Network Latency Prediction for Personal Devices: Distance-Feature Decomposition from 3D Sampling

***Abstract*—With an increasing popularity of real-time applications, such as live chat and gaming, latency prediction between personal devices including mobile devices becomes an important problem. Traditional approaches recover all-pair latencies in a network from sampled measurements using either Euclidean embedding or matrix factorization. However, these approaches targeting static or mean network latency prediction fall short for personal device latencies, due to unstable and time-varying network conditions, triangle inequality violation and unknown rank of latency matrices. In this paper, by analyzing latency measurements from the *Seattle* platform, we propose new methods for both static latency estimation as well as the dynamic estimation problem given 3D latency matrices sampled over time. We propose a distance-feature decomposition algorithm that can decompose latency matrices into a distance component and a network feature component, and further exploit the structured pattern inherent in the 3D sampled data to increase estimation accuracy. Extensive evaluations driven by real-world traces show that our proposed approaches significantly outperform various state-of-the-art latency prediction techniques.**

I. Introduction

Recent years have witnessed a dramatic growth of Internet traffic of personal devices, among which a large portion comes from mobile devices such as smartphones and tablets [1]. Due to the increasing popularity of interactive applications including live video chat (e.g., FaceTime, Skype, Google+) and gaming, understanding the latencies between personal devices has become essential to the operation of such real-time and delay-sensitive applications. A common idea to estimate end-to-end Internet latencies in a large network is to measure RTTs for only a subset of all pairs, based on which the missing latencies of other pairs are recovered. Existing solutions to such an estimation problem either relies on network embedding (e.g., Vivaldi [2], GNP [3]), which maps nodes into a space, so that their distances in the space predict their latencies, or applies matrix factorization [4] assuming the latency matrix has a certain low rank.

However, the unique characteristics of personal devices have posed great challenges to latency estimation. *First*, almost all existing approaches perform static network latency prediction, based on one incomplete matrix formed by current, mean or median RTTs, assuming the latencies are stable or unchanged, while in reality, latencies between personal devices could vary dramatically over time due to changing network connectivities. In other words, the prediction based on such 2D sampling fails to utilize the significantly useful structures inherent in the 3D data of delay matrices evolved over time. *Second*, network embedding algorithms such as Vivaldi [2], [5] often attempt to find the network coordinates of nodes in a Euclidean space. However, it is a widespread belief [4], [6], [7] that the triangle inequality may not hold for latencies among end users at the edge of the Internet. *Third*, matrix factorization schemes [4] assume a certain rank of the delay matrices to decide the dimensions of the factors. However, in reality, ranks of delay matrices of personal devices are either hard to know or unstable.

In this paper, we conduct an in-depth analysis of latency measurements collected from *Seattle* [8], an educational and research platform of open cloud computing and peer-to-peer computing. Seattle consist of laptops, servers, and phones, donated by users and institutions. Compared with another dataset we collected from the PlanetLab, we observe that the latencies between personal devices present different properties in latency distribution as well as time-varying characteristics.

Based on measurements from Seattle, we propose novel methods for both static and dynamic latency estimation problems. *First*, we propose the so-called "Distance-Feature (D-F) Decomposition" method which can decompose a given incomplete latency matrix into a distance matrix that models the impact of geographical distances on propagation delays, and a low-rank network feature matrix that models correlated network conditions among nodes. We propose an iterative learning process using Euclidean embedding and the Penalty Decomposition (PD) method for matrix completion as subroutines. The proposed decomposition avoids the shortcomings of both Euclidean embedding and matrix completion, while exploiting both of their strengths, since the symmetry and triangle inequality do not have to hold for network features, while the low-rank assumption is not imposed on distances.

More importantly, to predict changing latencies on the go, we propose the first dynamic recovery process to estimate the current missing latencies based on "frames" of incomplete latency matrices sampled in the past. By jointly applying different matrix transformation schemes, we convert the incomplete 3D data into structured 2D matrices, and extend the proposed D-F decomposition to apply to the transformed matrices, exploiting the inherent structures both within each frame and across frames.

We conduct extensive trace-driven simulations based on measurements from both Seattle and PlanetLab, and show that the D-F decomposition significantly outperforms state-of-the-art static recovery techniques, including matrix factorization and Vivaldi with a high dimension especially for the Seattle data. The dynamic recovery based on 3D sampling can further substantially enhance the prediction accuracy of changing latencies compared with static latency prediction algorithms.

The remainder of this paper is organized as follows. Sec. II reviews the related literature, followed by a comparative study of latency measurements from both Seattle and PlanetLab in Sec. III. We propose the distance-feature decomposition method for latency recovery in Sec. IV and study its performance through trace-driven simulations as compared to state-of-art algorithms. In Sec. V, we propose our dynamic latency estimation scheme based on the 3D data of latency matrices evolved over time and again conduct extensive simulations to evaluate its performance. The paper is concluded in Sec. VI.

II. Relationship to Prior Work

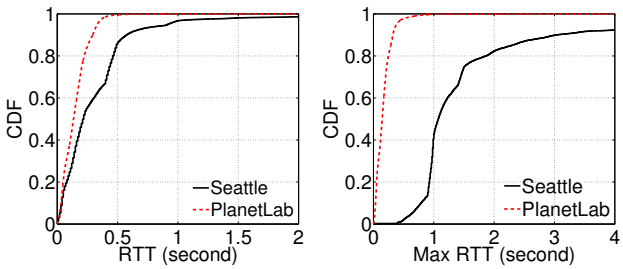
Network coordinate systems (NCSs) embed hosts into a coordinate space such as Euclidean space, and predict latencies by the coordinate distances between hosts [9]. In this way, explicit measurements are not required to predict latencies. Most of the existing NCSs, such as Vivaldi [2], GNP [3], rely on the Euclidean embedding model. However, such systems suffer a common drawback that the predicted distances among every three hosts have to satisfy the triangle inequality, which does not always hold in practice. Many studies [5], [10] have reported the wide existence of triangle inequality violations (TIV) on the Internet.

To overcome the TIV problem, matrix factorization is introduced in [11] and has recently drawn an increasing attention in the networking community [4], [12]. The key idea is to assume a network distance matrix is low-rank and complete it by factorizing it into two smaller matrices using methods such as Singular Value Decomposition (SVD) or Non-negative Matrix Factorization (NMF) [13]. The estimated distances via matrix factorization do not have to satisfy the triangle inequality. However, these systems actually do not outperform Euclidean embedding models significantly, due to reported problems such as prediction error propagation [6]. Besides, without considering the geographical distances between hosts that dictate propagation delays, they have missed a major chunk of useful information.

