Towards Network Triangle Inequality Violation Aware Distributed Systems*

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

ABSTRACT

Many distributed systems rely on neighbor selection mechanisms to create overlay structures that have good network performance. These neighbor selection mechanisms often assume the triangle inequality holds for Internet delays. However, the reality is that the triangle inequality is violated by Internet delays. This phenomenon creates a strange environment that confuses neighbor selection mechanisms. This paper investigates the properties of triangle inequality violation (TIV) in Internet delays, the impacts of TIV on representative neighbor selection mechanisms, specifically Vivaldi and Meridian, and avenues to reduce these impacts. We propose a TIV alert mechanism that can inform neighbor selection mechanisms to avoid the pitfalls caused by TIVs and improve their effectiveness.

Categories and Subject Descriptors

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

General Terms

Measurement, Performance, Experimentation

Keywords

Internet delay space, triangle inequality violations, analysis, neighbor selection, distributed system

1. INTRODUCTION

The Internet is an interesting environment where delay[1] measurements do not always "make sense". Often, even if node A is close to node B and node B is close to node C, node A can be very far from node C. That is, Internet delays often violate the triangle inequality. Numerous studies have reported the existence of triangle inequality violations (TIV) in the Internet delay space (e.g. [25, 4, 17, 39, 11, 35]). As discussed in [39], delay space 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.

In what ways does this property matter? Of course this property does not break the basic best-effort datagram delivery service provided by the Internet. However, this property can degrade the performance of distributed systems that assume the triangle inequality holds for Internet delays. Investigating such impacts and avenues to reduce them is the subject of this study.

For distributed systems whose performance is dominated by the efficiency of network communications, it is important that communication neighbors are selected to minimize delays. For example, in a tree-based overlay multicast system, a joining node needs to find an existing group member who is nearby to serve as its parent in the tree. This neighbor selection operation is crucial for many distributed systems including those that are based on structured overlays (e.g. [31, 23, 38, 22]) and unstructured overlays (e.g. [1, 6, 24]).

When neighbor selection is based on brute-force network measurements, the quality of the selected neighbor cannot be affected by delay TIVs. However, as the number of nodes in the system scales up, brute-force measurements become unattractive due to the communication overhead and the time it takes to collect the measurements. In order to perform neighbor selection without brute-force measurements, it is often useful to make certain assumptions about the properties of the delay space. Many existing solutions to this problem assume the triangle inequality holds for Internet delays in order to infer delays between nodes (e.g. [3, 34, 24, 8, 33, 7]).

Two representative solutions are Vivaldi [3], which is based on the network embedding approach, and Meridian [34], which is based on the recursive probing approach. Both solutions require a small amount of offline network delay measurements that help guide the selection of a nearby neighbor. Vivaldi is expected to be less accurate, but requires no online network measurements. On the other hand, Meridian requires online network measurements and is expected to be highly accurate. Both solutions rely on the triangle inequality assumption to infer the delays between nodes in the system.

How do TIVs impact the performance of these neighbor selection solutions? Is it possible to reduce the problems caused by TIVs? To shed light on these questions, we first analyze the severity of TIVs in several available delay data sets. This leads to the following observations. First, although it makes intuitive sense that the larger the delay of an edge (i.e. a path between two nodes), the more severe the TIV it causes, such generalization is not reliable

*This research was sponsored by the NSF under CAREER Award CNS-0448546. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. government.

[1]For simplicity, we will use "delay" in place of "round-trip delay."

IMC'07, October 24-26, 2007, San Diego, California, USA.

because the TIV severities of edges of similar delays are highly variable. Thus, it is not accurate to predict the TIV severity of an edge simply by considering its delay. Secondly, even if two edges AB and CD have very nearby end-nodes (i.e. A has small delay to C and B has small delay to D), the difference between their TIV severities is no smaller than the difference between two randomly chosen edges. Therefore, it is also not accurate to predict the TIV severity of an edge by simply considering the TIV severity of another nearby edge.

Even though these observations highlight complex and irregular characteristics in TIV that we cannot yet explain completely, we can identify the impact of TIV on Vivaldi and Meridian by studying their behavior under TIV. When faced with TIV, Vivaldi resolves the TIV by forcing edges to shrink or stretch in the embedding space. This results in oscillations in the embedding. Moreover, the magnitude of such oscillation is so large that the predicted delay for a 10ms edge can vary by as much as 175ms. This behavior hurts the neighbor selection performance of Vivaldi. For Meridian, the online recursive probing approach is expected to yield very good neighbor selection accuracy. Under ideal conditions where probing overhead is unrestricted and without TIV, the Meridian approach is expected to find the correct nearest neighbor. However, due to TIV, the ring structures created by Meridian contain inconsistencies. The result is that, even under ideal conditions, Meridian cannot find the nearest neighbor for 13% of the cases.

We continue the investigation by testing several strawman strategies for coping with TIVs. We test existing models for network coordinates that accommodate TIV [16, 11] but find that these methods do not benefit the neighbor selection problem. We also test the naive approach of removing edges that cause the most severe TIVs given the global information. What we learn is that Vivaldi and Meridian fail on this strategy for different reasons and thus even with full knowledge of TIVs, fine-grained strategies are needed to avoid the impact of TIVs in different neighbor selection mechanisms.

Finally, we propose a TIV alert mechanism. This TIV alert mechanism does not predict the severity of TIV of an edge. Rather, it simply identifies edges that are likely to cause severe TIVs. The design of this TIV alert mechanism stems from our basic observation that when a delay space with TIV is embedded into an Euclidean space, the edges that have severe TIVs tend to be shrunk in the resulting embedding. We explain this mechanism and show that it can help identify edges with severe TIVs. Furthermore, by incorporating this mechanism into Vivaldi and Meridian, it is possible to reduce the impact of TIV and improve their neighbor selection performance. We believe these findings serve as a first step towards building robust TIV-aware distributed systems.

