A Unified Network "Coordinate" System for Bandwidth and Latency

Venugopalan Ramasubramanian[‡] Dahlia Malkhi[‡] Fabian Kuhn[†]
Ittai Abraham[‡] Mahesh Balakrishnan[‡] Archit Gupta[*] and Aditya Akella[*]

[*]University of Wisconsin-Madison, Madison, WI 53706
[†]ETH, Zurich, Switzerland
[‡]Microsoft Research Silicon Valley, Mountain View, CA 94043

Abstract

Network coordinate systems, such as GNP and Vivaldi, provide virtual positions for networked hosts, which enable the hosts to connect to nearby peers, find the closest server, or organize themselves in a topologically-aware manner. Current network coordinate systems, however, only use latency to compute the positions, leaving out an important network metric—namely bandwidth.

In this paper, we present a unified approach that provides virtual positions based on both bandwidth and latency. The key intuition is that network latency and bandwidth are approximate tree metrics, that is, a set of distances that can be embedded in a tree. We first argue based on intuition and analysis of three real-world datasets why bandwidth and latency can be represented as tree metrics. Then, we present Sequoia, an accurate and light-weight system that provides virtual network positions by embedding bandwidth or latency on trees; the network positions computed by Sequoia are as easy to use as a set of coordinates. Finally, we present an evaluation based on the three datasets showing that: 1) Sequoia represents latency as accurately as Vivaldi in addition to being the first "coordinate" system for bandwidth; 2) it enables selection of the closest and the most-provisioned (highest bandwidth) server with low error and overhead; and 3) it computes topologically-aware trees, which can be used to organize a networked system efficiently.

1 Introduction

Latency-aware network applications are pervasive these days: Web-based services and content distribution networks (CDNs) often redirect client requests to the closest server while peer-to-peer systems and distributed hash tables (DHTs) prefer to select neighbors based on network proximity. Naturally, several systems have been designed and built to provide latency-centric functionalities. Systems such as IDMaps [8], Meridian [25], and Oasis [9] provide the capability for discovering closest-servers efficiently. Other systems such as GNP [16], Vivaldi [6], and PIC [5] provide a convenient set of coordinates for each host that can then be used to estimate latency and select proximal peers.

However, a totally different network property—namely bandwidth—has emerged as a crucial performance factor. With the increasing advent of online-media-streaming, podcasts, and movie/video downloads, clients of web-based multi-media services and hosts of peer-to-peer CDNs have a new need to select servers based on bandwidth in addition to latency. Current systems that compute network coordinates based on latency, unfortunately, fail for bandwidth; while, the suitability of other systems for bandwidth-based server selection has not been explored yet.

This paper, for the first time, presents an intuitive model for network bandwidth. The proposed model is based on the observation that under certain typical circumstances, bandwidth is a tree metric; that is, the set of bandwidth measures can be exactly embedded as distances on a tree. For instance, bandwidth is a tree metric when it primarily depends on the last-mile, access links. Claims from prior measurement studies supporting the prevalance of this instance and an analysis of a real-world bandwidth dataset presented in this paper indicate that Internet bandwidth is indeed an approximate tree metric.

Fortuitously, Internet latency also turns out to be an approximate tree metric. While this revelation may not be surprising since the Internet is quite hierarchical, our analysis of two real-world latency datasets confirm that end-to-end latencies closely follow the hierarchy.

The outcome of the above observations is an elegant and unified model to represent both latency and bandwidth. This paper explores the model of embedding network latency and bandwidth onto trees. More concretely, it presents the notion of *prediction trees*, where end hosts at the leaf level connected via a network of virtual inner nodes with carefully assigned link weights model latency or bandwidth. Note that systems that reconstruct the internal topology of the Internet already exist [15, 27].

However, unlike these systems that try to locate individual gateways and routers through expensive and intrusive measurement probes, our approach strives to build a "virtual" model (where internal nodes represent fake routers) through light-weight, end-to-end mechanisms.

Prediction trees provide key intrinsic advantages: First, predicting latency or bandwidth between two hosts in a tree requires a mere computation (just like coordinate-based network positioning systems) when the paths to a common ancestor are known. Thus, the paths to a common root host can serve as "coordinates" for each host. Second, finding the closest or best-provisioned server can be accomplished efficiently through a simple search along the path from the root to a leaf. Finally, a reasonably-balanced tree is a highly scalable structure where operations (such as search or join) can be accomplished with low overhead.

This paper presents a system called Sequoia that provides a variety of network-centric functionalities based on the above-described notion of *prediction trees*. Sequoia maintains a collection of virtual trees between the participating hosts and provides efficient latency/bandwidth prediction, server selection, and topology-aware clustering through easily-decentralized, light-weight mechanisms. It is resilient to violation of the triangle inequality condition in network measures, tolerates non-availability of some measurements, and shows good scalability with increasing number of hosts.

We envision that Sequoia would serve the needs of several networked systems and applications. First, path quality prediction is useful for neighbor selection in Distributed Hash Tables and peer-to-peer file sharing services (e.g., BitTorrent). Second, web services can use Sequoia to efficiently redirect clients to the closest server (like Meridian does) or to a well-provisioned server. Finally, hierarchical networked systems, such as overlay multicast systems [4, 28, 22] and network monitoring systems [19, 26] can leverage the intrinsic topology-aware hierarchy built by Sequoia to organize themselves.

Overall, this paper makes the following major contributions: First, it presents an elegant approach for representing network measures as tree metrics, providing intuitive and analytical reasons to argue that at least two important measures—latency and bandwidth—are approximate tree metrics. Second, it outlines the design and implementation of the Sequoia system, which constructs prediction trees for latency and bandwidth in a cost-efficient yet accurate manner. Finally, it demonstrates the new abilities that Sequoia provides for bandwidth-based server selection and topology-aware hierarchical clustering while highlighting Sequoia's ability to match the state-of-the-art in latency-based network positioning, all through an extensive evaluation driven by real-world datasets.

The rest of the paper has the following organization: Section 2 provides some background about tree metrics and makes a case for embedding network measures on trees. Section 3 then describes the Sequoia system in detail while Section 4 evaluates Sequoia. Finally, we discuss suitable applications and related work in Sections 5 and 6 and conclude in Section 7.

2 Background and Intuition

The key intuition behind this work is that Internet path measures such as bandwidth and latency are approximate tree metrics. In this section, we provide some background about tree metrics and present intuitive and analytical arguments to back up this intuition.

2.1 Tree Metrics

Consider a set D of pair-wise measurements of some network path property, say latency or bandwidth, between a set V of networked hosts. This set of measures D is a *tree metric* if there exists a tree T with non-negative weights such that $V \subseteq T$ and $d_V(u,v) = d_T(u,v)$ for all $u, v \in V$, where $d(u,v)$ represents the pair-wise path property. In other words, a set of measures is a tree metric if it can be derived from distances on a tree, that is, embedded on a tree. Note that in the above definition, the tree T may have additional nodes not present in the set V.

There is a convenient condition called the *Four-Points Condition* [3] (4PC) to verify whether a set of measures is a tree metric. The four-points condition states that for any four hosts w, x, y, and z ordered such that $d(w,x) + d(y,z) \leq d(w,y) + d(x,z) \leq d(w,z) + d(x,y)$, $d(w,y) + d(x,z) = d(w,z) + d(x,y)$. That is, of the three sums of distinct pairs of distances, the highest sum is equal to the second highest sum. A set of measures is a tree metric if and only if every set of four hosts satisfies the 4PC [3]. Figure 1 illustrates the four-points condition graphically.

