Tuning Vivaldi: Achieving Increased Accuracy and Stability

Benedikt Elser[1], Andreas Förschler[2], and Thomas Fuhrmann[1]

[1] Computer Science Department, Technical University of Munich, Germany
{elser,fuhrmann}@so.in.tum.de
[2] Computer Science Department, University of Karlsruhe, Germany
foerschler@so.in.tum.de

Abstract. Network Coordinates are a basic building block for most peer-to-peer applications nowadays. They optimize the peer selection process by allowing the nodes to preferably attach to peers to whom they then experience a low round trip time. Albeit there has been substantial research effort in this topic over the last years, the optimization of the various network coordinate algorithms has not been pursued systematically yet. Analyzing the well-known Vivaldi algorithm and its proposed optimizations with several sets of extensive Internet traffic traces, we found that in face of current Internet data most of the parameters that have been recommended in the original papers are a magnitude too high. Based on this insight, we recommend modified parameters that improve the algorithms' performance significantly.

1 Introduction

Self-organizing peer-to-peer systems cannot rely on a well-designed network topology. They rather form their overlay network topology on their own. For many such applications it is crucial that the resulting topology reflects the underlying topology of the Internet. Matching both topologies can, for example, reduce the latency which important not only for telephony applications. It can also increase the throughput, for example, when a file sharing application is able to find peers in the same organization.

Ideally, a peer would know which potential new neighbors provide a low latency. It is important that, to this end, we cannot first measure the latency and decide afterwards, because sampling the network in such a way has an extremely high overhead. It is thus necessary to predict the latencies.

A popular solution to this problem is the use of network coordinates. They assign positions in an euclidean space to the peers, so that their metric distance reflects the respective latencies. There are several such systems, but Vivaldi [1] has obtained the most attention. Its simple spring model is easy to understand and led to a practical implementation in the BitTorrent [2] client *Azureus* (now called Vuze).

The inclusion of the Vivaldi algorithm in Azureus sparked even greater interest in network coordinates. Ledlie et al. [3], e.g., proposed an optimization that changed the force computation in Vivaldi's spring model into a round based approach. Both this optimization of Vivaldi and the original algorithm contain several parameters. Their complex interaction offers various possibilities to fine tune the algorithm.

To the best of our knowledge, this fine tuning has not been studied systematically yet. In this paper we report on an in-depth analysis of Vivaldi and its optimizations. For our analysis we collected a number of latency matrices that complement other such data sets. We created a packet level simulator that uses these latency matrices to create a overlay between peers running the Vivaldi algorithm. Besides static latency matrices we also used dynamic trace-based input data for the simulator. Based on our analysis we then propose parameter choices that can improve the algorithms' performance significantly.

The rest of this paper is structured as follows: Section 2 gives an overview of Vivaldi and its proposed optimizations. Section 3 describes the data sets and simulation method that we used for our analysis. Section 4 discusses the most interesting results from our analysis. Section 5 gives a brief overview of related work. Section 6 concludes with an outlook to future work.

2 Overview of Vivaldi

The Vivaldi algorithm models a spring network: The peers are the endpoints of springs whose length is set to the actually measured round trip time (RTT) between the peers. The underlying metric combines the Euclidean coordinate distance with an additional 'height' displacement. Based on continued RTT measurements, the peers update their coordinates according to their displacement error estimations. Two constants c_c and c_e control these updates. Algorithm 1 gives a summary of this Vivaldi NE algorithm in pseudo code.

Studies of the Azureus client revealed problems with the algorithm in real live situations [3]. First of all, a node does not contact its peers equally often. The resulting imbalance has a negative impact on the global optimization process. Secondly, real world influences inevitably create spikes in the RTT measurements, which distort the coordinates of an otherwise stable system massively.

