Fast Overlay Tree Based On Efficient End-to-End Measurements

Xing Jin Yajun Wang S.-H. Gary Chan
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{csvenus, yalding, gchan}@cs.ust.hk

Abstract— **Most of the application-layer multicast protocols use end-to-end delay as their primary metric. However, for applications such as stored video delivery, meeting a certain target bandwidth requirement is of primary importance. In this paper, we present a centralized approach on how to build a fast overlay tree based on efficient end-to-end measurements.**

We first investigate how to infer underlay topology (in terms of connectivity) with low measurement cost. Given N end-hosts, traditionally full $N(N-1)/2$ traceroutes are needed to accurately determine the underlay topology. We propose a much faster heuristic (Max-Delta) where a server selects appropriate host-pairs to probe in parallel so as to reveal the most information on the underlay in each round. Given an inferred network topology, we then present the algorithm of Fast Application-layer Tree (FAT), which builds an overlay tree of a certain target bandwidth by estimating possible load on each underlay link.

Simulation results show that almost full measurements are needed to discover completely underlay topology. However, substantial reduction in measurements (by almost an order of magnitude) can be achieved if some accuracy, say 5%, can be sacrificed. As compared to traditional ALM protocols such as Narada and Overcast, FAT achieves high bandwidth, low link stress, and low RDP.

I. Introduction

In order to overcome current limitations in IP multicast, application-layer multicast (ALM) has been proposed. While many ALM protocols have been proposed (such as Narada [1], Nice [2], DT [3], etc.), most of them focus on reducing end-to-end delay between the source and the end-hosts. The *tree bandwidth*, defined as the minimum path-bandwidth of an overlay tree, is often considered as secondary or not at all. As a consequence, the bandwidth of the ALM tree constructed may be too low for applications with a certain target bandwidth requirement. We address in this paper how to construct an ALM tree to support these applications using efficient end-to-end measurements.

A major challenge in constructing ALM tree of a target bandwidth is that two seemingly disjoint overlay paths may share common underlay links, thereof reducing the tree bandwidth. Therefore, underlay topology is essential in building high-bandwidth overlay tree. We hence address the following two important issues:

- *Efficient inference of underlay information using end-to-end overlay measurements*
  We consider that one overlay path measurement can obtain the route sequence and link metrics of interest (such as delay and/or bandwidth). We will use traceroute as our measurement tool (though other tools may be also used).
  We address the following problem: given N hosts in an overlay network, how to infer the router-level topology (in terms of connectivity) with low number of path measurements? Obviously, in the worst case, $N(N-1)/2$ measurements are needed to construct an accurate topology, assuming overlay paths are symmetric. We are interested in building a highly accurate topology (discovering, say, 95% of the link information) with substantially fewer measurements.
  We introduce a heuristic called *Max-Delta*, where hosts utilize distance estimation tools like GNP or Vivaldi to estimate their coordinates [4], [5]. A server then collects host coordinates and chooses the best set of host-pairs for measurements in each round. Our simulation results show that even though almost full $N(N-1)/2$ traceroutes are needed to discover completely underlay topology, Max-Delta can construct the underlay topology with high accuracy with substantially fewer measurements (by almost an order of magnitude).
- *Construction of high-bandwidth overlay tree given an (inferred) underlay topology*
  Given the inferred underlay topology and link bandwidth, we propose how to construct application-layer tree to meet a certain target bandwidth requirement. If this cannot be met due to limited link bandwidth, the tree should achieve as high tree bandwidth as possible.
  We propose Fast Application-layer Tree (FAT), where the server constructs an overlay tree by choosing paths with lighter "load", thus avoiding those bottleneck links. Our simulation results show that it achieves high tree bandwidth, low link stress and low relative delay penalty (RDP).

This work has been supported, in part, by the Areas of Excellence (AoE) Scheme on Information Technology of the University Grant Council (AoE/E-01/99), and by grants of the Research Grant Council (HKUST6199/02E & HKUST6156/03E) in Hong Kong.