Another form of tree metric that is useful for modeling network path properties is the *ultra metric*. An ultra metric embeds network hosts into a hierarchically-separated tree, where any pair of hosts with the same least common ancestor also has the same distance. By least common ancestor, we mean the closest common ancestor in the tree. Note that, once again, the tree might include additional hosts not present in the set of measures. An ultra metric is a stricter form of tree metric in that every ultra metric is a tree metric while the inverse is not always true.

Similar to tree metrics, there is a convenient condition to verify whether a set of measures is an ultra metric. This condition, called the *Three-Points Condition* (3PC) [3], states that for any three hosts x, y, and z ordered such that $d(y,z) \leq d(x,z) \leq d(x,y)$, $d(x,y) =$

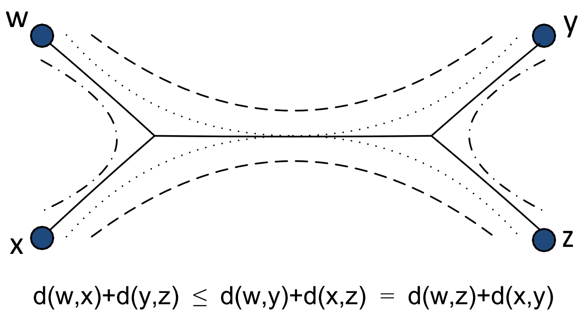


Figure 1: **Four-Points Condition: For any four nodes and the three sums of distinct pairs of distances between them, the highest sum is equal to the second highest iff the nodes and distances are embeddable in a tree.**

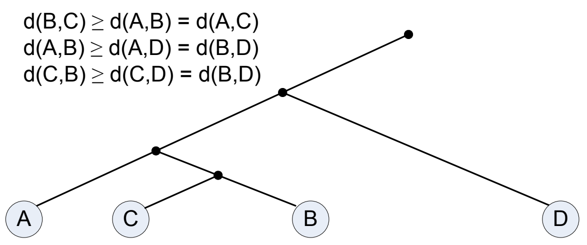


Figure 2: **Tree Showing an Ultra-Metric: Hosts B and C with a common ancestor to A have the same distance to A. Similarly for A and B with respect to D.**

$d(x, z)$. That is, of the three measures, the largest is equal to the second largest. A set of measures is an ultra metric if and only if every set of three hosts satisfies the 3PC [3]. Figure 2 illustrates an ultra metric and the three-points condition graphically.

Note that the above definition of the three-point condition applies to distance measures, where smaller is better. The corresponding 3PC for bandwidth would state the converse, that is, in any triplets, the two smaller bandwidths are equal.

2.2 Bandwidth and Latency as Tree Metrics

We first argue why tree metric is a fitting representation for bandwidth. We do this by presenting two hypothetical network models in which bandwidth measures turn out to be exact tree metrics.

Best Bandwidth Networks: Consider a network of hosts connected by routers and gateways where the path used to route packets between two hosts is the best bandwidth path, that is, the path with the highest bottleneck bandwidth, where the bottleneck bandwidth of a path is the minimum of the bandwidths of each link in the path.

We can trivially show that the set of bandwidths between the end hosts in the above network is an ultra metric. Assume, for instance, that there are a set of hosts, x, y, and z with $d(y, z) \geq d(x, z) \geq d(x, y)$ that violate the 3PC for bandwidth; that is, $d(x, z) \neq d(x, y)$. Then the path $x \rightarrow z \rightarrow y$ would have a higher bandwidth than the current path, indicating that the network is not using the best bandwidth paths—a contradiction.

Edge Bandwidth Networks: Consider a network of hosts connected by routers and gateways where the last-mile access links have lower bandwidths than the links at the inner core. That is, the bottleneck bandwidth between two end hosts only depends on the bandwidths of the access links connecting the two end hosts to the network and is totally independent of what routing policies are used for finding the paths.

In the above model too, we can trivially show that the set of bandwidths between the end hosts is an ultra metric. Consider any set of three hosts, x, y, and z with $a_z \geq a_y \geq a_x$, where a_x, a_y, and a_z are the bandwidths of the access links connecting x, y, and z. Then, $\min(a_y, a_z) \geq \min(a_x, a_z) \geq \min(a_x, a_y)$; and therefore, $d(y, z) \geq d(x, z) = d(x, y)$, the 3PC for bandwidth, since $d(u, v) = \min(a_u, a_v)$ in the above network model.

The Internet, of course, does not always satisfy the above models. In fact, the first scenario of best-bandwidth networks may arise only in rare instances; for example, when a CDN such as Akamai uses the best-provisioned paths in an overlay network[1]

However, the second scenario, where the path bandwidth depends on the last-mile, access links, is more common. This is especially true in the increasingly prevalent broadband networks as shown by a recent measurement study by Dischinger et al. [7]. Another study by Hu et al. [13] claims that, even in the Wide-Area Internet, 60% of paths between random end hosts have the bottleneck in the first or second hop. The conclusions of these studies indicate that Internet bandwidth could be close to tree metric.

End-to-end latency, on the other hand, typically depends on all the components in the path. Naturally, it is harder to understand why latency might fit into tree metrics as well. At a high level, the Internet has a hierarchical organization, with different tiers of ISPs (Tier 1, Tier 2, etc.). Yet, in practice, this organizational hierarchy does not translate to topological hierarchy since ISPs have peering relationships in complicated ways; Tier 1

[1]Akamai actually employs overlay routing on their EdgePlatform; although, we are not aware if the routes are optimized for bandwidth.

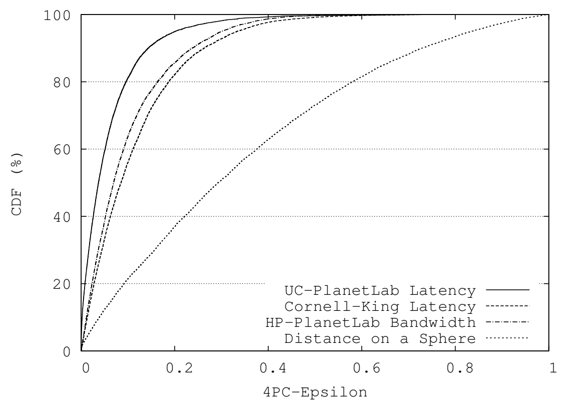


Figure 3: **CDF of ϵs for Different Datasets: The ϵs are generally small indicating that the datasets closely resemble tree metrics. In contrast, latencies between nodes distributed on the surface of a sphere have much bigger ϵs.**

ISPs peer with each other, Tier 2 ISPs have multiple Tier 1 parents and Tier 2 peers, and so on.

We believe that even though the Internet is a mesh with lots of shortcut paths, the shortcuts often do not contribute significantly to latency. It is known that the top tiers are more densely connected than the lower tiers [24]. Hence, the shortcuts tend to be present mostly in the top tiers, and a fast, over-provisioned top-tier would keep the impact of these shortcuts on end-to-end latency low compared to the effect of longer links in the edges of the Internet.

Next, we present further, quantitative evidence to show that Internet measures are approximate tree metrics.

2.3 Analysis of the Tree-ness of the Internet

We quantify the fit of a set of measures to the tree metric by measuring how well it satisfies the four-points condition. We use a parameter called the *4PC-ϵ* originally introduced in [1] to quantify the deviations from 4PC. The 4PC-ϵ for a set of four nodes w, x, y, and z ordered such that $d(w,x) + d(y,z) \leq d(w,y) + d(x,z) \leq d(w,z) + d(x,y)$ is the one that satisfies the equation $d(w,z) + d(x,y) = d(w,y) + d(x,z) + 2\epsilon \cdot \min\{d(w,x), d(y,z)\}$. The paper [1] provides more intuition about this definition of 4PC-ϵ.

The distribution of 4PC-ϵs show how close a network measure is to a tree metric. The 4PC-ϵs are zero for a perfect tree metric and at most one for an arbitrary metric.

