Decentralized, Accurate, and Low-Cost Network Bandwidth Prediction

Sukhyun Song, Pete Keleher, Bobby Bhattacharjee, and Alan Sussman
UMIACS and Department of Computer Science, University of Maryland
{shsong, keleher, bobby, als}@cs.umd.edu

***Abstract*—The distributed nature of modern computing makes end-to-end prediction of network bandwidth increasingly important. Our work is inspired by prior work that treats the Internet and bandwidth as an approximate *tree metric space*. This paper presents a decentralized, accurate, and low cost system that predicts pairwise bandwidth between hosts. We describe an algorithm to construct a distributed tree that embeds bandwidth measurements. The correctness of the algorithm is provable when driven by precise measurements. We then describe three novel heuristics that achieve high accuracy for predicting bandwidth even with imprecise input data. Simulation experiments with a real-world dataset confirm that our approach shows high accuracy with low cost.**

I. Introduction

Network bandwidth is an important factor in determining the performance of distributed computing applications. Since bandwidth measurements are generally expensive to perform, we would expect that networked applications can greatly benefit from the ability to predict pairwise bandwidth without performing full *n-to-n* measurements. For example, a peer-to-peer (P2P) computational grid system [1] could increase its performance by finding high-bandwidth nodes (and links) to store large scientific input or output datasets.

Unfortunately, however, there exists no effective framework that can predict bandwidth between hosts in a decentralized fashion. Euclidean coordinate spaces are not a good model for embedding bandwidth measurements. Accordingly, attempts [2] to use a traditional network coordinate system do not work well in predicting bandwidth, resulting in poor accuracy. Ramasubramanian et. al [2] claim that the Internet and bandwidth can be modeled by an approximate tree metric space that almost satisfies the four-point condition [3]. Based on this finding, they proposed a new method for bandwidth prediction, where bandwidth measurements are embedded as distances in an edge-weighted tree. The result centralized system has been shown to have reasonably high accuracy.

Inspired by the success of tree-embedding approach, our study focuses on decentralization. We feel that a fully decentralized bandwidth prediction system must consider four requirements: i) there must exist no centralized data structure and no centralized component, ii) bandwidth should be predicted accurately, iii) measurement traffic should be scalable as the number of hosts increases, and iv) the system should be able to adapt to network changes. Our study appeared originally in [4] as an extended abstract. The contributions of this paper are fourfold. First we describe the design of a decentralized bandwidth prediction system that satisfies all four requirements above. Especially, our algorithm does not require a centralized component (such as a set of landmark nodes) that all hosts must communicate with to measure bandwidth. Second, we provide a theoretically provable algorithm. The edge-weighted tree constructed by our algorithm embeds bandwidth measurements without any error when we assume that bandwidth measurements are exactly represented as a tree metric space. The third contribution is a set of three new heuristics that allow high prediction accuracy in the real Internet. Finally, we present simulation results validating the high accuracy and low cost for our algorithm.

The rest of the paper is organized as follows. We first discuss the underlying intuition behind this work in Section II. Section III describes the algorithm design, and presents the techniques used to achieve high prediction accuracy in the real world. Finally, Section IV evaluates our approach experimentally, and we conclude and discuss future work in Section V.

II. Background and Related Work

A. Definitions

- A metric space is an ordered pair (V, d) where V is a set of nodes and d is a metric (distance function) on V.
- An *edge-weighted tree* is a connected graph without cycles, and with non-negative edge weights.
- The *distance* between two nodes u and v on an edge-weighted tree T, denoted $d_T(u, v)$, is defined by the sum of weights of edges on the path from u to v.
- An edge-weighted tree T *induces* a metric space (V, d) if and only if T contains all nodes in V and $\forall u, v \in V$, $d(u, v) = d_T(u, v)$ holds.
- The *four-point condition (4PC)* on a metric space (V, d) states that for any set of four nodes $w, x, y, z \in V$, $d(w,x)+d(y,z) \le d(w,y)+d(x,z) \le d(w,z)+d(x,y)$ implies $d(w,y) + d(x,z) = d(w,z) + d(x,y)$.
- A metric space that satisfies 4PC is called a *tree metric space*.

