Supporting Network Coordinates on PlanetLab

Peter Pietzuch, Jonathan Ledlie, Margo Seltzer
Harvard University, Cambridge, MA
hourglass@eecs.harvard.edu

Abstract

Large-scale distributed applications need latency information to make network-aware routing decisions. Collecting these measurements, however, can impose a high burden. *Network coordinates* are a scalable and efficient way to supply nodes with up-to-date latency estimates. We present our experience of maintaining network coordinates on PlanetLab. We present two different APIs for accessing coordinates: a per-application library, which takes advantage of application-level traffic, and a stand-alone service, which is shared across applications. Our results show that statistical filtering of latency samples improves accuracy and stability and that a small number of neighbors is sufficient when updating coordinates.

1 Introduction

Collecting up-to-date latency measurements between nodes in an overlay network is important for many classes of applications. Proximity-aware distributed hash tables use latency measurements to reduce the delay stretch of lookups [15], content distribution systems construct network-aware trees to minimize dissemination times [1], and decentralized web caches need latency information to map clients to cache locations. Especially in a wide-area network, communication latencies have a significant impact on the overall execution time of operations.

To exploit network locality, today's overlay networks are left with the burden of performing their own network measurements. Developers must continually reinvent the wheel duplicating measurements when multiple network-aware overlays are sharing a single distributed testbed, such as PlanetLab [19]. Implementations that gather all-pairs latency measurements are only scalable for relatively small overlay deployments. For example, the all-pairs ping service managed by Stribling [18] has recently ceased operation because it became infeasible to obtain up-to-date measurements for over 500 PlanetLab nodes. In addition, measuring techniques that lead to good latency samples without suffering from high variance caused by measurement anomalies are non-trivial.

To address these issues, a *latency service* can provide applications with up-to-date estimates of network latencies between nodes. We describe our experience of maintaining such a service on PlanetLab based on *network coordinates*. Here, each overlay node maintains a coordinate obtained through an embedding of latency measurements in a metric space. The Euclidean distance between two coordinates is an estimate of the communication latency between the nodes. This enables nodes to infer latencies to remote nodes without the overhead of a direct latency measurement. The metric space interpolates non-existing measurements, which reduces the measurement overhead from $O(n^2)$ to linear in the number of nodes.

We discuss trade-offs between two different solutions for a network coordinate service: a dedicated, stand-alone *service*, which is shared among applications, and a per-application *library*, which exploits application-specific traffic for network coordinate updates. Our experience deploying network coordinates on PlanetLab reveals that coordinate stability and convergence is a challenge. We have developed two techniques to address this: statistical filtering of latency samples and decoupling low-level coordinate updates from the coordinates used by applications. We have found that our implementation of a latency service now provides network coordinates that are sufficiently stable and accurate to support our application needs, while keeping the measurement overhead small.

After a survey of existing work in Section 2, we present the APIs and trade-offs of our network coordinate service and library in Section 3. In Section 4, we show how statistical filtering and our delayed update technique greatly improves accuracy and stability and how a small number of measurement neighbors can lead to accurate coordinates. We conclude in Section 5.

2 Latency Service

A latency service enables overlay nodes to obtain latency estimates to other nodes. We adopt the following design goals for our latency service.

1. **Good accuracy.** Latency estimates between nodes should have a relatively low error but the required accuracy depends on the application. For example, if the latency estimate is used to select the nearest node, a certain error is tolerable as long as it does not affect the result. The latency service must also achieve its accuracy goal when network latencies are changing due to BGP route updates or congestion.
2. **Low measurement overhead.** The latency service should minimize latency probing to conserve network resources. Latency measurements should use application data packets between nodes when possible. Note that there is a tension between the achievable accuracy and the measurement overhead.
3. **Quick latency prediction.** Many applications require quick decisions based on latencies between nodes. The latency service itself should not introduce a long delay when queried for latency estimates.
4. **Scalability.** The design of the latency service must be

scalable in terms of the number of nodes in the network for which latency measurement are required.

