Incorporating change detection in network coordinate systems for large data transfers

Efthymia Tsamoura
Aristotle University of
Thessaloniki, Greece
etsamour@csd.auth.gr

Anastasios Gounaris
Aristotle University of
Thessaloniki, Greece
gounaria@csd.auth.gr

ABSTRACT

The performance of large scale applications, such as those enabled by service-oriented, grid and cloud technologies, heavily relies on aspects related to the network topology and latency. As such, predicting the actual communication latencies is of high interest. The current state-of-the-art solutions to the problem of estimating the latency among distributed nodes comprise algorithms that tend to build upon the notion of network coordinates (NCs). Since network conditions change continuously, NCs need to be updated very frequently and, thus, are prone to oscillations. We present a variant of the pioneer NC algorithm, called Vivaldi, which encapsulates a change detection mechanism to prohibit NCs updates unless the network conditions change significantly. The contribution of this paper is twofold: first, to assess the impact of change detection and, second, to evaluate the NC algorithms in a realistic service-based environment where real measurements refer to large data transfers, contrary to current approaches that collect feedback from much smaller data transmissions, such as pings. The evaluation shows that our variant improves both performance and stability with less overhead.

1. INTRODUCTION

Distributed computing has gained ground due to the significant computational capabilities that it provides. Modern grid and cloud computing infrastructure, through offering transparent access to a dynamic set of heterogeneous computational and storage resources, allow end users to deploy large-scale applications that have been deemed infeasible until very recently. One of the most important parameters that affect the performance of distributed applications, e.g., query processing [12], is network delay, which depends on network topology and traffic conditions. For this reason it is important to monitor and accurately estimate delays.

There exist several approaches to extract network delay data. A straightforward one is to continuously collect latency measurements among all pairs of participating hosts, as done by the *all pairs ping service* on PlanetLab; however, due to its limited scalability and its inherent drawbacks, i.e., it increases the amount of traffic that is sent over the network, this approach is not employed. More advanced methods try to predict the network delays among two arbitrary hosts utilizing information obtained from the underlying network. The methods fall into two categories: in the first one, a network delay model is built and updated utilizing end-to-end network delay measurements [8], while in the second one, a structural network model is constructed using BGP tables, traceroute and other measurements [7]. The pioneer algorithm of the first category is Vivaldi [4].

Vivaldi embeds the end-to-end measured network delays in an euclidean space, where each host is associated with network coordinates (NC), such that the distance between any two arbitrary NCs provides an estimate of the actual network delay. Vivaldi, and its extensions use measured delays that correspond to transmissions of small data sets, including ping messages. Due to the fact that the network delays continuously change during the lifetime of a distributed application, NCs are prone to oscillations. To improve stability, the work in [6] differentiates between system- and application- level NCs. System-level NCs are updated as soon as a new delay measurement becomes available, while the updates are not propagated to the applications unless a specific event has occurred, e.g., the system-level NCs have changed significantly.

This work aims to study an alternative technique for mitigating the oscillations in NCs in a real environment, which does not update the NCs at all unless the observed network delays change significantly. To this end, our technique employs a change detection algorithm. In addition, we focus on delays experienced during the transmission of larger data sets (at the orders of MBytes) through a web service interface, since such settings are common in practice. The efficiency of the proposed technique is evaluated with the help of a network delay dataset that is collected from PlanetLab ([3]). To the best of our knowledge, it is the first evaluation of NC-based algorithms using communication cost measurements referring to large transfers. Interestingly, our proposal not only performs less work because it does not update the NCs continuously and thus is characterized by improved stability, but also is more accurate.

The paper is organized as follows. Section 2 presents the related work. Section 3 presents the collected dataset. Our technique and the experimental comparison appear in Sections 4 and 5, respectively. Section 6 concludes the paper.

PCI 2013, September 19 - 21 2013, Thessaloniki, Greece

2. RELATED WORK

Our work mainly relates to the areas of (i) prediction, (ii) analysis and (iii) estimation of network delays. Before presenting the related work, we will disambiguate the terms latency, RoundTrip Time (RTT), which also called latency in the literature, and throughput. The RTT between two endpoints is defined as the total delay to transmit data from the sender to the destination endpoint plus the delay of the reply message sent by the destination endpoint to reach the sender endpoint. On the other hand, network throughput is defined as the average rate of successful data delivery over the network and is measured in bits per second, and is inversely proportional to network latency [12].

As stated in the introduction, there exists a plethora of work that aim to predict the data transmission delays among hosts without performing direct network delay measurements [8], [7]. In this work, we consider a variant of the Vivaldi algorithm that belongs to a broader category of methods that embed the network delay space into a metric space [4]. Many extensions to Vivaldi have been proposed in the literature; however, the majority of these methods assume static network delays. The evolution of network conditions is considered in [6] that proposes ways for rendering Vivaldi more stable, but still updates the NCs continuously. Our work shares the goal of [6] to improve stability but updates the NCs only when significant changes in the measured delays are detected.