We compute the distributions of 4PC-ϵs for three realworld datasets: 1) a *UC-PlanetLab Latency* dataset of round-trip times between PlanetLab nodes measured at University of Cincinnati [35], 2) a *Cornell-King Latency* dataset of latencies measured between DNS servers using the King technique [11] at Cornell University [29], and 3) a *HP-PlanetLab Bandwidth* dataset of available bandwidth measurements between PlanetLab nodes collected at HP Labs [33] using the pathChirp tool [20]. Table 1 summarizes the details about these datasets.

Figure 3 shows the CDF of 4PC-ϵs for each dataset. In this graph, we omit the node sets where a triangle inequality violation occurred since the definition of 4PC only applies to a metric. In our datasets, violations of the triangle-inequality occurred for roughly 15 to 40 % of the triplets. We discuss more about triangle-inequality violations later in this section.

At a high-level, Figure 3 shows surprisingly small values of 4PC-ϵs (less than 0.2 for 80% of values) indicating that all three datasets are approximate tree metrics. Further, as a sanity check, the figure also shows the distribution of 4PC-ϵs for a metric of latencies on the surface of sphere, as would be the case if the latencies between Internet hosts were based on geographical distance. Not surprisingly, the distribution of 4PC-ϵs for a sphere shows much less resemblance to a tree metric than latencies on the Internet. This also indicates that the prevalent approaches [6] for modeling Internet latency using coordinate-based models and Euclidean distances may not be the most fitting.

2.4 Assumptions

Overall, the above intuition and analysis indicates that approximating Internet path measures as tree metrics is a promising approach. Certainly, this approach makes a few assumptions about the path measures. Similar to prior coordinate-based approaches [16, 6, 5] for modeling latency, this approach also assumes that the network measure is a metric. The metric assumption implies at least two properties: 1) the measures are symmetric and 2) the triangle inequality holds. However, these properties do not always hold in the Internet. Policy-based routing (such as hot potato and cold potato) that do not select shortest or bandwidth-optimal paths and transient overheads such as queuing delay lead to violations of symmetry and the triangle inequality condition.

Our approach, described in the next Section, does not currently address asymmetric network measures. However, it is resilient to triangle inequality violations and works well in their presence as will be evident from our evaluation.

Finally, we are aware that the term *bandwidth* often refers to related yet different measures: *capacity*, the theoretical maximum bandwidth of a path in the absence of cross traffic, *available bandwidth*, the actual unused bandwidth at a given instant, and the *bulk-transfer throughput* achievable by a TCP connection [17]. The intuition provided above as well as our approach and techniques described below apply well to any of these bandwidth measures. The dataset we use to evaluate our approach, however, measured available bandwidth.

3 Sequoia

We next present Sequoia, a system that applies the above insights for embedding network measures such as latency and bandwidth onto trees. Sequoia strives to use these tree embeddings to provide network-aware functionalities such as path quality estimation, server selection, and hierarchical clustering to applications. The rest of this section provides an overview of Sequoia, details its design, and describes how it supports the above functionalities.

3.1 Overview

Sequoia constructs "virtual" *prediction trees* for a system of networked hosts through end-to-end measurements. The prediction tree for a set of networked end hosts looks as follows: The end hosts form the leaf nodes and are connected via a network of virtual inner nodes. The links in this virtual topology have weights that model a network measure, that is, latency or bandwidth.

The inner nodes are virtual in the sense that they are introduced for purely modeling purposes and are not expected to have any real-world associations. The tree topology also does not imply that the end hosts organize into a tree-based distributed system; the virtual trees could merely be an abstraction in the memory of a few or all hosts in the system. However, as explained later, Sequoia's virtual trees could be used to construct a hierarchically organized distributed system if required.

Sequoia designates an end host as a *lever* R for each virtual tree; the lever could be any of the end hosts or a specially provisioned landmark server. The lever acts as a reference for the tree; that is, link weights are assigned such that the tree path from the lever to an end host has the same distance as the real Internet path between the lever and the end host. Figure 4 illustrates a virtual tree for three end hosts and a lever.

The prediction tree provides a convenient way to estimate path qualities between participating hosts. Each host has a *distance label* that encodes the path of the host to the lever and the corresponding link weights. Two hosts can then quickly estimate the quality of their path without referring to other parts of the tree. For example, in Figure 4, hosts A and B have the distance label s,t and C the label t. Two nodes, say A and C, can estimate their distance by computing the path A,s,t,C from their distance labels.

In addition, a prediction tree enables three other network-centric functionalities. First, a participating end-host can use the prediction tree to discover a closest or best-provisioned peer node by merely looking at its immediate vicinity in the tree. For example, in Figure 4, host A can find its closest peer host B through a localized search. Second, an external end-host can use the tree to guide its search to find the best server. Such a search can start at the lever and direct itself in the tree along the path from the lever to the chosen server at the leaf. Finally, the prediction tree itself provides a network-aware hierarchical clustering of hosts. For example, in Figure 4, hosts A and B being closer to each other than host C form a lower-level cluster.

3.2 Design

The central process in Sequoia is prediction-tree construction. Sequoia's algorithms for constructing prediction trees are based on a basic heuristic that ensures that the constructed tree will provide zero distortion if the modeled path quality is an exact tree metric. That is, if we start with a tree metric, we get a zero-distortion prediction tree.

The join algorithm based on the above heuristic has the following key step: A host's position in the tree is determined by the lever and one other node called the *anchor*. A host, say B, tries to maintain exact distances to the lever R and its anchor A; it preserves the three distances d(A,R), d(B,R), and d(A,B), by introducing a virtual node s in the existing tree path between A and R. Note that computing the distances from s to R, A, and B such that it preserves the real distances is quite trivial (for instance, $d(s,R) = 0.5 \cdot (d(A,R) + d(B,R) - d(A,B))$).

A host B joining an existing prediction tree first finds a suitable anchor A in the current tree. In order to preserve zero-distortion for tree metrics, the anchor needs to be the host that maximizes the distance d(s,R) on the tree [3]. This distance $d(s,R) = 0.5 \cdot (d(A,R) + d(B,R) - d(A,B))$ is called the *Gromov product* and will be denoted as $(A|B)_R$ in the rest of the paper. This basic join algorithm is illustrated in Figure 6.

3.2.1 Anchor Tree

Even though the above algorithm provides an elegantly construction for prediction trees, it does not help in shaping the structure of the tree. Consequently, the resulting prediction tree may not have a nice balanced structure. For instance, if we start with distances on a linear chain of hosts, then the resulting prediction tree will also be a chain. This has profound impact on the scalability of our approach because the cost of basic primitives such as distance labels and server selection depends on the tree structure.

We propose an alternative, scalable abstraction called the *anchor tree*. An anchor tree is simply a tree showing anchor relationships of end hosts in the prediction tree. Figure 5 shows an example anchor tree for the prediction tree in Figure 4. Unlike the prediction tree, an anchor tree can be constructed in a well-balanced manner. Even

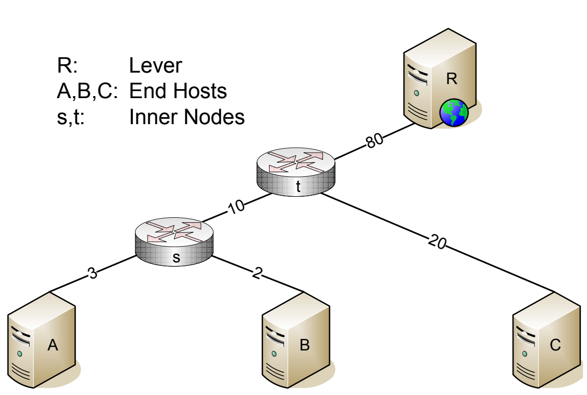


Figure 4: **Prediction Tree: An example prediction tree between three end hosts and a lever. The link weights model path qualities such as latency.**

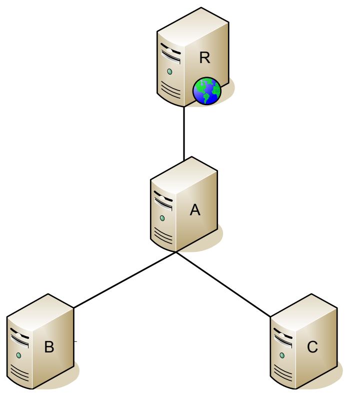


Figure 5: **Anchor Tree: An example anchor tree corresponding to the prediction tree in Figure 4.**

in the adverse case where the distances are induced by a linear chain, we can construct a well-balanced anchor tree.

