# Gossip-based distance estimation and failure detection distributed application

Alessandro Finocchi
*University of study of Rome Tor Vergata*
Rome, Italy
alessandro.finocchi@students.uniroma2.eu

***Abstract*—Like coordinates in an euclidean space, addresses in a network space can identify where an host is located, but while in the former we are able to define different concept of distance, for the latter there is no naive ideas of how to find the distance between two hosts, given their addresses.**
**Large-scale internet network can benefit from the ability of predicting communication latencies between nodes in a lot of contexts: load-balancing, data replication, choice of elaboration nodes in a cloud computing system and so on. Having information about the distance between nodes can help them taking better decision, preferring maybe a closer or a farther one depending on the situation.**
**Here comes Vivaldi, a simple, light-weight, fully distributed and also efficient algorithm to assign synthetic coordinates to hosts such that the distance between two coordinates can accurately predict the round-trip time between the hosts they represent. In this work a Vivaldi algorithm, more stable than the original one, has been studied, implemented and tested.**

***Index Terms*—Coordinate System, Gossip, Membership, Peer-to-Peer, Registry, Synthetic Coordinates, Vivaldi**

## I. Introduction

Synthetic coordinates systems allow an internet host to predict its round-trip time towards other hosts. Using the Vivaldi algorithm they can do this in a very efficient manner [1], but still other works [2] could afford to improve the performances of this algorithm and so they have been taken into account for the implementation of this work.
The purpose of the current study is to create a peer-to-peer network that hosts can connect to starting from a registry, and after receiving information about the other participant through a membership protocol, they can build a proper idea of their position in the network with respect to the others in order to know which host is closer and which one is farther: to do so Vivaldi algorithm computes the synthetic coordinates for each peer, creating a space that works as a map of the network.
As stated before, when run on real systems, it has been shown that the original algorithm does not produce stable and accurate coordinates [2], so starting from the initial work, the actual system has been developed adding a moving-percentile (MP) filter on the rtt probes, and heuristics on the coordinates gossiping and updates, which will improve these metrics.
The solution will be implemented using the Go programming language and the gRPC APIs, each component will be containerized with Docker and distributed on an AWS EC2's instance.

## II. Background

First thing first let's analyze the main concept that will be fundamental in understanding the solution design.

### A. Gossip protocols

Gossip (or epidemics) protocols define a distributed communication paradigm inspired by its human counterpart to efficiently disseminate information, but it can actually be used for different purpose; they tend to be deployed in contexts where both the scale and the dynamism of the underlying network make the usual communication protocol unpractical.
In this study two gossiping protocols has been used: the first for the membership layer of the host and the second for disseminating the Vivaldi's coordinates.

### B. SIR model

In a gossiping protocol the final goal is to spread the updates of a value to all other nodes, and if no new updates are injected, after some time $t$ all nodes will have the same value: in the information dissemination problem where updates are rare (and this is our scenario because when the network reach its stability updates become rare, we'll see that coordinates never stop) this is done using the SIR model, also called *complex epidemics* or *rumor mongering*.
A node can be in one of three state:

- *Susceptible (S)*: the node does not know about updates,
- *Infected (I)*: the node knows about the updates and is spreading them,
- *Removed (R)*: the node knows about the updates but doesn't spread them anymore.

The decision to transition to the removed state is made using the *feedback-counter* strategy which has been shown to have the shortest delay among the other possible variants in [3]:

- When: after having received a message which acknowledges that the exchange partner was already aware of the update (feedback)
- How: after a fixed number $k$ of feedback evaluation (counter)

### C. Membership management

The problem of maintaining the list of nodes participating in a protocol is a gossiping dissemination problem itself: in modern systems like data-centers and peer-to-peer networks, the diffusion of the membership list presents two important

issues: the scale and the dynamism: indeed there could be several thousand of machines that periodically go in and out the network due to the churn and/or the high rate of failures, and so this list must be continuously updated.
An important observation is that nodes do not actually need the complete membership list: in fact they need just a sample of such list, it may also be selected uniformly at random. This sample is called *partial view* and the problem of providing nodes with an up-to-date partial view is called *membership management* [3].