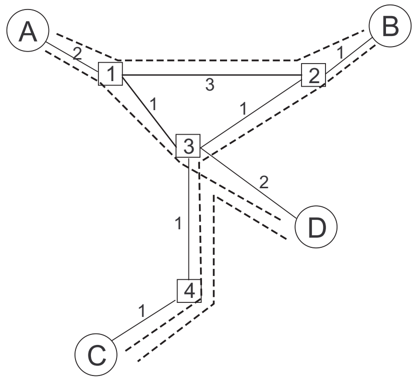


Fig. 1. An example of underlay network. The dashed lines indicate overlay paths among hosts. The labels along the links indicate normalized bandwidth.

The rest of the paper is organized as follows: In Section II, we discuss how to efficiently infer underlay topology by means of overlay measurements. In Section III, we describe FAT to build high-bandwidth overlay tree. In Section IV, we present illustrative simulation results. We conclude in Section V.

II. Efficient Topology discovery

Measurement tools such as traceroute, pathchar or Nettimer are often used to obtain underlay network information. Because these tools consume much time and network resources, hosts should probe the paths efficiently so that they reveal the most information on the underlay.

We consider that we can obtain the router-level path (in terms of link delay and router IDs) between any pair of hosts through some measurement tool such as traceroute. There are no anonymous routers or alias routers, and the path between a pair of hosts is unique and stable. This is reasonable because end-to-end Internet paths tend to be stable for significant length of time, such as a day [6]. For simplicity, we consider that the path is symmetric. Note that asymmetric paths do not affect our heuristics.

The problem is how to infer the router-level topology among a given group of N hosts with small number of measurements. As the router-level network is a sparse graph [7], we expect that exhaustive probing is not necessary. Figure 1 shows an example, where A, B, C, D are end-hosts and $1, 2, 3, 4$ are routers. The labels along the links are normalized bandwidth (explained later). The dashed lines indicate overlay paths among the hosts. As the figure shows, paths $A-B$, $B-C$, $C-D$ and $D-A$ have revealed all the underlay links, and therefore we need to measure only four instead of all the possible six paths.

If we do not need to know the underlay topology with full accuracy (for example, discovering the links with 95% accuracy), the number of measurements may be substantially reduced. The issue is then how to choose the most representative paths to probe.

Suppose a central server serves the host group in which a host can perform traceroute to any other hosts. Each path measurement reveals the router IDs and link delays (and possibly link bandwidth) along the path. The server utilizes host coordinates to select paths to probe in each round/iteration. These coordinates can be estimated by low-cost tools such as GNP or Vivaldi [4], [5]. In order to maximize the parallelism of measurements among hosts in each iteration (and hence reduce the time to infer the topology), the server assigns one probing target to each host. Hosts then probe their own targets and report their path measurements to the server. The server combines all these results and starts the next iteration. We study the following path selection heuristics at server:

- Random Probing (Random): In each iteration, a host randomly traceroutes an unmeasured host.
- Longest Path Probing (Longest): Usually a longer path in the Internet contains more hops (routers); therefore, among all the unmeasured paths, a longer one may contain more links/routers currently undiscovered. For each host in an iteration, the server therefore chooses the farthest (in coordinate space) unmeasured host as the host's target.
  The coordinates in GNP can be computed in $O(N)$ time. The pair-wise distance in coordinate space can then be computed in $O(N^2)$ time. For each host, we sort its adjacent paths in decreasing order, which takes $O(N \log N)$ time. Thus the computation complexity before path measurements is $O(N + N^2 + N * N \log N) = O(N^2 \log N)$. In each of the following iterations, selection of one probing target needs $O(1)$ computations and hence one iteration costs $O(N)$ computation.
  A concern of this scheme is that if a host is far from the other hosts in the group, that host would be susceptible to probe measurements from the other hosts. As a result, these traceroutes would repeatedly reveal the information towards the host instead of mutual information among all the hosts. This will lead to inefficiency in traceroute measurement.
- Max-Delta Probing (Max-Delta): For host a and b, denote the distance between them using their coordinates as $Euclidean(a, b)$, and the length of the shortest path between them on the partially constructed topology as $D_p(a, b)$. Let
  $$\Delta(a, b) = D_p(a, b) - Euclidean(a, b).$$
  In the first iteration, each host is assigned a probing target so that the overlay graph constructed from the measurements is connected. In the following iteration, for each host, the server then computes Δ for all unmeasured pairs and chooses the one with the maximal Δ as its probing target in the next iteration.
  The computation complexity is analyzed as follows. At the beginning of some iteration, suppose the partially underlay network contains V_p nodes (including hosts and routers) and E_p links. The computation of single-source shortest paths on a graph with V nodes and E edges takes $O(V \log V + E)$ time [8]; therefore, computing all D_p values is of complexity $O(N(V_p \log V_p + E_p))$. For

each host, it takes $O(N)$ time to choose the maximal Δ from at most N candidates. Therefore the total time complexity in this iteration is $O(N(V_p \log V_p + E_p + N))$.

Note that in an iteration, it is possible that a host is not assigned any target to probe. This is the case when all the paths from the host to others have been probed.

III. Fast Application-layer Tree (FAT)

Given the inferred network topology and link bandwidth, we present in this section how to construct an ALM tree which achieves a certain target bandwidth.

A. Tree Construction

Let M be the number of iterations the server performs in inference. There are at most MN overlay paths. Let BW_{req} (kbps) be the target bandwidth requirement of an application and BW_i (kbps) be the bandwidth of underlay link i. Further let W_i be the normalized bandwidth of link i, defined as $W_i = BW_i/BW_{req}$. Note that if link bandwidth is not known or available, we may set $W_i = 1, \forall i$. Denote C_i the number of overlay paths on link i. We define L_i, the "carried load" on the underlay link i, as

$$L_i = \begin{cases} \frac{C_i}{W_i}, & \text{if } C_i > W_i; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

To build a tree with high bandwidth, we should avoid the links likely to be congested, i.e., those with high L_i. Accordingly, we evaluate the "path load" of an overlay path p as

$$L(p) = \sum_{j \in p} L_j. \tag{2}$$

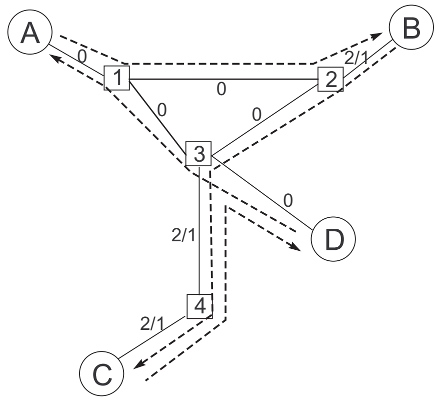
Thus, the server can compute $L(p)$ for all the probed overlay paths, and build a minimum spanning tree as the overlay tree.

We show a simple example in Fig. 2. Suppose the underlay network is the same as Fig. 1 ($N = 4$). Let $M = 1$, and the measurement paths are $A \to B$, $B \to C$, $C \to D$ and $D \to A$. These overlay paths are shown as dashed lines in Fig. 2(a), where the label along the underlay link is its "load" L_i defined in Equation (1). The server then computes the loads of overlay paths and constructs the overlay graph as shown in Fig. 2(b) with the labels indicating the load $L(p)$ according to Equation (2). The minimum-spanning tree $\{A-B, A-D, D-C\}$ is then used as the overlay tree. In this example, the resultant tree has normalized bandwidth 1 and thus is able to meet the target bandwidth requirement.