Ledlie et al. [3] proposed two improvements to the Vivaldi NE algorithm that compensate for these problems: A low pass RTT filer based on median values includes only plausible RTT values. A *neighbor decay formula* addresses the imbalanced measurement frequencies. To this end, each node keeps a list of its most recently used peers (*recent neighbor set*). It contains those peers with whom a node ran the Vivaldi algorithm at most a time e_t ago. The force vector F, which changes the peers' coordinates, is then modified as follows

$$\tilde{F} = \sum_{j=1}^{N} F_j \cdot \frac{a_{max} - a_j}{\sum_{i=0}^{n-1} a_{max} - a_i} \tag{1}$$

where $a_{max} \leq e_t$ is the maximum age of an entry in the recent neighbor set, a_k is the age of the kth entry, and F_k is the force pushing in the direction of that entry. Algorithm 2 summarizes this Vivaldi ND variant in pseudo code.

The most important contribution of this revised algorithm is that it does not only take the most recent measurement into account, but that many measurements jointly contribute to the coordinate adjustment. This greatly reduces spikes and other fluctuations.

Algorithm 1: Original Vivaldi Neighbor Error Algorithm [1]

Input:
x_i: local coordinate of node i
x_j: remote coordinate of node j
RTT: RTT to node j
e_i: error estimation of node i
e_j: error estimation of node j
c_c, c_e: constants

Output:
x_i: updated coordinate of node i
e_i: updated error estimation of node i

function vivaldiNE$(RTT, x_i, x_j, e_i, e_j, c_e, c_c)$
begin
  $w \leftarrow \frac{e_i}{e_i+e_j}$
  $e_s \leftarrow \frac{|\|x_i-x_j\|-RTT|}{RTT}$
  $e_i \leftarrow e_s \times c_e \times w + e_i \times (1-c_e \times w)$
  $\delta \leftarrow c_c \times w$
  $x \leftarrow x + \delta \times (rtt - \left\|x_i - x_j\right\|) \times u(x_i - x_j)$
  return (x_i, e_i)
end

In order to judge this and other improvements quantitatively, we use the following three indicators:

1. A node i's error is the median over all absolute errors between all node pairs (i, j). The *median error* of the system is the median of all node errors [4].

2. The *relative application-level penalty* (RALP) [5]

$$RALP \equiv \frac{1}{n} \cdot \sum \frac{v_i - p_i}{p_i}, \tag{2}$$

describes the penalty that a node experiences when choosing a peer based on network coordinates, as compared to the perfect choice that an omniscient oracle could recommend based on 'real' RTTs. Here, p a sorted list of *measured* RTTs between a peer and its neighbors, and v is a sorted list of *predicted* RTTs between the peer and its neighbors.

3. We define the *degree of stability* that the embedding reaches according to [3] as

$$stability \equiv \frac{\sum_i \Delta x_i}{\Delta t} \tag{3}$$

where Δx_i as the drift of x_i in the time period Δt.

3 Data sets and Simulator

In our analysis we used four data sets:

Algorithm 2: The Neighbor Decay Optimization of the Vivaldi Algorithm [3]

Input:
x: locale coordinate of node
$Y = \{y_1, y_2, \ldots, y_k\}$: coordinate of nodes from the k sized neighbor set
$R = \{rtt_1, rtt_2, \ldots, rtt_k\}$: RTT to nodes in the neighbor set
$A = \{a_1, a_2, \ldots, a_k\}$: time of last contact with all nodes in the neighbor set
t: constant damping adjustment of local coordinate, similar to VivaldiNE's c_c

Output:
x: updated coordinate of local node

```
1  function VivaldiND(x, Y, R, A, t)
2  begin
3      s ← 0
4      F ← 0
5      a_max ← max{a_1, a_2, ..., a_k}
6      for i = 1 to k do
7          s ← s + a_max − a_k
8      for i = 1 to k do
9          e ← rtt_i − ||x − y_k||
10         F_i ← e × u(x, y_k)
11         F ← F + F_i × (a_max − a_i) / s
12     return x ← x + t × F
13 end
```

- **Azureus-to-PlanetLab** is the trace from Ledile et al. [3]. It yields a 249x249 RTT matrix.
- **MITKing** is the data set of Dabek et al. used to derive the original Vivaldi algorithm [1]. It is based on measurements with the King technique among 1740 DNS servers.
- **KingBlog** is a dataset similar to the MIT King data. It was extracted from 2500 DNS servers [6].
- **Dynamic PlanetLab Dataset** is a dynamic trace of 13,4 million single measurements between 83 fully interconnected PlanetLab nodes, which we collected between March 6 and 9, 2009.