### D. Vivaldi synthetic coordinates

Maintaining the pairwise distance between 2 nodes in a network with $N$ connected hosts requires storing $O(N^2)$ distances in memory, so it wouldn't scale that well.
The heart of the Vivaldi algorithm is modeling the network as a *spring relaxation problem*: visually, one can imagine each peer being treated as a mass connected to all other masses in a pairwise way with a spring, which natural length is defined as the distance (rtt) between the two masses (hosts) it connects. The algorithm compresses all the springs down to the origin of a space, and then let them go, allowing the system to settle in right shape.
The coordinate model is really a predictive model and we want this to converge to the real world actual topology, and so given that the real world topology should be somewhat stable (servers usually don't go around), the model should converge over time to represent reality.

## III. Solution Design

Let's start with a simple overview of the proposed solution.

### A. Registry

There is a registry that runs two services: the *connection* service and the *heartbeat* service. The former allows incoming hosts to discover a list of other nodes already inside the network in order to allow the new ones to join the protocol activities, the latter permits the registry to keep an up-to-date view of the peers still connected, so that the host list provided during the connection contains peers actually connected to the network.

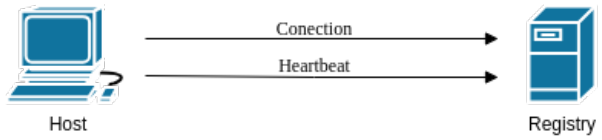


Fig. 1. At the beginning the node interacts with the registry

### B. The network

The peer-to-peer network is created using the registry that provides each host with a list of selected nodes. Once connected to them, the hosts starts 3 different protocols: the Membership protocol, the Vivaldi protocol and the Vivaldi gossiping.

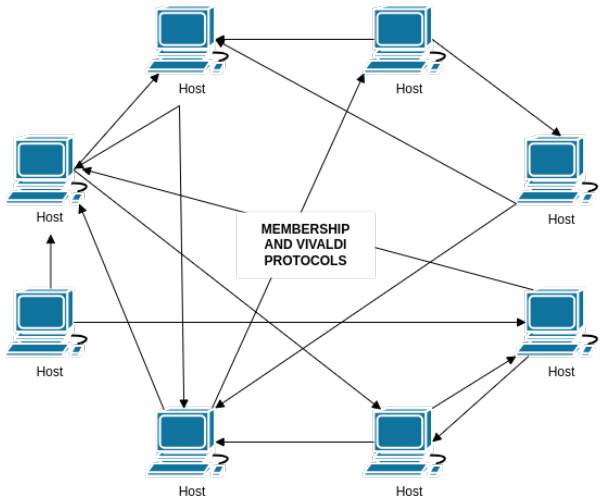


Fig. 2. Each node knows only a subset of the other participant

## IV. Solution Details

Let's now dive into the implementation aspects of the proposed solution.

### A. Registry

The registry can be queried with an address known by every other hosts.
Thanks to an heartbeat algorithm, the registry maintains a list of still-active nodes. In this way, in case of a failure, since the host would not communicate its exit from the network, while the other nodes tend to forget about it, the registry awaits for a certain number of timeout cycles, after which the host is considered disconnected and is removed from the list.
The connection with the registry is secured using gRPC connection with mutual TLS credentials for the host and registry, that's to say data are encrypted in both directions; this is done using the OpenSSL tool to create self-signed certificates, process CSRs (Certificate Sign Request) using the standard x509, and to create the public and private keys of both host and registry that will be used to secure the communication.

### B. Membership protocol

The implementation of the protocol comes from the work in [4]: each peer maintains a partial view periodically updated with a gossip procedure.

1) **Partial view**
   It's a list of $c$ node descriptors, each one containing the address of the node and its age, an index of how long the node has been part of the partial view.
2) **Parameters**
   - $c$: the size of the partial view.
   - $H$: healing, it defines how aggressive the protocol should be when it comes to removing potentially faulty nodes.
   - $S$: swapping, directly proportional to the priority given to the nodes pulled by another peer.