The rest of this paper is organized as follows. Section 2 evaluates the TIV characteristics of measured Internet delays. Section 3 explains how TIVs impact the performance of Vivaldi and Meridian. Section 4 explores several strawman strategies that deal with TIVs. Unfortunately, the benefits of these strategies are quite limited. Section 5 introduces a TIV alert mechanism, and evaluates the application of this mechanism in Vivaldi and Meridian. The related work is presented in Section 6 and Section 7 concludes this paper.

2. ANALYZING TIVS IN INTERNET DELAYS

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

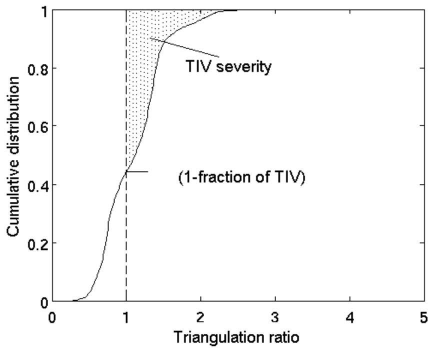

Figure 1: Illustration of the TIV severity metric

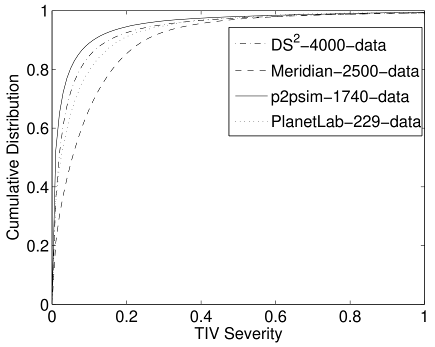

Figure 2: Cumulative distribution of TIV severity

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. [25, 4, 39, 11, 35]) 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 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 paper, 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.

2.2 Analysis of TIV Characteristics

Figure 2 shows the extent of TIVs found in 4 different measured Internet delay data sets: p2psim data (1740 nodes) [19], Meridian data (2500 nodes) [34], DS^2 data (4000 nodes) [35], 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 [35] 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 [35] 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. 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) [2]. 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 of edges within each bin. Figure 4 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

[2]Note that [11] uses a different definition for TIV and thus the results are different

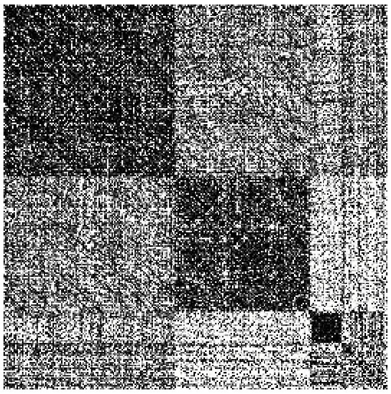

Figure 3: TIV severity by cluster (a white point represents the most severe TIV).

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 4 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 4 the median TIV severity has a peak for the edges around 500-600 ms. Similar irregular behavior can be observed in Figure 5, Figure 6 and Figure 7 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 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

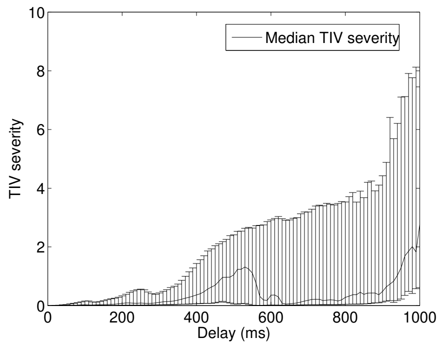

Figure 4: Relation between delay and TIV severity for DS^2 data. Error bar shows the 10%, median and 90%.

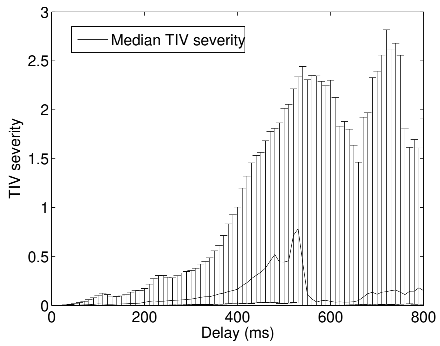

Figure 5: Relation between delay and TIV severity for p2psim data. Error bar shows the 10%, median and 90%.

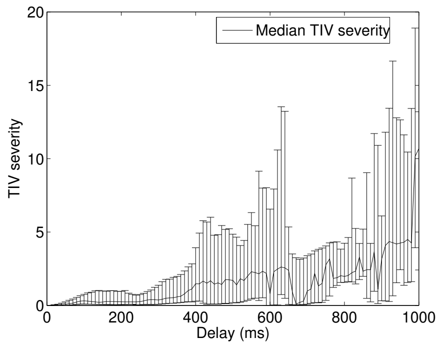

Figure 6: Relation between delay and TIV severity for Meridian data. Error bar shows the 10%, median and 90%.

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 9 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 addition, 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 paper, the experiments are performed on the DS^2 4000-node delay data set unless otherwise noted.