Beyond matrix factorization, the general matrix completion problem, including minimizing the rank of an incomplete matrix subject to limited deviation from known entries [14] and minimizing the deviation from known entries subject to a fixed rank [15], has also been widely studied recently for numerous applications in control, image recovery and data mining. Besides, measurement studies have been conducted for different kinds of networks, such as WiFi networks [16], Cellular networks [17], and 4G LTE networks [18], reporting the latencies and other properties. The latency measurement on Seattle is cross-network in nature, as Seattle involves many different types of nodes from servers to laptops and smartphones.

III. *Seattle* vs. *PlanetLab*: Measuring the Latencies

In this section, we characterize the latencies between personal devices according to the measurements we have collected from *Seattle* [8]. *Seattle* is a new open peer-to-peer computing platform that provides access to personal computers



(a) RTT distributions (b) Max RTT for each pair of nodes

Fig. 1. RTT distributions in Seattle and PlanetLab. a) CDFs of all measured RTTs. b) CDFs of the maximum RTT measured for each pair of nodes.

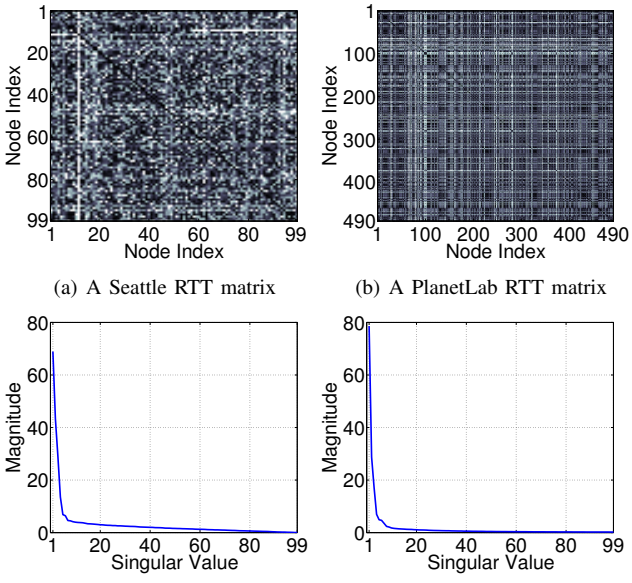
worldwide. In contrast to PlanetLab [19], which is a global research network comprised of computers mostly located in stable university networks, the Seattle nodes include many personal devices, such as mobile phones, laptops, and desktop computers, donated by users and institutions. Due to the diversity, mobility and instability of these personal devices, there is significant difference between Seattle and PlanetLab in terms of latency measurements.

We have collected the round trip times (RTTs) between 99 nodes in the Seattle network in a 3-hour period commencing at 9 pm on a day in summer 2014. The measurement has resulted in 688 latency matrices containing $6,743,088$ latencies, each of which has a size of 99×99 and represents the pairwise RTTs between 99 nodes collected in a 15.7-second timeframe. In the sequence, we may refer to each matrix as a "frame" in such 3D measurement data. Our data collection on Seattle was limited to 99 nodes because as a new platform that includes both personal computers and servers, Seattle is yet to receive more donations of personal devices. However, it will be clear in Sec. IV and Sec. V that the collected data is rich enough for the purpose of studying latency prediction algorithms.

As a benchmark dataset, we have also collected the RTTs between 490 PlanetLab nodes in a 9-day period in 2013 and obtained 18 matrices containing $4,321,800$ latencies, each of which has a size of 490×490 and represents the pairwise RTTs collected in a 14.7-hour timeframe. We compare the collected Seattle data and PlanetLab data in terms of inter-node RTTs, rank properties, and time-varying characteristics.

Round Trip Times. Fig. 1(a) shows that the Seattle RTTs are greater than those in PlanetLab, with values spread in a wider range. The mean RTT of the two datasets are 0.36 seconds for Seattle and 0.15 seconds for PlanetLab, respectively. While the largest measured RTT in PlanetLab is 7.90 seconds, the maximum RTT measured in Seattle is 90.50 seconds, which maybe because a corresponding node is not online, a frequent case for cellular devices out of the service region. The long tail in Seattle RTTs implies that triangle inequality violation may be prevalent in Seattle.

Rank of Latency Matrices. Fig. 2(a) and Fig. 2(b) plot the heat maps of a typical frame (one of the 688 latency matrices) in the Seattle data and a typical frame in the PlanetLab data (with white representing large RTT values and black representing small RTTs). We can observe that redundant patterns exist in Fig. 2(a) and Fig. 2(b) and obtain an intuitive



(a) A Seattle RTT matrix (b) A PlanetLab RTT matrix

(c) A Seattle RTT matrix (d) A PlanetLab RTT matrix

Fig. 2. The heat maps and singular values of a Seattle RTT matrix and a PlanetLab RTT matrix.

knowledge that the latency matrices in both datasets may be low-rank. We further perform singular value decomposition (SVD) [20] on both latency matrices, and plot the singular values of both latency matrices in Fig. 2(c) and Fig. 2(d). We can observe that the singular values of both matrices decrease fast. The 15th singular value of the Seattle latency matrix is 4.9% of its largest one, while the 7th singular value of the PlanetLab latency matrix is 4.7% of its largest one. This confirms the low-rank nature of Internet RTTs reported in previous measurements [21].

Time-Varying Characteristics. Unlike PlanetLab, since Seattle contains personal devices including laptops and mobile phones, the diversity and mobility of these personal devices may greatly affect the stability of latency measurements. Fig. 3 plots the RTT measurements evolved over time for 3 typical pairs of nodes in Seattle and PlanetLab, respectively. In contrast to the latencies in PlanetLab which almost remain unchanged over 9 days, the 3 pairs of nodes in Seattle have latencies that vary frequently even in only 30 minutes. The average standard deviation of the latencies between each pair of nodes is 0.36s in Seattle and 0.01s in PlanetLab.

To get a further idea about the evolution of the entire frame of data over time, we denote $M(t)$ the $n \times n$ matrix of RTTs measured at time t, where $M_{ij}(t)$ represents the RTT between node i and node j. Then, we define the Relative Varying Percentage (RVP) of $M(t)$ relative to the first matrix $M(1)$ as

$$RVP(t,1) = \frac{1}{n^2-n} \sum_{i,j=1,i\neq j}^{n} [M_{ij}(t) - M_{ij}(1)]/M_{ij}(1).$$

We compare the RVPs of the Seattle RTTs over time with those of the PlanetLab RTTs, by plotting the RVP of every frame at time t relative to the first frame of data in Fig. 4, which shows a huge difference between the two datasets. While the largest RVP of the PlanetLab frames over 9 days is only 0.09, the RVPs of the Seattle frames measured for

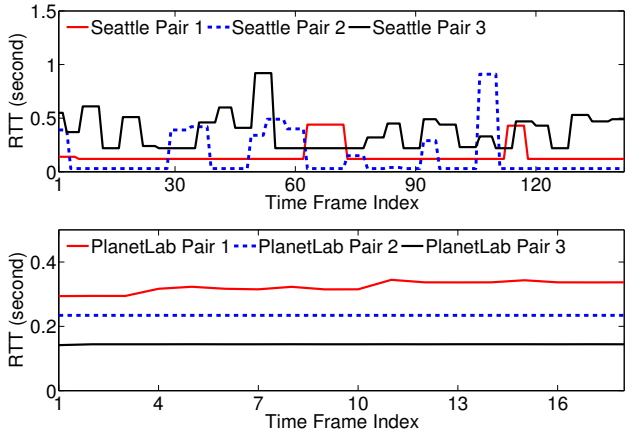


Fig. 3. The time-varying charactistic of latencies between 3 pairs of nodes in Seattle and PlanetLab.