In [11], [10], the network delay data is analyzed and intrinsic properties of such delays are identified. For example, the work in [11] studies the triangle inequality violation (TIV) property of the Internet delay space, while the work in [10] studies the structure of the Internet. This work does not deal with the dynamic nature of the network delays. The work in [9] deals with the issue of addressing the effects of routing changes on network delays. Our work aims to study the dynamic characteristics of network delay data collected from PlanetLab and present a NC-based prediction algorithm that is aware of the changes.

Regarding the collection of network delay measurements, one approach is to use the ping utility (e.g.,[1]). A different approach is followed by the King tool [5], which issues recursive DNS queries to measure the network delay between two DNS name servers and through this measurement it infers the actual network delay between two endpoints that are topologically close to these DNS servers. Another approach, which we follow, is to estimate latency by taking the inverse of network throughput [12].

3. PROFILES OF NETWORK DELAYS DURING LARGE DATA TRANSFERS

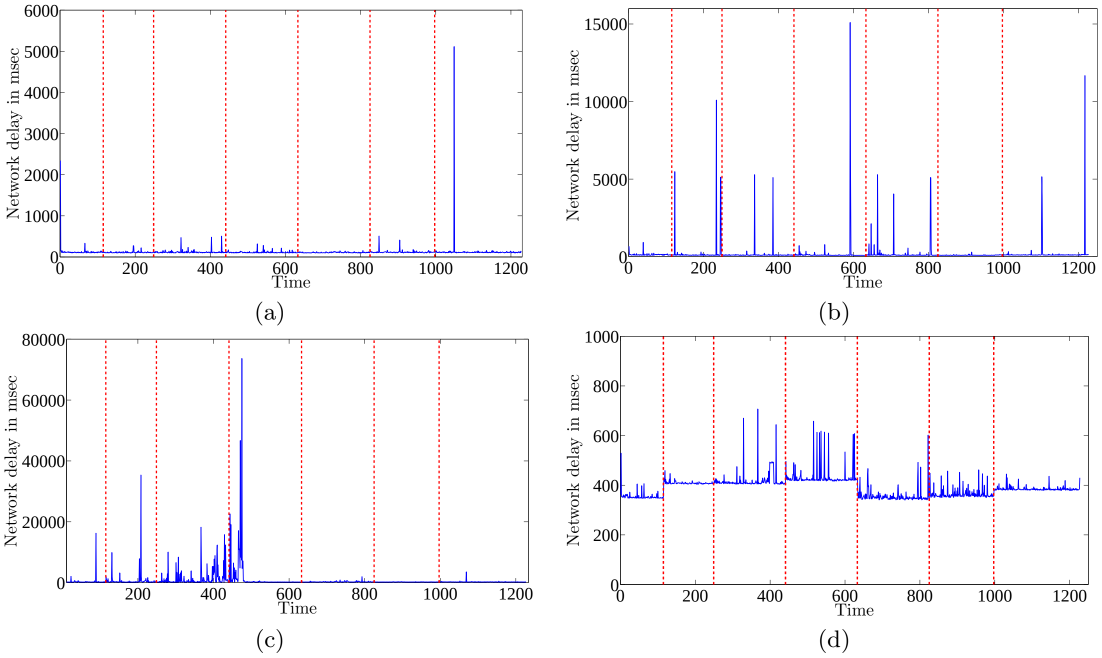
This section begins by presenting the methodology adopted to collect the latency measurements and it continues by presenting part of the most frequent patterns that the latency measurements of a link may follow and a statistical analysis of the collected measurements.

In total, three datasets were collected during a twenty-one day period between November 21 and December 19 2011 utilizing 136 randomly chosen hosts from PlanetLab EU [3]. The approach that we employed to create the three datasets is presented below. First, we created a set of all host pairs and then divided this host pair set into three subsets by randomly assigning each host pair to one of the subsets (and thus three datasets were created). During one day, we collected network latency measurements only for one host pair subset utilizing the following approach. For each host pair (h_1, h_2), we were initiating a synchronous transmission of 4 MB from h_1 to h_2 every 5 minutes. We chose to send 4 MB of data between the hosts in order to better distinguish the network delays among different host pairs and monitor those delays for large data transmissions. Each network latency measurement for the link connecting h_1 with h_2 corresponds to the total time spent in order to send the 4 MB of data from h_1 to h_2 plus the time to send an acknowledgment message of few bytes from h_2 to h_1.

The data transmissions were performed through web services. In particular, in each host, we deployed a web service with two methods, namely $send(String\ block, String\ destHost)$ and $get(String\ block)$. The first one is responsible for sending the string $block$ to the host named $destHost$. It performs that task by synchronously invoking the get method of the web service that is hosted on $destHost$ host with parameter $block$. Note that we have set $block$ to a string of 4 MB size. The web services are developed using the JAX-WS 2.0 API and the Apache Tomcat 6.0.9 Servlet/JSP Container.

