Tarantula: Towards an Accurate Network Coordinate System by Handling Major Portion of TIVs

Zhuo Chen[1], Yang Chen[2], Yibo Zhu[1], Cong Ding[2], Beixing Deng[1] and Xing Li[1]
[1] Department of Electronic Engineering, Tsinghua University, Beijing, China
[2] Institute of Computer Science, University of Goettingen, Goettingen, Germany
E-mail: chenzhuo08@mails.tsinghua.edu.cn, yang.chen@cs.uni-goettingen.de

***Abstract*—Network Coordinate (NC) systems provide an efficient and scalable mechanism to estimate latencies among hosts. However, many popular algorithms like *Vivaldi* suffer greatly from the existence of Triangle Inequality Violations (TIVs). Two-layer systems like Pharos and hierarchical Vivaldi have been proposed to remedy the impact of TIVs. They divide the whole space into several location-based clusters and run NC systems on both global layer and local layer. However, the two-layer model is only able to optimize the intra-cluster links relating to a limited portion of TIV triangles. In this paper, we propose a new NC system, Tarantula, which divides the space in a novel way. By categorizing the TIVs into three classes, we show that Tarantula handles a much larger portion of existing TIVs than two-layer systems. Moreover, we present two techniques to further strengthen the Tarantula system: 1) relate the updating step size in the Vivaldi algorithm used in Tarantula to ground-truth latency so as to improve the prediction for short links; 2) propose Dynamic Cluster Optimization to dynamically adjust clustering of hosts. Our experimental results show that Tarantula outperforms Pharos and Vivaldi significantly in terms of estimation accuracy. When implementing different NC systems in the application of server selection and detour finding, Tarantula again performs the best.**

I. INTRODUCTION

The Network Coordinate systems [6] map Internet hosts to a geometric space. By assigning each host a d-dimension vector, which is called coordinates, the system uses the geometric distance between two arbitrary hosts to predict the ground-truth latency. NC systems offer acceptable error for latency prediction and exert low overheads. For an amount of N nodes in real network, only $O(N)$ measurements are needed to predict the $O(N^2)$ latencies. So far, NC systems are widely used in many services, including locality-aware server selection [16], which is beneficial to BitTorrent-based file sharing [18] and cloud services [15]. Detour finding [17] is another promising application based on NC systems, which is useful in overlay routing [12] and multi-player online gaming [1]. In the next generation network, NC systems is again useful in areas such as route selection for IPv6 multihomed sites [8].

However, with the existence of Triangle Inequality Violations (TIVs), the coordinates mapping in popular Euclidean space systems such as Vivaldi [5], GNP [13] and PIC [4] can never be satisfactory [9], [11]. Several studies, therefore, have been made to remedy the negative effect of TIVs. Two-layer systems like Pharos [3] and hierarchical Vivaldi [7] divide the whole space into several subsets and run an NC system on each subset. The main idea is that if not all the three links in a triangle with TIV property (referred as TIV triangle) are in the same NC system, the impact of this TIV will be reduced. However, our analysis shows that only a limited portion of TIVs can be handled in this way. Therefore, we propose a new NC system, Tarantula, which divides the space with a novel approach, forming several cluster-based systems and subspace-based systems. With this space dividing method, the impact of a much larger portion of TIVs has been attenuated.

Besides the basic idea, we also propose different detailed design considerations. First, we observe that while short links (less than 50ms) account for a significant portion of all links, the estimation for them remains the worst with the Vivaldi algorithm. Therefore, we modify the Vivaldi algorithm used in Tarantula by relating the updating step size to the ground-truth latency, which greatly improves the estimation for short links. Second, we are considering different cluster dividing strategies. We find that the best performance appears when all the nodes in a cluster are geographically close to each other. This strategy, however, is hard to implement, for the geographical information of all hosts are sometimes not provided. Therefore, we propose Dynamic Cluster Optimization, which optimizes the clustering of space periodically.

