NetICE9: A Stable Landmark-Less Network Positioning System

Dragan Milić, Torsten Braun
Institute of Informatics and Applied Mathematics
University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
email: {milic|braun}@iam.unibe.ch phone: +41 31 511 {2633|2631}

***Abstract*—We propose NetICE9, a novel landmark-less method for embedding RTTs into virtual spaces. NetICE9 is inspired by VIVALDI, the most commonly used landmark-less approach. VIVALDI chooses its neighbors randomly and optimizes only towards one neighbor at a time. With NetICE9, we propose a solution to those drawbacks. NetICE9 significantly improves both the stability of the simulation and the precision of how RTTs are embedded into a virtual space. Our evaluation based on RTTs measured in the Internet show that NetICE9 significantly outperforms VIVALDI in terms of stability and precision of RTT prediction. NetICE9 improves RTT prediction also in comparison to GNP.**

I. Introduction

RTT (round trip time) is one of the most important properties of Internet communication due to its role as a limiting factor for the effective bandwidth of TCP connections [5] and its impact on delay-sensitive real-time applications. The idea behind an RTT prediction scheme is not to measure the RTT between each pair of hosts in the network, but to obtain more or less precise estimates of RTTs between all hosts in the network based on a small number of measured RTTs. Such an RTT prediction scheme can be used to optimize overlay network structures, to choose the nearest data source for file transfers, or to find an optimal route based on predefined QoS parameters.

Embedding RTTs into virtual spaces to obtain an efficient RTT prediction scheme has been a research topic for almost a decade [2], [3], [8], [13], [14], [17]–[19]. Numerous methods have been proposed to achieve this goal. All methods proposed can be classified as either landmark-based or landmark-less. The former use so-called landmarks, a set of designated hosts within the network. Those landmarks usually measure RTTs among each other and use this information to position themselves within a virtual space. All other hosts in the network measure RTTs to the landmarks and, in the virtual space, position themselves relative to the landmarks according to the measured distances. Landmark-less approaches, on the other hand, usually perform a distributed simulation of the physical system. Such a simulation incrementally decreases the total error for embedding of RTTs in the virtual space and converges toward an optimal solution. Landmark-based approaches are usually more stable than landmark-less ones, in the sense that the host positions in the virtual space are more stable. Landmark-based approaches usually have limited scalability and fault tolerance due to the fact that they heavily depend on landmarks as central components. Landmark-less approaches are usually completely distributed without any central components. This makes them very resilient to host failures and network outages. Landmark-less approaches are usually quite sensitive to violations of assumptions underlying the simulation. For example, one of the most commonly made assumptions is that the triangle inequality holds for the measured RTTs. As shown in [7], [9], [21], however, this is not always the case in the Internet. Such violations lead to undesired effects such as permanent oscillations of host positions in the simulation; or even worse, rotation and translation of the whole system within the virtual space. Those effects lead to unstable host positions, making them useful only for a limited time, even if there were no changes in the underlying network.

We propose NetICE9, an improvement of the VIVALDI [3] approach. Like VIVALDI, NetICE9 is a distributed simulation of a physical system. Each host calculates its position in the virtual space without knowing the whole system. The only information each host has is about its neighbors. Unlike VIVALDI, which simulates a system of hosts connected by springs, NetICE9 is a simulation of a crystallization process. This means that NetICE9 simulates the creation of a crystal structure in a virtual space, where each host corresponds to one atom within this structure. The forces determining the relative positions of the atoms are proportional to the difference between measured RTTs and distances in the virtual space representing them. In a crystal, those forces are in balance. Essentially, each atom is positioned in such a way that the forces of repulsion from and attraction to its neighbors are in balance.

In this paper, we also present results of our simulations based on RTTs measured in the Internet [6], [20]. Comparing the performance of NetICE9 and VIVALDI with different data sets, our simulations clearly show that our approach (NetICE9) yields significantly better results in terms of both the stability of host positions and the precision of the RTT embedding.

After discussing related work in Section II, Section III identifies the sources of instability of VIVALDI. Section IV introduces NetICE9 and Section V presents our evaluation methodology and simulation results. Section VI summarizes the proposed approach and evaluation results.

II. Related Work

VIVALDI [3] is a distributed simulation of a physical system, which consists of hosts positioned in a virtual n-dimensional Euclidean space. Every pair of hosts is connected by a virtual spring. The force stored in this spring is proportional to the distance between two hosts within the virtual space and the RTT measured between them. Each host periodically corrects its position within the virtual space by randomly choosing a neighbor, measuring the RTT to it and adjusting its position by moving towards or away from the chosen neighbor to lessen the force stored by the virtual spring (i.e. the host adjusts its position to reduce the embedding error). In order to reduce oscillations of the whole system, VIVALDI proposed that each step of an optimization (moving towards or away from the neighbor in order to relax the force of the spring) is not performed fully. Instead, the spring is relaxed only partially at each step. As an estimate of how much the spring should be relaxed, VIVALDI proposed using estimates of the embedding error of a host performing the optimization and its neighbor. This estimate of the embedding error is calculated at each step of the optimization. The goal is to prefer relaxing the springs to neighbors who have better estimates of their position. In addition to a host's position in the virtual space, VIVALDI also proposed the introduction of a non-negative component, the so-called "height vector". This component represents the non-metric part of the host position. The purpose of the height vector is to further increase prediction precision by more or less representing the distance of the host to the backbone of the Internet. Due to its simplicity (no sophisticated function minimization is needed) and because it does not need any infrastructure (landmarks or similar) VIVALDI has become very popular for different peer-to-peer applications [1], [4], [16].