In order to evaluate the different Vivaldi variants, we built a simulator that takes either the static RTT matrices or the dynamic RTT traces as input. The method of deriving a Vivaldi simulation from a static matrix of pairwise RTTs was introduced by Cox et al. [1]. We extended their idea with a dynamic trace-based simulation.

On startup, the simulator randomly chooses a neighborhood of 32 peers for each node. In the static case, the simulator also determines a sequence of RTT measurements. Here, making a measurement means to look up the RTT from the RTT matrix. In the dynamic case, the sequence of measurements in the trace is predefined. A node obtains the RTT values for all the peers in its respective neighborhood and calculates an estimated RTT according to the describe median filter.

There are two variants of how to determine the sequence of RTT measurements in the static case: In [1] a node starts a new measurement immediately after the previous

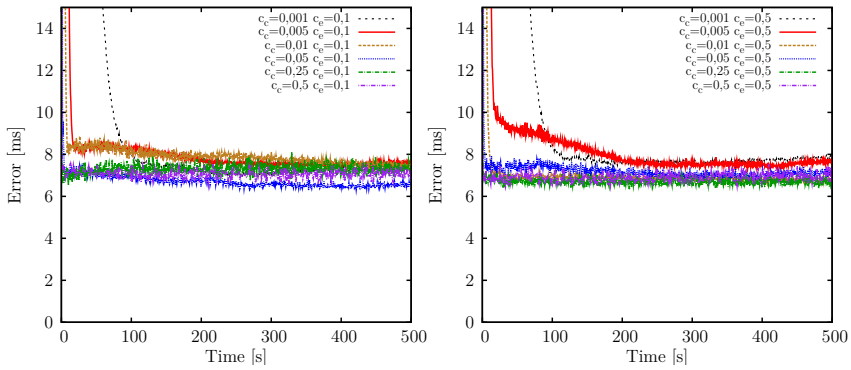

Fig. 1: Embedding error (Vivaldi NE CA, Azureus dataset)

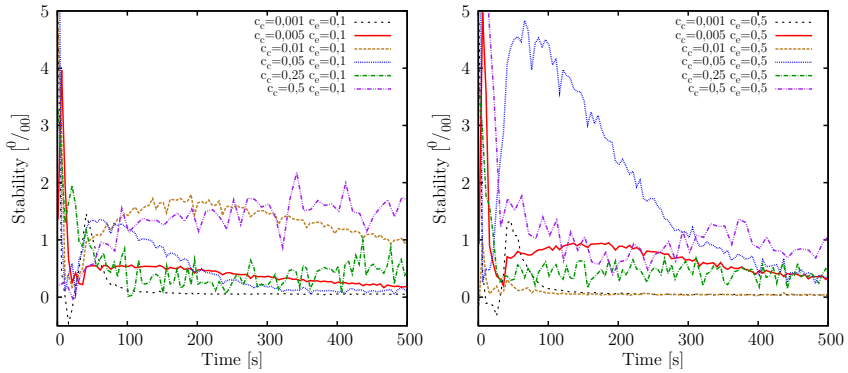

Fig. 2: Stability (Vivaldi NE CA, Azureus dataset)

measurement has completed (*continuous adjustment*, CA). This leads to an imbalance, because nodes with low RTT conduct more rounds of the Vivaldi algorithm. Another drawback is the huge traffic overhead that those continuous measurements cause. In order to avoid this imbalance and reduce the overhead, we propose to use a predefined average time interval for the measurement rounds (*uniform adjustment*, UA).