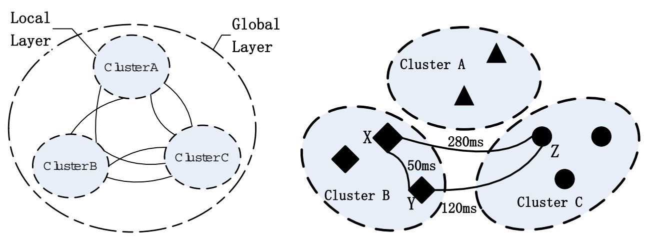
The rest of the paper is organized as follows. The Vivaldi and two-layer systems are reviewed in section II. In section III, we explain in detail why Tarantula is able to address more TIVs than two-layer systems. Some more technics to improve Tarantula are also discussed. In section IV, we evaluate Tarantula in terms of estimation accuracy. We also test Tarantula's performance in the application of server selection and detour finding. We conclude our paper in section V.

II. BACKGROUND

In an NC system, each host conducts scalable measurements to keep a set of numbers representing its own coordinates. Most NC systems apply the Euclidean space model, so the predicted distance between hosts i and j is calculated as

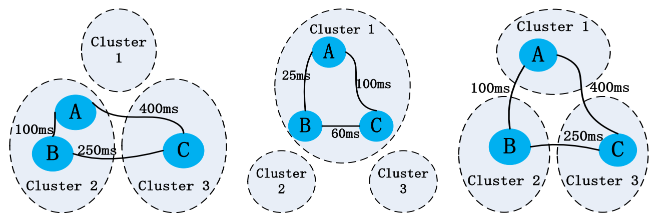
$$D_{i,j} = \|\vec{x}_i - \vec{x}_j\| = \sqrt{\sum_{k=1}^{d} (a_{ik} - a_{jk})^2} \qquad (1)$$

where $D_{i,j}$ represents the predicted distance and $\vec{x}_i = (a_{i1}, a_{i2}, ..., a_{id})$ is the d-dimension coordinates of node i. Specifically, we use $D_{i,j}$ as an approximation of the ground-truth latency between i and j, which is denoted as $L_{i,j}$. People



(a) System Structure (b) Reduction of TIV Interference

Fig. 1. Two-layer Model



(a) 1^{st} Class TIV (PL: 70% King: 67%) (b) 2^{nd} Class TIV (PL: 12% King: 16%) (c) 3^{rd} Class TIV (PL: 18% King: 17%)

Fig. 2. Different Classes of TIVs

usually use relative error (RE) as a metric to evaluate the estimation performance [5], [10].

$$RE = \frac{|D_{i,j} - L_{i,j}|}{\min(D_{i,j}, L_{i,j})} \tag{2}$$

A. Vivaldi

As the representative decentralized NC algorithm, Vivaldi models the network as a spring system. Suppose we connect every pair of hosts with a spring, the initial length of which is the measured latency between the two end hosts. Just as the spring system converges to minimize the potential energy of the system, the Vivaldi algorithm minimizes the sum of squares of the absolute errors (AEs) in a decentralized way:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} | \, \|\vec{x}_i - \vec{x}_j\| - L_{i,j}|^2 \tag{3}$$

In each round, one node i measures the latency to one of its neighbors, j, and calculate w, the weight meaning how confident we are about the current measurement. Then node i adapts its coordinates with a step size proportional to both w and the "force of the spring" - the difference between the estimated latency and the measured latency. Host i also modifies its local error ε_i based on the current measurement, which represents the average estimation error of the links connected to it. In bootstrapping period, all the coordinates are set to be zeros and local errors are set to be ones. We will further discuss the Vivaldi algorithm in section III-C.

B. Two-layer NC systems

[3] and [7] proposed two-layer models to improve the estimation accuracy. We use Pharos [3] instead of [7] as a representative in this paper, because [7] requires an oracle for clustering, which is not feasible in real Internet. Pharos divides the space into several clusters in a distributed way, and runs a Vivaldi system on every cluster, as well as the whole space (Figure 1(a)). We have implemented Pharos and deployed it in the PlanetLab test bed [17]. In Pharos, two layers are formed - a global layer and a local layer. Every host keeps different sets of neighbors and coordinates in different layers. The system then estimates intra-cluster latencies based on local coordinates and inter-cluster latencies based on global coordinates. In each round, one host updates its coordinates in either global-layer or local-layer based on the measured latency in the layer, respectively.