We are interested in latency measurements that are synchronized (at least roughly) so that they can reveal the global network conditions at a given time. The final datasets that we used include 3155 host pairs in total (952, 942, 1261 from the first, the second and the third dataset, respectively) with 1220 latency measurements each, where the $\tau-th$ measurement of any of these pairs is taken roughly at the same time. The reason that we did not use all the measurements taken is that due to hardware failures or memory leaks some Tomcat servers became unavailable during the measurement collection procedure. In order to avoid cases where the collected latency measurements from different links are not synchronized, we discarded the measurement collection procedure for all host pairs belonging to the same subset at the time at least one host could not send or receive data through its web service. Apart from that, some hosts were unavailable for one or more days during the measurement collection procedure and, as a consequence, we could not collect latency measurements for the links which had these hosts as endpoints. The latency measurements that are related with those hosts were excluded from the final datasets.

In the following, we show some representative patterns that the observed latency measurements of a link may exhibit along time. The pattern of Figure 1(a) has some latency spikes of short duration and of short amplitude (less than 1 *sec*) and these spikes are somehow uniformly distributed in time. The dashed vertical lines separate latency measurements collected during different days. In contrast to the pattern of Figure 1(a), the pattern shown in Figure 1(b) has a relatively higher number of latency spikes. These spikes have short duration, but larger amplitude (e.g., more than 5 *sec*). The pattern shown in Figure 1(c) comprises a different paradigm, where latency measurements have significantly larger duration and amplitude from the ones depicted on the two latter figures, e.g., some latency measurements have more than 50 *sec* amplitude. Continuing our discussion, Figure 1(d) shows an example of a link, where the measured network delays do not only have spikes, but the average network delay changes between days. For example, in Figure 1(d), the latency measurements that are collected during the second, the third and the fourth days exhibit a drift of approximately 50 *msec*.



(a)

(b)

(c)

(d)

Figure 1: Typical patterns that of the observed latency measurements of a link.

Figure 2(a) shows the cumulative distribution of the measured network latencies. In particular, for each pair of hosts, we took the 95% percentile out of the distribution formed by the measured latencies. The x-axis of Figure 2(a) shows the 95% percentiles that are collected from 3155 host pairs. The first observation is that the 90% of the host pairs sends 4MB of data within less than 5 *sec.*

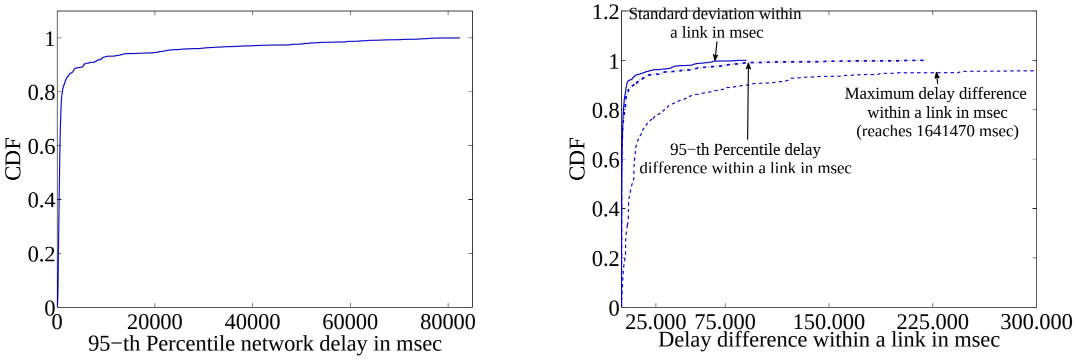
Next we study the changes that may happen within one network link along time. Figure 2(b) shows the cumulative distribution of the standard deviation of the latency measurements within a link. We can see that for the 90% of links, the standard deviation is less than 3.9 *sec.* For each pair of hosts, we also examined the maximum difference between the collected latency measurements. Figure 2(b) also shows the cumulative distribution of the maximum differences between the measurements of the links. We can see that the 90% of links may have a maximum delay difference of no more than 89 *sec.* Finally, for each pair of hosts, we took the 95% percentile out of the distribution formed by the differences of the collected latency measurements for the corresponding pair. Figure 2(b) shows the cumulative distribution of the collected 95% percentiles. We can see that for the 90% of the profiled links, the 95% percentile of the delay differences is more than 10 *sec.* From the cumulative distribution functions of Figure 2(b) we can conclude that the latency fluctuations that may happen within a link are not negligible and can be as high as two orders of magnitude large.

4. CHANGE-AWARE NC ESTIMATES

In this section, we present our proposal which extends the original Vivaldi algorithms, as well as, we provide brief descriptions of the original Vivaldi algorithm and its variant presented in [6]. We call the proposed Vivaldi variant CA-Vivaldi.

In Vivaldi, each of the remote hosts is embedded in an euclidian metric space, where each host maintains its coordinate and a confidence value associated with it [4]. Network embedding is done utilizing latency measurements collected among the hosts and the actual network delays can be estimated through the distances among the hosts' coordinates. During the embedding procedure, each host updates its coordinate every time a new latency value becomes available. This latency value corresponds to the actual network delay from the specific host to one of its neighbors and can be taken through any of the methods described in Section 2. Let $l^{\tau}_{i,j}, \tau > 0$, be the network delay between the $i-th$ host and its neighbor $j-th$ at time τ. In brief, when a latency measurement $l^{\tau}_{i,j}$ appears on node i, the latter updates its coordinates employing the following steps. First, it computes the error $\epsilon^{\tau}_{i,j}$ of this measurement as $\epsilon^{\tau}_{i,j} = l^{\tau}_{i,j} - ||\mathbf{x}^{\tau-1}_{i} - \mathbf{x}^{\tau-1}_{j}||$, where $\mathbf{x}^{\tau-1}_{i}$ and $\mathbf{x}^{\tau-1}_{j}$ are the coordinates of nodes i and i, respectively, at time $\tau - 1 > 0$. Second, it computes the direction along which node i has to be moved (according to the newly presented measurement) and, finally, it moves a small step in this direction. The neighbors of each node can be either chosen randomly, or, in the extreme case, can include all of the remaining network hosts. Full details regarding Vivaldi can be found in [4].