B. Bandwidth as a Metric and Treeness of the Internet

Higher values are considered better for bandwidth while closer is generally more desirable for distance in a metric space. So, Ramasubramanian et. al [2] used the *linear transform function* $d(u, v) = C - BW(u, v)$ to represent bandwidth as a metric, where $BW(u, v)$ is the bandwidth between nodes u and v, $d(u, v)$ is the distance in a metric space, and C is a constant. Representing bandwidth as a metric implies four properties: i) $d(u, v) \geq 0$, ii) $d(u, v) = 0$ if and only if $u = v$, iii) $d(u, v) = d(v, u)$, and iv) $d(u, w) \leq d(u, v) + d(v, w)$. The first property is satisfied by having a large value for C, for example, the expected maximum bandwidth. By setting $BW(u, u) = C$, we can also satisfy the second property. We satisfy the third property by setting both $BW(u, v)$ and $BW(v, u)$ to the average bandwidth of the forward and reverse directions. Even though no effective method has been found to directly address the last assumption, we provide several heuristics to accurately embed bandwidth information into a metric space in the real Internet, as described in Section III-D.

There are two pieces of evidence to verify that the Internet is close to a tree metric space for bandwidth. First, Ramasubramanian et. al [2] verify that a bandwidth dataset produce small values of parameter ε. ε was introduced by Abraham et. al [5] to quantify how closely a set of four nodes satifies 4PC. If all ε values in a metric space are zero, the metric space is a perfect tree metric space. Second, there is a theoretical model of network topology such that bandwidth between two nodes is bottlenecked in the first hop of routing path. It has been proved that a metric space for this model is a perfect tree metric space [2].

C. Approaches for Edge-Weighted Tree Construction

Theorem 2.1: A metric space (V, d) satisfies 4PC if and only if there exists an edge-weighted tree that induces (V, d).

By Theorem 2.1, we can expect to use an edge-weighted tree for bandwidth prediction. Buneman [3] proved Theorem 2.1 by providing the first algorithm to construct an edge-weighted tree for a given tree metric space. However, unlike our incremental iterative algorithm described in Section III, Buneman's recursive algorithm does not allow nodes to be incrementally added to existing trees. Since the resulting edge-weighted tree would not be expandable, we cannot directly apply the algorithm in practice when nodes dynamically join a distributed system.

Abraham et. al [5] proposes a tree construction algorithm for an approximate tree metric space. Even though the theoretical work has a contribution in providing upper and lower bounds on the accuracy of tree embedding, it suffers the same problem as Buneman's algorithm for practical uses because it also uses a non-incremental recursive algorithm.

Our research is inspired by the Sequoia system [2], which uses a tree-embedding model for bandwidth prediction and proposes an incremental iterative tree construction algorithm for the first time. We naturally use the same terms as the Sequoia authors do to explain our algorithm even though some terms have somewhat different meanings. Our study has several contributions relative to Sequoia. First, our system is fully decentralized and does not require a centralized component. To participate in the Sequoia system, each node must measure bandwidth with several nodes starting from a fixed landmark node called the lever node. On the other hand, each node joins our system by performing measurement with an arbitrary set of nodes starting at a random node. Second, we can prove that our algorithm constructs an edge-weighted tree that induces a tree

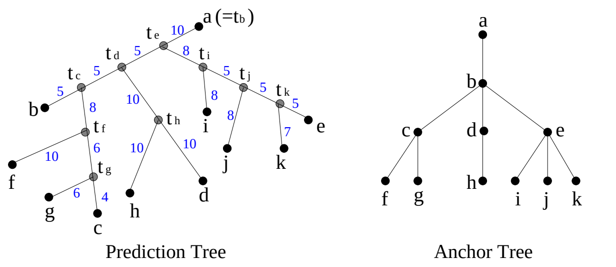


Fig. 1. A Prediction Tree and a Corresponding Anchor Tree

metric space without any error. Third, we succeed in achieving higher accuracy in a real world network by introducing three novel heuristics. Sequoia uses an algorithm that fits a perfect tree metric space directly in practice, and results in lower accuracy than our approach, as shown in Section IV.