The anchor tree and its corresponding prediction tree are equivalent; the former is just a more scalable representation of the latter. It provides the exact same distance estimates as the prediction tree although computing distance is a bit more tedious. The path from an end host to the lever also serves as the distance label in an anchor tree. Two end hosts can estimate the distance between them based on their distance labels. A simple distance computation algorithm is to construct a prediction tree based on the anchor relationships encoded in the two distance labels and then use the prediction tree to estimate the distance.

A well-balanced anchor tree has the following advantages over prediction trees: First, it provides short distance labels that scales logarithmically with system size. Second, the paths from the lever to hosts are also short enabling efficient searches for best servers. Finally, unlike the prediction tree, an anchor tree is composed only of real hosts and can be directly used to construct a distributed system.

3.2.2 Best Host Discovery

The anchor tree also facilitates a low overhead technique for construction of prediction trees. Note that, the tree construction algorithm outlined earlier involved the key step of finding an anchor that maximizes the Gromov product. This step might involve searching through all hosts in the system and measuring their distance to the joining host. Such an effort is unproductive as it requires $O(N)$ measurements in a system of N hosts.

Sequoia instead uses the anchor tree to prune this search space and find a suitable anchor with less measurements. We first outline a general search algorithm on the anchor tree for finding a host that meets a general search criterion with respect to a target host B. The search starts with the lever as the *candidate* host. It looks at all the hosts within a certain *search_depth* D from the lever and measures their distance to the target host. The best host from this set is chosen as the new candidate and the search is repeated until no progress can be made. Finally, the search returns the best host in the path from the lever to the last candidate. This search algorithm is illustrated in Figure 7.

The search criterion in the above algorithm can be set according to requirements. A criterion based on smallest latency or largest bandwidth enables server selection. For tree construction, the criterion is to maximize $d(A, R) - d(A, B)$, where B is the joining node. Note that the third term in the Gromov product, $d(B, R)$ is a constant during the anchor search.

The above search algorithm is a heuristic and is not guaranteed to find the correct anchor or the closest or the best-provisioned server. It is a practical trade-off to keep measurement cost low albeit, as shown in Section 4, an effective one. The number of measurements required by the search algorithm is also not bounded although we expect it to scale logarithmically on average if we bound the number of hosts a node can anchor.

3.3 Practical Issues

In this section, we discuss how Sequoia handles other major practical issues.

3.3.1 Bandwidth Representation

The previous discussion primarily talked about building prediction and anchor trees for a distance (latency) measure. There are two choices for representing bandwidth

Figure 6: **ConstructTree Algorithm: Basic prediction tree construction algorithm that preserves tree metrics for a set V of hosts.**

```
return tree $T := \mathbf{constructTree}(V \setminus \{R\}, R)$;

function ConstructTree(V,R):
  if $|V| > 1$ then
    choose next host $B \in V$
    choose anchor $A$ that maximizes $(A|B)_R$
    add virtual node $s$ between A and R at distance $(A|B)_R$ from $R$;
    add node $B$ at distance $(A|R)_B$ from $s$
    $T := \mathbf{constructTree}(V \setminus \{B\}, R)$;
  fi;
  return T;
end ConstructTree
```

Figure 7: **SearchTree Algorithm: Basic search algorithm to find a candidate host meeting some search criterion with respect to a target host B using the anchor tree.**

```
return candidate $C := \mathbf{SearchTree}(R, B)$;

function SearchTree(C, B):
  $S$ := all hosts at depth $D$ from $C$ in the anchor tree;
  $C'$ := best host $S$ that meets the search criterion;
  if $C \neq C'$ then
    $C := \mathbf{SearchTree}(C', B)$;
  fi;
  return $C$;
end SearchTree
```

on a prediction tree. The first is to use prediction trees as a black-box for modeling any approximate tree metric and represent bandwidth by treating it as a distance metric. The second is to build a bandwidth-specific prediction tree where the prediction tree will resemble an ultrametric tree, similar to Figure 2. One well-known way to build a bandwidth-specific prediction tree is to build a maximum- weighted spanning tree (MST) between the hosts using bandwidths as link weights and then defining the distance between two hosts in the MST as the weight of the smallest weighted-link in the path [3].

Sequoia chooses the first, black-box approach. We found it to have better accuracy than the second approach and easier to unify the methodology for representing latency and bandwidth. However, bandwidth has a converse semantics compared to latency—higher is better as opposed to smaller is better for latency. Hence, Sequoia reverses the order of bandwidth measures by subtracting each measure from a high constant before tree construction and subtracts the predicted value from the same constant before providing it to the application. Note that, the choice of this constant does not affect the outcome of the prediction in any way. It just needs to be high enough to avoid negative values after subtraction.

3.3.2 Multiple Trees

It turns out that the accuracy of the prediction tree depends on the choice of the lever and its location in the network. It is possible that the chosen lever is not well-suited to demarcate the relative positions of some hosts in the network, or the lever may not be able to measure its distance to some hosts. To mitigate the impact of bad levers and network problems, Sequoia uses multiple prediction trees referenced at distinct levers. Choosing the median distances estimated from the trees then helps in removing the outliers and improving the accuracy of prediction and server selection.

3.3.3 Balancing Anchor Trees

As highlighted earlier, a well-balanced anchor tree is crucial for the scalability of Sequoia. It is desirable that the anchor tree is not too deep (shorter paths from hosts to the lever) and not too fat (bounded children for each anchor). In practice, the anchor trees generally don't seem to appear too deep or fat (see Section 4).

Yet, we have a mechanism to build balanced anchor trees to handle worst-case scenarios. The ConstructTree algorithm in Figure 6 chooses the anchor when a new host joins the tree. It is possible to pre-determine the anchor choices in a desirable way by changing the order in which hosts join the tree. Our anchor balancing mechanism changes the join order so that each anchor has a bounded number of children and anchor paths are not unnecessarily deep. Unfortunately, this mechanism is expensive (requires $O(N \log N)$ computations for a system with N hosts), and we don't intend to employ it except in a bad case.

3.3.4 Triangle Inequality Violations

Network measures are not true metrics and often violate key properties such as the triangle inequality condition. It is crucial for Sequoia to be resilient to such violations.

First, observe that violations of triangle inequality has a simple effect on Sequoia's prediction trees; they produce negative weights on the links. This is because the

Gromov product can be negative if the triangle inequality does not hold. Negative link weights are not particularly a problem for computing distances on trees since trees are acyclic (there are no negative-weight cycles). Nevertheless, they might make a few distance estimates to come out negative. Sequoia simply ignores negative estimates and instead uses estimates from other trees.

But Sequoia also corrects the above aberration and avoids negative link weights by fixing the triangle inequality. It adds a large constant to each measured value before building the prediction tree and subtracts it back before presenting estimated distances to applications. Adding a large constant (similar to bandwidth representation) does not change the accuracy of prediction tree in any way. It simply results in the Gromov product being higher by that constant and consequently avoids any negative link weights.