5. **Simple application integration.** It should be easy to run the latency service and for an application node to obtain latency estimates. The latency service should have an intuitive API and any node should be able to use the latency service.

2.1 Previous Work

Several research groups have recognized the need for a latency service on the Internet. Unfortunately, many current proposals for latency services make a poor trade-off between accuracy and overhead, are not widely deployed, require changes to the network, or have scalability issues. In this section, we provide a survey of latency services and determine their compliance with our design goals.

The simplest latency service gathers all pairs latency information and makes this data available to all nodes via a centralized location, as exemplified by the *all pairs ping service* [18] on PlanetLab. Such an approach causes a large amount of measurement traffic because every node measures latencies to every other node.

IDMaps [5] is a latency service that attempts to minimize measurement traffic. It uses a network of *tracers* that proactively measure distances between themselves and to representative nodes from each address prefix. This information is used to create a virtual distance map of the Internet. Since only tracers measure latency, the overhead is kept low but the prediction error is determined by the distribution of tracer locations. Achieving a good distribution is hard because the physical network topology is not known in practice.

The *Internet Iso-bar* [2] system attempts to remove the requirement of topology knowledge by dividing network nodes into clusters depending on latencies. A node from each cluster is then selected to monitor intra- and inter-cluster latencies and to respond to latency queries. However, the accuracy of the system depends on how amenable the network is to clustering. The cluster size determines the measurement overhead.

Ratnasamy *et al.* propose a latency service that attempts to reduce the number of network measurements even further [14]. Nodes measure their network distance only to a small number of *landmark nodes* and use the results to partition themselves into *bins*. Nodes that fall within the same bin are deemed to be close. Although this scheme vastly reduces the measurement overhead compared to other systems, it also exhibits a high error due to the coarse-grained assignment to a fixed number of bins.

Nakao *et al.* observe that much of the network information that applications are interested in is already collected by lower network layers. They propose to exploit this through a *routing underlay* [11], which provides a standardized interface for applications to inspect the state and structure of the network. Although an underlay would provide efficient access to network information already gathered by routers, it requires changes to routers.

2.2 Network Coordinates

A latency service can be constructed using a *network embedding* [8, 12] that embeds measured latencies between nodes in a low-dimensional geometric space. Each node maintains a *network coordinate (NC)*, with the goal that the Euclidean distance between two NCs is an estimate of physical network latency. Two classes of algorithms were proposed to compute NCs: *landmark-based* schemes, such as *GNP* [12], *Lighthouses* [13], and *PIC* [3], which use a fixed number of landmark nodes for NC calculation, and *simulation-based* approaches, such as *Vivaldi* [4] and *BBS* [16], which model NCs as entities in a physical system. Since one of our design goals for the latency service is scalability, we adopt a fully-decentralized, simulation-based approach for our NC service.

```
VIVALDI($\vec{x_j}, w_j, l_{ij}$)
1  $w_s = \frac{w_i}{w_i + w_j}$
2  $\epsilon = \frac{|\|\vec{x_i} - \vec{x_j}\| - l_{ij}|}{l_{ij}}$
3  $\alpha = c_e \times w_s$
4  $w_i = (\alpha \times \epsilon) + ((1 - \alpha) \times w_i)$
5  $\delta = c_c \times w_s$
6  $\vec{x_i} = \vec{x_i} + \delta \times (\|\vec{x_i} - \vec{x_j}\| - l_{ij}) \times u(\vec{x_i} - \vec{x_j})$
```

Figure 1: *Vivaldi* update algorithm.

The *Vivaldi* algorithm calculates NCs as the solution to a spring relaxation problem. The measured latencies between nodes are modeled as the extensions of springs between massless bodies. A network embedding with a minimum error is found as the low-energy state of the spring system. Each node successively refines its NC through periodic updates with other nodes in its *neighbor set*.