In GNP [13], [14], m hosts are chosen as landmarks (denoted by $\mathcal{L}_1 \dots \mathcal{L}_m$). All other hosts (denoted by $\mathcal{H}$) in the Internet measure RTT distances (denoted by $\hat{d}_{\mathcal{H}\mathcal{L}_1}, \dots, \hat{d}_{\mathcal{H}\mathcal{L}_m}$) to the landmarks and use this information to determine their positions in a virtual space. The coordinates of a host in a virtual space (denoted by $\mathcal{C}_\mathcal{H} := (\mathcal{C}^1_\mathcal{H}, \dots, \mathcal{C}^n_\mathcal{H})$) are determined using multilateration, which is based on minimizing the error function f_e. In the case of GNP, this is the least-squares-error function:

$$f_e(\mathcal{C}_\mathcal{H}) \ := \sum_{i=1}^{m} (d(\mathcal{C}_\mathcal{H}, \mathcal{C}_{\mathcal{L}_i}) - \hat{d}_{\mathcal{H}\mathcal{L}_i})^2 \qquad (1)$$

The function minimization is performed using the Downhill Simplex [12] method. For calculating $d(\mathcal{C}_\mathcal{H}, \mathcal{C}_{\mathcal{L}_i})$, the positions of the landmarks in the virtual space ($\mathcal{C}_{\mathcal{L}_1} \dots \mathcal{C}_{\mathcal{L}_m}$) must be known. GNP computes coordinates of the landmarks by minimizing the following error function:

$$f_e(\mathcal{C}_{\mathcal{L}_1}, \dots, \mathcal{C}_{\mathcal{L}_m}) \ := \sum_{i,j \in \{1 \dots m\}, j>i} (d(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_{\mathcal{L}_j}) - \hat{d}_{\mathcal{L}_i\mathcal{L}_j})^2$$

III. VIVALDI's Instability

VIVALDI has numerous advantages over other RTT embedding schemes. Being fully distributed, it does not require any infrastructure and is very robust against churn. It also predicts RTTs pretty well, it scales well with the number of hosts participating and it is simple to implement. On the other hand, VIVALDI tends to be unstable. By this we mean that hosts permanently change their positions in the virtual space. There are two reasons for this behaviour: One reason are the local oscillations of the host's position relative to its neighbors. These oscillations are relatively small and result in a small change of RTT predictions.

The more severe reason for this instability is the movement of the whole system, i.e. all hosts participating in the distributed simulation performed by VIVALDI. Since VIVALDI never reaches a stable state, local oscillations never cease. Those small local oscillations of one host affect all hosts that have chosen it as one of their neighbors. This change then affects their neighbors and so on. As the final effect, the whole system does not stand still in the virtual space, but instead it translates and rotates.

Movement of the whole system in the virtual space does not significantly change the RTT predictions obtained by VIVALDI, since the relative positions of the hosts in the virtual space do not change much. But, if we consider how such positions are used, it turns out that the system's movement poses a problem. Positions of the hosts are usually used to predict RTTs. If the positions are constantly changing, we must obtain a new position of the host, to which we would like to have an RTT estimate. By doing so, we could also just perform an RTT measurement itself, since it involves the same amount of communication. Hence, being able to rely on a host position for a longer period of time is desirable.

Another additional reason for instability of VIVALDI is its use of height vectors. Height vectors are a non-metric component of each host's position. It quantifies how "far away" every host is to the "core" of the Internet. This distance from the metric core of the Internet is represented as a positive number and this number is used in order to refine RTT prediction made by VIVALDI. Same as the position in the metric space, the height vector of every end system is a product of the distributed simulation process.

Together with positions of end systems, calculating height vectors are also a result of the simulation process. The consequence of this is that the leveling of height vectors aditionally increases convergence time of VIVALDI. Another reason, why we don't consider height vectors in this work is that when height vectors are used, the positions of hosts are less usable for other purpose than RTT embedding. For example, if host positions should be used to perform some kind of geographic routing, height vectors must be discarded. This increases overall error of the found paths compared to an approach where height vectors are not used.

A. *How Bad Is the Instability?*

To illustrate the problem we performed the following experiment: We implemented VIVALDI in an event-based simulator [15]. For the network model, we used an RTT distance matrix obtained by the "all sites ping" experiment of Planet-Lab [20].

This RTT distance matrix contains RTTs for each pair of hosts participating in Planet-Lab [10]. The total number of hosts in our data is 217. Each host obtained a random choice of 18 neighbors, to which it performed RTT measurements and optimized its position in a virtual 5-dimensional Euclidean space. The goal of our experiment was to determine, for how long one could rely on the positions of the other hosts.