4 Results

In our extensive study [7] we analyzed all the proposed algorithm variants with the four datasets and with various parameter settings. Here we briefly discuss the most interesting results. All use the described Vivaldi variants with four dimensions and a height component. In case of uniform adjustment (UA), we chose an interval of 10 seconds.

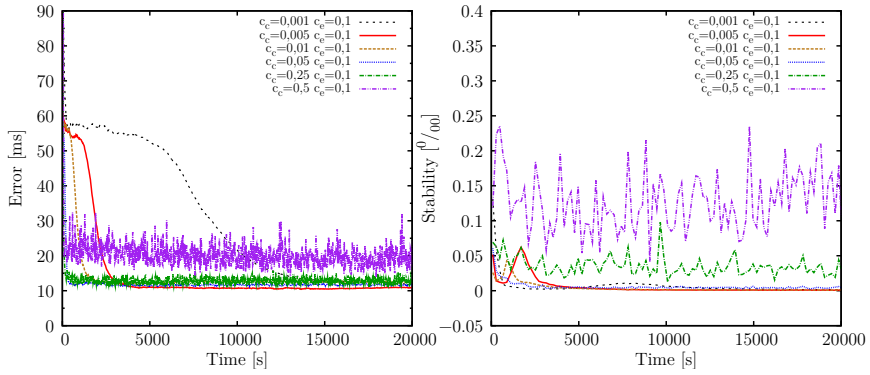

Fig. 3: Embedding error and stability (Vivaldi NE UA, Azureus dataset)

Fig. 1 shows the error, figure 2 the stability of the original Vivaldi Neighbor Error algorithm with continuous adjustment (CA) and different values for c_c and c_e using the Azureus-to-PlanetLab dataset. In accordance to Cox et al. [1] we find that smaller values for c_c and c_e lead to more stable coordinates. But as our analysis shows, this improved stability is only relevant in the initial phase: After 500 seconds, the stability of the embedding is similar for all parameter combinations. Moreover, we find through the course of our simulations that c_e has only little influence on the embedding error and the stability. A conclusion that is also supported from both figures.

Fig. 3 shows the same scenario for the uniform adjustment (UA) variant. Due to the much lower measurement frequency the shown time scale covers a larger range. In all settings we see that both stability and median error never reach the accuracy of the CA variant. Furthermore, we see that small c_c values delay the convergence enormously, whereas large values lead to great instabilities. For large c_c values the low stability also leads to a high, fluctuating embedding error. As a result we find an optimal parameter choice at $c_c = 0,005$. (We do not show different c_e values, because they have only little effect.)

Fig. 4 and 5 show the Neighbor Decay optimization of Vivaldi for different RTT probing intensities. In the CA variant in figure 4 we show the results for $e_t = \infty$, This parameter choice causes the peers to adjust their position relative to all the peers that they ever contacted. In the UA variant in figure 5 we use $e_t = 120\,\text{ms}$, where the 120 ms correspond to the median value of the RTTs. This choice causes the peers to adjust their position only relative to active peers, i.e. those that have just send their RTT information. Our results demonstrate that the UA variant produces the results from Ledlie et al. within a factor of two, while requiring only sparse RTT probing (10 sec versus 120 ms). However the results also show a rather poor performance of the algorithm in terms of the median embedding error. Furthermore we learn again that there is an optimal choice for the algorithm's parameter, namely $t = 0,005$.