Figure 1 shows how a new observation, consisting of the remote node's NC $\vec{x_j}$, its confidence w_j, and a latency measurement l_{ij} between the two nodes, i and j, is used to update the local NC. The *confidence* w_i quantifies how accurate a NC is believed to be. First, the *sample confidence* w_s (Line 1) and the *relative error* ϵ (Line 2) are calculated. The relative error ϵ expresses the accuracy of the NC in comparison to the true network latency. Second, node i updates its confidence w_i with an exponentially-weighted moving average (Line 4). The weight α is set according to the sample confidence w_s (Line 3). Also based on the sample confidence, δ dampens the change applied to the NC (Line 5). As a final step, the NC is updated in Line 6 (u is the unit vector). Constants c_e=c_c=0.25 affect the maximum impact an observation can have on confidence and NC, respectively [6]. We define the *stability* of a NC as its total change over time in $\mathrm{ms/s}$.

3 Architecture

A latency service based on NCs exploits several properties of NCs that help satisfy the design goals from Section 2.

- NCs achieve good accuracy on Internet topologies. Although an embedding error arises because Internet latencies violates the triangle inequality, these violations are not severe enough to prevent a metric embedding in practice. Previous work [4] has found a median relative error of 11%, which we confirmed on PlanetLab.
- Non-existent measurements between nodes are interpolated by the network embedding, thus reducing the measurement overhead. The trade-off between measurement overhead and accuracy is made explicit by NCs. The accuracy and convergence of NCs can be improved by increasing the measurement frequency and

extending the neighbor set.

- NCs provide almost instantaneous latency predication because they do not actively initiate new latency measurements to respond to latency queries. Active measurement approaches, such as Meridian [20], may introduce a non-trivial delay while a fresh latency estimate is being obtained.
- The decentralized algorithm for computing NCs makes the implementation scalable to a large number of nodes. We have successfully deployed a NC service on over 300 PlanetLab nodes.

To achieve simple application integration, we propose two different architectures: a stand-alone *NC service* and a per-application *NC library*. Both approaches have the advantage that they provide a correct implementation of NC to applications. As will be explained in Section 4, the application programmer does not have to deal with the complexity of latency measurement.

Network Coordinate Service. If the network infrastructure is cooperative and under control of a single authority, such as PlanetLab, an efficient solution is to deploy a *NC service* on all the nodes. Each application then accesses the locally running NC service. This has the advantage that the cost of inter-node measurements is amortized across all applications that share the service. A drawback of this approach is that parameters, such as the measurement frequency, which determines the convergence of the NCs, must be set globally for all applications.

```
double   estimateLat   (double[] remoteNC)
double[] getNC         ()
double   getConfidence ()
double   getRelError   ()
double   forceUpdate   (IPAddr remoteNode)
```

Above we show the API of the latency service that is part of our SBON deployment [17] on PlanetLab. The function `estimateLat` returns the latency estimate between a local and remote node given the remote node's NC. The local NC and confidence are returned by the `getNC` and `getConfidence` calls, respectively. A call to `getRelError` returns the current median relative error over the last n latency measurement that were used for coordinate updates. If the application needs an up-to-date latency to a remote node, a call to `forceUpdate` causes the NC service to perform a measurement to the remote node returning the observed latency. This API assumes that nodes in a distributed application are identified as an IP address and NC pair, `(IPAddr,NC)`. As a result, any node can obtain a latency estimate to another node about which it has learned.