To do so, we obtained the position of one host 2 s after the start of the simulation. We kept this position constant and, at different time points in the simulation, we used it to determine the relative embedding error to all other hosts in the system defined as:

$$\left|\frac{\text{measured rtt} - \text{predicted rtt}}{\text{measured rtt}}\right|$$

Figure 1 shows the result of the experiment. The whole system tends to move away from the position where that host was located after 2 s of simulation at a more or less constant speed. This motivated us to define a measure for the instability

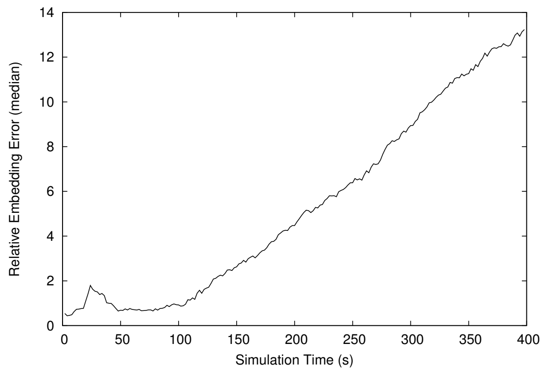

Fig. 1. Average relative embedding error of the fixed position of one host relative to changing positions of the whole system

of VIVALDI (and related approaches): average host speed. Each host "travels" through the virtual space by updating its position. At each position update, there is a distance the host has overcome, defined as the distance between the old and the new position of the host. We define the speed of a host as the sum of those distances (calculated from the beginning of the simulation) divided by the simulation time. This measure gives us an estimate, how unstable one system is: The larger the average speed of the hosts in the virtual space, the more unstable is the system.

In order to test the correctness of our VIVALDI implementation, we compared the behavior of the same simulation with two different RTT data sets. One data set is the same we used to obtain results from Fig. 1 (VIVALDI Planet-Lab). The second one is an RTT matrix we obtained by randomly placing 217 points in a 5 dimensional Euclidean space and calculating distances between them (VIVALDI metric). If our implementation of VIVALDI is correct, the embedding error of the VIVALDI approach using the VIVALDI Metric data, which stems from the Euclidean space, should reach 0 after some convergence time. At the same time, the average host

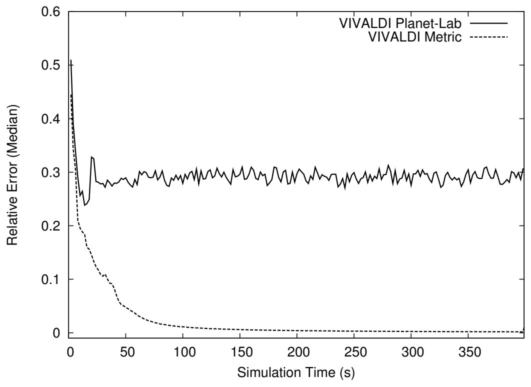

Fig. 2. Median of the relative embedding error using RTTs measured in Planet-Lab vs. distances originating from a metric space

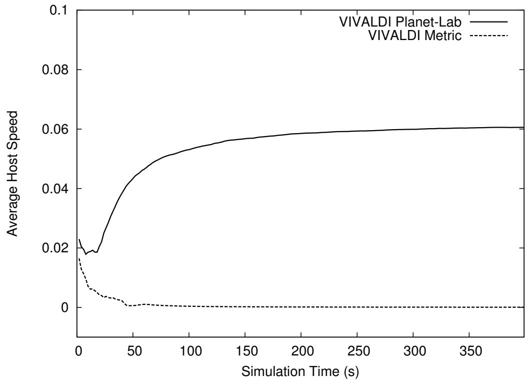

Fig. 3. Average (moving) speed of host using RTTs measured in Planet-Lab vs. distances originating from a metric space

speed should also reach 0. Figure 2 shows the median of the embedding error for the two data sets. The median of the relative embedding error of the simulation using the VIVALDI Metric data converges after 150 s of the simulation towards 0. At the same time, the relative embedding error of VIVALDI Planet-Lab remains more or less constant.

Fig. 3 shows the average host speed for both data sets. The average host speed of VIVALDI Metric reaches 0 at a the same time the relative embedding error reaches 0. On the other hand, the average host speed of VIVALDI Planet-Lab converges towards a constant value, meaning that the whole system is constantly moving through the virtual space.

B. Reasons for VIVALDI's Instability

Three hosts whose RTTs do not satisfy the triangle inequality may be sufficient to render a VIVALDI system unstable. For example, those RTTs could be 2.5, 3 and 8ms. Each time one host tries to correct its position relative to another host, it will either decrease the "predicted" value of the RTT 8ms or increase one of the other two RTTs (2.5ms or 3ms) to have larger predicted values. Since VIVALDI always optimizes one (randomly chosen) RTT prediction, it never reaches a stable state, since decreasing error in one direction increases the error in the other, which must be then compensated in turn.