In order to deal with fluctuations in the observed network latencies, which lead to significant oscillations of the NCs, the work in [6] introduced a smoothing step to the Vivaldi algorithm. According to [6], a host does not update its coordinate utilizing the exact new latency measurements, but employs a percentile of the latency measurements that have been presented to this host in the recent past. In order to employ this technique, also known as Moving Percentile (MP) filtering, each host must maintain for each of its neighbors a sliding window of historic latency measurements. For example, let $W_{i,j}$ be the sliding window, which stores a portion of the most recent latency measurements presented to host i. Then, every time a latency measurement is supplied



(a)

(b)

Figure 2: (a) Cumulative distribution of the 95% percentile values taken out of the distributions formed by the measured latencies of the profiled links. (b) Cumulative distributions for the standard deviation, the maximum delay difference and the $95-th$ percentile of differences of the latency measurements within a link.

to host i, the latter updates its coordinate using a percentile from $W_{i,j}$.

The rationale of the proposed variant can be summarized as follows; only when the network delay of a link changes along time, the coordinates of the endpoint hosts are updated. In other words, the host i updates its coordinate only when the supplied $l^{\tau}_{i,j}$ latency measurements show a drift. Detecting changes on the network delay of a link can be done using a variety of change detection algorithms (e.g., [2]). However, in this work, we employ a simple and computationally efficient approach. It is assumed that the network delay of a link changes when two consecutive latency measurements differ significantly. Following this rationale, a host i updates its coordinate at time τ only when $l^{\tau}_{i,j}$ differs from the most recent latency measurement $l^{\kappa}_{i,j}$, $0 < \kappa < \tau$, that the host i has used to update its coordinate with respect to its neighbor j, i.e., $|l^{\kappa}_{i,j} - l^{\tau}_{i,j}|/|l^{\tau}_{i,j}| > \theta, \theta > 0$.

The difference between the proposed Vivaldi variant and the two other techniques is that the former updates a host's coordinate only when a significant change in the measured network latencies is detected. As will be shown in Section 5, this change increases both the accuracy and the stability of the resulting network embedding, decreasing at the same time the total running time because of fewer NC updates.

5. EXPERIMENTAL EVALUATION

The goal of this section is to experimentally compare the proposed Vivaldi variant (CA-Vivaldi) along with the Vivaldi algorithm and the Vivaldi variant proposed in [6], also called MP-Vivaldi. The comparison is done in terms of accuracy and NC stability. We also study how the size of the neighborhood of each node, as well as, other parameters of the Vivaldi variants affect the above performance criteria. The performance of the considered algorithms for finding NCs is evaluated both utilizing the collected raw latency measurements and also a smoothed version of them. The purpose of smoothing the collected latency measurements was to eliminate the latency spikes of unit-width from the ground truth, assuming that these spikes do not reflect a highly loaded network but are due to highly loaded hosts. More details regarding smoothing the collected latency measurements are given at the end of this section. The main experimental conclusions are: (i) the proposed Vivaldi variant produces more accurate and more stable NCs than Vivaldi, (ii) the proposed variant is faster than Vivaldi, (iii) the Vivaldi variant proposed in [6] is less accurate (up to an order of magnitude) and less stable (up to an order of magnitude) than our variant and Vivaldi itself.

The following summarize the parameter values that we considered when we conducted the experiments. The default dimensions of the NCs are three. We also considered that each host might have a neighborhood of size two or eight, i.e., the NC are updated based on the measurement of either two or eight links. For parameters $|W|$ and α, which correspond to the size of the sliding window and its percentile when the hosts employ MP-Vivaldi, respectively, we considered the following assignments: $W = 4$ and $\alpha = 0.25$, $W = 4$ and $\alpha = 0.5$, $W = 6$ and $\alpha = 0.5$, $W = 8$ and $\alpha = 0.125$ motivated by the results in [6]. Regarding the proposed variant, the possible values that we consider for the change detection threshold parameter θ are 0.05, 0.1 and 0.25, respectively.

We conducted experiments utilizing all the three collected datasets, as well as, considering 5-dimensional NCs. However, since the derived experimental conclusions were similar, we only present the experimental results for the first dataset for 3 dimensions. Recall that the first dataset consists of 952 host pairs and for each host pair we collected latency measurements at $\tau = 1, \ldots, 1220$ timepoints.