Network Coordinate Library. In some cases, an application should include a module for latency estimation without relying on an externally running service. This is true for peer-to-peer applications that are deployed on a varying set of heterogeneous nodes. To address this, we also propose a *NC library* that any application can link against to support NCs. In order to avoid duplicating functionality, the library handles only the computation of coordinates but leaves the actual network communication for network probing to the application. This enables the application to exploit application traffic as much as possible for measurements.

```
void     updateNC      (IPAddr remoteNode,
                        double[] remoteNC,
                        double remoteConf,
                        double latency)
--------------------------------------------------
void     forceUpdate   (IPAddr remoteNode)
```

In addition to the functions provided by the stand-alone service, the NC library API has a function `updateNC` that is used by the application to feed in new network measurements from application-level traffic. Only if the application-level traffic is not frequent enough or does not cover a large enough set of nodes to compute an accurate NC does the library request additional latency measurements from the application. As will be explained in Section 4.2, the NC library monitors its relative error to decide if the NC is converging sufficiently. If this is not the case, it uses the `forceUpdate` callback to the application to request more diverse measurements by initiating a latency measurement to a new remote node.

4 Implementation Issues

Regardless of whether NCs are accessed through a service or a library, they must be designed to handle practical networking problems, such as measurement variation, data loss, and node failures. The focus of our work to date has been on measurement variation: creating an accurate and stable coordinate system using real world latency samples. In this section, we explain our solutions to handle non-ideal latency samples, which have a significant negative impact when left unfiltered. We also describe how measurement overhead can be controlled by tuning neighbor sets. As an overview, we found that:

- Latency samples for a particular link have a high variance. These raw samples can cause wild, temporary perturbations, which cascade across the coordinate space. We found that a statistical filter suppressed these anomalies and greatly improved accuracy and stability.
- Nodes' relative latencies change over time. We refine a filter to remove bad measurements while preserving changes in the underlying network. This sustains accuracy over time.
- Applications prefer stable coordinates. A distinction between *system-level NCs*, which are raw Vivaldi coordinates, and *application-level NCs*, which summarize a recent window of coordinate updates, help suppress unnecessary application activity. A new coordinate is only externally visible to an application after a significant change has occurred.

4.1 Measuring Latency

During our initial deployment of NCs on PlanetLab, we observed latency samples of as much as three orders-of-magnitude greater than the normal latency for a given link. When used for NC calculation, these samples induce a large coordinate change in a high confidence node, which, in turn, causes large shifts in the NC of its neighbors. Such changes keep propagating through the coordinate space, causing high instability, low convergence, and decreased accuracy, because coordinate shifts are not reflecting future measurements. Occasionally, but not always, we could attribute large values in our application-level measurements to high CPU load on one of the nodes.

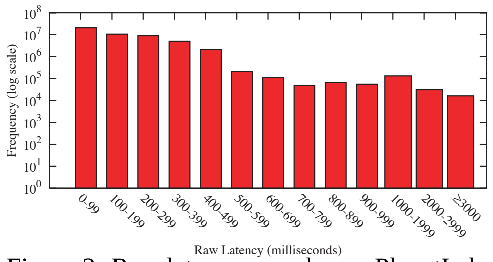

Figure 2: Raw latency samples on PlanetLab.

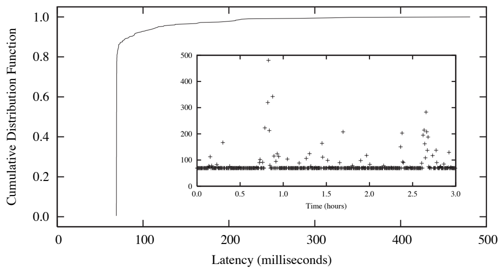

Figure 3: Kernel-level ping measurements.

To illustrate the extent of the anomalies, we show a distribution of application-level UDP latency samples between 269 nodes collected over three days on PlanetLab in Figure 2. Over 0.4% of the samples are greater than one second, larger than even a slow intercontinental link, and frequent enough to periodically distort the coordinate space. Not only is a significant fraction of samples large, but also individual links have extended tails: samples of a link tend to produce a consistent latency within a tight range, but then a tail of the samples can extend into the tens of seconds. Both the range and tail depend on the link. We found that feeding these raw samples directly into Vivaldi leads to poor accuracy and stability.