Fig. 4 illustrates such a behaviour, that we describe as "host chasing". At each simulation step, a host tries to reduce its RTT prediction to one other host. By doing so, it increases the embedding error towards all other hosts. This increased embedding error then gets corrected by another host, which in turn increases other embedding errors. The result of this behaviours is the translation of the whole system through the virtual space. Another drawback of VIVALDI arises from the

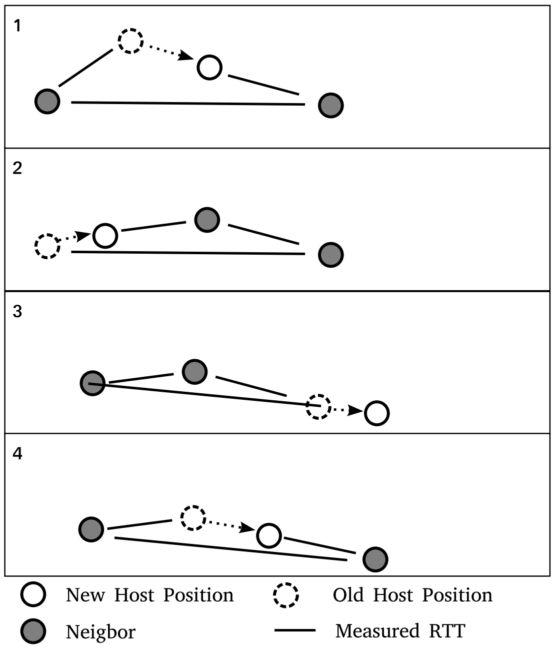

Fig. 4. An example of "host chasing" in VIVALDI caused by a triangle inequality violation

fact that VIVALDI does not require the "is a neighbor" relation to be bidirectional. A bidirectional "is a neighbor" relation means, that if host *A* optimizes its position relative to host *B*, then *B* should optimize its position relative to *A*. In practice, each host chooses a fixed number of random neighbor hosts and optimizes its position in the VIVALDI system relative to them. This means, if there is no optimal solution (i.e. some of the properties of the metric space are violated by the measured RTTs), hosts will be oscillating around their (theoretically) optimal position.

Fig. 5 show what could happend, when the "is a neighbor of" relation is only unidirectional. The host in the middle of the figure tries optimizing its position relative to the neighbors, which at each optimization step leads to a new position of the host, since the neighbors will never move themselves towards the host in the center. If the relation "is a neighbor of" would be bidirectional, the neighbors of the host in the center would also move towards that host, which would reduce oscillation of its position.

IV. NetICE9

A. Crystallization

The basic idea of NetICE9 is to simulate the process of crystallization. Crystallization is a process, where a solid

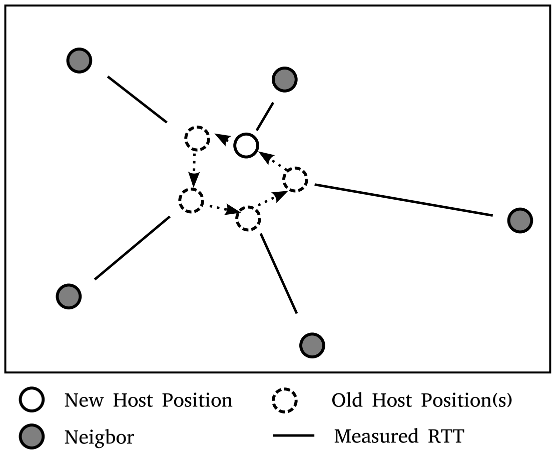

Fig. 5. An example of oscillations in VIVALDI caused by lack of bidirectional "is a neighbor of" relation

structure is formed incrementally. This structure grows when new atoms align themselves to their neighboring atoms, which are already part of the structure. The position of a newly added atom depends on the forces acting between it and its neighbors. The attracting and repelling forces between atoms are van der Waals forces. The final position of the atom is where those forces are in balance.

Within NetICE9, each host is considered to be an atom which needs to be embedded into a crystal structure. In analogy to the attracting and repelling van der Waals forces between atoms, we define the force between two hosts as the difference between the predicted and the measured RTT. If this difference is negative, there is a repulsion force between two hosts. On the other hand, a positive difference means the two hosts are attracted to each other.

Since we assume that the triangle inequality does not hold, the sum of the forces acting on a host is rarely zero. Instead of trying to find a position, where the sum of those forces is zero, we look to find the position, where the sum of these forces is minimal. To achieve this, we propose using a function minimization method. The function we are minimizing in order to find the host position is the least square function (similar to the one used by GNP).

B. NetICE9 Algorithm

The crystal structure of NetICE9 is self-organizing. Each host determines its own position within the crystal structure. Every host in NetICE9 is aware of a set of hosts which are its neighbors. To each of those neighbors, it periodically measures its RTT. At the same time, the host also queries each of its neighbors about their current position in the virtual space and their current speed (at what rate has it been changing its position recently). The host then uses this information in order to update its own position. Updating the position is done by minimizing the objective function:

$$f_e(\mathcal{C}_\mathcal{H}) := \sum_{i=1}^{m} \frac{1}{1 + (\mathcal{V}_{\mathcal{N}_i} \cdot \delta)} \cdot (d(\mathcal{C}_\mathcal{H}, \mathcal{C}_{\mathcal{N}_i}) - \hat{d}_{\mathcal{H}\mathcal{N}_i})^2$$

This objective function is very similar to the one used by GNP, with the difference that we are using neighbors as landmarks and that each neighbor is weighted according to its speed. The larger a neighbor's speed, the lower is its weight. We scale the speed with a constant δ. This constant has an impact on the convergence time. In our experiments we discovered that $\delta = 1000$ yields best results. Since the speed of each neigbor can theoretically be 0, we add 1 to the scaled speed of the neighbor in order to avoid division by zero. The whole process of updating the position of a host is described in Algorithm 1.