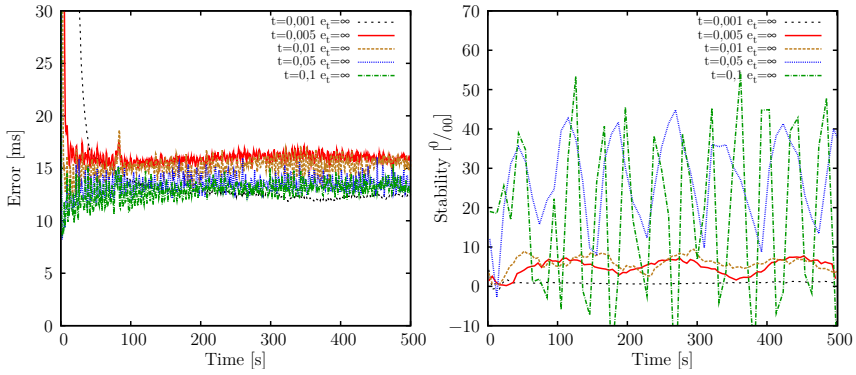

Fig. 4: Vivaldi ND with Azureus dataset (CA)

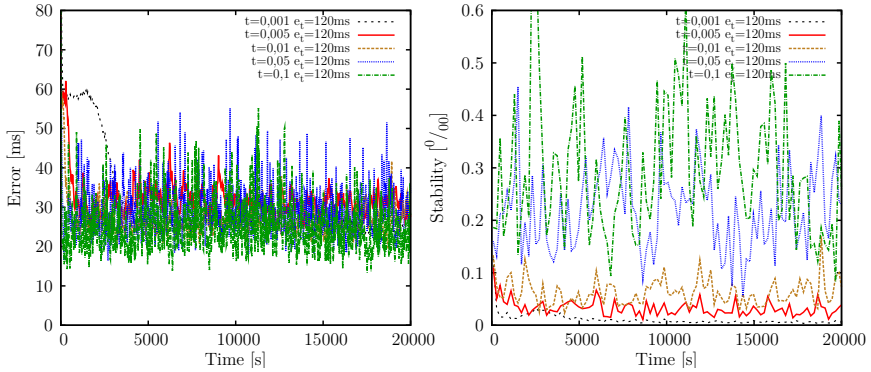

Fig. 5: Vivaldi ND with Azureus dataset (UA)

In order to complement our analysis with an application-level metric, we further analyzed the different variants using the RALP quality measure. Fig. 6 shows the results for continuous adjustment (left) and uniform adjustment (right) using the Azureus dataset, while figure 7 illustrates the same for the KING Blog dataset. Clearly, all variants improve over the random peer selection case, which does not use network coordinates at all.

Similarly to almost all of our measurements, the difference between the two adjustment variants depends on the dataset. Simulations with the KING dataset behave comparably for all variants (NE, ND, w/ and w/o CA, UA) of the Vivaldi algorithm, whereas the Azureus dataset exhibits a significant dependence on the parameter choices. In particular, when using the uniform adjustment variant, the original neighbor error algorithm outperforms the neighbor decay optimization. This indicates that the claimed improvement of ND [3] might not carry over to a general application 'in the wild', despite the suggestive title of the respective publication.

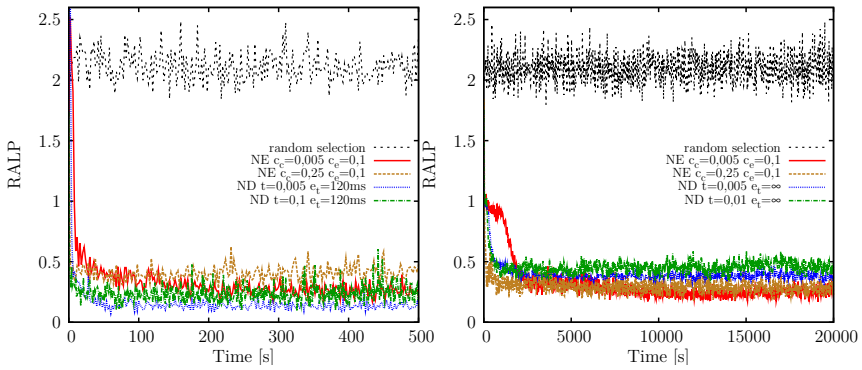

Fig. 6: RALP with Azureus data set, using CA (left) and UA (right)