Regarding the performance criteria of interest, accuracy is measured through the relative error measure. The relative error for the network link connecting the hosts i and j at time τ is given by $r^{\tau}_{i,j} = |||\mathbf{x}^{\tau}_i - \mathbf{x}^{\tau}_j|| - l^{\tau}_{i,j}|/l^{\tau}_{i,j}$. In this formula, $\mathbf{x}^{\tau}_i$ and $\mathbf{x}^{\tau}_j$ correspond to the coordinates of the hosts i and j, respectively, after the latency measurements that are taken up to and including the $\tau - th$ time point have been presented to the hosts and $l^{\tau}_{i,j}$ refers to the latency of the link connecting the hosts i and j measured at time τ. Stability refers to the degree of change of the positions of the found NC along time. Similar to [6], the stability of a host is quantified using the NC movement measure, which is computed at time $\tau > 2$ by $||\mathbf{x}^{\tau}_i - \mathbf{x}^{\tau-1}_i||$. As the collected latency measurements are in *msec*, the NC movement for a host at time τ is also measured in *msec*. The experimental

Approach \ Criterion	Median RE	Median TM
CA-Vivaldi (2 neighbors per host)		
(θ=005)	0.48	96.67
(θ=01)	0.48	100.89
(θ=025)	0.46	75.34
MP-Vivaldi (2 neighbors per host)		
($\|W\|$=4,α=0.25)	8	419.70
($\|W\|$=4,α=0.5)	3	270.8
($\|W\|$=6,α=0.5)	9	488.4
($\|W\|$=8,α=0.125)	4	192.59
Vivaldi (2 neighbors per host)		
	0.48	111.53

Approach \ Criterion	Median RE	Median TM
CA-Vivaldi (8 neighbors per host)		
(θ=005)	0.32	380.46
(θ=01)	0.40	365.87
(θ=025)	0.34	289.24
MP-Vivaldi (8 neighbors per host)		
($\|W\|$=4,α=0.25)	6	2318.40
($\|W\|$=4,α=0.5)	7	3644.50
($\|W\|$=6,α=0.5)	18	3744.75
($\|W\|$=8,α=0.125)	9	4298.68
Vivaldi (8 neighbors per host)		
	0.40	387.34

Approach \ Criterion	Median RE	Median TM
CA-Vivaldi (2 neighbors per host)		
(θ=005)	0.38	58.14
(θ=01)	0.40	101.49
(θ=025)	0.36	62.60
MP-Vivaldi (2 neighbors per host)		
($\|W\|$=4,α=0.25)	4	203.5
($\|W\|$=4,α=0.5)	20	479.73
($\|W\|$=6,α=0.5)	25	492.02
($\|W\|$=8,α=0.125)	6	216.84
Vivaldi (2 neighbors per host)		
	0.40	72.35

Approach \ Criterion	Median RE	Median TM
CA-Vivaldi (8 neighbors per host)		
(θ=005)	0.28	252.69
(θ=01)	0.32	209.39
(θ=025)	0.26	391.62
MP-Vivaldi (8 neighbors per host)		
($\|W\|$=4,α=0.25)	87	9697.34
($\|W\|$=4,α=0.5)	24	4832.85
($\|W\|$=6,α=0.5)	23	7671.64
($\|W\|$=8,α=0.125)	97	5356.46
Vivaldi (8 neighbors per host)		
	0.36	440.89

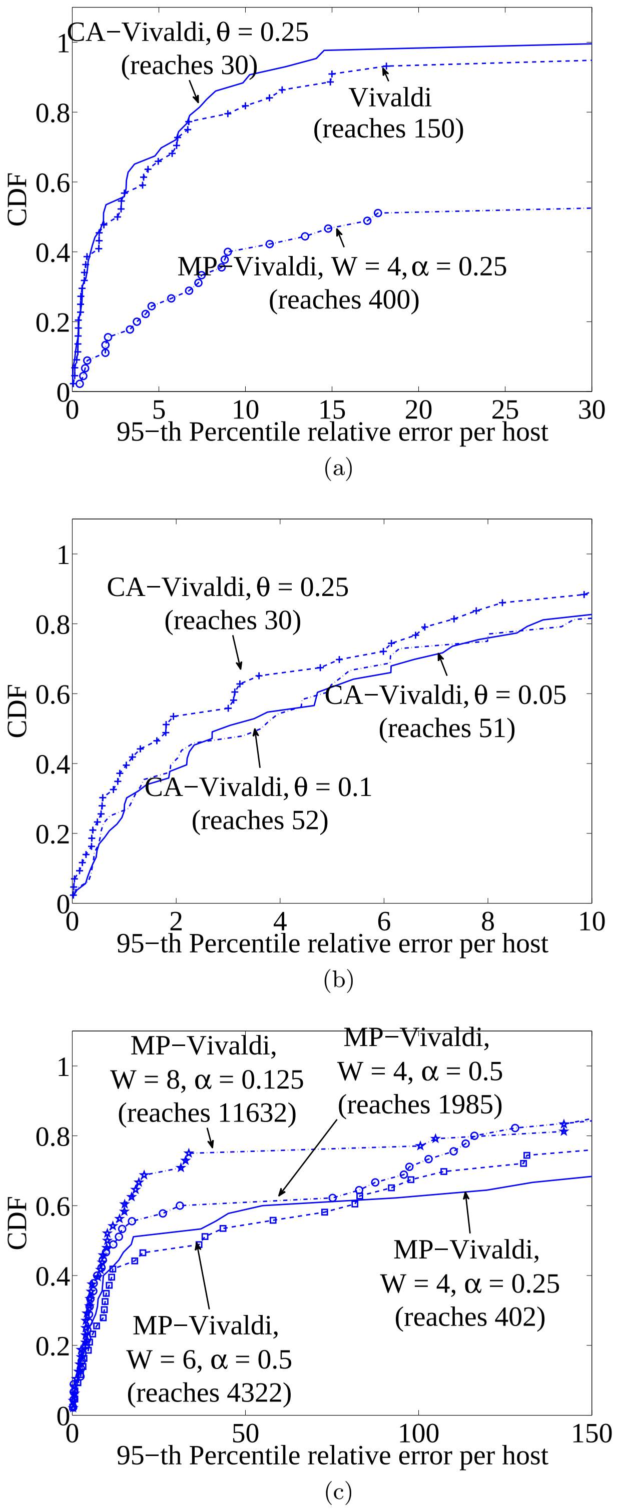
Table 1: Median relative error (RE) and median total movement (TM) of the proposed variant, Vivaldi and MP-Vivaldi for different parameter assignments and different neighborhood sizes. The results are aggregated along $\tau = 1, \ldots, 1220$. Upper-part: The collected latency measurements are utilized as ground truth. Lower-part: The collected latency measurements are pre-processed to remove spikes of unit-width.