The reason we use this algorithm is to determine the position of the host using all available information. VIVALDI uses a much simpler approach. The optimization is done towards only one neighbor. In the case where no optimal embedding is possible, this approach results in oscillating host positions (see Section III). In NetICE9 we perform the optimization as proposed in Algorithm 1. The host position will remain stable, if the positions, velocities, and RTTs remain constant. This greatly improves both stability and precision (reduces the embedding error) of the system.

Algorithm 1 NetICE9 algorithm for updating a host's position in a virtual space

```
Require: $N$ Set of neighbors
Require: $T_u$ Update interval
Require: $P_c$ Current position of the host
Require: $V_p$ Previous EWMA (exponentially weighted moving average) of host speed
Require: $\alpha$ Dampening factor for EWMA
  for $n \in N$ do
    $(RTT[n], POS[n], SPEED[n]) \leftarrow Query(n)$ {Query current speed (EWMA) and position of a neighbor and measure RTT to it}
  end for
  $P_p \leftarrow P_c$ {Store current host position as previous position}
  $P_c \leftarrow Minimization(RTT, POS, SPEED, P_p)$ {Determine new host position using function minimization. Use the previous position of the host as the starting point for the function minimization.}
  $delta \leftarrow |P_p - P_c|$ {Calculate distance between old and new position}
  $V \leftarrow delta/T_u$ {Calculate the host speed}
  $V_p \leftarrow \alpha \cdot V + (1 - \alpha) \cdot V_p$ {Update EWMA of host speed}
  return $P_c, V_p$
```

Computational overhead of this algorithm is significantly larger than the one proposed by VIVALDI. Most of this overhead is imposed by using function minimization. Since the computational overhead of the used function minimization depends on many parameters, such as complexity of the function and even the start point, we are not able to give an estimate of the computational overhead. On the upside, when the changes of the measured RTTs are not large, the function minimization terminates very quickly, since the starting point is near the optimum. This makes updates use almost constant computational overhead in most of the cases ($O(1)$).

C. Choice of Neighbors

Each host has to select a limited number of other hosts within the network as its neighbors. We assume that this choice of neighbors should not be random. VIVALDI [3] showed that using only the nearest neighbors yields inaccurate predictions for large RTTs. Instead, VIVALDI uses a mix of close as well as distant neighbors to give the host a sense of its global position within the network.

In NetICE9, we decided to use a neighbor selection strategy with a bidirectional "is a neighbor of" relation. We assumed that this would significantly reduce the rippling effect of local oscillations of hosts on the whole system, because the host oscillations would go both ways and cancel each other out. The problem was that there did not exist neighbor selection strategies which fulfilled both conditions: 1) it selects a mix of close as well as distant hosts as neighbors and 2) it ensures a bidirectional "is a neighbor of" relation. For this reason, we developed our own neighbor selection strategy called fisheye [11]. Fisheye is a topology-aware overlay network building protocol which is able to give each host a fisheye-view of a network. A fisheye-view of a network is a choice of c hosts from the network with special properties. Those properties are geographical diversity and fisheye-lens like distribution of neighbors. Geographical diversity means that the choice of neighbors is evenly distributed around the host. Fisheye-lens distribution of neighbors means that the density of chosen neighbors decreases with the distance to the host. We achieve those properties by performing a distributed gravity force minimization algorithm as described in [11]. Fig. 6 shows an example of such a choice of neighbors. An interesting feature

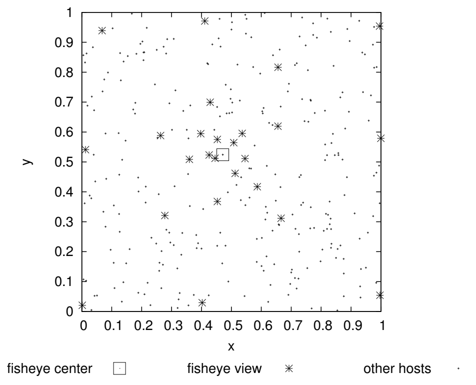

Fig. 6. Example of fisheye view calculated using the gravity force minimization algorithm.

of our fisheye approach is that it does not require embedding of hosts into a virtual space. Instead, it is based only on measured RTTs. We also developed a version of our overlay network building protocol which is able to create a fisheye overlay network with bidirectional "is a neighbor" relation. This overlay network is the one we are using as a neighbor selection for our NetICE9 approach.

V. Evaluation

To evaluate the precision and stability of our approach, we compared it to VIVALDI and GNP by implementing VIVALDI in an event-based network simulator [15].

A. Simulation Scenario and Parameters

The network model of the simulation is the one used by Dabek et al. in their original evaluation of VIVALDI [3]. Each message within the network is delayed by half the RTT between two hosts. RTT information for our model was obtained from RTT measurements from the Internet. For input data we had two data sets. One set (denoted as Planet-Lab) contains a full RTT distance matrix of 217 different hosts obtained from the "all sites ping" experiment [20]. The other data set (denoted as KING) contains a full RTT distance matrix of 462 hosts, which was obtained using the King [6] method.