III. Design

The overall design goal is to construct an edge-weighted tree that induces a tree metric space. This section describes details of the design starting from a centralized algorithm. The algorithm is extended to a partially decentralized algorithm that reduces measurement traffic without relying on any landmark nodes. Then we discuss how to build a fully decentralized algorithm by distributing data structures. Heuristics to improve prediction accuracy are provided last.

A. Centralized Algorithm to Construct a Prediction Tree

An edge-weighted tree embedding bandwidth information is called a *prediction tree* (Fig. 1). The number on each edge represents the weight of the edge. A leaf node in a prediction tree has degree one and represents each participating host in the system. An inner node with degree two or more is created when a new leaf node is added. The linear transform function $d(u, v) = C - BW(u, v)$ is used to represent bandwidth as a metric, and $BW_T(u, v) = C - d_T(u, v)$ for bandwidth prediction. For example, in Fig. 1 if $C = 100$, the predicted bandwidth value $BW_T(b, c)$ is 77 because $d_T(b, c) = 23$.

Algorithm 1: **AddNode**$(\mathbf{T}, \mathbf{x})$: Add a node x to a prediction tree T using a random base node.

```
1 Pick a base node z from any leaf nodes in T
2 Measure $d(x, s)$ for all leaf nodes $s$ in $T$
3 Find an end node $y$ that maximizes $(x|y)_z$
4 Put $x$'s inner node $t_x$ on the path $z \sim y$ where
  $d_T(z, t_x) = (x|y)_z$ holds
5 Add $x$ with edge $(t_x, x)$ of weight $d(z, x) - (x|y)_z$
```

A prediction tree starts with the first added node as a singleton, and the second node is added along with an edge that connects the two nodes and is weighted by their distance. The tree grows by iteratively adding nodes as shown in Algorithm 1. To add a new node x to a prediction tree, the algorithm chooses a node z called the *base node*, which can be any leaf node, and selects another node y called the *end node* that maximizes *Gromov product* $(x|y)_z$. The Gromov product of x and y at z, denoted $(x|y)_z$, is defined here as $(x|y)_z = \frac{1}{2}(d(z, x) + d_T(z, y) - d(x, y))$. x's *inner node* t_x is created and located on the path $z \sim y$ where $d_T(z, t_x) = (x|y)_z$. The algorithm then adds x to the prediction tree by creating an edge (t_x, x) of weight $d(z,x) - (x|y)_z$, so that $d_T(z,x) = d(z,x)$. The key idea in Algorithm 1 is to choose a random base node, and this provides an indispensable underpinning for a decentralized system relying on no fixed landmark node. From Theorem 3.1, we know that bandwidth information is correctly embedded into the prediction tree constructed by Algorithm 1.

Theorem 3.1: (Correctness of Algorithm 1) Given a tree metric space (V,d) and a node $x \in V$, let T be an edge-weighted tree that induces a subspace $(V \setminus \{x\}, d)$ of (V,d). If Algorithm 1 adds x to T and creates a new edge-weighted tree T', then T' will induce (V,d).

B. Reducing Measurement Traffic with an Anchor Tree

Lines 2∼3 of Algorithm 1 show a naive way to find an end node, which measures bandwidth to all n hosts in the system. It is important to reduce this number of measurements to a scalable amount in developing an efficient prediction system. This leads us to introduce an *anchor tree*.