results are summarized in Table 1.

In the first experiment, we compare the accuracy of the three alternatives for finding NC assuming a neighborhood of size two for every host. The rest of the parameters of the Vivaldi variants are set to $\theta = 0.25$, and $W = 4$, $\alpha = 0.25$, respectively. Following the presentation style adopted in [6], Figure 3(a) shows the cumulative distribution of the $95-th$ percentile relative error per host after all of the 1220 collected latency measurements have been supplied to each host pair. In particular, for each host i, we computed the relative errors for the links having the specific host as their source endpoint, $r_{i,j}^{\tau=1220}$, and then we took the $95-th$ percentile out of these relative errors. The set of the collected $95-th$ percentiles of relative errors that are computed for every host comprise the data of the x-axis. From this plot, we can observe that the accuracy of the proposed variant is higher than the accuracy of the other two alternatives. The maximum $95-th$ percentile relative error of the proposed variant is approximately 30 (the latency of a link that is estimated using NC is 30 times higher than the actual latency), while it is 130 and 400, for Vivaldi and for the MP-Vivaldi, respectively. Another observation is that all alternatives may be inaccurate by orders of magnitude. The cumulative distributions of the $95-th$ percentile relative error per host after the 50% and the 75% of the collected latency measurements have been supplied to each host pair, respectively, are also studied. Due to space limitations we will only mention that these cumulative distribution functions also show that the proposed variant is more accurate than the other two alternatives and the results do not change significantly.

In order to study how does the parameter θ affect the performance of the proposed variant, we repeated the previous experiment only for the proposed variant changing θ to 0.05 and 0.1, respectively, keeping the size of the neighborhood the same. Recall that parameter θ affects the sensitivity of the variant in delay fluctuations; as θ increases the variant becomes more robust to temporal delay fluctuations. Figure 3(b) summarizes the results of this experiment. We can see that the accuracy of the proposed variant increases as parameter θ increases, as well, while the the maximum $95-th$ percentile relative error of the proposed variant is lower than that of Vivaldi independently of θ. For example, the maximum $95-th$ percentile relative error of the proposed variant is approximately 50 for θ 0.05 or 0.1 (three times lower than the corresponding value when employing Vivaldi). Similar to Figure 3(b), Figure 3(c) shows the results after executing MP-Vivaldi for the following W, α parameter combinations: $W = 4, \alpha = 0.25$, $W = 4, \alpha = 0.5$, $W = 6, \alpha = 0.5$, $W = 8, \alpha = 0.125$. We can see that its accuracy changes significantly for different selections of the parameters W and α, however, it is always worse than that of the proposed variant and Vivaldi. Especially, in some cases, the maximum $95-th$ percentile relative error exceeds 1000.

We have summarized in the upper-left part of Table 1 the median relative errors of any of the considered three alternatives for finding NCs. These errors are computed taking into account all of the links at any timepoint $\tau = 1, \ldots, 1220$, i.e., each cell in the second column of the upper-left part of Table 1 is computed as $median_{\forall i,j,i\neq j,\tau\in\{1,\ldots,1220\}}\{r^{\tau}_{i,j}\}$. This table shows that the median relative error of the proposed variant is lower than that of Vivaldi, while the accuracy of MP-Vivaldi is approximately an order of magnitude worse than that of the other two solutions.



(a)

(b)

(c)

Figure 3: $95-th$ Percentile relative error per host after all of the latency measurements taken at $\tau = 1, \ldots, 1220$ have been supplied to each host pair. The size of each host's neighborhood is 2.