In each simulation run we started a new instance of a host every 100ms until all hosts were active. This means that all hosts are active by the time the simulation has been running for 21.7 s in the case of the Planet-Lab data or after 46.2 s in the case of the KING data set. For each embedding we used a 5-dimensional Euclidean space as the virtual space. We limited the number of chosen neighbors (c) to 18. We kept each simulation running for 800 s (simulation time).

B. Impact of Neighbor Selection on VIVALDI

First we noticed in our simulations that selecting neighbors uni-directionally at random may not be the best choice. In order to confirm this, we performed a simulation of VIVALDI with three different neighbor selection strategies.

- *RAND* selects c neighbors randomly for the optimization. This strategy is the one used by VIVALDI.
- *FISHEYE-UD* uses the fisheye in a unidirectional mode. Each host chooses a fisheye view of the overlay network. The relation "is a neighbor of" is not guaranteed to be bidirectional in this strategy.
- *FISHEYE-BD* is a fisheye in a bidirectional mode. This strategy differs from the unidirectional fisheye in that the resulting overlay network is guaranteed to be two-way (bidirectional "is a neighbor of" relation).

Figures 7 - 10 show the average embedding error and host speed for all three proposed neighbor selection strategies for both data sets. The simulation results show that, as we expected, using a better overlay network can result in more stable host positions. Figures 8 and 9 show an interesting result: Using a fisheye overlay without bidirectional "is a neighbor of" relation actually results in an even more unstable system. This is because the fisheye method of host selection increases the effect where hosts are "chased" through the virtual space. For that reason, the prediction error of FISHEYE-UD is even worse than that of RAND. As soon as the choice of the neighbors is bidirectional, the "chasing"-effect disappears and the systems becomes both more stable and more precise compared to a random choice of neighbors (RAND).

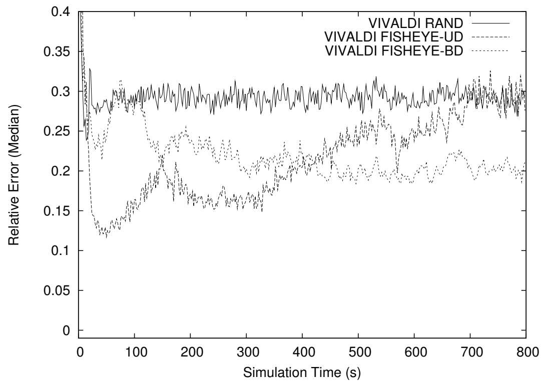

Fig. 7. Median of embedding error for VIVALDI using different neighbor selection strategies with Planet-Lab data.

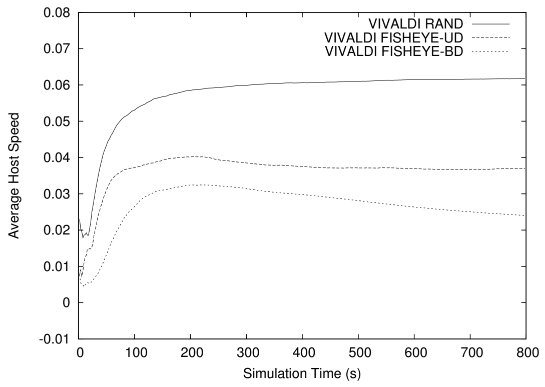

Fig. 8. Average host speed in the virtual space for VIVALDI using different neighbor selection strategies with Planet-Lab data.

C. NetICE9 vs. VIVALDI

Our second improvement to VIVALDI is to position a host relative to all known neighbors simultaneously instead of optimizing the position relative to one neighbor at a time. As before, we compared the median of the relative embedding errors and average host speed in the virtual space for VIVALDI using a random neighbor choice, VIVALDI with a bidirectional fisheye neighbor choice, and NetICE9 (a combination of bidirectional fisheye neighbor choice and positioning relative to all neighbors). We performed this comparison for both the Planet-Lab and the KING data set, see Figures 11 - 14.

D. Prediction Error

Since the median of the relative prediction error just gives an estimate of the average prediction error, we decided to take a closer look at the overall distribution of the relative RTT embedding error. In order to achieve this, we compared CDFs (cumulative distribution functions) of predicted distances. In Figures 15 and 16, present the CDFs of the relative embedding error for NetICE9, GNP, VIVALDI RAND and VIVALDI FISHEYE-BD. Those CDFs have been obtained by comparing relative embedding errors of each of those systems after running the simulation for 400 seconds. In order to have comparable GNP results we also used embedding into a 5

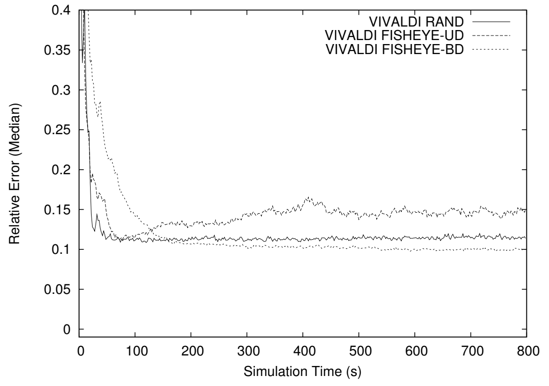

Fig. 9. Median of embedding error for VIVALDI using different neighbor selection strategies using KING data.