3. UNDERSTANDING THE PROBLEMS CAUSED BY TIVS

Many distributed systems rely on neighbor selection mechanisms to create overlay structures with good network performance. Vivaldi and Meridian are two representative solutions to the nearest neighbor selection problem. They represent two interesting but very different design points. Vivaldi is based on network embedding techniques. The benefit of Vivaldi is that it only requires very few offline measurement probes. However, Vivaldi is not very accurate at finding the nearest neighbor. On the other hand, although Meridian can find the nearest neighbor much more accurately, Meridian requires online measurement probes. Although the basic principles of Vivaldi and Meridian are quite different, they both make the assumption that, the underlying Internet delay space is a metric space in which triangle inequality holds among all the delays. A recent study [11] 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. Moreover, to our knowledge, the impact of TIV on Meridian has not been previously studied. In this section, we try to understand how TIVs impact the performance of Vivaldi and Meridian.

3.1 Principles Underlying Vivaldi and Meridian

Vivaldi is a distributed network coordinate system that aims to embed the network delays into a low dimension metric space. While any metric space can potentially be used, this paper uses a 5D Euclidean space for simplicity. Regardless of what metric space is used, it is important to note that all metric spaces obey the triangle inequality and are therefore incompatible with delay TIV. In Vivaldi, each node is assigned a virtual coordinate and the network delay between a pair of nodes is estimated by the Euclidean distance given by their coordinates. To compute the coordinates for each node, Vivaldi simulates a physical spring system. Each pair of nodes (i, j) corresponds to a spring with a rest length set to the measured delay between i and j. The current length of the spring is the estimated Euclidean distance between the nodes. The potential energy of the spring is proportional to the square of the displacement from its rest length, which is actually the square of the estimation error. Vivaldi uses an adaptive procedure to minimize the spring energy. Each node has several neighbors in the Vivaldi system. At each step, when a node measures the delay between itself

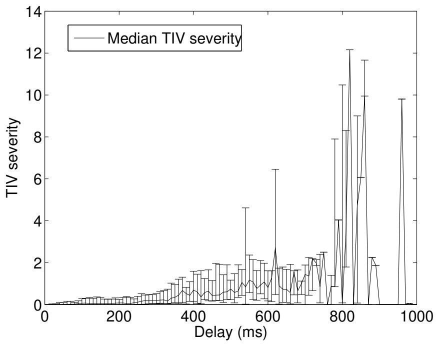

Figure 7: Relation between delay and TIV severity for PlanetLab data. Error bar shows the 10%, median and 90%.

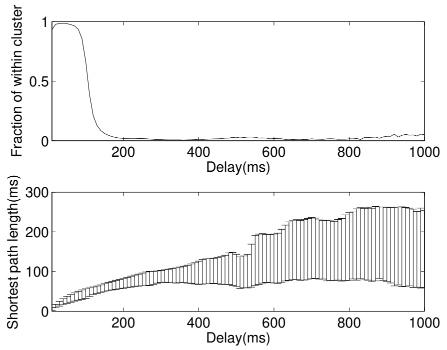

Figure 8: Shortest path length for edges of DS^2 data at different delays

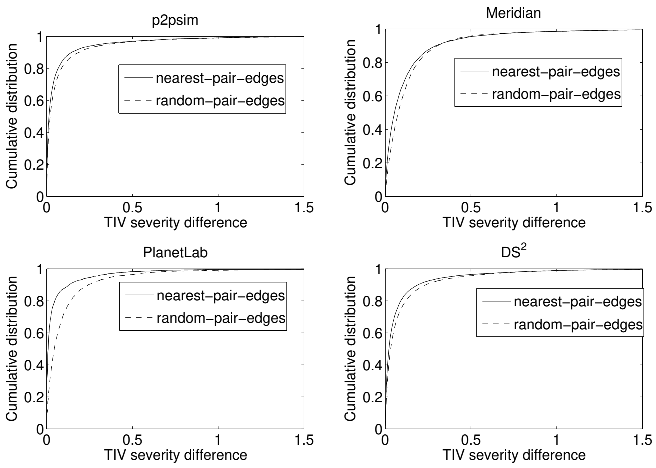

Figure 9: Proximity property of TIVs

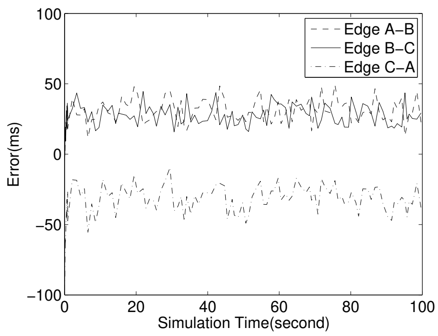

Figure 10: Vivaldi error trace for a simple 3-node network with TIV

and one of its neighbors, the node will be pulled toward or pushed away from that neighbor to a new position that decreases the energy of the corresponding spring. The system evolves so that the nodes move to the positions that minimize the squared error of the measured delays. Ideally, if network delays are perfectly embeddable in the Euclidean space, Vivaldi can generate a set of coordinates that predict all network delays very well. However, real network delays are not Euclidean and violate the triangle inequality. Thus, in general, it is not possible to predict all the network delays accurately. Many edges will still have high prediction error in Vivaldi.