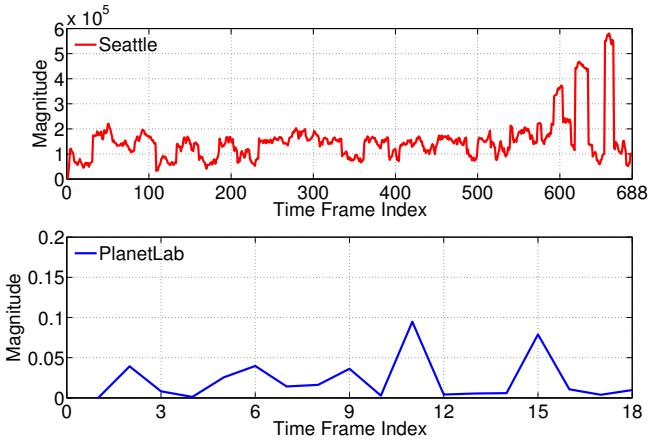


Fig. 4. The relative varying percentage of every measured latency matrices to the first measured latency matrix in Seattle and PlanetLab.

3 hours vary from 0 to 5.8×10^5 with a mean of 1.5×10^5. This demonstrates the time-varying nature of Seattle latencies, which makes it hard to predict the latency between two Seattle nodes. Traditional network coordinate embedding is not suitable to model the latencies in personal device networks. For example, if a Seattle node is a cellphone, whenever the phone user moves, its coordinate will change greatly according to the changes in surrounding network environments.

IV. Static Latency Estimation via Distance-Feature Decomposition

In this section, we propose a new algorithm for the static network latency recovery problem, given a frame of latency matrix with missing values. Our new algorithm exploits both the underlying geographical distances and the low-rank structure of the RTT matrix at hand. Traditional Euclidean embedding [2], [3] assumes symmetry and triangle inequalities for pairwise latencies, which may not be true in reality, especially for mobile devices with poor connectivity. On the other hand, matrix factorization approaches [4] rely on low-rank structures in the latency matrix. However, such low-rank structure may not exist if the nodes indeed reside in a Euclidean space. A toy example of 3 nodes in a 2D plane (e.g., with pairwise

distances as $3, 4, 5$) shows that the 3×3 latency matrix (i.e., $[(0, 3, 4)^\mathsf{T}, (3, 0, 5)^\mathsf{T}, (4, 5, 0)^\mathsf{T}]$ in this case) has a full rank in general cases.

Our algorithm combines the strengths of both methods by modeling the pairwise latencies with two components: a *distance component*, representing geographic information that dictates propagation delay, and a *network feature component*, representing correlated network connectivity. The essence of our algorithm is a learning process that iteratively decomposes both components.

A. The Static Network Latency Prediction Problem

Let $\Re^n$ denote the n-dimensional Euclidean space. The set of all $m \times n$ matrices is denoted by $\Re^{m \times n}$. Assume a network contains n nodes, and the latency matrix measured between these nodes is $M \in \Re^{n \times n}$, with M_{ij} representing the RTT between node i and node j. We use Θ to denote the set of index pairs $(i,\ j)$ where the measurements M_{ij} are missing. For missing entries $(i,\ j) \in \Theta$, we denote their values as $M_{ij} = \mathsf{unknown}$. We define the sample rate R as the percentage of known entries in M.

The static prediction problem is—given an RTT matrix M with missing entries, recover the values of the missing entries. We let $\hat{M} \in \Re^{n \times n}$ denote the recovered RTT matrix.

B. Iterative Distance-Feature Decomposition

We assume that the RTT matrix M can be decomposed into a symmetric distance matrix $D \in \Re^{n \times n}$ and an asymmetric network feature matrix $F \in \Re^{n \times n}$, i.e.,

$$M_{ij} = D_{ij} F_{ij},\ 1 \le i,\ j \le n, \tag{1}$$

where D_{ij} represents the distance between nodes i and j in a Euclidean space, and F_{ij} represents the "network connectivity" from node i to node j: a smaller F_{ij} indicates a better connectivity. The rationale is that while the geographical distance of two nodes on the earth dictates the propagation delay between them, other factors such as network congestions and node status can also affect the RTT values.

We assume that *only* the network feature matrix F is low-rank. This is because there exists correlation between network connectivities on all incoming (or outgoing) links of each node. Another interpretation is through feature vectors. If the rank of F is r, F can be represented by

$$F = F_l^\mathsf{T} F_r,\ F_l \in \Re^{r \times n},\ F_r \in \Re^{r \times n}. \tag{2}$$

We call the i^{th} column of F_l, denoted by f_l^i, the *left feature vector* of node i, which represents the network feature from node i to other nodes. Similarly, we call the i^{th} column of F_r, denoted by f_r^i, the *right feature vector* of node i, which represents the network feature from other nodes to node i. Hence, the network connectivity from node i to node j can be determined by the feature vectors, i.e., $F_{ij} = {f_l^i}^\mathsf{T} f_r^j$.

Our model overcomes the shortness of both Euclidean embedding and low-rank matrix completion, since symmetry and triangle inequalities only need to hold for the distance matrix D but not F, and the low-rank property is only assumed for network connectivity F but not D.