An anchor tree is a rooted unweighted tree where each node represents a host in the system. The first added node in the system becomes the root node of the anchor tree, and the second node becomes the child of the root node. Then the anchor tree grows along with a prediction tree following Algorithm 2, which is an improved version of Algorithm 1 with respect to end node search time. When adding a new node x to a prediction tree, the algorithm moves up and down on an anchor tree starting from a random base node z until finding a Gromov product maximizer y. At each hop, the algorithm creates a set of candidate nodes to be an end node (CAND) by considering the currently visited node and all its neighbors, and measures $d(x,s)$ for each candidate node s. The algorithm then moves in the direction that locally maximizes the Gromov product until reaching a global maximizer. Once x is added to the prediction tree in the same way as in Algorithm 1, x is also added to the anchor tree by becoming a child of x's *anchor node*. x's anchor node is defined as a node that was previously added to the prediction tree along with the edge that x's inner node t_x is located on. For example, assuming that nodes in Fig. 1 are added to the system in an alphabetical order, when adding h to a prediction tree, h's inner node t_h is located on edge (t_d, d). Node d is defined as h's anchor node because edge (t_d, d) is created when d was added. Using Lemma 3.2 and mathematical induction, we can prove that a prediction tree constructed by Algorithm 2 correctly induces a tree metric space. The lemma deals with the case where part of a prediction tree can be excluded from end node searching.

Lemma 3.2: Given a tree metric space (V,d) and a node $x \in V$, let T be an edge-weighted tree that induces a subspace $(V \setminus \{x\}, d)$ of (V,d). For three leaf nodes z, y, and w in T, let t be an inner node on path $z \sim y$ at distance $(y|w)_z$ from z. Let S be a set of leaf nodes in all the subgraphs connected to (i.e., rooted at) the inner nodes on path $t \sim w$. If $(x|y)_z \geq (x|w)_z$, then $(x|y)_z \geq (x|s)_z$ $\forall s \in S$.

The number of measurements needed to run Algorithm 2. depends on the number of visited nodes and the degree (the number of neighbors) of each visited node. Those numbers are both dependent on the shape of the anchor tree and the locations of the base node and end node. Also, for a given metric space, an anchor tree has different shapes from different node addition orderings. The best case is when the measurement starts at a leaf node and ends at its parent node that has no other child nodes, so takes $O(1)$ measurements. For some poor orderings for adding nodes, the algorithm can produce a long chain-style anchor tree of $O(n)$ depth or a shallow anchor tree with a node of $O(n)$ degree, so that the worst case number of measurements taken is $O(n)$.

Algorithm 2: **FastAddNode**$(\mathbf{T}, \mathbf{A}, \mathbf{x})$: Add a node x to a prediction tree T and an anchor tree A, skipping some unnecessary measurements.

```
1  Pick a base node $z$ from any leaf nodes in $T$
2  $y \leftarrow z$
3  while true do
4      CAND $\leftarrow \{y, y$'s parent and child nodes in $A\}$
5      Measure $d(x,s)\ \forall s \in$ CAND
6      MAX $\leftarrow \mathrm{argmax}_{s \in \mathsf{CAND}} (x|s)_z$
7      if $y \in$ MAX then break else $y \leftarrow$ one node in MAX
8  Put $x$'s inner node $t_x$ in $T$ on the path $z \sim y$ where
   $d_T(z, t_x) = (x|y)_z$ holds
9  Add $x$ to $T$ with edge $(t_x, x)$ of weight $d(z,x) - (x|y)_z$
10 Let $x$'s anchor node $a$ be a node that was previously
   added with the edge $t_x$ is located on
11 Add $x$ to $A$ as $a$'s child node
```

C. Fully Decentralizing with Distributed Structures

A *distance label* is assigned to each node, so that we can construct a prediction tree in a distributed fashion. Node x's distance label contains all anchor nodes on the path from the root node to x in the anchor tree. The distance label also contains the corresponding distance values between anchor nodes and inner nodes. Suppose there are k anchor nodes in x's distance label, from x's anchor node a_1 to the root node a_k. a_{i+1} is a_i's anchor node, and t_{a_i} is a_i's inner node for $1 \leq i \leq k-1$. Then x's distance label is denoted by:

$$(a_k \xrightarrow[d_T(t_{a_{k-1}}, a_{k-1})]{d_T(a_k, t_{a_{k-1}})} a_{k-1} \ \cdots \ a_2 \xrightarrow[d_T(t_{a_1}, a_1)]{d_T(a_2, t_{a_1})} a_1 \xrightarrow[d_T(t_x, x)]{d_T(a_1, t_x)} x)$$