Wong et al. proposed another system, Meridian, which is based on active online probing and recursive query. Meridian forms a loosely-structured overlay network and uses online measurements to solve neighbor selection problems. Each Meridian node keeps track of a fixed number of other nodes in the system as its members. Then a Meridian node organizes all its members into a finite number of concentric, non-overlapping rings based on the measured delays between itself and these members. The rings have exponentially increasing radii. The i-th ring has inner radius $r_i = \alpha s^{i-1}$ and the outer radius $R_i = \alpha s^i$, where α is a constant and s is the multiplicative factor. One Meridian node needs to keep up to k members for each of its rings. To find the closest neighbor to a target node, a random Meridian node N is chosen to start a recursive query. Node N first measures the delay d between itself and the target. Then N simultaneously queries all of its ring members whose delays are within $(1-\beta)\times d$ to $(1+\beta)\times d$ from N, where $\beta < 1$ is an acceptance threshold. The idea is that, if the triangle inequality holds, then any ring member that is within $\beta \times d$ from the target node should be among this set of ring members. These ring members measure their delays to the target online and then report back to N. The query is then forwarded to the ring member who is closest to the target. This process repeats so that the query is gradually forwarded to the node that is closest to the target, or until the termination condition is satisfied. The termination condition is if next Meridian node's delay to the target is longer than $\beta \times d$. TIV can reduce the effectiveness of Meridian. This is because TIV makes the ring membership information unreliable. Two nearby nodes should be placed in the same ring or very close rings of a Meridian node. However, TIVs can make the two nearby nodes have very different delays to the Meridian node and thus be misplaced. Such placement mistakes hurt performance because the true closest node to the target may be misplaced and never be considered during the query forwarding process.

3.2 Illustrating the TIV Problems

In the following, we use concrete scenarios to show how Vivaldi and Meridian may behave when there are triangle inequality violations.

3.2.1 Vivaldi

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.

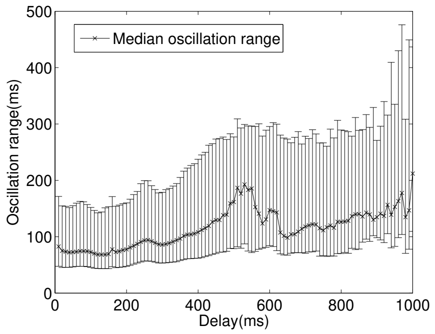

Figure 11: Distribution of the oscillation range of all the edges

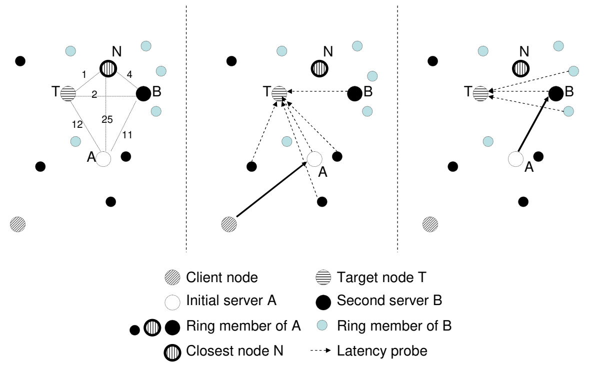

Figure 12: A client node sends a "closest neighbor to target T" request to a Meridian node A, which determines its delay d to T and asks its ring members within $(1-\beta)\times d$ and $(1+\beta)\times d$ to probe T (β is the constant acceptance threshold). The request then is forwarded to another Meridian node B, which is the closest to T among A's eligible ring members. Similarly B also asks its relevant ring members to probe T and finally finds out that B itself is closest to T. B is returned to the client as the found closest neighbor even though the real closest neighbor is N. Meridian fails to find N due to incorrect ring membership caused by triangle inequality violations.

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 10 shows the error trace of edge AB, BC, and CA over 100 second simulation time. Here the error is defined to be $(euclidean\_distance - measured\_delay)$.

As we can see from Figure 10, 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 $(max(prediction\_distance) - min(prediction\_distance))$ and collect the oscillation range for all the edges during a 500s simulation period. In Figure 11, 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.

3.2.2 Meridian

Meridian assumes that the triangle inequality property holds in the underlying delay space. So, intuitively, if two close-by nodes are both selected as ring members of another node, then they should be placed in the same or very close rings of that node. This assumption is no longer true with TIVs, and thus Meridian's performance is inevitably affected.

In Figure 12, a simple example is presented to show how TIV can affect Meridian's performance. A client node issues a request for the closest neighbor to target T. Meridian performs a recursive query and ends up with the chosen neighbor B while the real closest neighbor is N. The left picture in Figure 12 explains why Meridian fails to find the real closest neighbor. As can be seen from the picture, four nodes form four triangles, three of which violate the triangle inequality property. For example,AT is 12 and TN is only 1, but AN is 25. In this example, N and B will be placed in different rings of A although they are close to each other. So when A is chosen as the initial node to start the query, A will not ask N to probe T because N is too far. What is worse, after node B is determined as the second node to continue the query, B still cannot ask N to probe T since it is still relatively far from N due to a triangle inequality violation. So in this example, the Meridian system fails to find the real closest neighbor mainly because of the two triangle inequality violations. Note that the fact that N is involved in triangle inequality violations does not mean that N will never be found as the closest neighbor to T. If the client chooses another Meridian node as the initial node to start the recursive query, it is still possible for the Meridian system to find the real closest neighbor N, but the existence of TIV increases the difficulty for Meridian to find the closest neighbor.

Meridian uses active online probings during the recursive query, so Meridian is not as sensitive to TIVs as Vivaldi. In addition, Meridian already can tolerate some TIVs. For example, if β is set to a large value, then more ring members are allowed to probe the target. This can mask the TIV-induced placement errors in the ring membership. In the above example, a large β (though unrealistic) may allow N to be selected to probe the target and thus Meridian will find the correct closest neighbor to T. The downside of using a large β is increased probing overhead. This simple mechanism however is not sufficient to handle severe TIVs.