We also tried using kernel-level ping measurements and found that they suffered from a similar baseline and extended tail. Figure 3 shows the results of a three hour set of ping measurements using the `ping` program between two PlanetLab nodes (`berkeley` to `uvic.ca`). The data shows that 82% of the samples fall within 1ms of the median, but that the largest 5% are 2–7 times the median. Even though the measurements are being timestamped by the kernel, there are many large measurements that would jolt a stable coordinate system. In addition, as the subgraph shows, the deviations from the baseline measurement are not clustered all at one time, but occur throughout the trace; they do not signal shifts, but aberrations. Because kernel-level measurements would need to be filtered also and do not have the benefit of application-level traffic, we decided to find a way to incorporate samples with a high variance into the NC computation.

Although we estimate that approx. 90% of links fall into the type shown in Figure 3, a small percentage do exhibit multi-modal behavior. If multi-modal behavior was on a short time scale, it would be unclear what value would be appropriate to feed into the NC update algorithm; it might perhaps require a more complicated link description (*e.g.,* a PDF). However, we have not seen this behavior in practice. Instead, we have found cases of long-term, periodic bi-modal link latencies, as shown in Figure 4 (`ntu.edu.tw` to `6planetlab.edu.cn`).

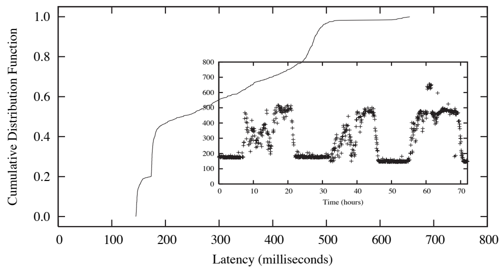

Figure 4: Long-term, periodic, bimodal latency samples.

That this behavior is long-term is important for two reasons: (1) it appears reasonable to summarize each link with a single current baseline latency and (2) NCs need continuous maintenance because this baseline latency changes over time.

Statistical Filtering. We explored three obvious but ineffective approaches before arriving at our final solution for latency signal extraction. First, we tried using *simple thresholds*: if an observation is larger than a constant, it is ignored. This did not work because one link's normal latency was well into the range of the tail of another link. Second, we applied an *exponentially-weighted moving average (EWMA)* to each link's sample history. We found that this performed worse than no filter at all because it weighted outliers too strongly, even with unusually low weights. Finally, we tried a more Vivaldi-specific solution: *lowering confidence* in response to high load. However, because sample variance can only partially be attributed to load, this solution was also not effective.

Instead, we found that a non-linear *moving percentile (MP)* filter greatly improved accuracy and stability. The MP filter takes two parameters: a window size of samples and the percentile of these samples to output. It removes noise and, based on the window size, responds to changes in the underlying signal. Before presenting our experimental results, we introduce a technique layered on top of the filtered raw coordinate.

Application-Level NCs. Our latency service makes a distinction between *system-level* and *application-level* NCs. The former are raw Vivaldi coordinates, which are updated with each observation. The latter are the application's idea of the local NC, updated only when a statistically significant change in the system-level NC has occurred. While some applications may want to access the raw value, many others prefer updates when the system-level NC exhibits sustained change compared to its past.

We found two successful heuristics for setting application-level NCs, both based on a change detection algorithm that uses sliding windows [7]: RELATIVE and ENERGY. Both compare a current window of coordinates to a window starting at the most recent application-level coordinate update. RELATIVE compares the two windows based on the amount of change relative to the nearest known neighbor. ENERGY compares them based on a statistical test that measures the Euclidean distance between two multidimensional distributions. Both update the application-level coordinate to the centroid of a recent window of coordinates and both heuristics allow the raw coordinate to "float" in a given region. As long as the coordinate does not leave the region and other major

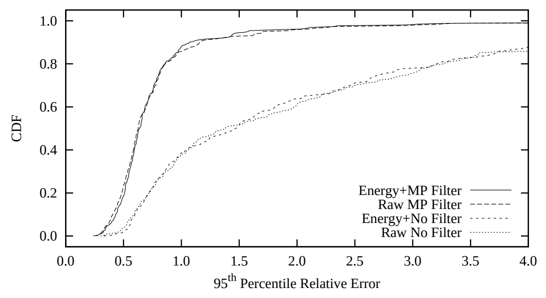

Figure 5: Accuracy on PlanetLab.