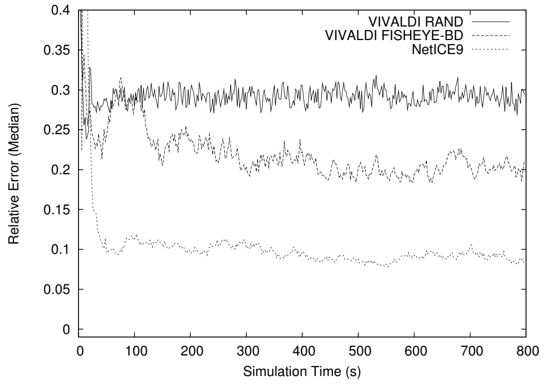

Fig. 11. Median of embedding error for NetICE9 compared with VIVALDI using Planet-Lab data.

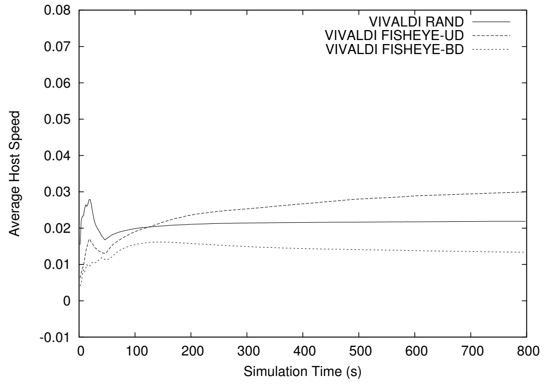

Fig. 10. Average host speed in the virtual space for VIVALDI using different neighbor selection strategies with Planet-Lab data.

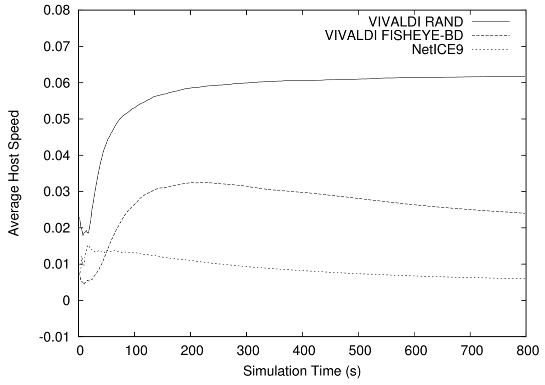

Fig. 12. Average host speed in the virtual space for NetICE9 compared with VIVALDI using Planet-Lab data.

dimensional virtual space using 18 landmarks. To reduce the impact of randomness, we compared the results of 30 simulation runs, each of them made with a different initial seed for the pseudo random number generator used by the simulator. This seed both influences the choice of neighbors, landmarks and initial starting points for function minimization. Figures 15 and 16 show that NetICE9 clearly outperforms VIVALDI in terms of relative error of RTT embedding. Moreover, NetICE9 outperforms the embedding of GNP. We explain this by the fact that NetICE9, due to its distributed nature, uses a larger portion of available RTT measurements. This enables NetICE9 to obtain a better embedding compared to GNP, since GNP uses only RTTs to landmarks, which are all the same for all other hosts. Fig. 16 shows that using the bidirectional fisheye overlay network improves the performance of VIVALDI almost to the level of the performance of GNP.

VI. Conclusion

We have identified two major sources of VIVALDI's instability: 1) it does not choose its neighbors symmetrically. 2) it optimizes towards only one neighbor at a time and disregards all information known about the other neighbors. In order to avoid these problems, we proposed NetICE9, which simulates the creation of a crystal structure. In this crystal structure, every atom (i.e. every host) positions itself relative to its surrounding atoms (hosts). In order to have a stable system, we chose the surrounding of each atom such that the whole crystal remains stable (i.e. the atoms of the crystal do not move). To achieve this, we propose using such a choice of neighbors that the"is a neighbor of" relation is bidirectional (A is a neighbor of B iff. B is a neighbor of A). Motivated by VIVLADI results, the choice of neighbors should be geographically diverse. This means that for an ideal RTT prediction scheme, a mix of close and remote neighbors should be used. One choice of neighbors fulfilling both of those properties is the fisheye [11] overlay network.

We have compared NetICE9 approach to VIVALDI and GNP using a simulation based on RTTs measured in the Internet. Our evaluation focused mainly on aspects of stability and precision of RTT embedding. NetICE9 outperforms both VIVALDI and GNP in the terms of precision of RTT embedding. NetICE9 proved to be more stable than VIVALDI.

References

[1] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson. Design and implementation trade-offs for wide-area resource discovery. *ACM Trans. Internet Technol.*, 8(4):1–44, 2008.

[2] M. Costa, M. Castro, A. Rowstron, and P. Key. Pic: Practical internet coordinates for distance estimation. In *International Conference on Distributed Systems*, Tokyo, Japan, March 2004.