The following experiment is used to quantitatively show how of-

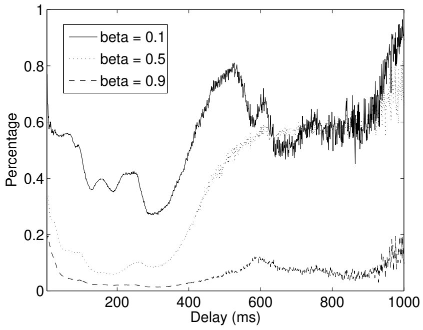

Figure 13: Percentage of Meridian ring members misplaced

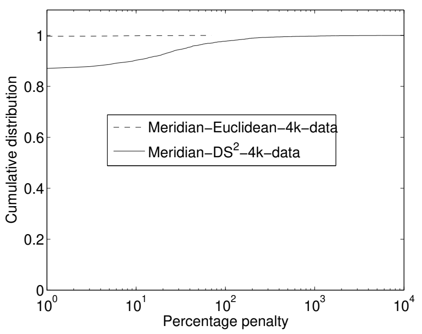

Figure 14: Neighbor selection performance of Meridian with ideal settings

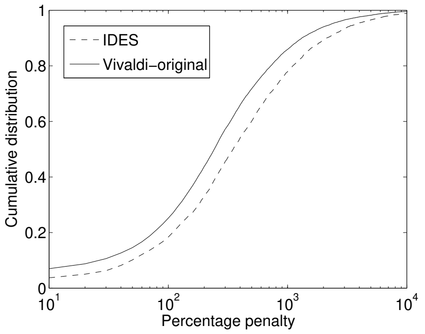

Figure 15: Neighbor selection performance for IDES

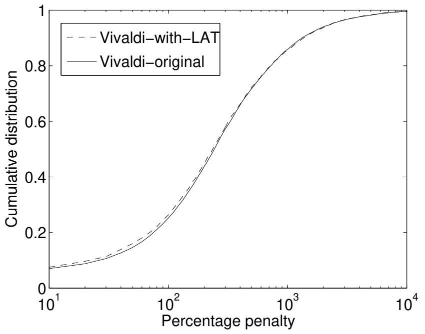

Figure 16: Neighbor selection performance for Vivaldi-LAT

ten Meridian will make a mistake due to TIVs. For the DS^2 data set, given any node N_i, for any other node N_j that has delay of d_{ij} to N_i, we find the set of nodes that are within the delay of $\beta \times d_{ij}$ to N_j and then we count the number of nodes in the set whose delays to N_i are not in the range from $(1-\beta) \times d_{ij}$ to $(1+\beta) \times d_{ij}$. This condition indicates that such nodes will cause placement errors in the ring membership. Figure 13 shows the result for different β values. The x-axis organizes the results for all node pairs N_i and N_j by the delay d_{ij}. The y-axis shows the percentage of nodes that will cause placement errors. As we can observe, larger β gives Meridian better tolerance to TIVs but larger β also increases the probing overhead significantly. If we set β to 0.5 just as recommended in [34], we can see that placement errors are frequent especially with respect to d_{ij} larger than 400 ms. Even for d_{ij} less than 400 ms, placement mistakes occur 10% to 30% of the times.

Next, we run Meridian simulations to study how TIVs will affect its performance in practice. Simulations are run with two different data sets, one is an artificial Euclidean matrix and the other is the DS^2 delay matrix. We use $\beta = 0.5$ for both simulations. For each data set, we randomly pick 200 nodes as Meridian nodes and use the other 3800 nodes as clients. In order to filter out all the factors that can potentially degrade Meridian's performance, we let each Meridian node to use all other 199 Meridian nodes as its ring members, then we turn off the termination condition, i.e., we allow Meridian to continue to search even when the new Meridian node's delay to the target is longer than $\beta \times d$. This provides an idealized setting for Meridian. Note that in reality, we usually do not use all other Meridian nodes as ring members and we have to use the termination condition to limit overhead. Thus, this experiment tries to show an upper bound of Meridian's performance. Figure 14 shows the results. Meridian nearly always finds the closest neighbor when run over the artificial Euclidean data set where the triangle inequality is satisfied. The reason that the performance of the Meridian on Euclidean data is not perfect is, given the target T, occasionally the distance between the closest server R and current chosen Meridian node M is not within the range between $(1-\beta) \times d_{MT}$ and $(1+\beta) \times d_{MT}$ and M has no other ring members within that range at all, so M has to stop. For the measured delay data, as can be seen, severe TIVs can clearly affect Meridian's performance even it is run under idealized settings.

4. STRAWMAN SOLUTIONS TO DEAL WITH TIVS IN NEIGHBOR SELECTION

Recent studies have reported the inaccuracy of network embedding caused by TIVs and several techniques have been proposed to accommodate TIVs [16, 11]. Thus, we would like to determine how much these techniques can help improve the performance of neighbor selection. Furthermore, we would also like to determine whether removing edges that have large TIV severity can help reduce the impact of TIVs on the neighbor selection mechanisms.

4.1 Neighbor Selection Experiment Methodology

For the rest of this paper, 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 [35].

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

$$\frac{(delay\_to\_selected - delay\_to\_optimal) \times 100}{delay\_to\_optimal}$$

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

For Meridian, we set the parameters as follows: $k = 16$ nodes per ring at most, 11 rings per node, multiplicative increase factor $s = 2$, $\beta = 0.5$, $\alpha = 1$. We select a random subset of 2000 nodes as Meridian nodes and build Meridian rings among them. This is done so that there are enough Meridian nodes to construct reasonable Meridian rings. The remaining 2000 nodes act as clients. One closest neighbor selection test is performed per client. A client sends its closest neighbor request to a random Meridian node. Again, we record the percentage penalty for each test. The experiment is run 5 times using 5 different random subsets of 2000 nodes as Meridian nodes. Results reported are cumulative over the 5 runs.

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 [16] 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. [11] 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 $e_x = \frac{\sum_{y \in S} (d_{xy} - \hat{d}_{xy})}{2|S|}$.