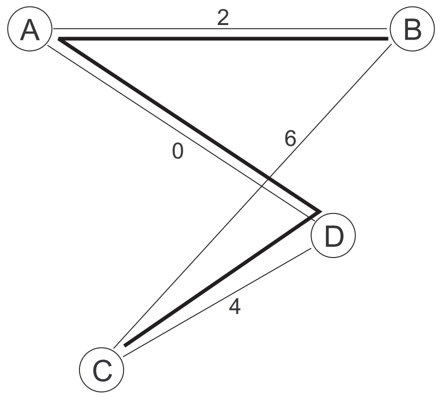
B. Algorithm Analysis

We analyze the algorithm complexity here. Assume there are x_p different underlay links in overlay path p. To compute parameter C_i for all underlay links, we need to go through all these overlay paths. The complexity is

$$O\left(\sum_{p=1}^{MN} x_p\right). \tag{3}$$



a) Step 1: The dashed lines are overlay paths among hosts. The labels along the underlay links are L_i.



b) Step 2: Construct the overlay graph. The labels along the overlay paths are $L(p)$. A MST is thus constructed.

Fig. 2. An example of FAT tree construction.

To compute load metric L_i as defined in (1), we need to go through all the distinct underlay links in probed overlay paths. This overhead is lower than (3), since we only need to go through the distinct underlay links.

The complexity of computing load $L(p)$ for all the overlay paths is the same as Equation (3).

Furthermore, to compute MST in a graph with n nodes and e edges, traditional Prim's algorithm with Fibonacci heaps can achieve $O(e + n \log n)$ complexity [8], i.e., $O(MN + N \log N)$ in FAT.

Therefore the total time complexity is

$$O\left(\sum_{p=1}^{MN} x_p\right) + O(MN + N \log N). \tag{4}$$

Some research has shown that the average number of routers in an overlay path in Internet is about 16. Thus we can regard x_p as a constant, and Equation (4) is then reduced to $O(MN + N \log N)$. This is the overall time complexity of FAT. If $M = O(\log N)$, it is reduced to $O(N \log N)$.

IV. Illustrative Simulation Results

In this section we evaluate our network inference methods and FAT on Internet-like topologies. We generate a number of (10) *Transit Stub* topologies with GT-ITM [9]. Each topology contains 3200 routers and about 20000 links. End-hosts are randomly put in the network. An end-host is connected to a stub router with 1ms delay, while the delay of core links is given by the topology generator.

A. Network Inference

We define the following metrics for network inference:

- Percentage of links (β), defined as the ratio of the number of links in the inferred topology to the total number of links in the actual underlay topology. Every link appearing in the inferred topology is an actual link on the underlay. Therefore, β is equal to 1.0 if and only if the inferred topology is completely accurate.
- Measurement load (R), defined as the number of overlay measurements performed by a host. The average measurement load among all the hosts in a complete inference procedure is bounded by $(N-1)/2$.

We evaluate the three measurement heuristics in Section II, denoted as "Random", "Longest" and "Max-Delta" in the following figures.

Figure 3 shows the average measurement load per host to achieve $\beta = 0.95$. The line $Y = (N-1)/2$ indicates the theoretical upper bound to discover the network with full accuracy. As the group size increases, R increases. *Max-Delta* achieves substantially fewer measurements (by almost an order of magnitude) as compared to the full measurements. Its average measurement load does not sensitively increase with the group size, which means that it is quite scalable to large group. *Longest Path Probing* does not perform well. We find that in *Long Path Probing* there are often a small subset of hosts being probed by many other hosts. These hosts therefore cannot help much to discover new links. These are also the hosts which cannot find any target to probe in later iterations. Therefore, all measurements are given to the remaining hosts, leading to inefficient inference.

Figure 4 shows the average measurement load per host for different β requirements, where the group size N is 256. To discover the accurate underlay, i.e., $\beta = 1.0$, these three heuristics have similar measurements requirements. However, if only a highly accurate approximate graph is desired, say, β around 0.95, *Max-Delta* achieves much lower measurement load.