3) **Peer selection**
   The policy implemented to choose the destination of the next membership information exchange is selecting uniformly at random the receiver.
4) **View propagation**
   Once a peer has been chosen, the peer have to exchange the information: following the results obtained in [4] the strategy implemented has been the push-pull one for better performances.
5) **View selection**
   The parameters that determine how view selection is performed are H and S: the hosts have been implemented to be able of behaving in 3 different way:
   - *blind*: $H = S = 0$,
   - *healer*: $H = c/2, S = 0$
   - *swapper*: $H = 0, S = c/2$

### C. Vivaldi protocol

Vivaldi was inspired by analogy to a real-world, and thus three-dimensional, Euclidean, mass-spring system. The algorithm can use other coordinate spaces too by redefining the coordinate difference and sum, vector norm and scalar multiplication operations. The implementation for the usual euclidean space is the following.

**Algorithm 1** Vivaldi

// Node $i$ pulls coordinates from node $j$ and updates
// its own coordinates.
// $rtt$: the round-trip time between node $i$ and $j$
// $\vec{x}_j$: the coordinates of node $j$
// $e_j$: the error of node $j$

**Require:** $(rtt,\ \vec{x}_j,\ e_j)$
1: $rtt = \text{Filter}(rtt, j)$
2: $w = e_i/(e_i + e_j)$
3: $e_s = |\ \|\vec{x}_i - \vec{x}_j\| - rtt|/rtt$
4: $\alpha = c_e \cdot w$
5: $e_i = \alpha \cdot e_s + (1 - \alpha) \cdot e_i$
6: $\delta = c_c \cdot w$
7: $\vec{x}_i = \vec{x}_i + \delta \cdot (rtt - \|\vec{x}_i - \vec{x}_j\|) \cdot u(\vec{x}_i - \vec{x}_j)$

Each node updates its coordinate $x_i$ and error $e_i$ with each new latency observation. An observation consists of the remote node's coordinate $x_j$, its error $e_j$, and a new measurement of the latency between the two nodes. First, a weight $w$, the confidence, is assigned to this observation based on how certain nodes i and j are relative to one another (Line 2). Second, they find how far off the observation was from what was expected based on the coordinates: $e_s$ is the relative error of this measurement (Line 3). Third, node i updates its error $e_i$ with an exponentially-weighted moving average. Unlike most EWMAs, however, the $\alpha$ is not fixed, instead it is weighted according to how much trust is given to the current observation (Lines 4-5). If this causes node i's error to go above one or below zero, it is forced to remain in bounds. Eventually coordinates are updated (Lines 6-7): $\delta$ is the pull of this observation on the coordinate, it dampens the magnitude and direction of the change applied to the coordinate. Constants $c_e$ and $c_c$ affect the maximum change an observation can have on error and coordinates, respectively. It's been used used

$$c_c = c_e = 0.25$$

which are the same values used in the original authors' Vivaldi simulator: these steps allows faster convergence even though further studies [5] found out that more stable results can be achieved using

$$c_c = 0.005 \qquad c_e = 0.1$$

Following the work in [2], to gain more stability an rtt filter has been used: in particular, the best one has been found to be the *MP filter*, which uses an history window of the last $h$ samples and take the $p$-th percentile in order to exclude those samples that happens to be several orders of magnitude grater the expected one (this can happen, as shown in [2] the $0.4\%$ of samples are grater than 1 second), which periodically distort the entire coordinate system, and this filter has been shown to provide a substantial improvement in both stability and accuracy of the coordinates; the parameters used has been chosen to be

$$h = 16$$
$$p = 25$$

Other filters has been implemented too, the *raw* and the *EWMA* ones, in order to compare the obtained results.
The system can also use another type of coordinates. In [1] has been shown that better performances can be achieved using a non-Euclidean space with an *height vector*: the height vector model is implemented redefining the usual vector operation as follow:

$$[\vec{x}, x_h] - [\vec{y}, y_h] = [(\vec{x} - \vec{y}), x_h + y_h]$$
$$\|[\vec{x}, x_h)\| = \|\vec{x}\| + x_h$$
$$\alpha \cdot [\vec{x}, x_h] = [\alpha \cdot \vec{x}, \alpha \cdot x_h]$$

### D. Vivaldi gossiping

The gossiping of the coordinates has been implemented using the SIR model as discussed before, but with some improvements. The only usage of MP filter still permits coordinate to change too much with respect to what someone would desire [2].
The idea is to maintain two types of coordinates:

- The *system coordinates*, the ones computed by the Vivaldi algorithm continuously,
- The *application coordinates*, the ones spreaded among all the nodes in the network, which are updated only when there are significant changes in the system coordinates.