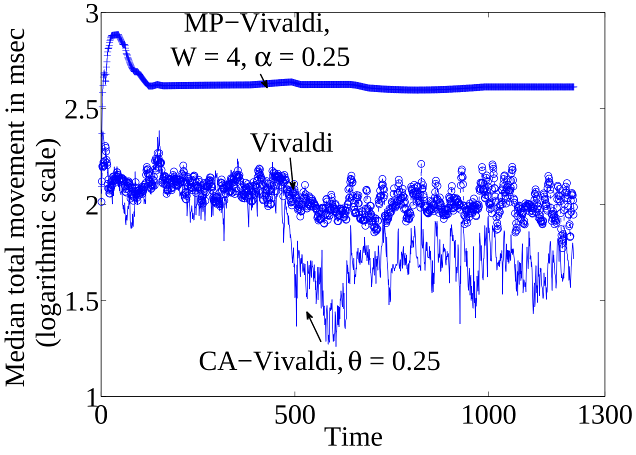
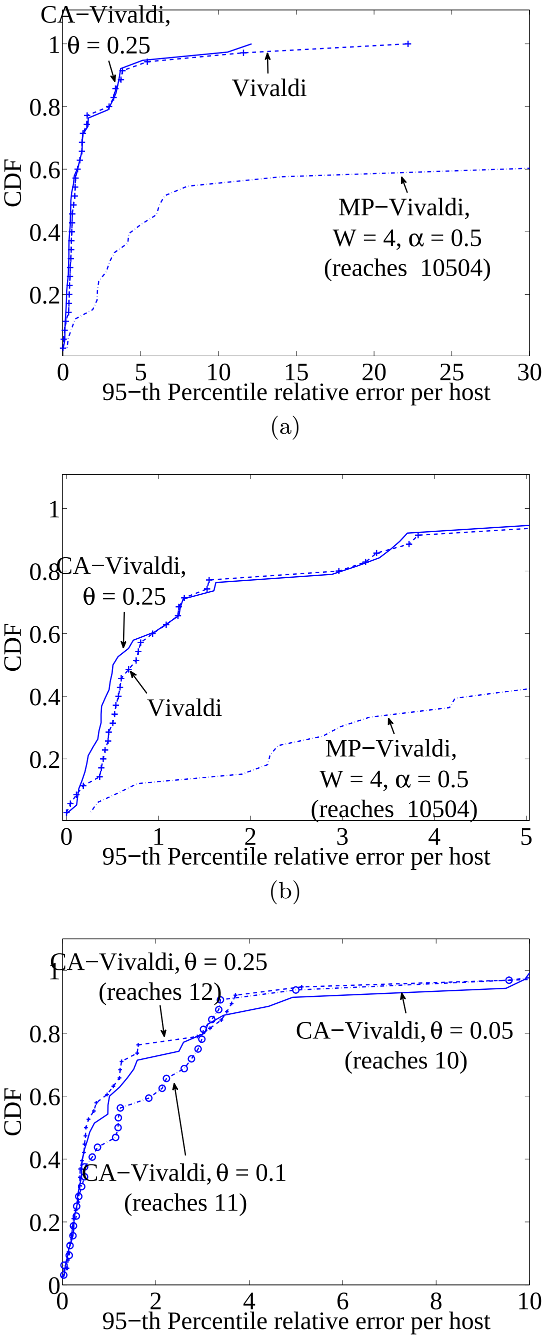


Figure 4: Median total movement of all hosts along $\tau = 1, \ldots, 1220$. The size of each host's neighborhood is 2.

Apart from accuracy, we also study the stability of the proposed variant. Figure 4 shows the median total movement of all hosts along time. In particular, at any timepoint $\tau > 0$ we computed the total movement of a host i that us incurred during updating its coordinate for the latency measurements $l^{\tau}_{i,j}$, $i \neq j$. Figure 4 is created after taking the median total movement of the hosts at a specific timepoint $\tau > 1$. From this figure, we can see that the proposed variant is more stable than Vivaldi and the MP-Vivaldi variant. Especially, as time passes by, the proposed variant becomes more stable, which is not the case for Vivaldi and the MP-Vivaldi variant.

The third column in upper-left part of Table 1 shows the median total movement of the hosts for the studied Vivaldi alternatives for different settings of their parameters aggregating along $\tau = 1, \ldots, 1220$. For example, the value of the cell that contains the median total movement of the hosts when employing Vivaldi is the median of the values of the line shown in Figure 4 that corresponds to Vivaldi. This table further supports our claim that the proposed variant is more stable than Vivaldi and its MP-variant.