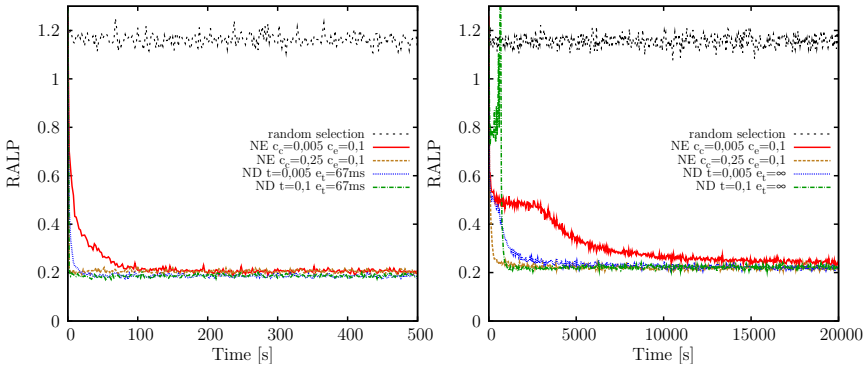

Fig. 7: RALP with King data set, using CA (left) and UA (right)

In order to better understand highly dynamic systems and their differences to static systems, we analyzed the PlanetLab RTT trace. Fig. 8 shows the embedding error for the original NE algorithm and the ND optimization with the median filter proposed by Ledile et al. [3], figure 9 shows the same data without this enhancement. First of all, we do not observe any significant effect of the median filter. Furthermore, the advantage of the neighbor decay optimization manifests most, when it is compared to a high c_c parameter in the original neighbor error algorithm. When $c_c = 0,01$ NE and ND behave almost identically.

Fig. 10 illustrates the same setup stability wise. All measurements show large instabilities for large values of c_c and t respectively. Again we conclude that the choice of large values for c_c and t does not only lead to faster convergence (as stated in literature), but also to an increased instability that in the end leads to a worse quality of embedding. As a result, we conclude that even though the neighbor decay optimization can yield a

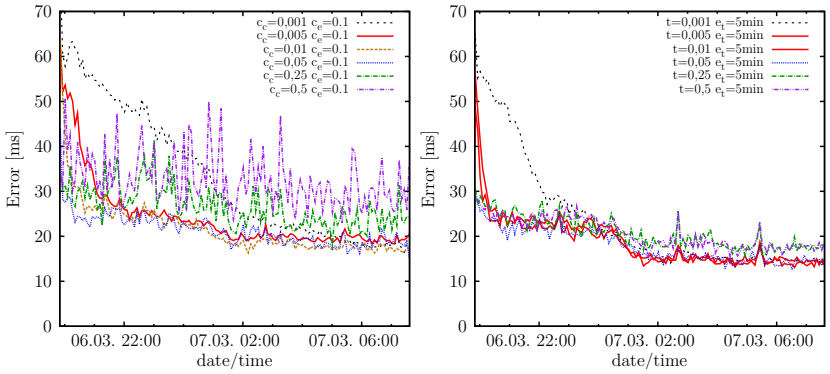

Fig. 8: Vivaldi NE (left) vs. ND (right) with median filter

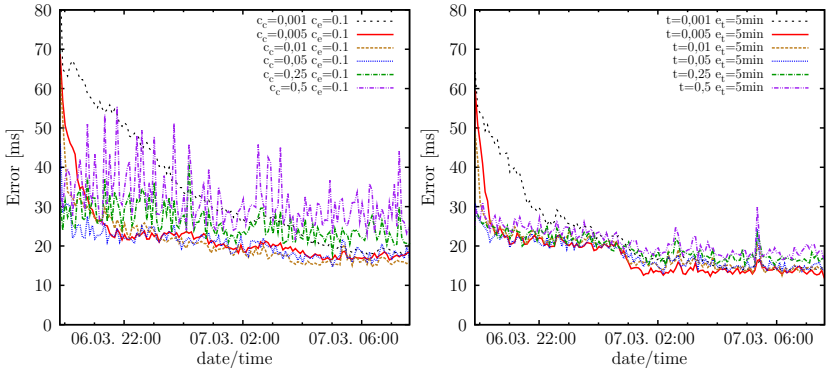

Fig. 9: Vivaldi NE (left) vs. ND (right) without median filter

better stability in principle, it does not improve the embedding for the moderate values of t and c_c that overall yield the best results.