As these figures show, it is not easy to discover accurate topology. None of the heuristics performs extremely better than others in this case. This is in accord with our basic knowledge of Internet, that it is quite difficult to decide which link should be in a path given a source and a destination, especially when they are in different AS. On the other hand, although it is difficult to obtain accurate topology, the network distance estimation can guide us to discover a highly accurate topology efficiently.

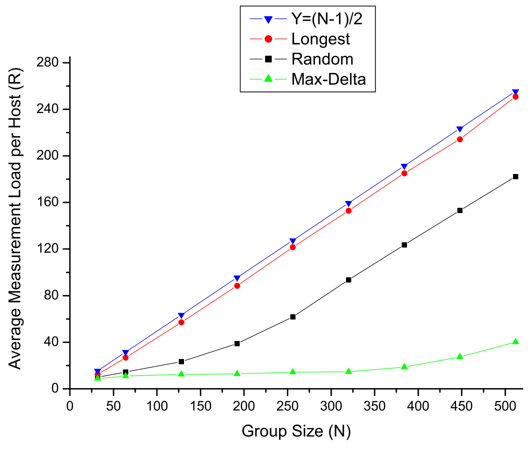


Fig. 3. Average measurement load on each host (R) to achieve $\beta = 0.95$.

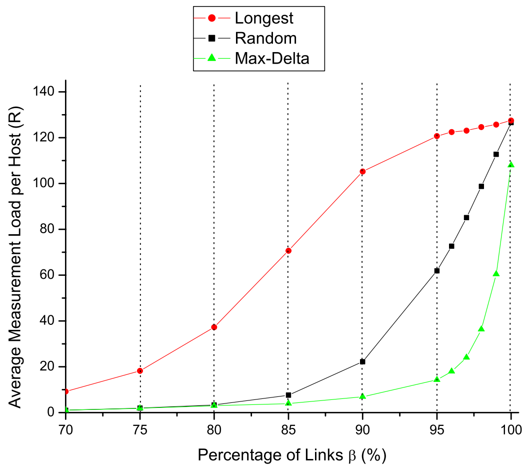


Fig. 4. Average measurement load per host for different β requirements ($N = 256$).

B. FAT Performance

We use tree bandwidth (defined in Section I), link stress and RDP (defined in [1]) to evaluate FAT. We also implement Narada and Overcast for comparison. Narada is one of the pioneering ALM protocols and its performance can serve as benchmarks. Overcast targets creating high bandwidth channels from one source to all receivers.

For FAT, it uses Max-Delta method to infer underlay topology. M is set to be $\lceil 2\log_2 N \rceil$. For Narada, each host's fanout range (the minimum and maximum number of neighbors each member strives to maintain in the mesh) is $3-6$. We use normalized bandwidth in parameter settings and result discussion. The target bandwidth is 1. We assume the normalized bandwidth is uniformly distributed between 3 and 5. In each simulation, all the schemes share the same underlay network

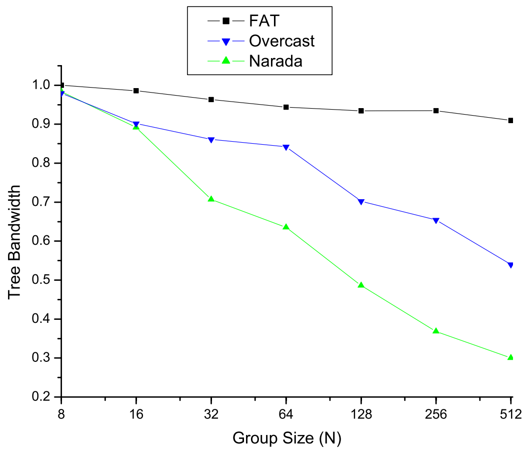


Fig. 5. Tree bandwidth vs. different group sizes.