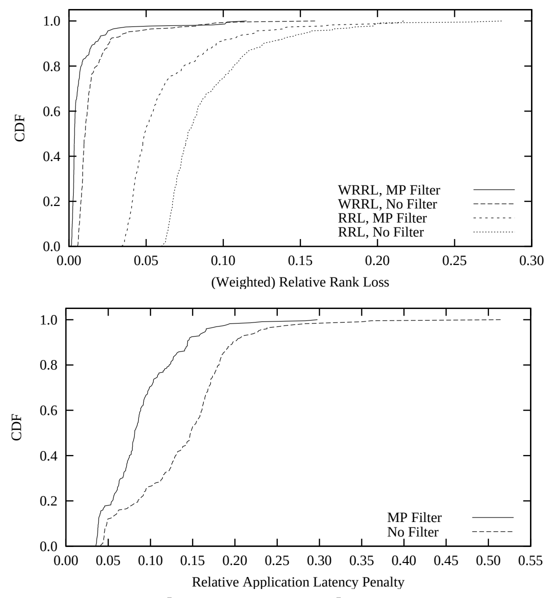

Figure 6: Application-oriented Accuracy Metrics.

changes in the network do not occur, application updates will be suppressed. Application-level NCs increase stability without decreasing accuracy.

Accuracy Results. Experimentally, we determined that a low percentile (*e.g.*, 25^{th}) and a window size of 4–8 samples or larger gives good results with the latency samples seen on PlanetLab. Links are moderately consistent: most follow the pattern seen in Figure 3, but about 10% that in Figure 4. Windows that are too large suppress network changes that should be reflected in the NCs: short windows are more effective than long ones, keeping the required state low. Figure 5 shows results from using the MP filter and the ENERGY heuristic with 270 PlanetLab nodes. It shows that with the MP filter only 14% of the nodes experience a 95^{th} percentile relative error greater than one, while 62% of those without the filter do. The enhancements combine to reduce the median of the 95^{th} percentile relative error by 54%.

In this experiment we measure accuracy as the coordinate's ability to predict the next sample along that link. For each observation, we compute the relative error, that is, the difference between the predicted and actual latency, divided by the actual latency. Each node then has a collection of relative errors from its samples; the figure shows the 95^{th} percentile out of this distribution, collected for the second half of a 4 hour run.

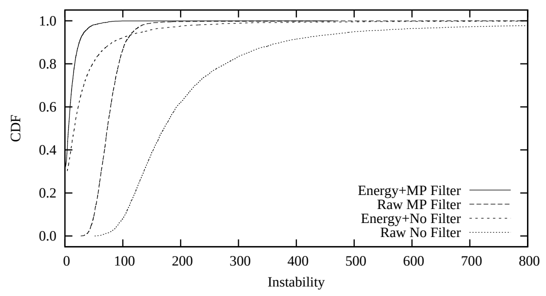

Figure 7: Instability on PlanetLab.