Finally we also checked our finding in the dynamic setting using RALP (cf. fig. 11). Here we find that the neighbor decay optimization shows a better performance than the original Vivaldi NE algorithm. Again, $c_c = 0.005$ and $c_e = 0.1$ outperform all other parameter combinations in the NE case. For the ND case $t = 0.005$ is the best choice.

5 Related Work

Finding neighbors with low RTT in an overlay network is an important issue. Several other works besides Vivaldi address the same problem. Global network positioning (GNP) [8] proposed to use RTT measurements to landmark servers as components for network coordinates. Vivaldi adopted the idea of network coordinates, but replaced the

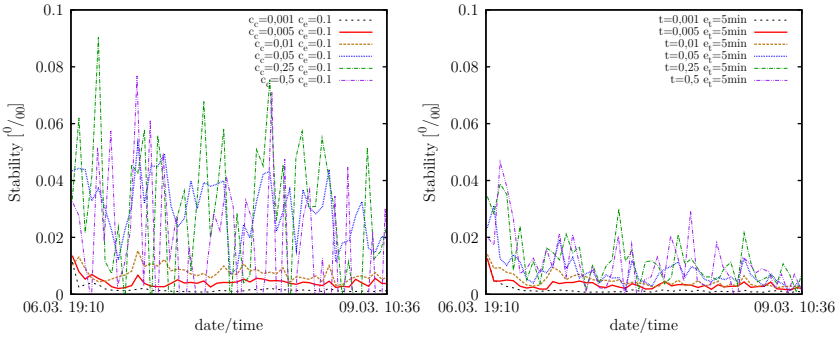

Fig. 10: Vivaldi NE (left) and ND (right) with median filter

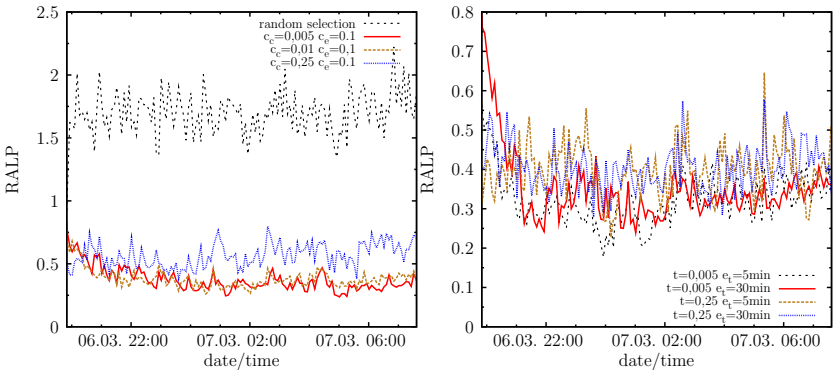

Fig. 11: RALP with NE (left) and ND (right) for the dynamic data trace

fixed set of landmarks with its spring approximation model. This makes Vivaldi a self-organizing system in the sense that is independent from a landmark infrastructure.

Meridian [9] is a gossip based system. It sorts a node's peers into exponentially growing buckets according to the respectively measured RTTs. When a node queries one of its peers for recommended further peers, it obtains answers from the bucket that matches the querier's RTT. This algorithm induces an iterative process that allows the peers to find proximate peers. Comparing Medidian to Vivaldi, we see that the need for direct measurements between the peers leads to increased costs that are not present in network coordinate systems [10].