Algorithm 1 Iterative Distance-Feature Decomposition

1: $D^0 := M$
2: **for** $k = 1$ to maxIter **do**
3: Perform Euclidean Embedding on D^{k-1} to get the complete matrix of distance estimates $\hat{D}^k$
4: $F_{ij}^k := \begin{cases} \frac{M_{ij}}{\hat{D}_{ij}^k} & \forall (i,\ j) \notin \Theta \\ \text{unknown} & \forall (i,\ j) \in \Theta \end{cases}$
5: Perform Matrix Completion (4) on F^k to get the complete matrix of network feature estimates $\hat{F}^k$
6: $D_{ij}^k := \begin{cases} \frac{M_{ij}}{\hat{F}_{ij}^k} & \forall (i,\ j) \notin \Theta \\ \text{unknown} & \forall (i,\ j) \in \Theta \end{cases}$
7: **end for**
8: $\hat{M}_{ij} := \hat{D}_{ij}^{\mathsf{maxIter}} \hat{F}_{ij}^{\mathsf{maxIter}},\ 1 \le i,\ j \le n$

To learn both the distance matrix D and network feature matrix F from a latency matrix M with missing entries, we propose an iterative algorithm, described in Algorithm 1, that incorporates both Euclidean embedding and low-rank matrix completion as subroutines. Denote the estimated matrix D and F at iteration k as $\hat{D}^k$ and $\hat{F}^k$. First, we initialize D^0 to be the original latency matrix M with missing entries. In each iteration, we estimate the distance matrix $\hat{D}^k$ with Euclidean embedding. We then obtain the remaining ratio matrix between M and $\hat{D}^k$, which is the incomplete network feature matrix F^k. By applying low-rank matrix completion on F^k, we get the estimated complete network feature matrix $\hat{F}^k$ at iteration k. We then divide M by $\hat{F}^k$ to get D^k, which is the input for Euclidean embedding in the next iteration, and so on. After a few iterations, $\hat{D}$ and $\hat{F}$ will approach the real geographical distance component and the network factor, respectively. Finally, the predicted latency between nodes i and j is given by (1).

The two critical subroutines in our algorithm are Euclidean embedding on D^k and low-rank matrix completion on F^k. There are various algorithms available for these two tasks. We apply the Vivaldi algorithm [2] for Euclidean embedding, and the Penalty Decomposition (PD) method [15] for low-rank matrix completion.

1) **Euclidean Embedding:** Given the input matrix $M \in \Re^{n \times n}$, Vivaldi predicts network latencies by assigning every node a coordinate and estimating the latency between two nodes by their Euclidean distance, i.e., the estimated latency $\hat{M}_{ij}$ between nodes i and j is given by

$$\hat{M}_{ij} = \|x_i - x_j\|, \tag{3}$$

where x_i is the coordinate assigned to node i.

2) **Low-Rank Matrix Completion:** Given an input matrix $X \in \Re^{m \times n}$ with missing entries, the problem of low-rank matrix completion is to find a complete matrix $\hat{X}$ by solving

$$\begin{aligned} \underset{\hat{X} \in \Re^{m \times n}}{\text{minimize}} \quad & \text{rank}(\hat{X}) \\ \text{subject to} \quad & |\hat{X}_{ij} - X_{ij}| \le \tau,\ (i,\ j) \notin \Theta, \end{aligned} \tag{4}$$

where τ is a parameter to control the error tolerance on known entries of X [15].

We utilize the Penalty Decomposition (PD) method [15] to solve the low-rank matrix completion problem in Algorithm 1. The PD method can solve general rank minimization problems like the following:

$$\begin{array}{ll} \underset{X}{\text{minimize}} & f(X) + \nu \text{ rank}(X) \\ \text{subject to} & g(X) \leq 0,\ h(X) = 0,\ X \in \Phi \cap \Psi, \end{array} \tag{5}$$

for $\nu \geqslant 0$, where Φ is a closed convex set and Ψ is a closed unitarily invariant convex set in $\Re^{m\times n}$, and $f : \Re^{m\times n} \to \Re$, $g : \Re^{m\times n} \to \Re^p$ and $h : \Re^{m\times n} \to \Re^q$ are continuously differentliable functions.

The PD method solves problem (5) by reformulating it as

$$\begin{array}{ll} \underset{X}{\text{minimize}} & f(X) + \nu \text{ rank}(Y) \\ \text{subject to} & g(X) \leq 0,\ h(X) = 0,\ X \in \Phi,\ Y \in \Psi, \end{array} \tag{6}$$

and defining a corresponding quadratic penalty function as

$$\begin{aligned} P_\varrho(X, Y) = f(X) + \nu \text{ rank}(Y) \\ + \frac{\varrho}{2}(\|[g(X)]^+\|_2^2 + \|h(X)\|_2^2 + \|X - Y\|_F^2), \end{aligned} \tag{7}$$

where $\varrho > 0$ is a penalty parameter, $[\cdot]^+$ denotes the non-negative part of a vector that $x^+ = \max(x, 0)$ given a vector $x \in \Re^n$, and $\|\cdot\|_F$ is the Frobenius norm of a real matrix $X \in \Re^{n\times n}$, i.e., $\|X\|_F = \text{Tr}(XY^T)$, with $\text{Tr}(\cdot)$ denoting the trace of a matrix. Then the PD method minimizes (7) by alternately solving two subproblems: minimizing over X with Y fixed and minimizing over Y with X fixed, each of which can be approximately solved by a block coordinate descent (BCD) method, which is widely used to solve large-scale optimization problems [22].

It is easy to see that problem (4) is a special case of problem (5) with $f(X) \equiv 0,\ p = q = 0,\ \nu = 1,\ \Psi = \Re^{m\times n}$ and

$$\Phi = \{X \in \Re^{m\times n} :\ |X_{ij} - M_{ij}| \leq \tau,\ (i, j) \in \Theta\}. \tag{8}$$

Thus, the two subproblems to be alternately solved are in the form of

$$\begin{array}{ll} \underset{X}{\text{minimize}} & \{\|X - A(Y)\|_F^2 :\ X \in \Phi\}, \\ \underset{Y}{\text{minimize}} & \{\text{rank}(Y) + \varrho\|Y - B(X)\|_F^2 :\ Y \in \Re^{m\times n}\} \end{array} \tag{9}$$

for some $\varrho > 0,\ A,\ B \in \Re^{m\times n}$, respectively. Thus, the PD method can be suitably applied to solve (4). Please refer to [15] for more details about the PD completion method.

C. Performance Evaluation

We evaluate our algorithm based on both the Seattle data and PlanetLab data, in comparison with various state-of-the-art approaches. We define the relative estimation error (RE) on missing entries as $|\hat{M}_{ij} - M_{ij}|/M_{ij}$, for $(i,\ j) \in \Theta$, which will be used to evaluate prediction accuracy.

1) **Comparison with Other Algorithms:** We compare our algorithm with the following approaches:

- **Vivaldi** with dimension $d = 3$, $d = 7$, and $d = 3$ plus a height parameter;
- **PD matrix completion (MC)** directly applied to the latency matrix M;
- **DMFSGD Matrix Factorization** [4] that attempts to approximate M by the product of two smaller matrices $U \in \Re^{r\times n}$ and $V \in \Re^{r\times n}$, i.e., $\hat{M} = U^T V$, such that a loss function based on $M - \hat{M}$ is minimized, where r is the assumed rank of $\hat{M}$.

For our method, the Euclidean embedding part on D is done using Vivaldi with a low dimension of $d = 3$ without heights.