3.3.5 Non-availability of Measurements

Network measurements sometimes fail; a few hosts may not respond to measurement probes, or firewalls and intermediate gateways may block the probes. We designed Sequoia to use opportunistic meaurements wherever possible and to be resilient to measurement failures whenever it occurs. Sequoia simply ignores unavailable measurements; it sets the Gromov product to negative infinity if one of the component measurements is not available. In the case that a host cannot measure itself to a lever, Sequoia uses the other prediction trees referenced at different levers.

3.3.6 Changing Measures

Finally, network measures are not constant; latency and available bandwidth change often. Instead of modifying the prediction trees to reflect each change, we propose a simple approach to adapt to dynamic changes. Sequoia maintains a sliding window of trees constructed at different times. Periodically, it constructs a new tree based on the oldest prediction tree (at the same lever) but with new, updated measurements. A weighted median could help pick fresher distance estimates from multiple trees.

3.4 Architecture

Sequoia is conducive to be deployed in many ways. In this section, we make a few remarks about the choices for Sequoia's architecture.

Centralized Service: Sequoia could be a centralized Web service like iPlane [15], exporting a query interface to external clients. Clients could query Sequoia to estimate a network property to a targeted host or choose the best server from a set of target hosts. Sequoia in turn could take advantage of opportunistic measurements reported by the clients or explicitly instruct clients to perform measurements to target hosts. This architecture might require additional effort to keep Sequoia responsive and available.

Partially Centralized System: A better architecture for Sequoia is to keep the participating hosts actively informed about their current "coordinates", that is, distance labels. A centralized server leverages measurements observed by the hosts, builds prediction (and anchor) trees efficiently, computes the distance labels, and informs the hosts. Hosts, in turn, can use the distance labels to perform latency/bandwidth prediction and server selection independently without consulting the centralized server.

Fully-Decentralized System: Finally, Sequoia could be a distributed system with no centralized server. The anchor trees provide a convenient distributed organization for such a deployment. Each host could maintain each host on its path to the lever as neighbors . The SeachTree algorithm in Figure 7 can be converted into a network protocol for finding the anchors and best servers through the above neighbor relationships (similar to peer-to-peer systems such as Gnutella [32]). Levers would serve as bootstrap hosts that new hosts can contact to join Sequoia with lever failures compensated by other levers in the system. Finally, if anchor hosts fail then their parents in the anchor tree can take over their role naturally.

4 Evaluation

We next present an evaluation of Sequoia. First, we show Sequoia's basic ability to represent the underlying datasets accurately. Second, we demonstrate Sequoia's practical benefits by using server selection as a targeted application. Finally, we discuss a few structural properties of Sequoia trees and highlight their topological correlations with the real world.

The evaluation is driven by the three real-world datasets (UC-PlanetLab, Cornell-King, and HP-PlanetLab) mentioned in Section 2. Table 1 summarizes the properties of these datasets. The datasets represent different network properties (two latency and one available bandwidth) measured on geographically spread-out hosts in the Wide-Area Internet (two between PlanetLab [30] hosts and one, Cornell-King, between infrastructure (DNS) servers). They are also of widely-different scales (125, 396, and 2500 hosts) and sometimes highly incomplete (only 40% of measurements in HP-PlanetLab). Finally, none of them are strict metrics as they have a significant amount of triangle-inequality violations (up to 40% in HP-PlanetLab).

Our evaluations were performed on an implementation of Sequoia's tree construction algorithms in C♯. We construct different numbers of Sequoia trees with randomly chosen levers for each dataset, use the above-described datasets to drive the measurements, and keep track and

Dataset	Measure	Technique	Hosts	Measurements	$\triangle <>$ Violations
UC-PlanetLab	Latency	Ping	125	15625	15 %
Cornell-King	Latency	King [11]	2500	3123750	22 %
HP-PlanetLab	Bandwidth	pathChirp [20]	396	65077	40 %

Table 1: **Summary of Datasets**

report the number of measurements required to construct the trees. All numbers reported in this Section are based on the decentralized tree-construction algorithm that uses selected, partial measurements from the datasets.

4.1 Prediction of Network Properties

We first present the error in using Sequoia to model and predict network properties. We represent the *prediction error* as a relative error: the ratio of the absolute difference between the predicted and the true values over the true value, that is, $abs(tree\_value - graph\_value)/graph\_value$. In the presence of multiple Sequoia trees, the predicted, tree value was taken as the median of the values estimated from all the trees.

We compare the accuracy of Sequoia models against Vivaldi [6], the well-known coordinate-based approach to model network latencies, using the Vivaldi simulator [31] built at Harvard University. For Vivaldi, we used the default value of three coordinates, set the number of neighbors to include all hosts for the two complete datasets and 50% of hosts for the third, incomplete dataset, and the number of iterations to match the size of the data set so that both Vivaldi and Sequoia would use a similar number of measurements in the best case.

Figure 8 shows the CDF of relative errors for Sequoia with 1, 5, and 15 trees in comparison to Vivaldi for the three datasets. We will first discuss the two latency datasets: UC-PlanetLab (Figure 8(a)) and Cornell-King (Figure 8(b)).

For both latency datasets, Sequoia's prediction accuracy is comparable to Vivaldi. Of course, Sequoia's accuracy depends on the number of trees used and, in general, improves with more trees. Moreover, the number of trees required to achieve a satisfactory level of accuracy depends on the size of the dataset; Sequoia 5 and Sequoia 15 do well for UC-PlanetLab and Cornell-King datasets respectively. Moreover, the presence of triangle inequality violations do not seem to deter effectiveness of Sequoia or Vivaldi. Finally, both Sequoia and Vivaldi tend to have a heavy tail of high errors. In practice, the heavy-tail does not affect the usefulness of either of the systems as most applications can tolerate the occasional errors. In theory, fortunately, treating network properties as approximate tree metrics provides a good handle in characterizing the worst-case performance of tree embedding algorithms as shown in [1].

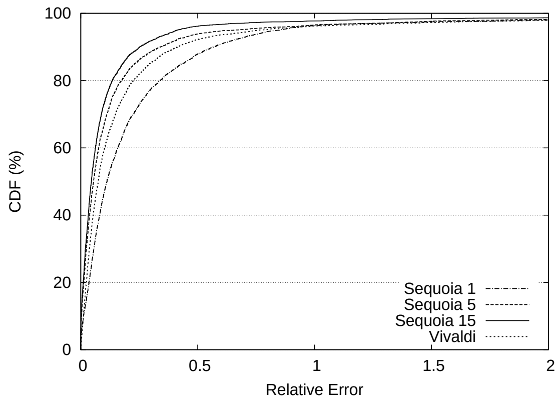
For the bandwidth dataset (Figure 9), Sequoia is clearly able provide a reasonably accurate representation while Vivaldi fails. The inability of Vivaldi to model bandwidth is not surprising as Vivaldi was not expected to work in the first place; bandwidth measurements cannot be intuitively tied to a coordinate space with Euclidean distances whereas it fits well into tree metrics as explained in Section 2. Sequoia's prediction errors for bandwidth, however, seem to be higher than the errors for latency. We will show in the next section that Sequoia's bandwidth representation already has significant practical benefits. Finally, Sequoia 1 appears to perform really bad for this dataset; closer examination showed that this aberration is just the result of a bad random choice for the lever, and our use of median values across multiple trees is a good idea for eliminating such aberrations and outliers.

The previous figures showed that Sequoia provides good prediction accuracy. A natural inquiry is to understand the number of measurements required to achieve that accuracy. We next show the distribution of the number of measurements used for finding the anchors while each new host tried to join the trees. We plot a distribution because the number of measurements varies for each host. We plot the measurement overhead for different datasets as a CDF in Figure 10. Since each dataset has a different number of hosts, we plot the measurements as a fraction of the total number of hosts measured when a new host joins (x-axis).

