WZN07], the edges that cause severe TIVs are highly likely to have a low prediction ratio. Since the faked delays cause severe TIVs, we should be able to use the prediction ratio metric to detect them.

Figure 5.11(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.

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 5.11(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. Figure 5.11(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 protect good nodes' coordinates effectively. 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.

5.4.3 Summary

In this section, we consider the coordinates security problem from the perspective of defending Byzantine faulty nodes in network coordinates systems. We study the coordinates security in two parts: security of coordinates computation and security of delay measurement. Our results show that, the accountability protocol can completely protect coordinates computation part. Furthermore, we discuss two strategies to protect delay measurement. The results demonstrate that, a TIV detection mechanism can effectively detect faked delays and reduce the impact of faked delay attacks on network coordinates.

Chapter 6

Discussion and Conclusion

In this thesis, we have studied several design issues on the accuracy, stability and security of network coordinates systems. The findings of this thesis reveal positive and negative points about network coordinates systems.

We have studied the impact of TIVs on network coordinates systems, and analyzed the properties of TIVs in the Internet delays. The negative point is our results show that the TIVs among network delays can significantly degrade the accuracy of network coordinates and cause coordinates oscillation in the Vivaldi system. Our analysis results highlight the irregular behavior of the TIVs in the Internet delays, which implies the difficulty of modeling the TIVs among the Internet delays. The positive point is, we have found a TIV alert mechanism that leverages the network embedding error to identify the edges causing severe TIVs. The evaluation results demonstrate the TIV alert mechanism can be used to reduce the impact of TIVs on the neighbor selection performance of network coordinates systems. The application of TIV alert mechanism is not limited to network coordinates systems. The idea of TIV alert can be used to provide TIV awareness in many other distributed systems. For example, in paper [WZN07], we have demonstrated that the TIV alert mechanism can be used to provide TIV awareness in another neighbor selection mechanism, the Meridian system [WSS05], and reduce the impact of TIVs on Meridian. In [LLS07], the authors have shown that the TIV alert mechanism can provide TIV awareness in overlay networks and can be used to find shorter overlay paths.

Our study on stability problem leads to another positive point on network coordinates system design. Although all the existing decentralized solutions fail to provide stable network coordinates, we have found a new model of stabilizing coordinates by eliminating

error can achieve coordinates stability without hurting accuracy in decentralized network coordinates systems.

We have proposed an analytical model to study the performance of statistical detection mechanisms in protecting network coordinates systems from malicious attacks. Our analysis and results expose the limitations of statistical detection mechanisms for securing network coordinates. We have found that, all the statistical detection mechanisms have a performance bottleneck, and all the existing statistical detection mechanisms cannot achieve an acceptable level of security against aggressive attacks. This highlights the difficulties of securing network coordinates in decentralized network coordinates systems. However, this problem can be solved. We have studied the coordinates security problem from the perspective of defending Byzantine faulty nodes. Our results show that, an accountability protocol can completely protect the coordinates computation part and a TIV alert detection mechanism can effectively protect the delay measurement part.

The findings in this thesis provide new guidelines on the design and application of network coordinates systems.

Centralized and decentralized network coordinates systems represent different design points. With our new findings, we can re-visit the trade-off between centralized and decentralized network coordinates systems. From the perspective of coordinates accuracy, centralized and decentralized network coordinates systems can achieve similar performance [DCKM04]. From the perspective of system scalability, centralized network coordinates systems assume a set of landmarks and suffer from the scalability problem. In contrast, decentralized network coordinates systems have good scalability. From the stability perspective, based on our results, centralized and decentralized network coordinates system can both achieve coordinates stability with good accuracy. From the security perspective, because all the nodes in centralized network coordinates systems use a set of trusted landmarks to compute their coordinates, they do not suffer from security problem. Although the evaluation results indicate the difficulty of securing network coordinates in decentralized systems, our results show that this problem can be solved. However, we can

	Accuracy	Scalability	Stability	Security
Centralized	Yes	No	Yes	Yes
Decentralized	Yes	Yes	Yes	Yes

Table 6.1 : Trade-offs between centralized and decentralized network coordinates systems

see that securing network coordinates in decentralized systems is a much harder problem, which requires additional verification protocols that significantly increase the complexity of network coordinates systems. These trade-offs are summarized in Table 6.1.

These findings also provide guidelines for applications that need network coordinates. Because of TIVs among the Internet delays, network coordinates cannot perfectly predict network delays. Our results show that a simple TIV alert mechanism can significantly reduce the impact of TIVs on network coordinates and improve the application performance. However, we do not have a general way of using TIV alert. We have shown that how to leverage TIV alert in neighbor selection application. For other applications, there are different strategies of using TIV alert.

For applications that require stable network coordinates, our findings indicate that a decentralized system can provide the needed stability without management overhead of landmarks. For applications with high security requirement, our findings show that, coordinates can be secured in decentralized systems, but it will significantly increase the system complexity. In this case, centralized network coordinates systems might be good choices since they can guarantee the security of network coordinates by design.