Defining accuracy as relative error produces a low-level metric that may not sufficiently capture application impact. Recently, Lua *et al.* proposed **relative rank loss** (*rrl*) to calculate how well coordinates capture the relative ordering of (all) pairs of neighbors [10]. Thus, for each node x, if $(d_{xi} > d_{xj} \wedge l_{xi} < l_{xj})$ or $(d_{xi} < d_{xj} \wedge l_{xi} > l_{xj})$, then the distances d between coordinates have to led to an incorrect prediction of the relative latencies l, presumably inducing an application-level penalty due to the wrong preference of a farther node. While *rrl* quantifies the *probability* of incorrect rankings, we wanted a metric that captures the *magnitude* of each rank misordering as well. For some applications, choosing the absolute nearest neighbor is important; however, often the extent of the error should be penalized: an error of 1ms is less severe than one of 100ms. **Weighted rrl** (*wrrl*) captures this by taking the sum of the latency penalties l_{ij} of pairs ranked incorrectly, normalized over all possible latency penalties. However, *wrrl* does not express the percentage in lost latency that an application will notice when using NCs. To approximate this quantity, we sum the relative latency penalty l_{ij}/l_{xi} for all pairs that are incorrectly ranked; we call this third metric the **relative application latency penalty** (*ralp*).

We illustrate how the MP filter affects these three metrics in Figure 6. The top graph portrays that while the probability of incorrect rankings (*rrl*) can range up to almost 30% for the worst case node, the latency penalty due to incorrectly ranked neighbors (*wrrl*) is 11% of the maximum in the worst case. The median *ralp* metric is 15% when using raw latency inputs, improving to 8% with the filter. We computed the "true" latency between nodes as the median for that link. In summary, our results indicate that the MP filter improves NC accuracy on PlanetLab for application-oriented operations, such as node ranking.

Stability Results. Unstable coordinates are problematic. Consider the situation where a node's coordinate is moving in a circle compared to using the centroid of that circle. If one is using the coordinate for a one-time decision (*e.g.*, finding the nearest node to initialize a Pastry routing table or finding a nearby web cache), unstable coordinates make a good decision less likely because it depends on the particular time the coordinates are compared. When coordinates are used for periodic decisions (*e.g.*, a proximity-based routing table update or re-positioning processing operators in a stream-based overlay), changing them may involve application-level work; unstable coordinates will induce updates based simply on their instability, not any fundamental change in relative node positions.

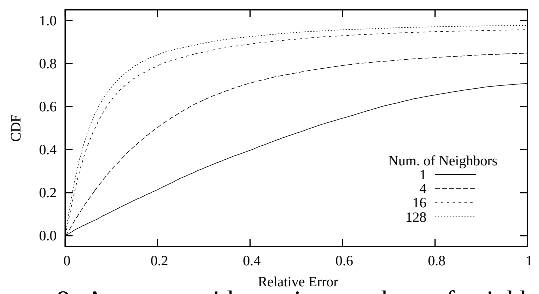

Figure 8: Accuracy with varying numbers of neighbors.

We measure stability as the amount of change in coordinates per unit time in *ms/sec*. This captures the amount of oscillation around a particular coordinate. Both the MP filter and application-level coordinates serve to suppress insignificant change. As shown in Figure 7, ENERGY dampens the filter's updates: 91% of the time it falls below even the minimum instability of the raw filter. Combined, the median instability is reduced by 96%. More detail on the MP filter and on the application-update heuristics can be found in our technical report [9].

4.2 Limiting Measurement Overhead

One of the advantages of the NC library is that it takes advantage of application-level traffic to keep NCs up-to-date. This implies that a lack of samples must induce additional measurements to more nodes, but only when accuracy can be significantly improved. If nodes have a small neighbor set (*e.g.*, two), their accuracy to their neighbors and confidence w_i is high, but their accuracy to the rest of the system (overall accuracy) is low. As the number of neighbors increases, confidence and accuracy to neighbors decrease slightly, but overall accuracy improves.

In Figure 8, we show how overall accuracy varies with the number of neighbors. Accuracy increases asymptotically as the number of neighbors approaches the number of nodes. As shown, only 16 neighbors is a sufficiently good substitute for a fully connected graph. This means that regular application-level traffic to a small number of nodes is sufficient to support NCs on PlanetLab.