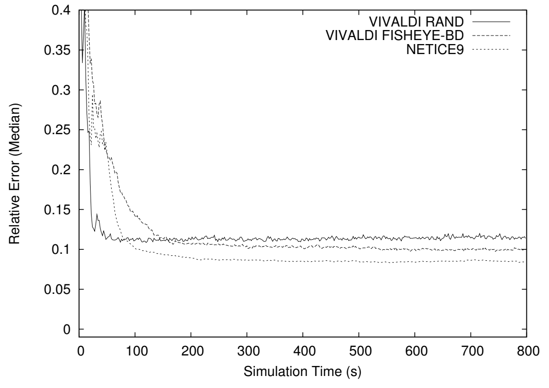

Fig. 13. Median of embedding error for NetICE9 compared with VIVALDI using KING data.

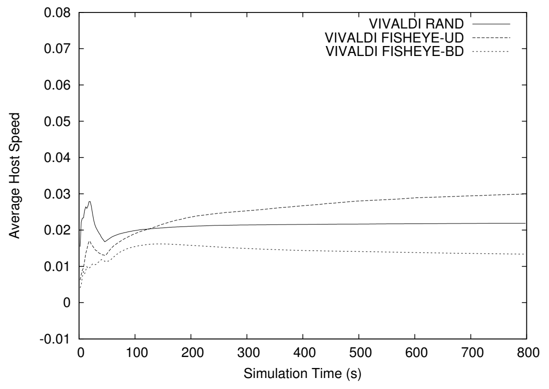

Fig. 14. Average host speed in the virtual space for for NetICE9 compared with VIVALDI using KING data.

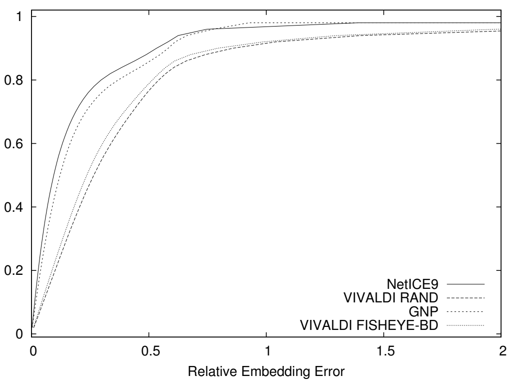

Fig. 15. Comparison of embedding error CDFs using Planet-Lab data.

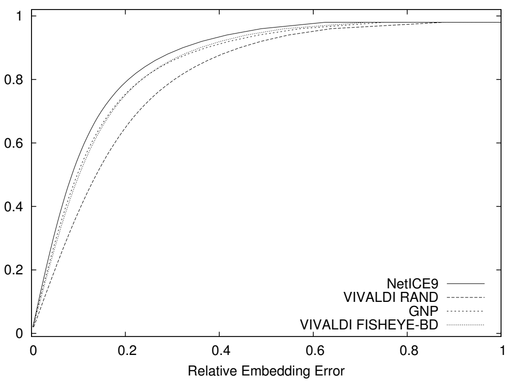

Fig. 16. Comparison of embedding error CDFs using KING data.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04*, pages 15–26, New York, NY, USA, 2004. ACM Press.

[4] A. Dufour and L. Trajković. Improving gnutella network performance using synthetic coordinates. In *QShine '06: Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks*, page 31, New York, NY, USA, 2006. ACM.

[5] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. Technical report, Lawrence Berkeley National Laboratory, 1997.

[6] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM Internet Measurement Workshop 2002*, 2002.

[7] S. Lee, Z.-L. Zhang, S. Sahu, and D. Saha. On suitability of euclidean embedding of internet hosts. In *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 157–168, New York, NY, USA, 2006. ACM.

[8] H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Internet Measurement Confgerence 03*, October 2003.

[9] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *In IMC*, 2005.

[10] R. McGeer. Planetlab: a worldwide deployment infrastructure for the next generation of network services. Technical report, CERN, Geneva, 2004. CERN, Geneva, 13 May 2004.

[11] D. Milic and T. Braun. Fisheye: Topology aware choice of peers for overlay networks. In *The 34th IEEE Conference on Local Computer Networks (LCN)*, Zurich, Switzerland, October 20-23 2009.

[12] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[13] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *IEEE Infocom02*, New York / USA, June 23-27 2002.

[14] T. S. E. Ng and H. Zhang. A network positioning system for the internet. In *USENIX 2004*, pages 141–154, Boston MA, USA, June 2004.

[15] OMNET++, avail. online:http://www.omnetpp.org, 2007.

[16] C. Pelsser. Using virtual coordinates in the establishment of inter-domain lsps. In *CoNEXT '05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 274–275, New York, NY, USA, 2005. ACM.

[17] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Trans. Netw.*, 12(6):993–1006, 2004.

[18] Y. Shavitt and T. Tankel. On the curvature of the internet and its usage for overlay constructi on and distance estimation. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Com puter and Communications Societies*, pages 374–384, Hong Kong / PRC, March 7-11 2004.

[19] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Internet Measurement Confgerence 03*, October 2003.

[20] C. Yoshikawa. Planetlab all-sites-pings experiment, 2006.

[21] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang. Measurement based analysis, modeling, and synthesis of the internet delay space. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 85–98, New York, NY, USA, 2006. ACM.