This differentiation aim to reduce the traffic of coordinate updates: indeed a coordinate change could initiate a cascade of other changes, and we would like this to happen only when coordinates change remarkably.

The mechanism used to do this is a window based mechanism which makes use of two windows (starting and current

windows respectively): given a stream of subsequent system coordinate

$$SC = \{\vec{c}_0, \dots \vec{c}_n\}$$

if the length of the windows is $k$ then the windows are respectively

$$W_s = \{\vec{c}_0, \dots \vec{c}_k\}$$
$$W_c = \{\vec{c}_{n-k}, \dots \vec{c}_n\}$$

At each new system coordinate computed by the Vivaldi algorithm

- if the two windows are not both of size $k$ then the new sample is added to both $W_s$ and $W_c$,
- otherwise no more samples are added to $W_s$, and $W_c$ slides to add $\vec{c}_i$ and drop $\vec{c}_{i-k-1}$

When the two windows are full, at each new sample after sliding $W_c$ the two window sets are tested for difference using two heuristics, and if at least one of them declare the sets to be different, then here it comes the "significant change", or *change point*, that the system was awaiting, so it can update the coordinates and spread the application ones. When a change point happens, $W_s$ and $W_c$ are cleared and the process begins again from the start.
The heuristics used to declare the change point are 2:

- **Relative**
  Given the centroid $\mathcal{C}(W)$ of a cluster of points $W$, the coordinates of the nearest known neighbor $\vec{r}$ and a relative error threshold $\epsilon_r$, if

  $$\frac{\|\mathcal{C}(W_s) - \mathcal{C}(W_c)\|}{\|\mathcal{C}(W_s) - \vec{r}\|} > \epsilon_r$$

  then let the system coordinates be

  $$\vec{c}_s = \mathcal{C}(W_c)$$

- **Energy**
  Define the energy distance between two sets $A$ and $B$ as

  $$e(A,B) = \frac{n_1 n_2}{n_1 + n_2} \left( \frac{2}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left\| \overrightarrow{a_i} - \overrightarrow{b_j} \right\| - \frac{1}{n_1^2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_1} \|\overrightarrow{a_i} - \overrightarrow{a_j}\| - \frac{1}{n_2^2} \sum_{i=1}^{n_2} \sum_{j=1}^{n_2} \left\| \overrightarrow{b_i} - \overrightarrow{b_j} \right\| \right)$$

  given a threshold $\tau$ to determine when sets diverged, if

  $$e(W_s, W_c) > \tau$$

  then let the application coordinates be

  $$\vec{c}_a = \mathcal{C}(W_c)$$

From [2] the parameters chosen are

$$\epsilon_r = 0.3 \qquad \tau = 8$$

Using such window-based heuristics fewer updates to $\vec{c}_a$ would occur, leading to greater stability.
In conclusion, a retention policy has been applied to the list of coordinates stored in each host in order to forget about those hosts who are no more part of the network: every 30 seconds an host forgets about those coordinates older than 2 minutes, but simply doing this would lead to deleting every coordinates if they stop changing, because their rumor mongering would stop too, so apart from the change point, application coordinates are spreaded every 60 seconds in order to refresh the age for the network peers.

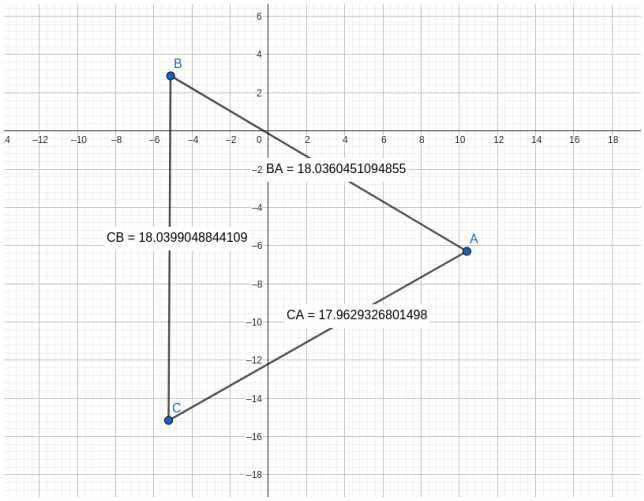


Fig. 3. 3 host in a 2-dimensional space once converged

## V. Results

We run two different test: the first to verify the correctness of the implementation, the second to simulate a scale-reduced real-world scenario.