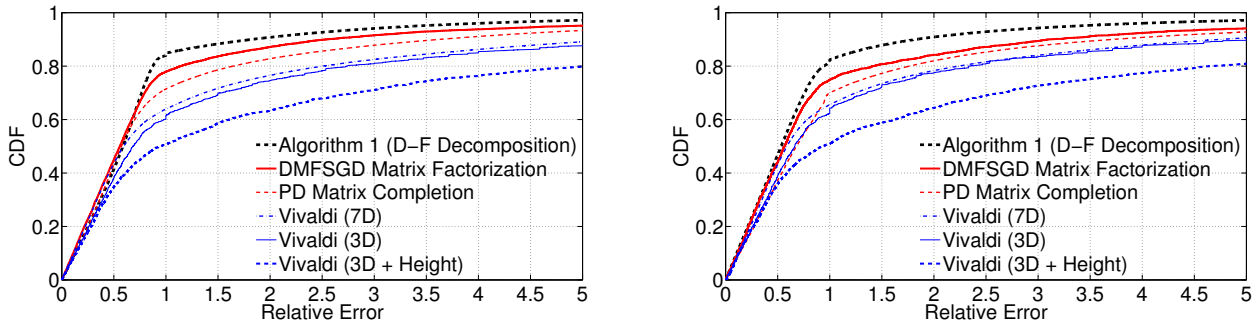
We randomly choose 100 frames from the 688 frames in the Seattle data. For PlanetLab data, as differences among the 18 frames are small, we randomly choose one frame to test the methods. Recall that the sample rate R is defined as the percentage of known entries. Each chosen frame is independently sampled at a low rate $R = 0.3$ (70% latencies are missing) and at a high rate $R = 0.7$, respectively.

For DMFSGD, we set the rank of $\hat{M}$ to $r = 20$ for Seattle data and $r = 10$ for PlanetLab data, respectively, since the 20^{th} (or 10^{th}) singular value of M is less than 5% of the largest singular value in Seattle (or PlanetLab). In fact, $r = 10$ is adopted by the original DMFSGD work [4] based on PlanetLab data. We have tried other ranks between 10-30 and observed similar performance. We plot the relative estimation errors on missing latencies in Fig. 5 and Fig. 6, for the Seattle data and PlanetLab data, respectively, under 6 methods.

For the Seattle results in Fig. 5, we can see that the D-F decomposition outperforms all other algorithms by a substantial margin. We first check the Vivaldi algorithms. Even if Vivaldi Euclidean embedding is performed in a 7D space, it only improves over 3D space slightly, due to the fundamental limitation of Euclidean assumption. Furthermore, the 3D Vivaldi with a height parameter, which models the "last-mile latency" to the Internet core [2], is even worse than the 3D Vivaldi without heights in Seattle. This implies that latencies between personal devices are better modeled by their pairwise core distances multiplied by the network conditions, rather than by pairwise core distances plus a "last-mile latency".

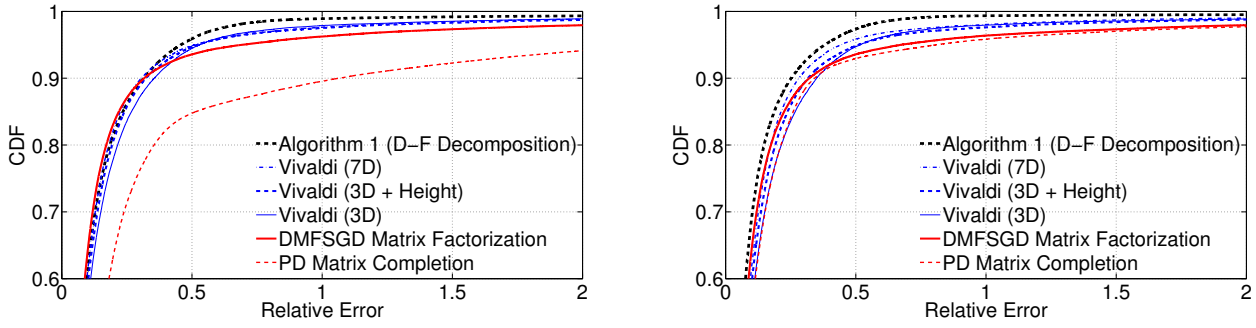
We now look at the matrix completion algorithms in Fig. 5. Both PD matrix completion and DMFSGD are inferior to our algorithm because they solely rely on the low-rank assumption, which may not hold for the pairwise core distances; as has been pointed our in the beginning of Sec. IV, the low-rank property seldom holds if the core distances indeed reside in a Euclidean space randomly.

For the PlanetLab results in Fig. 6, our algorithm is only slightly better than other algorithms, which again implies the much different behavior of Seattle and PlanetLab latencies. The improvement in PlanetLab is not as great as in Seattle, because network conditions in PlanetLab are more stable, which makes the network feature matrix F less useful. This



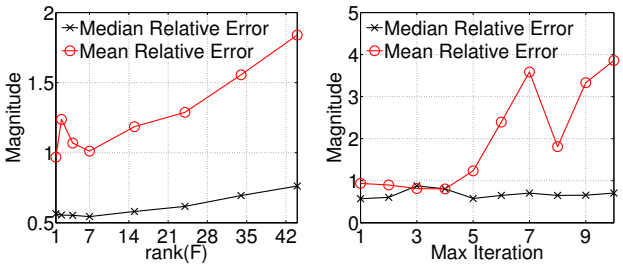
(a) Sample rate $R = 0.3$ (b) Sample rate $R = 0.7$

Fig. 5. The CDFs of relative estimation errors on missing values for the Seattle dataset, under sample rate $R = 0.3$ and $R = 0.7$. The legends follow the same order as the curves at relative error $= 1.0$



(a) Sample rate $R = 0.3$ (b) Sample rate $R = 0.7$

Fig. 6. The CDFs of relative estimation errors on missing values for the PlanetLab dataset, under sample rate $R = 0.3$ and $R = 0.7$. The legends follow the same order as the curves at relative error $= 0.5$.



(a) rank(F) (b) maxIter

Fig. 7. Influence of rank(F) and maxIter for the Seattle dataset.

fact again shows the unique strength of our algorithm to cope with unstable personal device networks.

2) **Impact of Parameters:** We investigate the impact of three parameters to our algorithm: the sample rate R, the rank of network feature matrix rank(F) and the number of iterations maxIter. Fig. 5(a) and Fig. 5(b) reveal the robustness of our algorithm at both high ($R = 0.7$) and low ($R = 0.3$) sample rates. We have also tested other sample rates and observed similar results.