By dividing the space into several clusters, two-layer systems also separate the links in some TIV triangles and place them in different NC systems, thus reducing the negative effect of these TIVs [17]. For example, in Figure 1(b), XYZ are the nodes of a TIV triangle. X and Y lie in the same cluster, while Z is located in the different cluster. Although link XY belongs to both global layer and local layer system, it is estimated in the local layer system, where links XZ and YZ are excluded. In this way, the estimation of XY can be optimized, for it has got rid of the impact from the TIV of XYZ. We therefore call link XY the optimizable link [17].

III. System Design

In this section, we describe the structure of our new system, Tarantula, which produces more optimizable links and reduces the impact of more TIVs than two-layer systems. We also present an approach to improve the prediction for short links. The clustering strategies are discussed in the last sub-section.

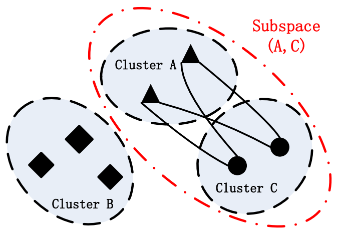
A. TIV characterization in two-layer NC systems

When the whole network is divided into several clusters, we can categorize the TIVs into three classes [17]: 1) two of the links in the TIV triangle are inter-cluster links while the other one is an intra-cluster link; 2) all three links are intra-cluster links; 3) all three links are inter-cluster links. This classification is illustrated in Figure 2, which also shows the proportions of different classes of TIVs existing in PlanetLab (PL) and King [5] data sets when dividing the space into three clusters. The cluster dividing strategy used here is to let every host join the cluster with the closest anchor [17], which will be further discussed in section III-D. As we can see, the first class TIVs account for more TIVs while the second and third classes of TIVs are not negligible.

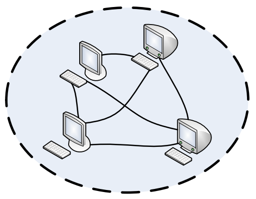
Based on this classification, we see that in a 1st class TIV triangle, the intra-cluster link is optimizable with the two-layer model as discussed in section II-B. However, inter-cluster links are not, since they are estimated in the global layer system, where the intra-cluster link in this TIV also exists. Similarly, all the links in the 2nd or 3rd class TIV triangles are not optimizable in two-layer systems. That said, the two-layer model is only able to partially handle the 1st class TIVs.

B. Design of Tarantula

Tarantula divides the whole space into three clusters (Cluster A, B and C). We choose three clusters because there are

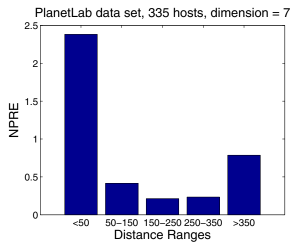


(a) Subspace (A,C)-based System

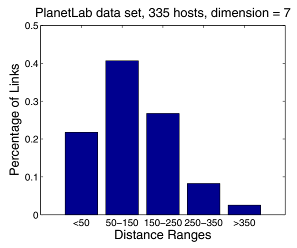


(b) Cluster-based System

Fig. 3. Structure of Tarantula



(a) Estimation Error



(b) Distribution of Links

Fig. 4. Vivaldi Performance for Different Distance Ranges

actually three major clusters in real network - America, Asia-Pacific, and Europe [17]. Every two clusters then constitute a subspace, which are denoted as Subspace (A,B), (A,C) and (B,C), so that each node belongs to one cluster and two subspaces. We run a Vivaldi system on each cluster or subspace (Figure 3). Figure 3(a) is an example illustrating the Subspace (A,C)-based system. In Tarantula, the intra-cluster latencies are estimated using the cluster-based system, and the inter-cluster latencies are predicted using the subspace-based system. That said,

$$D_{i,j} = \begin{cases} \|\vec{x}_{i,cluster} - \vec{x}_{j,cluster}\|, & Cls(i) = Cls(j), \\ \|\vec{x}_{i,sub(i,j)} - \vec{x}_{j,sub(i,j)}\|, & Cls(i) \neq Cls(j). \end{cases} \quad (4)$$

where $Cls(i)$ is the cluster to which i belongs and $sub(i,j)$ represents the subspace consisting both hosts i and j. The updating step of Tarantula structure is shown in Algorithm 1. In each round, one host updates its coordinates in either a cluster-based or subspace-based system based on the latency observation in the system, respectively.