For example, node d in Fig. 1 has $(a \xrightarrow[25]{0} b \xrightarrow[20]{10} d)$ as its distance label, because $d_T(a, t_b) = 0$, $d_T(t_b, b) = 25$, $d_T(b, t_d) = 10$, and $d_T(t_d, d) = 20$. Since a distance label is equivalent to a partial prediction tree, the distance between two nodes can be estimated with a simple computation. The participating nodes build an overlay network that directly matches the structure of the anchor tree. We can use Algorithm 2 to construct these distributed structures with a slight modification. A joining node x first chooses a random base node z from the system in the same way as a bootstrapping node is identified in a general structured P2P system. x measures bandwidth to several nodes while moving around the overlay network until finding an end node y. Then x determines its anchor node a and its distance label by using the distance labels of z and y to figure out where x's inner node t_x is located on a prediction tree. Finally, x becomes a's child node by notifying a of its join event.

Our system maintains and restructures the overlay network in response to a changing network environment by having each node send periodic heartbeat messages to its neighbor nodes. We first discuss dealing with failover. When a node m fails, one of m's child nodes c takes over m's role, and m's other child nodes become c's child nodes. c should be chosen such that c's inner node t_c is the closest to m among the inner nodes on the path from m to m's inner node t_m. This is because path $t_m \sim c$ must be long enough to contain all the other inner nodes that were originally put on path $t_m \sim m$. t_m becomes c's new inner node. c must adjust its distance label to reflect this change. c's takeover event is propagated down to the nodes in c's new subtree, so that they can update their distance labels. For example, in Fig. 1, if e fails then k takes over its role because t_k is closer to e than any other inner nodes on path $e \sim t_e$. i and j become k's child nodes.

When bandwidths change dynamically over time, we restructure the part of the system where bandwidth changes occur. When x detects a significant difference between $d(x, y)$ and $d_T(x, y)$ for some y that x has measured before, it leaves the system, then joins again using y as its base node. x's rejoin process is normally faster than an initial join process, because x can utilize bandwidth measurements it has already made.

D. Tolerating Imperfect Data

Although the Internet approximates a tree metric space, directly applying the previously described algorithms in practice will result in low accuracy for bandwidth prediction. We now describe useful heuristics to improve prediction accuracy.

1) Error Minimization: Deciding the position of a new node x in a prediction tree by finding a Gromov product maximizer guarantees perfect prediction accuracy in an ideal world, as shown by Theorem 3.1. However, when the 4PC does not hold in a metric space, this is no longer an effective approach to achieve high accuracy. We therefore modify the algorithm to find x's position that minimizes relative prediction error, rather than finding the Gromov product maximizer. In the first phase, x collects measurement data. At each hop, x computes its temporary position in a prediction tree as for the base node and each candidate end node. With each of x's temporary positions, x estimates relative prediction errors, using the formula $\frac{|BW(x,t) - BW_T(x,t)|}{BW(x,t)}$, for each node t out of n_t nodes that x has measured so far. Then x moves to the candidate node that minimizes the average of the n_t relative prediction errors. Once an error minimizer is found, x determines its final position in the second phase with the collected measurement data. For each of all possible pairs of nodes in the set of n_t measured nodes, again, x computes its temporary position and estimates the average relative prediction error. The position that minimizes the error is selected as x's final position. Note that while only the pairs with the single base node are considered in the first phase, all the possible pairs are used to find the best position of x in the second phase.

We have found that minimizing relative error shows better accuracy on a real-world bandwidth dataset than moving to the local Gromov product maximizer. Since x uses the bandwidth data it has already collected, this heuristic does not cause any additional measurements to be made. The heuristic also does not affect accuracy in the ideal world scenario.