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 15 and Figure 16 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

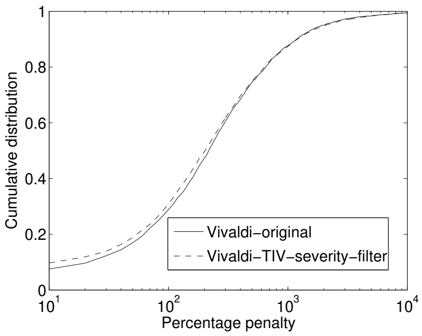

Figure 17: Neighbor selection performance for Vivaldi with TIV severity filter

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.

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 or by Meridian ring construction. Neighbor selection performance of this approach is shown in Figure 17 and Figure 18.

From Figure 17, 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.

For Meridian, as can be seen from Figure 18, the TIV outlier removal approach actually degrades Meridian's performance. Meridian relies on recursive routing to find the closest servers, but the TIV filter approach may inevitably remove some edges that are needed for Meridian to route queries to the closest nodes. We observe that, certain rings of a Meridian node may become under-populated by upto 50%. When a query needs to be routed through a member of that ring, the under-population of that ring may cause Meridian to fail to route the query further, resulting in sub-par performance.

In summary, given the global information, simply excluding some TIV outliers to clean the delay matrix does not help to improve any of the two neighbor selection mechanisms mentioned above. Hence even with full knowledge of TIVs, we still need specifically refined strategies for different applications to avoid the impact of TIVs.

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 systems like Vivaldi and Meridian and improve their performance.

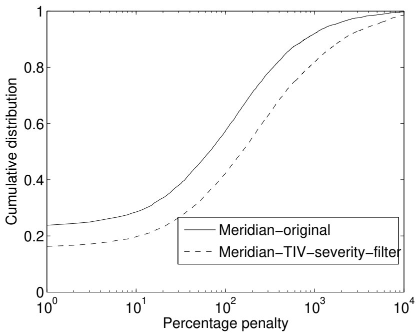

Figure 18: Neighbor selection performance for Meridian with TIV severity filter

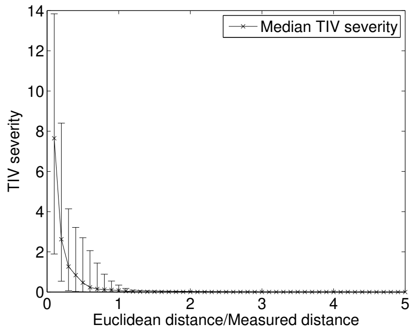

Figure 19: TIV severity for edges with different prediction ratios

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 number 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 network 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 $\frac{euclidean\_distance}{measured\_distance}$

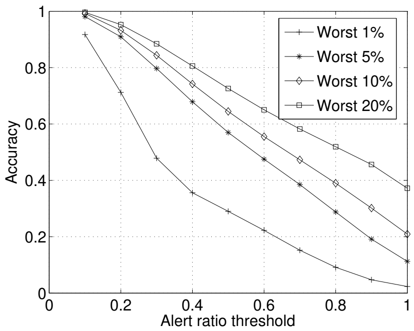

Figure 20: Accuracy of TIV alert mechanism

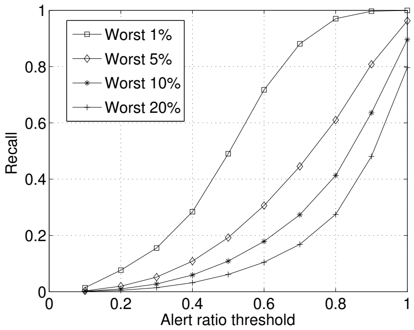

Figure 21: Recall rate of TIV alert mechanism

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 19 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 19, 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 20 and Figure 21. 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 21, 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 Vivaldi and Meridian.

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 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 $\frac{euclidean\_distance}{measured\_delay}$. 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 22 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 23 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.

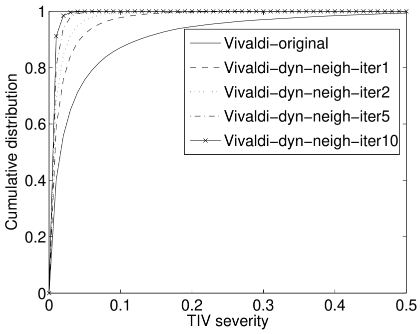

Figure 22: TIV severity of Vivaldi neighbor edges

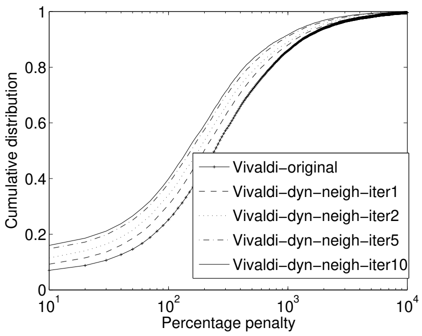

Figure 23: Neighbor selection performance of dynamic neighbor Vivaldi

5.3 Using TIV Alert Mechanism in Meridian

The operations of Meridian can be separated into two stages: ring construction and online recursive query. In the ring construction stage, each Meridian node needs to select $O(logN)$ other Meridian nodes as its ring members and then organizes them into concentric rings of exponentially increasing radii. After the ring construction stage, participating Meridian nodes form a loosely-structured overlay, on which we can perform the second stage operations, online recursive query. The second stage performs recursive query routing and active probing to search for the nearest neighbor.

The goal is to show that even straight-forward application of the TIV alert mechanism to ring construction and recursive query stages can reduce the impact of TIV on Meridian's performance. More sophisticated ways to use the TIV alert mechanism than those presented here may exist.