The NC library must decide when adding neighbors would significantly increase accuracy. However, a node cannot know its accuracy only by examining the relative error to its neighbors. Instead, it must estimate the overall relative error to all nodes. We propose that a node periodically samples a random node to test the current accuracy of its NC. If the tested accuracy is below a threshold, which is based on the expected accuracy of NCs on the Internet, it is likely that an increase of the neighbor set will reduce the relative error. The test node is then added permanently to the neighbor set. Similarly, if removing a node temporarily does not decrease accuracy (again by sampling a new node), the decrease is made permanent.

5 Conclusions

Up-to-date latency information as provided by a latency service is crucial for many distributed applications. Network coordinates are an efficient and scalable mechanism for obtaining latency estimates. However, any practical implementation must handle the variance of latency samples and minimize measurement overhead, while ensuring stable and accurate coordinates. In this paper, we have described the APIs of a network coordinate service and a library. We have also shown how statistical filtering addresses sample variance, how the distinction between system- and application-level coordinates improves coordinate stability, and how the use of application-level traffic for coordinate updates can reduce overhead. We believe that a network coordinate service can add network awareness to a wide range of applications and become one of a set of standard services for planetary-scale applications.

References

[1] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A Large-scale and Decentralized App-level Multicast Infrastructure. *JSAC*, 20(8), Oct. 2002.

[2] Y. Chen, K. H. Lim, R. H. Katz, and C. Overton. On the Stability of Network Distance Estimation. *SIGMETRICS Perform. Eval. Rev.*, 30(2), 2002.

[3] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proc. of ICDCS'04*, Tokyo, Japan, Mar. 2004.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of ACM SIGCOMM'04*, Portland, OR, Aug. 2004.

[5] P. Francis, S. Jamin, V. Paxson, et al. An Architecture for a Global Internet Host Distance Estimation Service. In *Proc. of INFOCOM'99*, New York, NY, Mar. 1999.

[6] T. Gil, F. Kaashoek, J. Li, et al. p2psim. `www.pdos.lcs.mit.edu/p2psim`.

[7] D. Kifer, S. Ben-David, and J. Gehrke. Detecting Change in Data Streams. In *Proc. of the 30th Int. Conf. on Very Large Data Bases*, Toronto, Canada, August 2004.

[8] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and Embedding Using Small Sets of Beacons. In *Proc. of FOCS'04*, Rome, Italy, Oct. 2004.

[9] J. Ledlie and M. Seltzer. Stable and Accurate Network Coordinates. TR 17-05, Harvard University, July 2005.

[10] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the Accuracy of Embeddings for Internet Coordinate Systems. In *Proc. of IMC'05*, Berkeley, CA, Oct. 2005.

[11] A. Nakao, L. Peterson, and A. Bavier. A Routing Underlay for Overlay Networks. In *Proc. of the ACM SIGCOMM'03 Conference*, Karlsruhe, Germany, Aug. 2003.

[12] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of INFOCOM'02*, New York, NY, June 2002.

[13] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris. Lighthouses for Scalable Distributed Location. In *Proc. of IPTPS'03*, Berkeley, CA, Feb. 2003.

[14] S. Ratnasamy, P. Francis, M. Handley, B. Karp, and S. Shenker. Topology-Aware Overlay Construction and Server Selection. In *Proc. of INFOCOM'02*, June 2002.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of Middleware'01*, Nov. 2001.

[16] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *Proc. of INFOCOM'03*, San Francisco, CA, Mar. 2003.

[17] Stream-Based Overlay Network. `www.eecs.harvard.edu/˜prp/research/sbon`, Feb. 2005.

[18] J. Stribling. All-Pairs-Pings for PlanetLab. `www.pdos.lcs.mit.edu/˜strib/pl_app`, Sept. 2004.

[19] The PlanetLab Consortium. PlanetLab. `www.planet-lab.org`, 2002.

[20] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Proc. of SIGCOMM'05*, Aug. 2005.

This material is based upon work supported by the National Science Foundation under Grant No. 0330244.