We then study the impact of the achieved rank(F) from the PD matrix completion part in our algorithm by tuning τ in (4) to indirectly control the produced rank(F). Recall that rank(F) also represents the dimension of node left/right feature vectors. Fig. 7(a) shows how the median and mean of relative estimation errors change as rank(F) varies. Our experimental experience suggests that the best results are usually achieved when $1 \leq$ rank(F) ≤ 10 (the best result is achieved at 7 in this figure).

Finally, Fig. 7(b) evaluates the impact of the number of iterations, which shows the best accuracy is often achieved in just the 2^{nd} or 3^{rd} iteration. The performance degrades when more iterations are performed due to the overfitting effect.

V. Dynamic Latency Estimation via 3D Sampling

Traditional network latency estimation [2], [4], [11] all attempt to predict static (median/mean) network latencies. However, as shown in Sec. III, the latencies between personal devices may change over time. This motivates us to study the dynamic latency estimation problem to fill the missing entries in the current frame T based on a window of frames measured from $t = 1$ to $t = T$. We call such an approach dynamic latency estimation from "3D sampling", since the network latencies are sampled over time, where each frame of data contains only partial measurements of RTTs.

As shown in Fig. 3, although the latency between a pair of Seattle nodes changes frequently, it may stay in a state for a while before hopping to a new state. Therefore, if we utilize this autocorrelation between frames at different time in addition to the inter-node correlation in the network feature matrix, we may improve the prediction accuracy for the current frame.

Suppose we have measured the latency matrix of n nodes for T time frames, where T is called the prediction window. Denote the incomplete matrix measured at time t by $M(t)$, then our objective is to predict the missing entries in the current latency matrix $M(T)$ based on $M(1), \ldots, M(T)$.

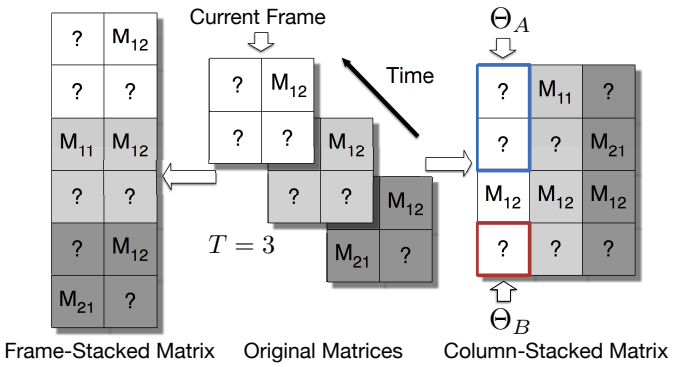
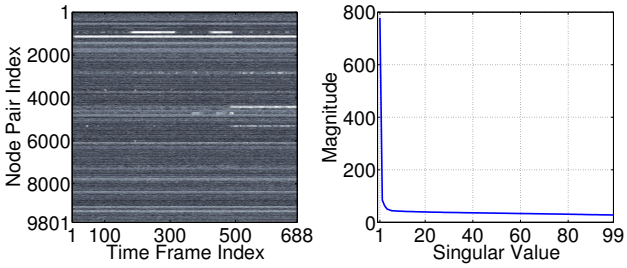


Fig. 8. The frame-stacking and column-stacking operations.



(a) Column-Stacked Matrix Heat Map (b) Column-Stacked Matrix SVD

Fig. 9. The heat map and singular values of the column-stacked Seattle dataset. The size of the compound matrix is 9801×688, and every column of the matrix contains all the latencies measured in one frame.

A. D-F Decomposition from 3D Sampled Data

The main idea of our algorithm is to stack the latency matrices measured at different times in different ways to obtain 2D compound matrices, and then exploit the low-rank nature of the compound matrices. We use two kinds of stack operations: frame-stacking and column-stacking, as illustrated in Fig. 8. In column-stacking, every latency matrix $M(t) \in \Re^{n \times n}$ is transformed into a column vector $V(t) \in \Re^{n^2}$ containing the latencies of all pairs of nodes. Then, the *column-stacked matrix* $\Omega \in \Re^{n^2 \times T}$ consists of vectors $V(t)$ ordered by their measured time. In frame-stacking, we directly concatenate all the measured latency matrices sorted by time to form the *frame-stacked matrix* $\mho \in \Re^{nT \times n}$.

Fig. 9 shows the heat map and singular values of the column-stacked matrix that consists of 688 frames of the Seattle data. As we can see, the heat map reveals the low-rank nature of the compound matrix: even though the column-stacked matrix has a size of 9801×688, the 22^{nd} singular value of the matrix is only 5% of the largest one. This implies that the latency matrices measured at different times are highly correlated while evolving with time.

Given a prediction window of T frames of latency matrices, we use Θ to denote the set of index pairs (i, j) for which $M_{ij}(T)$ are missing at the current time T. We can further divide Θ into two subsets:

$\Theta_A = \{(i,j) | M_{ij}(t)$ is known for at least one $t \in \{1, \ldots, T-1\}\}$

$\Theta_B = \{(i,j) | M_{ij}(t)$ is missing for all $t \in \{1, \ldots, T-1\}\}$

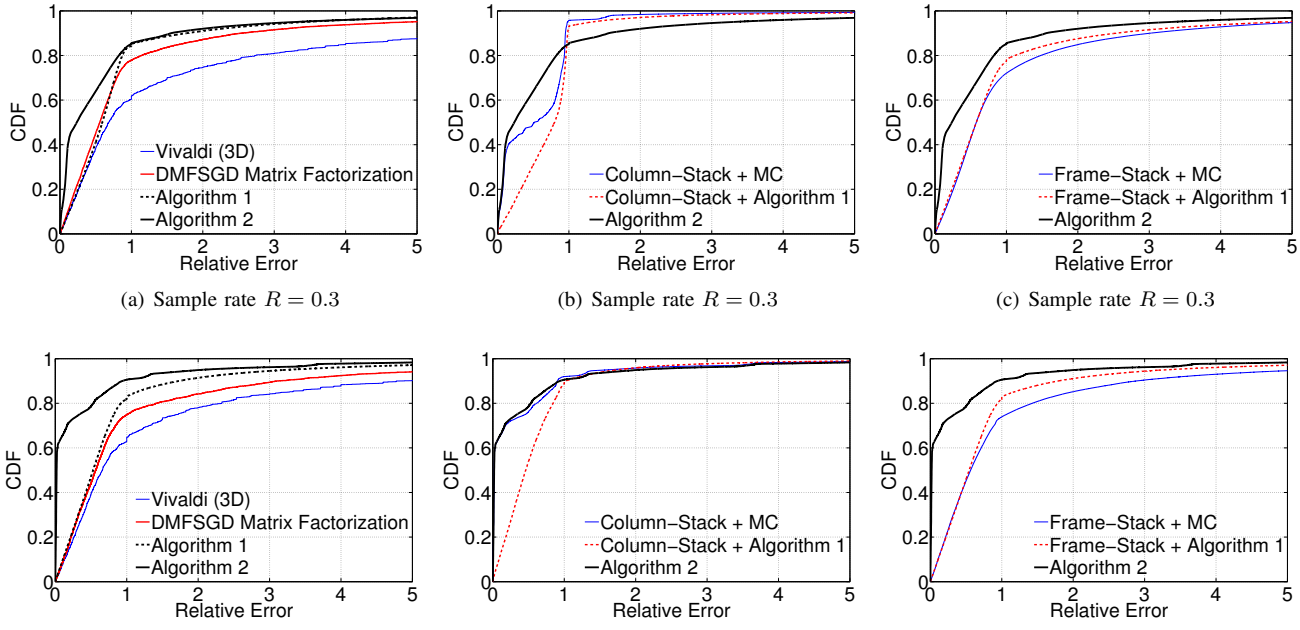
We use two different procedures to recover the missing pairs in Θ_A and Θ_B. For Θ_A, we simply apply a PD matrix completion on the *column-stacked* matrix Ω to recover the missing values. The predicted values of $M_{ij}(T)$ are denoted by $\hat{M}_{c,ij}(T)$ for $(i,j) \in \Theta_A$. The intuition is that when some past values $M_{ij}(t)$ were measured, we could directly take advantage of the low-rank property of Ω, i.e., the auto-correlation of measurements across time as shown in Fig. 9, to estimate the current missing values.