Algorithm 1 Tarantula

```
for $i$ in $Hosts$ do
    $j = Get\_Cluster\_Neighbors(i)$
    $Update\_Cluster\_Coordinates(i, j)$
    for $subspace$ in $\{(A,B),(B,C),(A,C)\}$ do
        if $i$ in $subspace$ then
            $j = Get\_Subspace\_Neighbors(i, subspace)$
            $Update\_Subspace\_Coordinates(i, j, subspace)$
        end if
    end for
end for
```

Moreover, in a spring system, if we remove some springs, the actual lengths of other springs would be closer to initial lengths. Considering the analogy between Vivaldi and the spring system, we remove the intra-cluster links in any subspaces in Tarantula in order to improve the estimation for inter-cluster links (Figure 3(a)). Practically, in a subspace-based system, one host does not select neighbors in the same cluster. This modification will not impair the prediction accuracy of intra-cluster links, for these links are estimated in a cluster-based system.

Thanks to the new space dividing approach, we have made a much larger proportion of links to be optimizable in Tarantula compared with two-layer systems. First, in the 1st class TIV triangles, which account for about 70% of total TIVs (Figure 2(a)), while two-layer systems can only make the intra-cluster link optimizable, Tarantula ensures all the links to be optimizable. For example, in Figure 1(b), X, Y and Z form a 1st class TIV. The intra-cluster link XY is optimizable in the same way as in the two-layer structure. In addition, we use the NC system built on Subspace (B,C) to predict inter-cluster links XZ and YZ in Tarantula. These predictions will not be affected by the TIV of XYZ, since the link XY only belongs to the system running on Cluster B. Therefore, all the inter-cluster links in the 1st class TIV triangles have become optimizable in Tarantula. Second, we can similarly prove that all links in the 3rd class TIV triangles are optimizable in Tarantula. In contrast, two-layer structures cannot handle this class of TIVs at all, which, according to Figure 2(c), accounts for as much as 15% of total TIVs.

C. Improvement for short links

The original Vivaldi algorithm minimizes the sum of squares of AEs. However, people usually cares more about minimizing RE rather than AE, since for links of different lengths, our tolerance to AE is totally different. For instance, there are two links with the actual lengths being 1 and 100. We also have two groups of estimation values: (a) 5 and 96, (b) 0.9 and 90. Using the AE minimum strategy, we would choose (a), where the prediction for shorter link would probably mislead us. On the contrary, the RE minimum strategy would choose (b), which seems more reasonable. Based on this, we also believe that the AE minimum strategy in original Vivaldi would lead to relatively bad prediction for short links.

To test our thoughts, we simulate Vivaldi using the PlanetLab data set which consists of 335 hosts. The dimension is set to be 7 throughout this paper. All the other parameters are the same as [5]. We divide the links into five groups based on link lengths and calculate the ninetieth percentile RE (NPRE [5], [3]) of latency estimation within each group (Figure 4(a)). We also record the percentage of links in each group (Figure 4(b)). As we can see, links less than 50ms account for more than 20% of all links, but the NPRE of this range of links is much higher than that of longer links.

Therefore, in the Vivaldi algorithm used in Tarantula, we apply a modification to the spring model in order to reduce the overall REs, especially that of short links. This time, the force

Algorithm 2 $Vivaldi\_In\_Tarantula(L_{i,j}, x_j, \varepsilon_j)$

```
$w = \frac{\varepsilon_i}{\varepsilon_i + \varepsilon_j}$
$\varepsilon_s = |\,\|\vec{x}_i - \vec{x}_j\| - L_{i,j}| / L_{i,j}$
$\varepsilon_i = \varepsilon_s \times c_e \times w + \varepsilon_i \times (1 - c_e \times w)$
$\delta = \min(c_c \times w \times \frac{\mu}{L_{i,j}}, 1)$
$\vec{x}_i = \vec{x}_i + \delta \times (L_{i,j} - \|\vec{x}_i - \vec{x}_j\|) \times u(\vec{x}_i - \vec{x}_j)$
```

of the spring, or the updating step size, is also proportional to the inverse of the measured latency:

$$\vec{F}_{i,j} = \frac{|\,\|\vec{x}_i - \vec{x}_j\| - L_{i,j}|}{L_{i,j}} \tag{5}$$