### A. Test of correctness

The first result tests the correctness of the implementation and it's been developed in two different ways using the usual euclidean space.
Human beings are not able to imagine above the dimensions where they live, but in a 2-dimensional scenario, having 3 hosts, each pair far the same rtt, they should be dislocated as an **equilateral triangle** laying on the euclidean bi-dimensional space used. If the rtt is equal to 18 ms, then we expect the sides of the triangle to approximate this quantity.
When the system reached out a relative error smaller than 0.05, one of the result obtained can be seen in Figure 3.

- $A(10.38, -6.258)$
- $B(5.12, 2.88)$
- $C(-5.22, -15.15)$

The same experiment has been developed in a 3-dimensional space with 4 hosts, always far the same rtt: in this case the form they achieve should be similar to a **regular tetrahedron**, and indeed the result when each host obtained an error less than 0.05 is like in Figure 4.

- $A(-2.19, -2.558, -7.04)$
- $B(1.26, 12.881, 0.94)$
- $C(-7.42, 0.56, 10.10)$
- $D(9.64, -2.42, 6.31)$

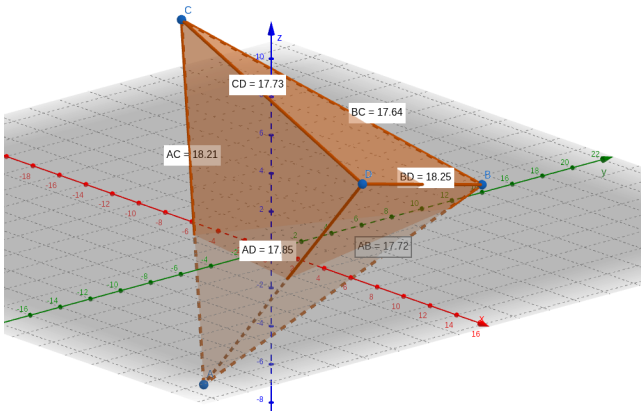


Fig. 4. 4 host in a 3-dimensional space once converged

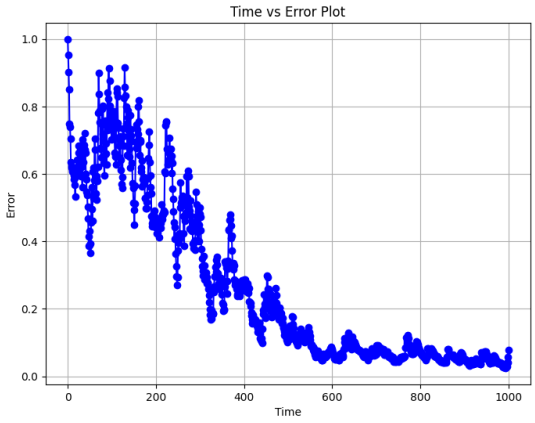


Fig. 5. Relative error with 3d height vector coordinates and 20 hosts

As we may see the average predicted distance between each pair of points, both in the 2-dimensional and 3-dimensional case, approximates well the real distance, which is 18 ms: in order to accomplish such a result an important aspect of the algorithm has been the **unit vector of null vector**, which has to be random as explained in [1], it can't be deterministic. Indeed, if it was be deterministic, since all hosts start from the space origin, then the first displacement of each host would be along the same straight line, and the same would be for all the next ones because all the $u(\vec{x}_i - \vec{x}_j)$ would lay on that straight line, so it would be like to reduce the system to a 1-dimensional euclidean space, and it would be too weak to represent a way more complicated topology with hundreds of thousands of hosts.
To avoid scenarios where the points aligned all over a straight line, a further improvement have been taken: the points don't start from the origin itself but in a random point inside the interval $[-0.5, 0.5]^d$, where $d$ are the dimensions of the coordinates: in this way the points don't start aligned and it's far more complicated for them to become it.