For Θ_B, we assume each $M(t)$ can be decomposed into a distance matrix $D(t)$ and network feature matrix $F(t)$, and apply a variation of the proposed Algorithm 1 (D-F Decomposition) to the *frame-stacked* matrix $\mho$ to recover the missing values. This requires us to iteratively apply Euclidean Embedding for every frame of the distance component $D(t)$ in the prediction window $t \in \{1, \ldots, T\}$, and apply a PD matrix completion to the entire frame-stacked matrix $\mho_F$ formed by all the network feature frames $F(t)$ for $t \in \{1, \ldots, T\}$. Therefore, Algorithm 1 must be extended to handle the frame-stacking (before PD matrix completion) and the *unstacking* (before Euclidean embedding).

Θ_B is treated differently, because for $(i,j) \in \Theta_B$ where no past value of $M_{ij}(t)$ was measured, the column-stacked matrix is not useful to predict $M_{ij}(T)$, since the entire row in Ω composed of $M_{ij}(t)$ for $1 \le t \le T$ is missing. In this case, the completed values of $M_{ij}(T)$ will be meaningless, because the rank of Ω will not change if we scale up or down the entire estimated row. Therefore, we utilize frame-stacking to exploit the low-rank nature of $\mho_F$, containing frame-stacked network feature matrices $F(t)$ for $1 \le t \le T$.

Algorithm 2 D-F Decomposition for 3D Sampled Data

procedure PREDICTING MISSING PAIRS IN Θ_A
  Column-stack $M(t)$ for $1 \le t \le T$ to get Ω. Perform matrix completion (4) on Ω to get the complete matrix $\hat{\Omega}$. Unstack $\hat{\Omega}$ to get $\hat{M}_c(t)$ as an estimate of each $M(t)$.
end procedure
procedure PREDICTING MISSING PAIRS IN Θ_B
  Initially, let $D^0(t) := M(t)$
  for $k = 1$ to maxIter **do**
    Perform Euclidean embedding on $D^{k-1}(t)$ to get the complete estimated matrix $\hat{D}^k(t)$ for each t.
    $$F^k_{ij}(t) := \begin{cases} \frac{M_{ij}(t)}{\hat{D}^k_{ij}(t)} & \forall (i,\ j) \notin \Theta \\ \text{unknown} & \forall (i,\ j) \in \Theta \end{cases}$$
    Frame-stack $F^k(t)$ for $1 \le t \le T$ to get $\mho^k_F$. Perform matrix completion (4) on $\mho^k_F$ to get the complete estimated matrix $\hat{\mho}^k_F$. Unstack $\hat{\mho}^k_F$ to get $\hat{F}^k(t) \in \Re^{n \times n}$.
    $$D^k_{ij}(t) := \begin{cases} \frac{M_{ij}(t)}{\hat{F}^k_{ij}(t)} & \forall (i,\ j) \notin \Theta \\ \text{unknown} & \forall (i,\ j) \in \Theta \end{cases}$$
  end for
  $\hat{M}_{f,ij}(T) := \hat{D}^{\text{maxIter}}_{ij}(T) \hat{F}^{\text{maxIter}}_{ij}(T),\ 1 \le i,\ j \le n.$
end procedure
$$\hat{M}_{ij}(T) := \begin{cases} \hat{M}_{c,ij}(T) & \forall (i,\ j) \in \Theta_A \\ \hat{M}_{f,ij}(T) & \forall (i,\ j) \in \Theta_B \\ M_{ij}(T) & \forall (i,\ j) \notin \Theta \end{cases}$$



(a) Sample rate $R = 0.3$ (b) Sample rate $R = 0.3$ (c) Sample rate $R = 0.3$

(d) Sample rate $R = 0.7$ (e) Sample rate $R = 0.7$ (f) Sample rate $R = 0.7$

Fig. 10. The CDFs of relative estimation errors on the missing values in the current frame with sample rate $R = 0.3$ and $R = 0.7$ for the Seattle dataset.

Algorithm 2 describes the detailed steps of our sparse recovery process based on 3D data.

B. Performance Evaluation

1) **Comparison with Other Algorithms:** We test our dynamic prediction algorithm on 50 3D matrices, each randomly selected from the Seattle dataset. Every 3D matrix contains $T = 3$ latency frames. The objective is to recover all the missing values in the last frame. We compare our algorithm with the static prediction methods described in the previous section. Besides, we also compare to four other methods:

- **Column-Stack+MC:** column-stack the latency matrices and perform PD matrix completion on Ω;
- **Column-Stack+Algorithm 1:** column-stack the latency matrices and perform D-F decomposition on Ω;
- **Frame-Stack+MC:** frame-stack the latency matrices and perform PD matrix completion on $\mho$;
- **Frame-Stack+Algorithm 1:** frame-stack the latency matrices and perform D-F decomposition on $\mho$;

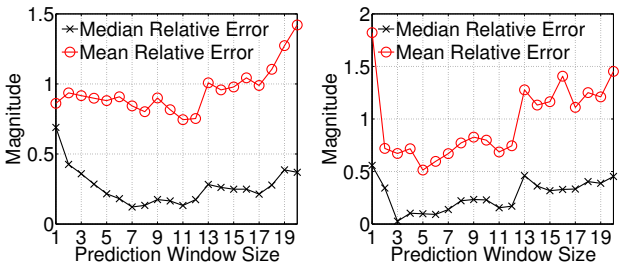
Notice that when we perform D-F decomposition on Ω or $\mho$, we perform Euclidean embedding for each unstacked latency frame individually and perform matrix completion on the big stacked network feature matrix in each iteration.