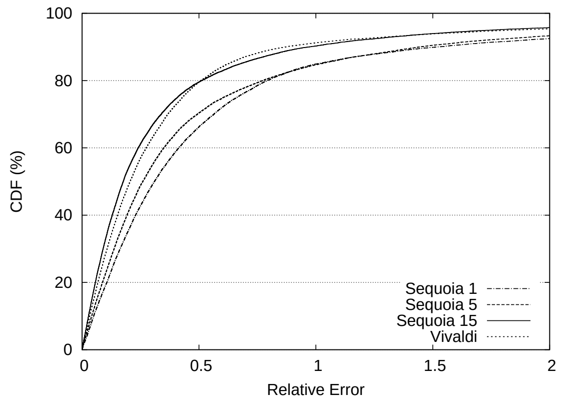
Overall, Figure 10 shows that tree construction only consumes a small fraction of the measurements. Moreover, the curves are fairly steep indicating that most hosts use a similar number of measurements. Occasionally, however, a host might perform a large number of measurements (as shown by the tail). The most important observation is that the fraction of measurements required decreases with the number of hosts in the datasets; its much lower (about 5% median) for the largest dataset (Cornell-King) compared to (about 20% median) the smallest dataset (UC-PlanetLab). This trend indicates that the measurement overhead scales sub-linearly with the number of hosts. In general, we expect the scaling to be logarithmic since the search is typically a walk down the tree.

4.2 Server Selection

Next, we demonstrate some practical benefits of Sequoia by showing how well it enables selection of closest and



(a) UC-PlanetLab Latency



(b) Cornell-King Latency

Figure 8: **Relative Error in Predicting Latency: Sequoia and Vivaldi have comparable accuracy in predicting latency .**

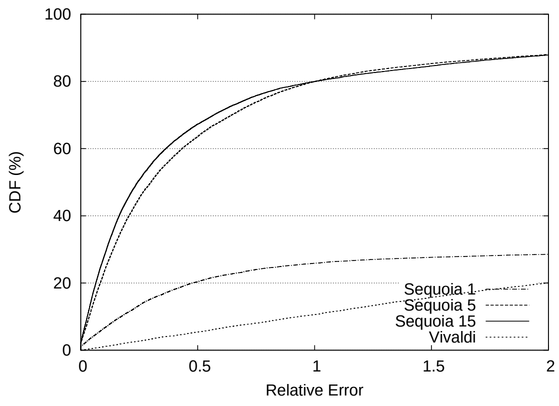


Figure 9: **Relative Error in Predicting Bandwidth: Sequoia shows an ability to represent available bandwidth with reasonable accuracy.**

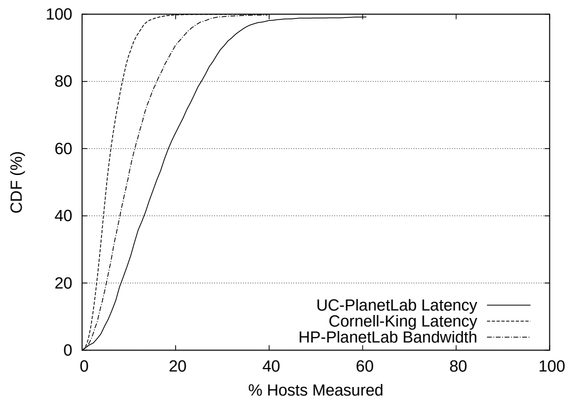


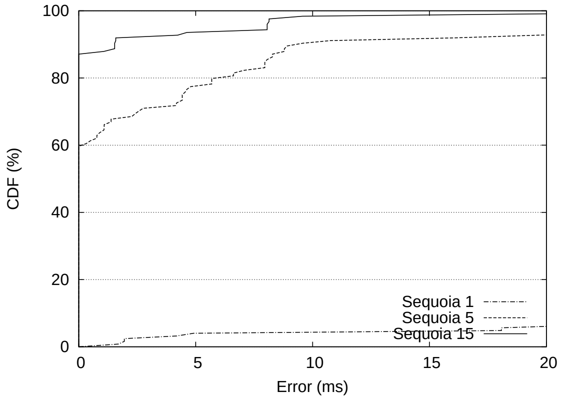
Figure 10: **CDF of Measurements Performed for Tree Construction: Tree construction requires measurements to a small fraction of hosts and the amount of measurements scales sub-linearly with the number of hosts.**

best-provisioned (highest bandwidth) hosts.

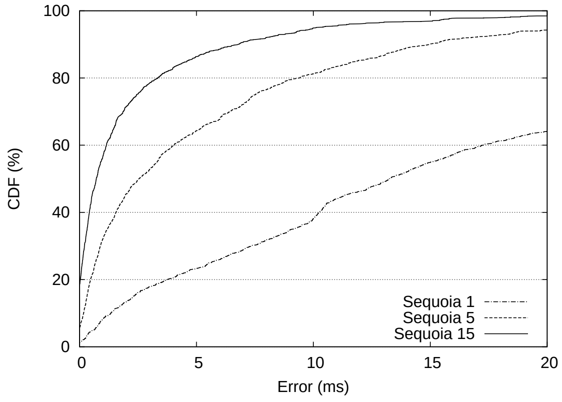
A *target host* trying to find the best server can be both an existing participant of the system or an external entity. In the former case, if the system employs Sequoia, the target is already part of the Sequoia trees; it can then find the best server through the tree search algorithm described in Section 3 without requiring additional measurements. In the latter case, where the target is an external entity, it might need to perform active measurements to guide its search. This results in a trade-off between the number of measurements performed and the quality of the best server found.

Here we show that with Sequoia, an external target can find a good quality server with a moderate measurement overhead. We show this through simulations where we choose each host in the dataset to be an external entity seeking to find the closest or the best-provisioned server for a latency or a bandwidth dataset respectively. The Sequoia trees built for the remainder of the dataset is then used to find the best server. The search algorithm on the Sequoia trees is restricted to only search at depth zero at each step, that is, as the anchor tree is descended from the lever only the children of each candidate is used to guide the search. Finally, the best host out of the candidates found by each tree is chosen as the selected server.

Figures 11 and 12 present the results for closest and best-provisioned server selection. They plot the quality of server selection as the error in the latency or bandwidth of the best host found by Sequoia versus the network measure to the best server in the dataset. We plot the absolute error for closest-server selection and the relative error for best-provisioned-server selection since



(a) UC-PlanetLab Latency



(b) Cornell-King Latency

Figure 11: **Error in Selecting Closest Server: Sequoia finds closest servers with small error margins.**

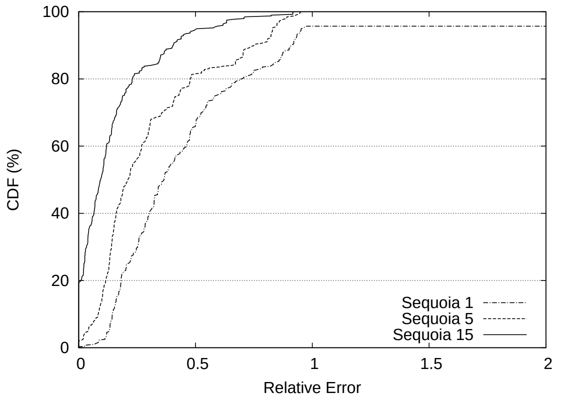


Figure 12: **Error in Selecting Best-Provisioned Server: Sequoia is able to identify better-provisioned servers with acceptable error margins.**

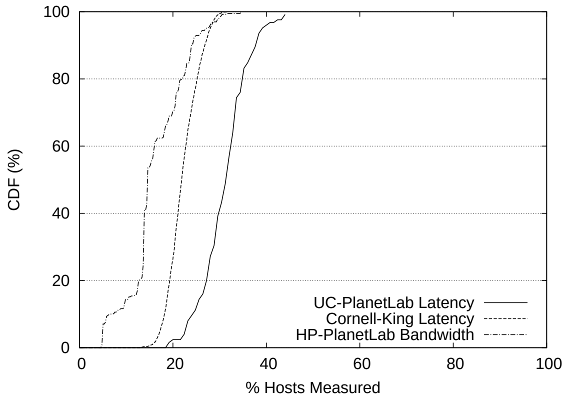


Figure 13: **CDF of Measurements Performed for Server Selection: Server selection requires measurements to a small fraction of hosts, and the amount of measurements scales sub-linearly with the number of hosts.**

bandwidth tends to have orders of magnitude variation.