With this modification, the shorter links will move a larger step in each round compared with the original algorithm while longer links move smaller steps, which results in a better estimation for short links. In fact, we are no longer minimizing the value shown in Equation 3. Instead, we solve

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{|\,\|\vec{x}_i - \vec{x}_j\| - L_{i,j}|^2}{L_{i,j}} \tag{6}$$

so that we have attached more importance on minimizing REs than original Vivaldi algorithm. Practically, we also bound the step size to be no more than the difference between estimated latency and measured latency. So, in the updating step, we set

$$step\_size = \min(c_c \times w \times \frac{\mu}{L_{i,j}}, 1) \times (|\,\|\vec{x}_i - \vec{x}_j\| - L_{i,j}|) \tag{7}$$

where c_c is a tunable parameter also shown in original Vivaldi, and μ is set to compensate for the factor of $\frac{1}{L_{i,j}}$. The modified Vivaldi algorithm in Tarantula is shown in Algorithm 2.

D. Dynamic Cluster Optimization

We compare the performance of three cluster dividing strategies using Tarantula structure: 1) Random cluster strategy: place a host in a random cluster. 2) Random anchor strategy: randomly choose three anchors; each host joins the cluster with the closest anchor. 3) Close-together strategy: select an anchor in each big continent, and hosts join clusters in the same way as 2). We again do our experiments on PlanetLab data set, since we have gained the geographical information of every host in it. We implement five experiments for both first and second strategies and calculate the average so as to reduce random deviation. As a result, we find that the close-together strategy (Average RE = 0.27), in this case, performs largely better than Random cluster strategy (Average RE = 0.48) and Random anchor strategy (Average RE = 0.39). Experiments on other data sets have shown similar results, so we believe the best clustering strategy is to place geographically close hosts in the same cluster.

However, the close-together strategy is hard to implement, for we sometimes cannot know the geographical information of all hosts in prior to choosing anchors. Also, one host may wrongly choose a cluster because of the unstable network when it pings the anchors. Therefore, we propose Dynamic

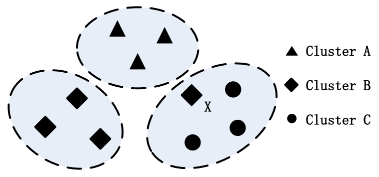


Fig. 5. Bad Clustering

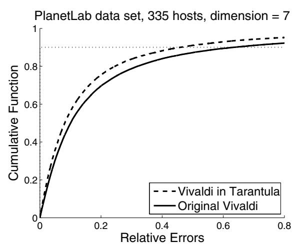
Cluster Optimization (DCO), an approach to dynamically optimize clustering in Tarantula. DCO periodically identifies the unfitted hosts, those who are not suitable for the current cluster, and moves them to a new cluster. We start describing DCO from the method used to identify unfitted hosts.

1) Finding unfitted hosts: Actually, the hosts which are not suitable for the current cluster tend to have large local errors, because although we have improved the estimation accuracy of short links, it still remains higher than that of longer links (Figure 6(b)). Suppose now host X is wrongly assigned to cluster B but it should be placed in cluster C (Figure 5). In the system running on Subspace (B,C), the links connected to X tend to be shorter than other links in the system, since all intra-cluster links are removed. That said, the local error of X is probably larger than the local errors of other hosts in Subspace (B,C). Based on this, we view the hosts with extremely large local errors in a subspace-based system as unfitted hosts. Practically, we set the bound to be 0.15, so that hosts with local error larger than 0.15 will be moved to the other cluster within the subspace. The algorithm of DCO is displayed in Algorithm 3, where $subs$ is short for subspace.