Fig. 10(a) and Fig. 10(d) compare Algorithm 2 with the static prediction algorithms. For both low and high sample rates $R = 0.3$ or $R = 0.7$, Algorithm 2 that exploits 3D sampled frames significantly outperforms the static latency prediction methods. It verifies the significant benefit of utilizing historical information, and reveals the strong correlation between different latency frames over timeline. By exploiting the low-rank structure of the column-stacked latency matrix Ω and the frame-stacked network feature matrix $\mho_F$, Algorithm 2 takes full advantage of the implicit information in the 3D data.

Fig. 10(b) and Fig. 10(e) compare Algorithm 2 with two other methods based on the column-stack operation: perform PD matrix completion on Ω, and perform D-F decomposition on Ω. Compared with the method that perform PD matrix completion on Ω, our Algorithm 2 outperforms it a lot when the sample rate is low ($R = 0.3$). The improvement is due to the different treatment to latencies for node pairs $(i,\ j) \in \Theta_B$ in our algorithm. When the sample rate is high ($R = 0.7$), the difference between Algorithm 2 and the method that performs PD matrix completion on Ω is tiny. Because the proportion of node pairs $(i,\ j) \in \Theta_B$ will be small if sample rate is high. For the method that performs D-F decomposition on Ω, it is even worse than performing PD matrix completion directly on Ω. This reveals the fact that we can benefit more from historical values of M_{ij} when they are available rather than using network condition correlations between different nodes for estimation.

Fig. 10(c) and Fig. 10(f) compare Algorithm 2 with two other methods based on the frame-stack operation: performing PD matrix completion on $\mho$, and performing D-F decomposition on $\mho$. As we can see, our algorithm outperforms both of them at both high ($R = 0.7$) and low ($R = 0.3$) sample rates. Furthermore, compared with performing PD matrix completion on $\mho$, the effect of performing D-F decomposition on $\mho$ is better, which again implies that utilizing the low-rank structure of the network feature matrix is more reasonable than utilizing the low-rank property of the original latency matrices.

Through all the comparisons above, we show the benefits of incorporating historical latency frames and prove the necessity of different treatments to unknown node pairs $(i,\ j) \in \Theta_A$ and $(i,\ j) \in \Theta_B$, i.e., the column-stack operation is suitable for node pairs $(i,\ j) \in \Theta_A$ and the frame-stack operation is better for node pairs $(i,\ j) \in \Theta_B$. It is shown that the combined



(a) Sample rate $R = 0.3$ (b) Sample rate $R = 0.7$

Fig. 11. Influence of the prediction window T for the Seattle dataset.

strategy in our hybrid Algorithm 2 is optimal.

2) **Impact of Prediction Window T:** Fig. 11 shows how the median and mean relative estimation errors for missing values in frame T vary when the prediction window T increases. We make two interesting observation. *First*, the best performance is achieved by $T = 3$ when the sample rate is 0.7, but is achieved by $T = 7$ when the sample rate is 0.3. *Second*, the prediction errors increase if we add more frames. When the sample rate R is high, a few recent frames are enough to predict the current missing latencies. However, when R is low, the latency between each pair of nodes is less frequently measured, and thus more historical frames are needed to recover the current latencies. However, once we have obtained enough information from some historical frames, adding more frames will hurt performance, since the rank of the column-stacked matrix will increase, making it harder to complete the stacked matrix with low error.

VI. Concluding Remarks

Based on measurements collected from the Seattle network that consists of user-donated personal devices, in this paper, we study the new challenges in estimating the less stable and time-varying latencies in personal device networks. We propose the distance-feature decomposition algorithm that avoids the defects of both Euclidean embedding and matrix factorization. By decomposing the network latency matrix into a distance matrix and a network feature matrix, our approach is able to capture the underlying geographical distance as well as varying network conditions among the nodes. To predict changing latencies, we further formulate the dynamic network latency estimation problem that aims to predict the current missing latencies based on frames of incomplete latency matrices collected in the past, and extend our distance-feature decomposition algorithm for such 3D sampled data, with the aid of a hybrid matrix transformation scheme. Extensive evaluation based on both Seattle and PlanetLab data shows that our algorithms outperform state-of-the-art network embedding algorithms (e.g., high-dimensional Vivaldi with/without heights) and matrix factorization (e.g., DMFSGD) by a substantial margin, especially for personal device networks. The prediction accuracy is further significantly improved by exploiting the structure inherent in the 3D sampled data through the proposed hybrid dynamic estimation mechanism.

References

[1] M. Z. Shafiq, L. Ji, A. X. Liu, and J. Wang, "Characterizing and modeling internet traffic dynamics of cellular devices," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 2011.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, 2004.

[3] T. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, 2002.

[4] Y. Liao, W. Du, P. Geurts, and G. Leduc, "DMFSGD: A decentralized matrix factorization algorithm for network distance prediction," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, 2013.

[5] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild." in *NSDI*, vol. 7, 2007.

[6] Y. Chen, X. Wang, C. Shi, E. K. Lua, X. Fu, B. Deng, and X. Li, "Phoenix: A weight-based network coordinate system using matrix factorization," *Network and Service Management, IEEE Transactions on*, vol. 8, no. 4, 2011.

[7] G. Wang, B. Zhang, and T. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007.

[8] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Seattle: a platform for educational cloud computing," in *ACM SIGCSE Bulletin*, vol. 41, no. 1, 2009.

[9] B. Donnet, B. Gueye, and M. A. Kaafar, "A survey on network coordinates systems, design, and security," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 4, 2010.

[10] S. Lee, Z.-L. Zhang, S. Sahu, and D. Saha, "On suitability of euclidean embedding of internet hosts," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, 2006.

[11] Y. Mao, L. K. Saul, and J. M. Smith, "Ides: An internet distance estimation service for large networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 12, 2006.

[12] Y. Liao, P. Geurts, and G. Leduc, "Network distance prediction based on decentralized matrix factorization," in *Networking 2010*. Springer, 2010.

[13] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, 1999.

[14] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, 2009.

[15] Y. Zhang and Z. Lu, "Penalty decomposition methods for rank minimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds. Curran Associates, Inc., 2011.

[16] K. LaCurts and H. Balakrishnan, "Measurement and analysis of real-world 802.11 mesh networks," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.

[17] J. Sommers and P. Barford, "Cell vs. wifi: on the performance of metro area mobile connections," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, 2012.

[18] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proc. of the 10th international conference on Mobile systems, applications, and services*, 2012.

[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, 2003.

[20] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, no. 5, 1970.

[21] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[22] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications*, vol. 109, no. 3, 2001.