Figures 11 and 12 show that server selection through Sequoia works well in practice. Using a reasonable number of trees (5 for the UC-PlanetLab and HP-PlanetLab datasets and 15 for the Cornell-King dataset), Sequoia finds a closest server within 10 ms about 80% of the time and a best-provisioned server with less than 50% error 80% of the time. The quality of selection of the best-provisioned host might appear to be inadequate from these numbers. However, server bandwidths vary in orders of magnitude and the challenge often is in selecting a 10 Mbps server over a 1 Mbps server. Moreover, the effectiveness of Sequoia's server selection can be further improved by increasing the number of trees used.

Figure 13 shows the other half of the tradeoff, namely, the measurement overhead while performing server selection. It plots the CDF of the measurement overhead in units similar to Figure 10 for the number of trees mentioned in the previous paragraph. Overall, this figure indicates that the number of measurements required was a small fraction of the total number of hosts in each case, with the fraction scaling sub-linearly with the number of hosts in the dataset as expected.

4.3 Topological Properties

Finally, we discuss the properties of the tree topology that Sequoia constructs. We first show the path lengths of each host to the lever in the anchor tree; recall that this path represents the distance label or the "coordinates" associated with each host. Short path lengths are desirable as they reduce the memory footprint for storing distance labels of hosts.

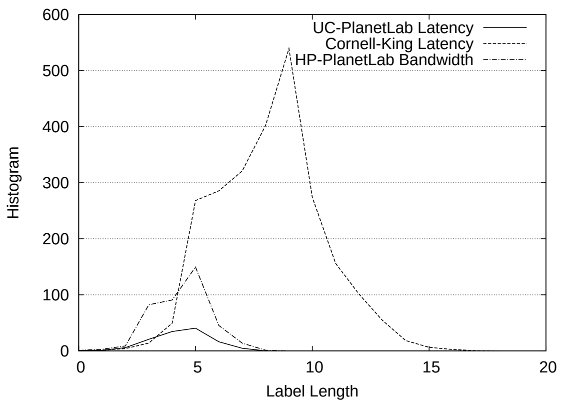


Figure 14: **Distribution of Lengths of Distance Labels: The distance labels computed based on the path to the lever are typically small and scales sub-linearly with the number of hosts.**

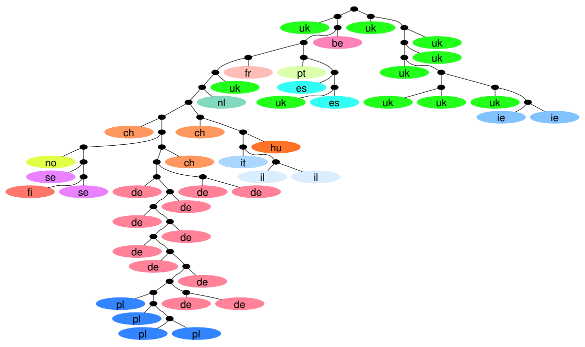


Figure 15: **Prediction Tree for PlanetLab Hosts in Europe: The prediction tree seem to separate hosts in different countries in Europe into well-defined regions.**

Figure 14 shows the distribution of the host-lever path lengths in the anchor trees for each dataset. As expected, the path lengths are variable since Sequoia's tree-construction algorithms don't guarantee a perfectly-balanced tree. Yet, they are typically small and show low variance. For instance, in the largest Cornell-King dataset of 2500 hosts, the label lengths are all under 16 with the mode of the distribution being a reasonable 9 per tree. Even though these numbers are much higher than the typical number of coordinates in Vivaldi or GNP, its still small enough for the capacities of modern systems.

While we presented aggregate statistics so far, the trees that Sequoia constructs also provide surprising revelations. We show a portion of the Sequoia tree constructed for the UC-PlanetLab latency dataset in Figure 15. At a high level, we found that the Sequoia tree was able to isolate hosts in different continents in the world into well-defined regions in the tree (sub-trees). This figure shows the European portion. The hosts are shaded by countries for clarity.

The Sequoia tree isolates hosts in different regions of Europe into well-defined clusters. For instance, there is a cluster at the top-right consisting of hosts in UK and Ireland (ie) and another at at the bottom-left with hosts in Poland (pl) and Germany (de). Hosts in larger regions, Spain (es) and Portugal (pt) and Norway (no), Sweden (se), and Finland (fi) are also well-separated. The clustering is not perfectly geographic, however; a couple of UK nodes seem to be wrongly clustered.

We believe that this result strongly supports our intuition for treating network properties as tree metrics. It indicates that the Internet is largely hierarchical, more hierarchical in some regions (Europe) than others (USA), and the observed latencies follow the hierarchy. This result also opens out Sequoia to be valuable for a wider-class of applications that benefit from building topology-aware overlays or hierarchical distributed systems.

4.4 Summary

Overall, the evaluation substantiates three key contributions of Sequoia. First, Sequoia provides the novel ability to construct an intuitive model for bandwidth using a small set of measurements and enable practical applications to perform effective bandwidth-based server selection. Second, Sequoia extends the same intuition to model latency while providing the same ease-of-use and accuracy as a state-of-the-art, coordinate-based latency model. Finally, Sequoia's tree models are well-correlated with the Internet topology making it a promising tool to build topology-aware systems. These observations were drawn using real-world datasets with the usual inconsistencies and vagaries making a strong case for Sequoia's practicability.

5 Applications

Sequoia promises unique abilities to networked systems. In this section, we discuss how Sequoia could benefit different network applications.

Server Selection: Several applications often have the need to select a "best" host from a set of other hosts based on some quality criterion such as distance, bandwidth, load, or a combination of such criteria. Typical scenarios where this need arises include: a) peer-to-peer structured DHTs such as Chord [23] and Pastry [21], which try to connect peers with other closer peers as neighbors, b) peer-to-peer file sharing services and content distribution networks such as BitTorrent, in which, a peer host likes to download torrents from another closer, well-provisioned peer, and c) clients of online video streaming services that like to enrich their experience by connecting to a closer server with high bandwidth. We already showed how Sequoia enables selection of closest and best-provisioned hosts.

Constraint Satisfaction: A few applications require host selection based on more complex constraints compared to the simple criterion above. For instance, Voice-over-IP (VoIP) services, such as Skype, often try to locate an intermediate relay node with good quality paths to two end hosts, while online gaming systems with multiple players, such as XBox LIVE, benefit from a well-placed coordination server with good paths to all the client hosts.

Locating a server with good connections to multiple target hosts often requires extensive search among the set of hosts in the system. Sequoia can serve as an efficient data structure for resolving such constraint satisfaction queries. For instance, the common ancestor in the distance labels of the target hosts might be a good starting point for doing such searches.

Hierarchical Organization: Finally, several distributed systems build a topology-aware hierarchy between the participating hosts: application-level multicast and video streaming protocols such as End System Multicast [4] and Bayeux [28], distributed network monitoring systems such as Astrolabe [19] and SDIMS [26], peer-to-peer overlays such as Meridian [25] and Coral [10]. Sequoia provide an inherent, topology-aware hierarchy for such distributed systems. Even though our tree models are virtual, that is, intermediate nodes are not real hosts, the virtual tree can be easily converted into a real, topology-aware tree by using known clustering protocols such as [19].