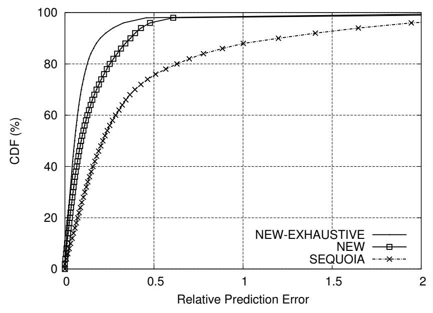
2) Rational Transform Function: A linear transform function $BW_T(u, v) = C - d_T(u, v)$ is used for bandwidth prediction in the base system. Unlike the ideal world scenario, $d_T(u, v)$ might not be equal to $d(u, v)$ in the real network. If $d_T(u, v)$ is much larger than $d(u, v)$, that can result in predicting a negative bandwidth value and will decrease overall prediction accuracy. To overcome this problem, we use a *rational transform function* $d(u, v) = \frac{C}{BW(u,v)}$ for embedding and $BW_T(u, v) = \frac{C}{d_T(u,v)}$ for prediction, so that the predicted bandwidth is always positive even when $d_T(u, v)$ is overestimated. As does the linear function, the rational transform function inverts ordering of bandwidth after performing the transformation. The second property of a metric space, as described in Section II, is satisfied by setting $BW(u, u) = \infty$. An additional benefit of this change is that it adds no extra costs into the system, similar to the error minimization heuristic.

3) Deep Search: The previously discussed node join algorithm considers only direct neighbors as candidate nodes to be an end node. We can modify that to take advantage of additional candidates by using indirect two-hop neighbors. Unlike the other two heuristics, however, deep search does require more measurements with additional candidates.

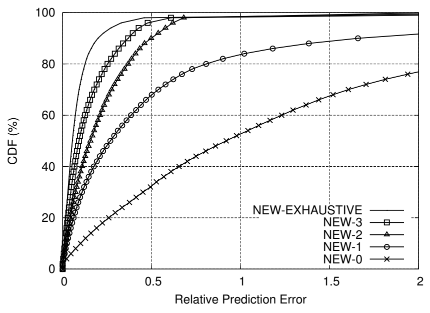
IV. Evaluation

This section evaluates our approach by examining i) accuracy and cost, ii) heuristic efficacy, and iii) scalability of measurement traffic. Our simulations are based on the HP-PlanetLab dataset described in [2]. This dataset contains available bandwidth measurements between PlanetLab nodes collected at HP Labs [6] using pathChirp [7]. Since the raw dataset is incomplete and has many unmeasured pairs of nodes, we first extracted measurements for the 190 nodes (out of 459) that give a full *n-to-n* asymmetric matrix containing bandwidth measurements. We then converted the matrix to a symmetric one by averaging bandwidth values from forward and reverse directions for each pair of nodes. The treeness of the dataset is measured with ε values [5]. The preprocessed dataset produces 0.186 for the average ε when the rational transform function is used with $C = 10000.0$ (Mbps).

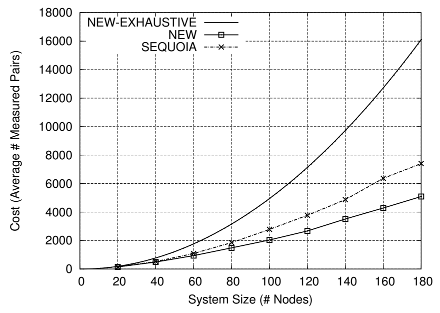
We show experimental results for three different approaches in Fig. 2. **NEW** shows results from our system using all three heuristics: error minimization, rational transform function, and deep search. **NEW-EXHAUSTIVE** is a special case of NEW that uses exhaustive *n-to-n* measurements to construct a prediction tree with the highest prediction accuracy. Each joining node performs measurements with all existing nodes in the system and finds its best position. While this approach is only theoretical, not practical, it provides an upper bound on



(a) Accuracy



(b) Heuristic Efficacy



(c) Scalability

Fig. 2. Experimental Results

the potential results for the real algorithms. **SEQUOIA** refers to our simulation of the centralized Sequoia algorithm [2].