Ring Construction - In the ring construction stage, each Meridian node needs to sample a set of other Meridian nodes as its ring members, measures the delays between itself to all its ring members and then puts all ring members into appropriate rings based on the measured delays. As we have shown in Figure 13, this simple ring construction procedure cannot handle all TIVs. Severe TIVs can mislead a Meridian node to put two close-by nodes that ought to be placed in the same or very close rings in two very different rings. So the idea here is to use the TIV alert mechanism to help adjust the ring memberships in order to accommodate those severe TIVs. We can either bring the node placed in the far ring back to the close ring or push the node placed in the close ring to the far ring. We assume an independent network embedding mechanism, say, Vivaldi, provides the prediction ratios for the TIV alerts.

The membership adjustment algorithm works as follows: for any Meridian node M, we need to check each of its ring members M_i to see whether the prediction ratio of the edge between M and M_i is within a safe range. That is, if the prediction ratio of the edge is smaller than certain threshold t_s or larger than another threshold t_l, we need to place that corresponding node into rings not only based on the real measured delay but also based on the predicted delay. So, in the worst case, a ring member will be placed into two rings. In this section, we always use $t_s = 0.6$ and $t_l = 2$. These threshold values by no means represent the optimal setting and we have not yet determined the optimal setting. The goal is simply to show that using reasonable thresholds can already provide benefits.

Online Recursive Query - During the recursive query stage, the Meridian system recursively hands off the query to a node that is closer to the target. In order to limit the number of iterations and the corresponding probing overheads, [34] suggests to use an acceptance threshold β to determine when to stop the recursive query. If, in one iteration, the chosen Meridian node cannot find at least one node that is closer than β times the distance to the target, it will stop searching. We believe that TIVs in the underlying delay space often misguide Meridian to prematurely stop at a suboptimal node. The idea here is: when the chosen Meridian node cannot find one node that is closer than β times its distance to the target, it will check the prediction ratio of the edge between itself and the target, if the prediction ratio is smaller than certain threshold t_s, then the chosen Meridian node will select another set of its ring members to restart the query based on the predicted delay to the target. The goal is that if the edge between the current Meridian node and the target causes severe TIVs, then we may have a better chance to find a node that is closer to the target by using the predicted delay to choose a subset of ring members to query. Again, we set the threshold t_s to 0.6.

We evaluate the effectiveness of applying the TIV alert mechanism to Meridian using the normal setting, where there are 2000 Meridian nodes out of 4000 nodes and $\beta = 0.5$, $\alpha = 1$, $s = 2$, $k = 16$ and $l = 4$. Figure 24 shows the neighbor selection results of Meridian after using the TIV alert mechanism. As we can see, the TIV alert mechanism does improve Meridian's performance. Because some ring members are placed into 2 rings and because we need to restart query based on the predicted delay occasionally, the online probing overhead increases by about 6%. For comparison, allowing the same additional probing overhead by increasing the value of β in regular Meridian provides less performance improvement.

We also evaluate our techniques under another setting where there are only 200 Meridian nodes out of 4000 nodes and each Meridian node uses all other 199 Meridian nodes as its ring members. Figure 25 shows the result. There are three sets of results. "Meridian original" uses the acceptance threshold $\beta = 0.5$. "Meridian TIV alert" stands for Meridian with the TIV alert mechanism used. "Meridian no termination" uses the idealized settings that is the same as what we used in Section 3.2.2. As can be seen, if all Meridian nodes are used as ring members, Meridian's performance is already very good. But after applying the TIV alert mechanism, we can still improve its performance with only about 5% more online probing overhead. Note that the performance of Meridian after applying the TIV alert mechanism is even better than the idealized Meridian which disables the termination condition and also causes about 5% more online probing overhead. This improvement is because Meridian is made aware of TIVs and cope with TIVs directly.

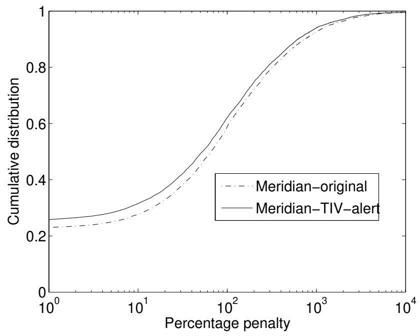

Figure 24: Neighbor selection result of Meridian using TIV alert mechanism. This technique causes 6% more on-demand probes.

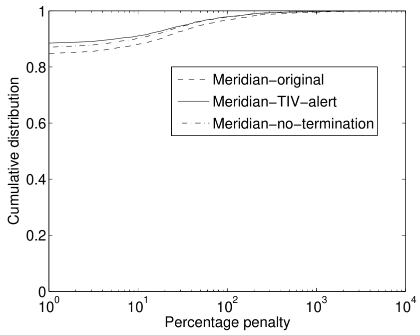

Figure 25: Neighbor selection result of Meridian using TIV alert mechanism. This technique causes 5% more on-demand probes.

In summary, a simple application of the TIV alert mechanism can improve Meridian. Although the magnitude of improvement is modest, more effective ways to apply the TIV alert mechanism may exist.

6. RELATED WORK

Our work on the impact of triangle inequality violations on neighbor selection mechanisms is closely related to many previous studies on performance analysis of network coordinate systems.

Network coordinate systems are regarded as a promising approach for neighbor selection because of its simplicity and scalability to predict $O(N^2)$ pair-wise delays by a constant number of measurement on each node. The basic idea of this method is to embed the Internet delay space into a geometric space. Following the basic idea of network embedding, several schemes have been proposed to set up an efficient and scalable network coordinate systems. Centralized methods such as GNP [17], Big-Bang [27], PCA [32, 12] and Hyperbolic [28] require a fixed set of landmarks which could be a bottleneck of the system. Therefore, some decentralized methods such as NPS [18], Vivaldi [3], PIC [2] and Lighthouses [20] were proposed to improve the scalability of network coordinate systems. We choose Vivaldi as a typical network coordinate system in our study. The findings and techniques we studied in this paper can potentially be applied to other network coordinate systems.