Ono [11] derives peer proximity from querying a content distribution network (CDN) to resolve a vector of DNS names. It assumes that peers that receive similar results from the CDN reside in the same AS and wile likely have a low mutual RTT. This is similar to GNP, but replaces the RTT measurements to the landmark servers with the CDN queries. Aggarwal et al. introduce in [12] the idea of an oracle that pursues a similar goal as Ono. But instead of living on a CDN the oracle is an ISP-operated recommen-

dation server that allows peers to obtain lists of proximate peers. Contrary to Ono and the oracle we aim at improving fully decentralized solutions that do not require the ISP to support the overlay network.

6 Conclusion

In this paper we studied Vivaldi and its proposed optimizations both with static RTT matrices and dynamic RTT traces. We found that in the original works, most constants are chosen too high to produce stable coordinates in dynamic real world settings. Seemingly the respective authors decided to sacrifice the stability of the resulting embedding to obtain a faster convergence. We believe this to be a bad trade, especially because most parameters produce similar relative embedding errors.

As a result of our analysis we recommend the following values: $c_e = 0.1$, $c_c = 0.005$, $t = 0.005$, and $e_t = 30\,\text{min}$. In our analysis we also spotted rare cases of especially poor performance of the Vivaldi algorithm family. We therefore recommend using conservatively low parameters to also safeguard against such cases.

Especially, we could not reproduce the claimed drastic superiority of the Neighbor Decay optimization method proposed by Ledlie et al. [3]. The only effect we could reproduce was a faster convergence, which we consider not as important as stability.

What struck us at most in our study were the large differences between the datasets. Even when comparing static data among each other, we found significantly different reactions of the different algorithm variants. The differences between the results from the static RTT matrices and the dynamic RTT traces were even greater. Ledlie et al. [3] argue that this depends on the magnitude of RTTs that the algorithm receives. We could not confirm this hypothesis and therefore recommend further research in that area, to better understand the cause of the large deviations.

References

1. Cox, R., Dabek, F., Kaashoek, F., Li, J., Morris, R.: Practical, distributed network coordinates. In: Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II), Cambridge, Massachusetts (2003)
2. Cohen, B.: Incentives build robustness in bittorrent. In: Proceedings of the first Workshop on Economics of peer-to-peer systems, Berkley, California (2003)
3. Ledlie, J., Gardner, P., Seltzer, M.: Network coordinates in the wild. In: Proceedings of USENIX NSDI'07, Cambridge, Massachusetts (2007)
4. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: Proceedings of the ACM SIGCOMM '04 Conference, Portland, Oregon (2004)
5. David R. Choffnes and Fabian B. Bustamante: What's Wrong with Network Positioning and Where Do We Go From Here? Technical Report NW-EECS-09-03 (2009) available online `http://www.eecs.northwestern.edu/research/tech_reports/number`.
6. Syrah: King-Blog-Dataset, accessed 30. December 2008. `http://www.eecs.harvard.edu/~syrah/nc` (2006) Variant with at least 8 neighbors was picked.
7. Andreas Förschler: Einbettung von Peer-to-Peer-Netzwerkgraphen zur Vorhersage von Paketumlaufzeiten (2009) Diplomarbeit, Universität Karlsruhe.

8. Ng, T.S.E., Zhang, H.: Predicting Internet Network Distance with Coordinates-Based Approaches. In: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) 2002, New York (2002)
9. Wong, B., Slivkins, A., Sirer, E.G.: Meridian: a lightweight network location service without virtual coordinates. In: Proceedings of the ACM SIGCOMM '05 Conference, Philadelphia, Pennsylvania (2005)
10. Pietzuch, P., Ledlie, J., Seltzer, M.: Supporting network coordinates on PlanetLab. In: WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems, Berkeley, California (2005)
11. Choffnes, D.R., Bustamante, F.E.: Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. Comput. Commun. Rev. **38**(4) (2008) 363–374
12. Aggarwal, Vinay and Feldmann, Anja and Scheideler, Christian: Can ISPS and P2P users cooperate for improved performance? Comput. Commun. Rev. **37**(3) (2007) 29–40