Bibliography

- [AEG⁺06] J. Aspnes, T. Eren, D. K. Goldenberg, A. S. Morse, W. Whiteley, Y. R. Yang, B. D. O. Anderson, and P. N. Belhumeur. A theory of network localization. *IEEE Transactions on Mobile Computing*, 5(12):1663–1678, December 2006.
- [AM04] I. Abraham and D. Malkhi. Compact routing on euclidean metrics. In *Proceedings of PODC 2004*, July 2004.
- [BK05] R.A. Bazzi and G. Konjevod. On the establishment of of distinct identities in overlay networks. In *Proceedings of ACM PODC'05*, July 2005.
- [CCRK03] 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.
- [CL99] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of ACM OSDI*, February 1999.
- [DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceeding of ACM SIGCOMM*, August 2004.
- [dLUB04] C. de Launois, S. Uhlig, and O. Bonaventure. A stable and distributed network coordinate system. In *Technical report, University Catholique de Louvain*, December 2004.
- [EGW⁺04] T. Ern, D. Goldenberg, W. Whitley, Y. R. Yang, A. S. Morse, B.D.O. Anderson, and P. N. Belhumeur. Rigidity, computation, and randomization of network localization. In *Proceeding of IEEE INFOCOM'04*, April 2004.

- [FJP⁺99] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniwicz, and Y. Jin. An architecture for a global Internet host distance estimation service. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.
- [GKK⁺04] 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.
- [Hen92] B. Hendrickson. Conditions for unique graph realization. *SIAM J. Comput.*, pages 65–84, 1992.
- [HKD07] 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.
- [JH97] D. J. Jacobs and B. Hendrickson. An algorithm for two dimensional rigidity percolation: the pebble game. *J. Comput. Phys.*, February 1997.
- [KMB⁺07] M.A. Kaafar, L. Mathy, C. Barakat, K. Salamatian, T. Turlitti, and W. Dabbous. Securing internet coordinate embedding systems. In *Proceeding of SIGCOMM'07*, August 2007.
- [KMTD06] M.A. Kaafar, L. Mathy, T. Turlitti, and W. Dabbous. Virtual networks under attack: Disrupting internet coordinate systems. In *Proceeding of CoNext'06*, December 2006.
- [Lam70] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Eng. Math.*, pages 331–340, 1970.
- [LGP⁺05] 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.

- [LGS07] J. Ledlie, P. Gardner, , and M. Seltzer. Network coordinates in the wild. In *Proceeding of USENIX NSDI'07*, April 2007.
- [LHC03] H. Lim, J. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proceedings of IMC*, Miami, FL, October 2003.
- [LLS07] C. Lumezanu, D. Levin, and N. Spring. Peer wise discovery and negotiation of faster path. In *Proceedings of HotNets-VI*, November 2007.
- [LPMS07] J. Ledlie, P. Pietzuch, M. Mitzenmacher, and M. Seltzer. Wired geometric routing. In *Proceedings of IPTPS 2007*, February 2007.
- [LPS06] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *Proceeding of International Conference on Distributed Computing Systems*, July 2006.
- [LZSS06] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of Euclidean embedding of Internet hosts. In *Proc. SIGMETRICS 2006*, June 2006.
- [MS04] 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.
- [NGD⁺07] 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 ACM and USENIX NSDI'07*, April 2007.
- [NZ02] T. S. E. Ng and H. Zhang. Predicting Internet networking distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM*, June 2002.
- [NZ04] T. S. E. Ng and H. Zhang. A network positioning system for the internet. In *Proceedings of USENIX Annual Technical Conference*, June 2004.

- [p2p] p2psim. <http://www.pdos.lcs.mit.edu/p2psim/>.
- [PCW⁺03] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proceedings of IPTPS*, 2003.
- [PLS05] 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.
- [SCH⁺99] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *Proceedings of ACM Sigcomm*, August 1999.
- [SKW04] A. Slivkins, J. Kleinberg, and T. Wexler. Triangulation and Embedding using Small Sets of Beacons. In *Proceedings of FOCS*, 2004.
- [Sli04] A. Slivkins. Distributed Approaches to Triangulation and Embedding. In *Proceedings 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
- [Sli06] A. Slivkins. Network Distance Estimation with Guarantees for All Node Pairs. Technical report, Cornell University, 2006.
- [ST03] 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.
- [ST04] 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.
- [TC03] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of IMC*, Miami, FL, October 2003.

- [WSS05] Bernard Wong, Aleksandrs Slivkins, and Emin Gun Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proceedings of ACM SIGCOMM*, August 2005.
- [WZN07] 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.
- [ZHLF06] R. Zhang, Y. Hu, X. Lin, and S. Fahmy. A hierarchical approach to internet distance prediction. In *Proceedings of IEEE ICDCS*, Lisboa, Portugal, 2006.
- [ZLPG05] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy Griffin. Internet routing policies and round-trip times. In *the 6th anual Passive and Active Measurement Workshop*, Boston, MA, March 2005.
- [ZNA⁺06] 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.
- [ZNR07] 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.
- [ZTH⁺06] 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.