We repeat the above experiments increasing the size of the neighborhood from 2 to 8. The results are shown in Figures 5(a)-(c) and 6 and the upper-right part of Table 1. From these figures we can conclude the following regarding accuracy: (i) the accuracy both of the proposed Vivaldi variant and Vivaldi is higher comparatively with their accuracy when the neighborhood size was two, (ii) the accuracy of the proposed Vivaldi variant is higher than that of Vivaldi for $\theta = 0.25$, while it is approximately the same for $\theta = 0.05$ and $\theta = 0.1$, and (iii) the accuracy of MP-Vivaldi is worse than that of the aforementioned algorithms. An interesting phenomenon that we observed for MP-Vivaldi is that its accuracy is worse than that when the size of each host's neighborhood was 2, e.g., see the median relative error columns from the upper-left and the upper-right parts of Table 1 for $|W| = 4, \alpha = 0.5$, $|W| = 6, \alpha = 0.5$ and $|W| = 8, \alpha = 0.125$. Regarding stability, the total movements of the hosts increase, which is expected as the size of the neighborhood of every host is bigger. As it was the case for a two-size neighborhood, the proposed variant is more stable than Vivaldi and MP-Vivaldi.



(a)

(b)

(c)

Figure 5: $95-th$ **Percentile relative error per host after all of the latency measurements taken at** $\tau = 1, \ldots, 1220$ **have been supplied to each host pair. The size of each host's neighborhood is 8. Figure (b) has been created after zooming in Figure (a).**

We close this section by presenting the results derived after feeding the algorithms with a smoothed version of the collected latency samples. As discussed in Section 3, the collected latency data comprise many abrupt and of high amplitude latency measurements. Since we collected this dataset through utilizing web services, part of these latency spikes may be a byproduct of overloaded Tomcat servers, instead of a highly loaded network. We made the assumption that latency spikes of unit-width, are due to highly loaded endpoints and, as such, must be removed from the ground truth. The task of smoothing is performed as follows: for the latency measurements $l^{\tau}_{i,j}$ of a host pair i, j we substituted $l^{\tau}_{i,j}, \tau > 2$, with the average of the values $l^{\tau-1}_{i,j}$ and $l^{\tau+1}_{i,j}$ if $l^{\tau}_{i,j}$ is five times higher both from $l^{\tau-1}_{i,j}$ and $l^{\tau+1}_{i,j}$. The results for a two-size and eight-size neighborhood per host are summarized in the lower-left and lower-right parts of Table 1, respectively. From these sub-tables we can see that the accuracy of both the proposed Vivaldi variant and Vivaldi becomes higher, while the proposed Vivaldi variant is more accurate relatively with the other two alternatives. Similar conclusions can be drawn for stability.

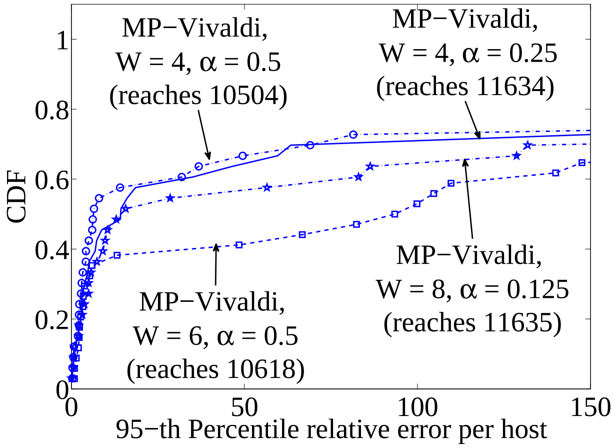


Figure 6: $95-th$ **Percentile relative error per host after all of the latency measurements taken at** $\tau = 1, \ldots, 1220$ **have been supplied to each host pair. The size of each host's neighborhood is 8.**

6. CONCLUSIONS

Predicting the actual delays among the endpoints of a network is a problem of high importance in modern distributed applications. This paper proposes a novel solution for predicting the actual network latencies building upon the state-of-the-art Vivaldi algorithm, which relies on the notion of NCs. The proposed Vivaldi variant is guided by changes in the underlying network; as such, it updates the hosts' NCs only when the network latencies change significantly. The experimental comparison shows that the proposed variant improves both accuracy and NC stability. Another contribution of this paper is that, in contrast to existing work, the evaluation is performed after collecting real latency measurements corresponding to large data transfers from PlanetLab; and as such is of interest for many real-world distributed applications.

7. ACKNOWLEDGMENTS

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program ”Education and Lifelong Learning” of the National Strategic Reference Framework

(NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

8. REFERENCES

[1] Skitter, http://www.caida.org/tools/measurement/skitter/.

[2] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM ICDM*, pages 443–448, 2007.

[3] D. E. Culler. Planetlab: An open, community-driven infrastructure for experimental planetary-scale services. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, pages 15–26, 2004.

[5] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. In *IMW*, pages 5–18, 2002.

[6] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *ICDCS*, pages 74–83, 2006.

[7] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: an information plane for distributed services. In *OSDI*, pages 367–380, 2006.

[8] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFCOM*, pages 170–179, 2002.

[9] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu. Understanding network delay changes caused by routing events. In *SIGMETRICS*, pages 73–84, 2007.

[10] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella. On the treeness of internet latency and bandwidth. In *SIGMETRICS*, pages 61–72, 2009.

[11] G. Wang, B. Zhang, and T. S. E. Ng. Towards network triangle inequality violation aware distributed systems. In *IMC*, pages 175–188, 2007.

[12] X. Wang, R. C. Burns, A. Terzis, and A. Deshpande. Network-aware join processing in global-scale database federations. In *ICDE*, pages 586–595, 2008.