We first constructed the systems for the three approaches by adding 190 nodes in the same order. Fig. 2(a) shows the CDF of the relative errors for all possible pairs among the 190 nodes. The relative bandwidth prediction error between node u and v is defined as $\frac{|BW(u,v)-BW_T(u,v)|}{BW(u,v)}$. NEW is more accurate than SEQUOIA, with more than 90% of the node pairs having a relative error less than 0.5, and closely tracks NEW-EXHAUSTIVE. We also estimated the total number of measurements needed to construct the system. NEW-EXHAUSTIVE causes 17955 measurements between all 190 nodes. NEW requires 5652 measurements, much fewer than the 11328 for SEQUOIA. Sequoia's high costs come from the top-down node join process with a fixed base node in an anchor tree. A new node x performs measurements starting at the root node r and its child node r_c, and passes through other nodes in a top-down way. The position of x's inner node t_x in a prediction tree is always determined on the path with the base node r fixed at one end. So, it is more likely that t_x is positioned on path $r \sim r_c$ in a prediction tree, and x becomes r_c's child node in an anchor tree. r_c in SEQUOIA actually has 113 child nodes. This high degree of high-level node causes heavy traffic as the next joining node performs measurements in a top-down way. On the other hand, our approach allows t_x to be positioned on a path with a random base node. So, there is a smaller chance that x chooses r_c as a parent, resulting in r_c's 27 child nodes in NEW. All the experiments performed with 10 different node addition orderings show similar results to those shown.

Fig. 2(b) shows how effectively the three heuristics discussed in Section III-D raise the accuracy of our system. **NEW-0** shows results for our system without any heuristics, **NEW-1** is the system with error minimization applied to NEW-0, **NEW-2** is the system with the rational transform function applied to NEW-1, and **NEW-3** is the system with deep search applied to NEW-2. (NEW-3 is equal to NEW in Fig. 2(a).) Those curves show how accuracy improves as the heuristics are applied one by one. Use of the heuristics allows accuracy to approach that of the exhaustive test case.

Fig. 2(c) shows scalability results, in terms of the measurement traffic needed to construct a system. For each system size, we created five different datasets, and we constructed a system 20 times with a different node join order for each dataset. Then we collected the total number of measurements to construct a system and computed an average value in each system size. NEW shows lower cost than SEQUOIA. The cost for SEQUOIA and NEW increases almost linearly while the exhaustive testing takes quadratic number of measurements.

V. Conclusions and Future Work

This paper has presented a decentralized and scalable system that accurately predicts pairwise bandwidths with low cost. Our node join algorithm does not require any fixed infrastructure, such as landmark nodes, that all hosts must measure bandwidth to and from. The algorithm also uses novel heuristics to achieve accuracy on imprecise, real-world, datasets. Simulation results show high accuracy, low cost, and scalability. We are currently extending this work in several ways. First, we are studying a cluster search algorithm to find a set of k nodes with the minimum interconnection bandwidth b. Second, we intend to use our system as the underlying technology for resource discovery in a P2P desktop grid [1].

Acknowledgment

We thank Venugopalan Ramasubramanian and the other developers of Sequoia for providing us with the HP-PlanetLab dataset used to evaluate our algorithm in Section IV.

References

[1] J.-S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman, “Resource discovery techniques in distributed desktop grid environments,” in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing - GRID 2006*. IEEE Computer Society Press, Sep. 2006.

[2] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, “On the treeness of internet latency and bandwidth,” in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems - SIGMETRICS/Performance 2009*. ACM, Jun. 2009.

[3] P. Buneman, “A note on the metric properties of trees,” *Journal of Combinatorial Theory, Ser. B*, vol. 17, pp. 48–50, 1974.

[4] S. Song, P. J. Keleher, B. Bhattacharjee, and A. Sussman, “Decentralized network bandwidth prediction,” in *Proceedings of the 24th International Symposium on Distributed Computing - DISC 2010*. Springer, Sep. 2010.

[5] I. Abraham, M. Balakrishnan, F. Kuhn, D. Malkhi, V. Ramasubramanian, and K. Talwar, “Reconstructing approximate tree metrics,” in *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing - PODC 2007*. ACM, Aug. 2007.

[6] “S3: Scalable sensing service.” [Online]. Available: http://networking.hpl.hp.com/s-cube/

[7] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, “pathchirp: Efficient available bandwidth estimation for network paths,” in *Proceedings of the 4th Passive and Active Measurement Workshop - PAM 2003*. Springer, Apr. 2003.