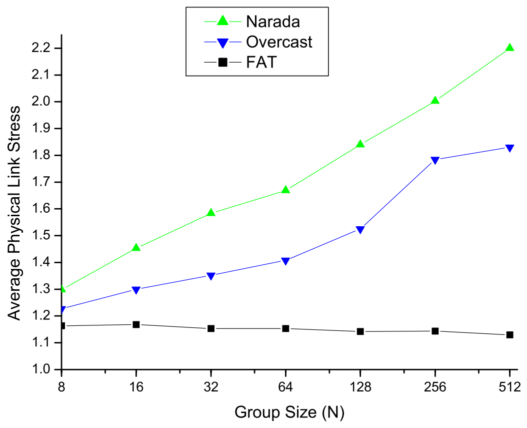


Fig. 6. Average link stress vs. different group sizes.

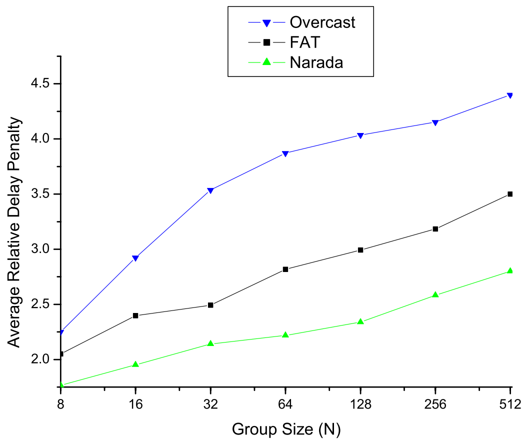


Fig. 7. Average RDP vs. different group sizes.

topology and bandwidth setting.

Fig. 5 compares tree bandwidth achieved by different schemes. As the group size increases, tree bandwidth decreases. FAT can achieve the maximum tree bandwidth among all these schemes. Even in large group size (> 100) case it can achieve near required bandwidth. While Overcast's performance is not well in large group size. This is due to its greedy tree construction method. Narada performs the worst for it does not optimize bandwidth in tree construction.

Fig. 6- 7 shows physical links stress and RDP of different schemes. FAT achieves much lower stress than Narada and Overcast. This indicates that the transmission load has been evenly distributed among underlay links. Its stress does not vary sensitively with group size. This confirms the tree bandwidth curve is Fig. 5.

The RDP of FAT is lower than Overcast. Because Overcast always tries to insert a new host as far from the source as possible. On the other hand, FAT strives to bypass heavy "load" paths and is inherent to circumvent long or many-hop paths.

V. Conclusion

In this paper we study how to build an ALM tree to meet a certain target bandwidth requirement. Since overlay paths may share bottleneck underlay links, it is difficult to solve this problem solely at overlay. Therefore, we propose to use network measurement tools to infer underlay network and utilize these information to build an ALM tree. We study how to efficiently infer underlay information. Furthermore, we propose FAT (Fast Application-layer Tree) scheme to build high-bandwidth overlay tree on top of the inferred underlay. We do simulations to evaluate our network inference heuristics and FAT. The results show that our inference method is effective with very low network measurements to get an accurate topology, and FAT can achieve higher bandwidth than Narada and Overcast.

References

[1] Y. H. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE JSAC*, vol. 20, pp. 1456–1471, Oct. 2002.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. ACM SIGCOMM'02*, pp. 205–220, Aug. 2002.

[3] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with Delaunay triangulation overlays," *IEEE JSAC*, vol. 20, pp. 1472–1488, Oct. 2002.

[4] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM'02*, pp. 170–179, June 2002.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. ACM SIGCOMM'04*, Aug. 2004.

[6] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of Internet path properties," in *Proc. ACM SIGCOMM IMW'01*, Nov. 2001.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *Proc. ACM SIGCOMM'99*, pp. 251–262, Sept. 1999.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

[9] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM'96*, vol. 2, pp. 594–602, Mar. 1996.