While relative error evaluations and theoretical analysis [30, 29] demonstrate the remarkable accuracy of network coordinate systems, some recent studies explore the limitations of these schemes in applications. Lua et al. [13] evaluate the network coordinate schemes by application oriented metrics such as relative rank loss and closest neighbors loss, and show that, these schemes perform poorly under the new metrics. Zhang et al. [37] study the performance of these network coordinate algorithms under overlay multicast, server selection and overlay construction. The major findings are that network coordinate schemes are inadequate for the applications in terms of the prediction accuracy, and the high error on short links has a big impact on application performance. In parallel, many Internet measurement studies repeatedly confirm the wide-spread TIVs found in the Internet delays [39, 11, 35]. In [39], Zheng et al. argue that, TIV is a natural feature of Internet structure and routing policies and it poses a big problem on network coordinate systems. Lee et al. [11] analyze the error of network coordinate schemes and claim that the inaccuracy of Euclidean embedding is caused by a large degree of TIVs in the Internet delays. Furthermore, some recent studies [21, 10, 9] report the stability problems of Vivaldi in practical deployment, but this problem is out of the scope of this paper.

Several techniques have been tried to deal with TIVs in network coordinate systems by introducing TIVs to the delay prediction. IDES [16] assigns an incoming and an outgoing vector to each node 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. This method removes the constraint of triangle inequality. Lee et al. [11] proposes to add a localized adjustment term (LAT) to Euclidean coordinates to account for the non-Euclidean effect. However the evaluation results in this paper have shown that, these techniques do not improve the neighbor selection performance of network coordinate system. In [36], a hierarchical approach is proposed to improve the performance of network coordinate system. The idea of this approach is, each node is assigned with multiple coordinates and the network delays in different scales are predicted by different set of coordinates. The difficulty of this approach is, it is very hard to predict which scale an edge belongs to without measuring it. All the above results inspire our work to understand the impact of TIVs on network coordinate systems.

Faced with the inaccuracy problem of network coordinate systems, Wong et al. [34] proposes a totally different neighbor selection mechanism, Meridian, which is based on recursive searching. The Meridian system still assumes triangle inequality in the Internet delays. To the best of our knowledges, this paper is the first work to study the impact of TIVs on the Meridian system and to propose techniques to deal with TIVs in the Meridian system.

In addition to network coordinates and the Meridian system, some other location techniques requiring information beyond end-to-end delay have also been proposed. The OASIS system [5] selects closest servers using geographical locations. iPlane [14, 15] and S^3 [26] propose to predict network delays with network topology map information gathered by $traceroute$ measurements. Although these techniques are quite different from Vivaldi and Meridian, iPlane and S^3 are still based on the triangle inequality assumption for network delays. Thus, the findings of this paper can potentially be helpful for improving the performance of iPlane and S^3 in face of TIVs in the future.

7. CONCLUSIONS

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 in two representative neighbor selection mechanisms (Vivaldi and Meridian) 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 and Meridian to become TIV-aware. We believe these findings serve as a first step towards building robust TIV-aware distributed systems.

Acknowledgment

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

8. REFERENCES

[1] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.
[2] 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.
[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceeding of ACM SIGCOMM*, August 2004.
[4] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniewicz, and Y. Jin. An architecture for a global Internet host distance estimation service. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.
[5] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazieres. Oasis: Anycast for any service. In *Proceedings of ACM NSDI*, May 2006.
[6] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.
[7] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth restricted metrics. In *Proccedings of ACM Symposium on Theory of Computing*, 2002.
[8] C. Kommareddy and B.Bhattacharjee. Finding close friends on the internet. In *Proceedings of the Ninth International Conference on Network Protocols(ICNP)*, November 2001.
[9] J. Ledlie, P. Gardner, , and M. Seltzer. Network coordinates in the wild. In *Proceeding of USENIX NSDI'07*, April 2007.
[10] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *Proceeding of International Conference on Distributed Computing Systems*, 2006.
[11] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of Euclidean embedding of Internet hosts. In *Proc. SIGMETRICS 2006*, June 2006.
[12] H. Lim, J. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proceedings of IMC*, Miami, FL, October 2003.
[13] 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.
[14] H. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani. A structural approach to latency prediction. In *Proceedings of Internet Measurement Conference*, Rio de Janeiro, Brazil, 2006.
[15] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun

Venkataramani. iPlane: an information plane for distributed services. In *Proc. OSDI 2006*, November 2006.

[16] 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.

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

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

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

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

[21] 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)*, 2005.

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

[23] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[24] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[25] 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.

[26] P. Sharma, Z. Xu, S. Banerjee, and S. Lee. Estimating network proximity and latency. *ACM Computer Communication Review*, pages 39–50, 2006.

[27] 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.

[28] 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.

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

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

[31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

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

[33] Marcel Waldvogel and Roberto Rinaldi. Efficient Topology-Aware Overlay Network. In *First Workshop on Hot Topics in networks (HotNets-I)*, October 2002.

[34] Bernard Wong, Aleksandrs Slivkins, and Emin Gun Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proceedings of ACM SIGCOMM*, August 2005.

[35] 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.

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

[37] 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.

[38] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for wide-area fault-tolerant location and routing. *U.C. Berkeley Technical Report UCB//CSD-01-1141*, 2001.

[39] 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, 2005.