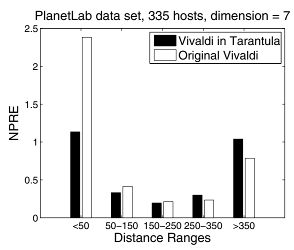
Algorithm 3 Dynamic Cluster Optimization

```
**while** forever **do**
  **for** $i$ in $hosts$ and $subs$ in $\{(A,B),(B,C),(A,C)\}$ **do**
    **if** $i$ in $subs$ and $\varepsilon_{i,subs} > bound$ **then**
      $Join\_New\_Cluster(i, subs)$
      $Calculate\_New\_Coordinates(i, subs)$
    **end if**
  **end for**
  $Wait(Update\_Interval)$
**end while**
```

2) Coordinate Mapping: In the coordinates calculating step in Algorithm 3, one choice is to set all the new coordinates to zeros. However, if this method is used, we need much more time for the system to converge every time after optimizing clustering. Our approach, called Coordinate Mapping, utilizes the current coordinates of some hosts to calculate the new coordinates of a host which changes its cluster. For instance, when node X is about to move to cluster C in Figure 5, its new coordinates in the Subspace (A,C)-based system are calculated like this: 1) Select some of X's current neighbors in the Subspace (A,B)-based system, which all lie in cluster A. 2) Calculate the distances from X to these neighbors using the current coordinates in Subspace (A,B). 3) Use simplex downhill algorithm [13] to calculate the new coordinates of

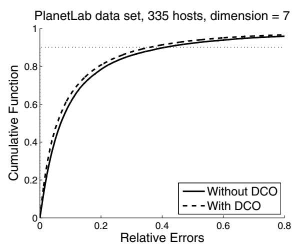


(a) Overall

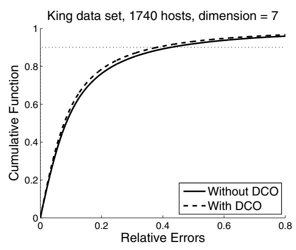


(b) In Different Distance Ranges

Fig. 6. Relative Error After Improvement in Vivaldi



(a) PlanetLab



(b) King

Fig. 7. Performance of Dynamic Cluster Optimization

X based on the current coordinates of these neighbors in the Subspace (A,C)-based system and the distances calculated in 2). In this way, X can get its exactly right new coordinates given that all hosts are perfectly mapped in the NC system in both Subspace (A,B) and (A,C). While the NC system could not provide perfect estimation accuracy, Coordinate Mapping still reduces the convergence time after cluster optimization. We can calculate X's new coordinates in other subspace-based or cluster-based systems with similar steps.

IV. Evaluation

In comparing the estimation accuracy of different systems, two popular data sets are used - the PlanetLab data set (see section III-C) and the King data set consisting of the RTT values among 1740 Internet naming servers [5]. We also test Tarantula's ability in the application of server selection [16] and detour finding [17]. The simulation is carried out in the MATLAB 2008 environment. In each round, the Vivaldi system updates the coordinates of one host based on the observation of latency to one of its neighbors. However, in Pharos and Tarantula, we need two and three measurements to neighbors in each round, respectively. Therefore, we run Vivaldi, Pharos and Tarantula for 6000, 3000, 2000 rounds so that the total traffic in all systems are the same.

A. Accuracy

1) Improvement for Short Links: We first test the performance of the modified Vivaldi algorithm used in Tarantula which was mentioned in section III-C. We tune the parameter μ to 64. As shown in Figure 6(a), the overall NPRE of modified Vivaldi is 22.8% less than that of original method. We also see that the improvement mainly focuses on short links (Figure 6(b)), which meets our expectation.

2) Dynamic Cluster Optimization: To simulate the uncertainty in clustering, we set three anchors based on geographical information. However, every host has a probability of 20% to join the wrong cluster. Then we compare the performance of Tarantula system with and without DCO. Practically, DCO adapts clustering once every 200 rounds. Figure 7 shows that with the use of DCO, Tarantula becomes less sensitive to the initial clustering and achieves a higher accuracy.

3) Overall Performance: We finally compare the overall estimation accuracy of different systems. All the abovementioned improvements for Tarntula have been applied to the final system. This time, μ is tuned to be 40 in a cluster-based system and 80 in a subspace-based system. The CDF of RE is plotted in Figure 8(a) and 8(d), from which we can see that Tarantula is much more accurate than other systems. In particular, the NPRE of Tarantula is about 25% less than that of Pharos and 39% less than that of Vivaldi in PlanetLab.