6 Related Work

Efforts for mapping and modeling Internet topology broadly fall into two categories: 1) end-to-end approaches that fit end-to-end measurements to predetermined models and 2) end-to-middle approaches that construct topology maps by probing routers and gateways in the core.

In the end-to-end category are several network positioning systems that fit Internet latency to simple topology models. GNP [16], Vivaldi [6], ICS [14], and PIC [5] assign synthetic coordinates from a low-dimensional (2-8) Euclidean space to each node and predict inter-node latencies by computing the Euclidean distance. Among other approaches, OASIS [9] use closest known geographic coordinates of the hosts to model position, while IDMaps [8] uses triangulation with respect to well-placed landmarks to position the nodes. While the above techniques give good accuracy for predicting network latency, their suitability for other path measures such as bandwidth has not been explored.

Sequoia is a similar modeling system based on end-to-end measurements that introduces a new, more insightful approach applicable to multiple network measures. It models bandwidth in addition to latency while providing a similar, convenient distance-label abstraction as the coordinates-based approaches. Moreover, Sequoia's topology models show a surprising similarity to the Internet.

In contrast to the above end-to-end approaches, recent systems such as iPlane [15] and S^3 [27] have taken up the effort to measure the core of the Internet. They probe a large number of known routers and gateways (available in public sources such as Route Views [34]) with a wide-range of tools that can measure or estimate inter-link latency, loss-rate, and bandwidth using PlanetLab hosts as vantage points. Through extensive measurements and clever inference mechanisms, they build a topological map that serves as a useful information service for the Internet.

Such a network information service based on measurements of the core is definitely in a better position to provide an accurate view of the Internet. However, a small to medium-sized networked system may not find it cost-effective to repeatedly measure a large portion of the Internet core. It could, on the other hand, consult a third-party service to obtain the necessary information. But the service may not be monitoring the part of the network where the system is deployed (in an enterprise network, for example), may be forbidden from use due to policy reasons, or may not be affordable. Sequoia promises a cost-efficient yet effective alternative, which can be independently and repeatedly deployed.

Finally, a large body of theoretical work [18, 12, 2], and most recently [1], exists on the topic of of embedding a metric space into a tree. These works provide lower bounds on the accuracy of tree embeddings and provide algorithms with proven upper bounds for constructing tree models.

7 Conclusions

Bandwidth is a critical network property, awareness of which substantially enriches the experience of clients of online video streaming services and participants in peer-to-peer content distribution systems. Yet few systems exist today that facilitate clients and peers to make preferential selection of high bandwidth servers over low bandwidth ones. This paper presented an enabling system called Sequoia that provides bandwidth-awareness to applications.

In that process, Sequoia introduces a new approach for representing bandwidth and latency with a single unifying model. This approach based on embedding network measures into trees appears to fit well into the largely hierarchical Internet. Sequoia leverages this fit to provide both bandwidth-based and latency-based functionalities for path quality prediction, server selection, and hierarchical clustering. This paper validated the above intuition, presented the design and implementation of Sequoia, and evaluated its benefits. Overall, tree-embeddings of network measures turns out to be a greatly promising approach to facilitate network-awareness.

References

[1] I. Abraham, M. Balakrishnan, F. Kuhn, D. Malkhi, V. Ramasubramanian, and K. Talwar. Reconstructing Approximate Tree Metrics. In *Proc. of Symposium on Principles of Distributed Computing (PODC)*, Portland, OR, Aug. 2007.

[2] I. Abraham, Y. Bartal, and O. Neiman. Embedding Metrics into Ultrametrics and Graphs into Spanning Trees with Constant Average Distortion. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, New Orleans, LA, Jan. 2007.

[3] P. Buneman. A Note on Metric Properties of Trees. *Journal of Combinatorial Theory,*, 17:48–50, 1974.

[4] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. In *Proc. of Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Santa Clara, CA, June 2000.

[5] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, Mar. 2004.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of ACM SIGCOMM Conference*, Portland, Oregon, USA, Aug. 2004.

[7] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characeterizing Residential Broadband Networks. In *Proc. of SIGCOMM Internet Measurement Conference (IMC)*, San Diego, CA, Oct. 2007.

[8] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, 2001.

[9] M. Freedman, K. Laskhminarayanan, and D. Mazières. OASIS: Anycast for Any Service. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006.

[10] M. Freedman and D. Mazières. Sloppy Hashing and Self-Organizing Clusters. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb. 2003.

[11] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. of SIGCOMM Internet Measurement Workshop (IMW)*, Marseille, France, Nov. 2002.

[12] A. Gupta. Steiner Points in Tree Metrics don't (Really) Help. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Washington, DC, Jan. 2001.

[13] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang. A Measurement Study of Internet Bottlenecks. In *Proc. of INFOCOM Conference*, Miami, FL, Mar. 2005.

[14] H. Lim, J. Hou, and C.-H. Choi. Constructing Internet Coordinate System based on Delay Measurement. In *Proc. of ACM/SIGCOMM Internet Measurement Conference (IMC)*, Oct. 2003.

[15] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *Proc. of the Usenix Conference on Operating Systems Design and Implementation (OSDI)*, Nov. 2006.

[16] E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, 2002.

[17] R. Prasad, M. Murray, C. Dovloris, and kc Claffy. Lower Bounds on the Distortion of Embedding Finite Metric Spaces in Graphs. *Discrete & Computational Geometry*, 19, 1998.

[18] Y. Rabinovich and R. Raz. Lower Bounds on the Distortion of Embedding Finite Metric Spaces in Graphs. *Discrete & Computational Geometry*, 19, 1998.

[19] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[20] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Avalable Bandwidth Estimation for Network Paths. In *Proc. of Passive and Active Measurement Workshop*, San Diego, CA, Apr. 2003.

[21] A. Rowstorn and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, Nov. 2001.

[22] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-scale Event Notification Infrastructure. In *Proc. of Workshop on Networked Group Communications*, London, UK, Nov. 2001.

[23] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[24] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterzing the Internet Hierarchy from Multiple Vantage Points. In *Proc. of the Infocom Conference*, New York, NY, June 2002.

[25] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service Without Virtual Coordinates. In *Proc. of ACM SIGCOMM Conference*, Philadelphia, PA, Aug. 2005.

[26] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. In *Proc. of SIGCOMM*, Porland, OR, Aug. 2004.

[27] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. S^3: A Scalable Sensing Service for Monitoring Large Networked Systems. In *Proc. of Workshop on Internet Network Measurement*, Pisa, Italy, Sept. 2006.

[28] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalabe and Fault-Tolerant Wide-Area Data Dissemination (NOSSDAV). In *Proc. of Workshop on Network and Operating Systems Support for Digital Audio and Video*, Port Jefferson, NY, June 2001.

[29] Meridian: A Lighweight Approach to Network Positioning. `http://www.cs.cornell.edu/People/egs/meridian`.

[30] PlanetLab: An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services. `http://www.planet-lab.org`.

[31] Network Coordinate Research at Harvard. `http://www.eecs.harvard.edu/~syrah/nc/`, 2006.

[32] The Gnutella 0.4 Protocol Specification. http://dss.clip2.com/GnutellaProtocol0.4.pdf, 2000.

[33] S^3: Scalable Sensing Service. `http://networking.hpl.hp.com/s-cube`.

[34] University of Oregon Route Views Project. `http://www.routeviews.org`.

[35] All-Sites-Pings for PlanetLab. `http://ping.ececs.uc.edu/ping/`, 2006.