### B. Test of reduced-scale real world scenario

Now that we have an idea of the right behavior of the algorithm, we may use a real-world configuration scenario, reduced to the capacity of our resources, to run and see more serious result.
This time the used configuration includes 20 hosts, each pair far the same rtt; if we try using a 3-dimensional euclidean space we find out that the system won't converge at all, because it's impossible to set 20 points in a 3-dimensional space to have the same distance from each other, we could increase the number of dimensions but this solution wouldn't scale in a real world scenario with hundreds of thousands of host, so we adopt the non-euclidean space with the 3-dimensional height vector space, where we are able to set up $n$ points to the same distance with each other: indeed the solution should be for every node to have the same coordinates fixed in any point $\vec{x}$, and the height vector as the half of rtt. In this way we obtain that the predicted distance between two nodes $i$ and $j$ tends to the correct value

$$\begin{aligned}\text{rtt\_predicted} &= \|(\vec{x}_i, x_{ih}) - (\vec{x}_j, x_{jh})\| \\ &= \|\vec{x}_i - \vec{x}_j\| + x_{ih} + x_{jh} \\ &= \|\vec{x} - \vec{x}\| + \frac{\text{rtt\_real}}{2} + \frac{\text{rtt\_real}}{2} = \text{rtt\_real}\end{aligned}$$

## VI. Discussion

### A. Limits of the system

There are several causes that make the abosolute convergence of the coordinates impossible.

- Even when stabilized the protocol can't stop coordinates from rotating: this effect triggers the gossiping and so there is traffic even when coordinates seem stable.
- As described in [6] there are several causes that prevents the accuracy to be completely stable, and between them a source of problems is the drift: imagine to have peers that spent some time communicating with each other and figured out a sort of stable arrangement between them. The problem is that the origin isn't one of the points involved in the Vivaldi protocol, peers can actually be arranged correctly in any portion of the space, so a node can be 0 seconds or 5 years far from the origin, thus the coordinates can become huge, and overflows can happen.
- Another problem addressed in [7] is the high number of Triangle Inequality Violation (TIV) which limit the accuracy in local portion of the network.

### B. Future improvements

We can recognize several key features for enhancement of the system. Future improvements will focus on:

- Avoid network partitions in the membership protocol: this is an important aspect because their presence makes it impossible for the system to converge to the actual topology of the network.
- Let the coordinate not to diverge because of drift.
  A simple technique solves this problem and it's called "Gravity" [6]: the origin becomes the centroid of the

cluster of points, it is treated as a center of mass that attracts every other node, so at every updated a very small pushed toward the origin is applied. Nevertheless, the coordinates can still rotate but this is not a problem because the distance stays the same.

- Implement the coordinate snapshotting as in the serf tool [8], a decentralized gossip-based algorithm for cluster membership and failure detection which uses 8-dimensional height vector coordinates (this choice allows the system to be robust to every type of network topology the user actually deploys): the idea is that when an host shuts down or crashes, at the moment of its restart it will be very likely that the new coordinate is very close to last position it was before disconnecting, so if the host stores its coordinates on the disk and restores them when back, the effect of churn gets decreased, in fact the origin is more likely to be much farther from reality.
  So coordinates snapshotting makes the overall network a lot more stable.

## REFERENCES


[1] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, page 15–26, New York, NY, USA, 2004. Association for Computing Machinery.

[2] J. Ledlie, P. Pietzuch, and M. Seltzer. Stable and accurate network coordinates. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 74–74, 2006.

[3] Alberto Montresor. *Gossip and Epidemic Protocols*, pages 1–15. John Wiley Sons, Ltd, 2017.

[4] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8–es, aug 2007.

[5] Benedikt Elser, Andreas Förschler, and Thomas Fuhrmann. Tuning vivaldi: Achieving increased accuracy and stability. In Thrasyvoulos Spyropoulos and Karin Anna Hummel, editors, *Self-Organizing Systems*, pages 174–184, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[6] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, page 22, USA, 2007. USENIX Association.

[7] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of euclidean embedding for host-based network coordinate systems. *IEEE/ACM Transactions on Networking*, 18(1):27–40, 2010.

[8] https://www.serf.io/.