B. Application: Server Selection

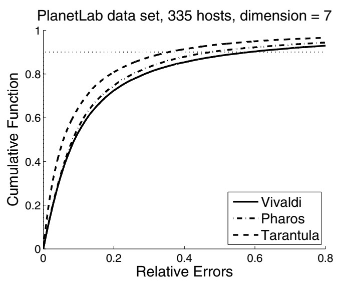
We apply different systems in the application of server selection. Practically, some nodes are randomly selected from each trace as servers, while other hosts play the role of clients, which try to select the nearest server. Since the hosts in PlanetLab data set are significantly less than those in the King data set, the servers we set in PlanetLab are also less. We calculate the stretch [16] for every client, which is defined as the actual distance to the closest server selected based on the NC system, divided by the distance to the actual closest server. Finally, we use the ninetieth percentile stretch (NP Stretch) as a metric to evaluate the performance of different NC systems. In practice, we do the experiments for 50 times and use the average NP Stretch in displaying.

Figure 8(b) and 8(e) show that Tarantula greatly reduces the NP Stretch in server selection compared with Vivaldi and Pharos. For instance, the NP Stretch is about 2.01 when selecting from 32 servers in the PlanetLab data set using Tarantula, which is 55.33% smaller than that of Pharos and 72.24% smaller than that of Vivaldi.

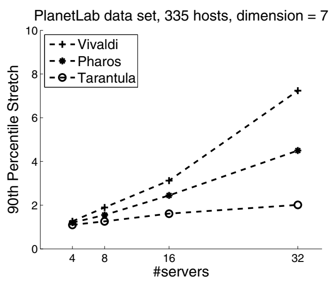
C. Application: Detour Finding

We can actually benefit from existing TIVs in real network [17]. For instance, X, Y and Z are three nodes in a TIV triangle and $L_{X,Y} + L_{Y,Z} < L_{X,Z}$, we can choose Y as a relay point in the real network so as to make a message travel faster from X to Z by first sending it to Y and then diverting it to Z. That is to say, we have found a detour from X to Z.

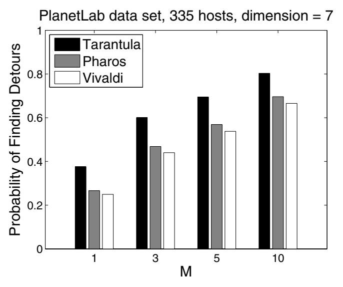
When trying to find detours, for example, from X to Z, we enumerate every other node as a relay host Y_i and calculate the predicted detour distance: $D_{X,Y_i} + D_{Y_i,Z}$. We then select M nodes, Y_{k_1} to Y_{k_M}, with the smallest predicted detour distances. With extra measurements to links XY_{k_i} and $Y_{k_i}Z$, we can pick up a node Y_{min} among the M nodes such that $L_{X,Y_{min}} + L_{Y_{min},Z}$ is the smallest. If this actual detour distance is smaller than $L_{X,Z}$, then we have successfully found a detour from X to Z. We denote the number of links with detours existing in real network as N_{detour}, and the number of



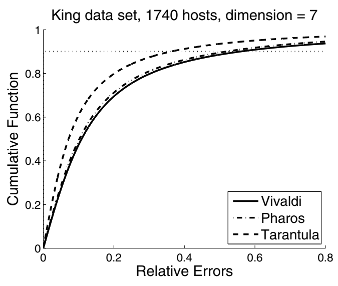
(a) PlanetLab: relative error



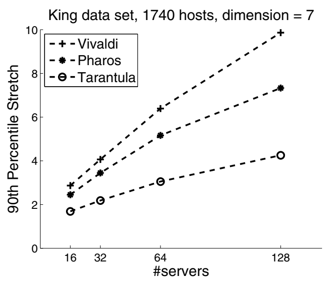
(b) PlanetLab: server selection



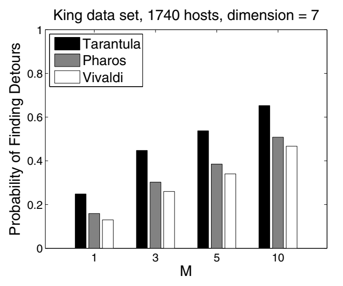
(c) PlanetLab: detour finding



(d) King: relative error



(e) King: server selection



(f) King: detour finding

Fig. 8. Compare the Performance of Different NC Systems

links that we can find detours as n_{detour}. Then the probability of finding detours is defined as: $P_{detour} = \frac{n_{detour}}{N_{detour}}$.

This probability is used to evaluate the ability of detour finding. We show the performance of different systems in Figure 8(c) and 8(f). When only making one extra measurement ($M = 1$), Tarantula's probability of finding detours is about 30% larger than that of Pharos. As M grows larger from 1 to 10, detour finding with Tarantula system is again much more efficient than Pharos and the Vivaldi system.

V. Conclusion

In this paper, we categorize the existing TIVs into three classes, and find out that existing two-layer NC systems are only capable of handling a small portion of TIVs. Our new NC system, Tarantula, further addresses the TIV problem. Our contributions are three folds: 1) Tarantula uses a novel way to divide the space and deals with a much larger portion of TIVs. 2) We modify the Vivaldi algorithm used in Tarantula so that short links are better predicted. 3) We propose DCO which periodically moves the unfitted hosts to a more suitable cluster. With real Internet measurement traces, we show that Tarantula outperforms Pharos and Vivaldi greatly in estimation accuracy. Moreover, Tarantula performs the best in the application of server selection and detour finding. We intend on expanding Tarantula structure to help improving the matrix factorization model based NC systems such as Phoenix [2].

VI. Acknowledgements

This work is supported by the National Basic Research Program of China (No.2007CB310806) and National Science Foundation of China (No.60850003).

References

[1] S. Agarwal, J. Lorch. Matchmaking for Online Games and Other Latency Sensitive P2P Systems. In Proc. of ACM SIGCOMM, 2009.
[2] Y. Chen, X. Wang, C. Shi, et al. Phoenix: A Weight-based Network Coordinate System Using Matrix Factorization. To appear in IEEE Transactions on Network and Service Management, 2011.
[3] Y. Chen, Y. Xiong, et al. Pharos: Accurate and Decentralised Network Coordinate System. IET Communications, 2009, 3(4):539-548.
[4] M. Costa, M. Castro, A. Rowstron, et al. PIC: Practical Internet Coordinates for Distance Estimation. In Proc. of IEEE ICDCS, 2004.
[5] F. Dabek, R. Cox, and F. Kaashoek. Vivaldi: A Decentralized Network Coordinate System. In Proc. of ACM SIGCOMM, 2004.
[6] B. Donnet, B. Gueye, et al. A Survey on Network Coordinates Systems, Design, and Security. IEEE Communication Surveys and Tutorial, 2010.
[7] M. A. Kaafar, B. Gueye, F. Cantin, et al. Towards a Two-tier Internet coordinate system to mitigate the impact of triangle inequality violations. In Proc. of IFIP-TC6 Networking, 2008.
[8] C. de Launois, S. Uhlig and O. Bonaventure. Scalable Route Selection for IPv6 Multihomed Sites. In Proc. Networking 2005.
[9] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In Proc. of NSDI, 2007
[10] S. Lee, Z. Zhang, S. Sahu, et al. On Suitability of Euclidean Embedding of Internet Hosts. In Proc. of ACM SIGMetrics/Performance, 2006
[11] E.K. Lua, T. Griffin, M. Pias, et al. On the Accuracy of Embeddings for Internet Coordinate Systems. In Proc. of ACM IMC, 2005.
[12] C. Lumezanu, R. Baden, Dave Levin, Neil Spring, et al. Symbiotic Relationships in Internet Routing Overlays. In Proc. of NSDI, 2009
[13] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In Proc. of INFOCOM, June 2002.
[14] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In Proc. of INFOCOM, April 2003.
[15] P. Wendell, J. W. Jiang, et al. DONAR: Decentralized Server Selection for Cloud Services. In Proc. of ACM SIGCOMM, 2010
[16] R. Zhang, C. Tang, Y. C. Hu, S. Fahmy and X. Lin. Impact of the Inaccuracy of Distance Prediction Algorithms on Internet Applications: an Analytical and Comparative Study. In Proc. of IEEE INFOCOM, 2006.
[17] Y. Zhu, Y. Chen, et al. Taming the Triangle Inequality Violations with Network Coordinate System on Real Internet. In Proc. of ACM ReArch, 2010.
[18] Azureus bittorrent. http://azureus.sourceforge.net/.