Inferring Characteristics of Multicast Trees

Xiaowei Yang (yxw@lcs.mit.edu)
Li-wei Lehman (lilehman@mit.edu)
Laboratory for Computer Science, MIT

December 17, 1998

1 Introduction

The heterogeneity of the Internet presents a challenging environment for multicast applications. Receivers from different parts of the multicast tree have different loss characteristics and bandwidth constraints. In order to cope with the heterogeneity problem, many have proposed to embed multicast servers within the multicast tree for local optimization. For example, LBRM [6] and RMTP [12] use statically configured repair servers within the network to perform local loss recovery. Appropriate placement of these repair servers can minimize the loss recovery overhead of local regions on the entire group. For another example, RMX[5] and Media Gateways[1] have proposed to spawn software agents within the network to perform rate adaptation through data transformation and real-time media transcoding in order to accommodate heterogeneous bandwidth constraints. However, dynamic and intelligent placement of these multicast servers and transcoding agents remain an open issue. Ideally, one should deploy these local optimization servers/agents dynamically by partitioning the multicast tree into highly correlated loss neighborhoods or subtrees sharing the same bottleneck links. Placing repair servers based on loss neighborhoods achieves efficient recovery scoping, since all receivers behind the same bottleneck link tend to require the same retransmissions. Placement of transcoding agents should be based on where bandwidth mismatches occur. In both cases, dynamic placement of these servers rely on a good understanding of the bottleneck locations and bandwidth constraints associated with each parts of the multicast tree.

In this paper, we present an algorithm to infer the characteristics of multicast tree by using pure end-to-end measurements. Our algorithm performs both topology and performance discovery. By analyzing the loss correlations between receiver's loss bitmaps, the multicast tree is partitioned into local clusters with similar loss characteristics and bandwidth constraints. Our goal is to first determine which receivers share the same bottleneck links and form local clusters with highly correlated losses. We then estimate the bandwidth capacity of the bottleneck link which connects each local cluster to the rest of the multicast group.

2 Overview

2.1 End-to-end measurments

Since end-to-end information is comparatively easier to obtain, end-to-end measurements have been widely adopted. If we view a network as a blackbox, a sending stream as the input signal, the corresponding receiving stream as the output signal, all internal cross traffics as noises, end-to-end measurements can be described as: given measured output and some particular input, how much we can tell about the structure or characteristics of the blackbox and noises. The input signal is described by packet sequence number and inter packet time (i.e. packet interdeparture time). So is the output. Information which can be deduced/derived from received sequence number space are packet loss statistics and packet reorderings. Further information can be inferred from direct measurements. For example, TCP infers a congestion signal from a packet loss. In [11], packet loss statistics are used to infer topologies of multicasting trees. Similarly, as bottleneck bandwidth and cross traffics both can shape packet interarrival time, from the interarrival time, we can infer bottleneck bandwidth and patterns of cross traffics [10], [2].

2.2 General Approach

From the above analysis, we perceive that end-to-end measurements are highly limited – we do not have much available information and most conclusions

are based on probabilistic guesses. In this paper, we studied the effectivenss of inferring multicasting tree purely based on end-to-end information. Inferring characteristics of multicasting tree consists of three steps.

- Step 1: Inferring bottleneck bandwidth seen by each receiver.
- Step 2: Inferring the logical tree topology.
- Step 3: Inferring bottleneck location.

Step 1 can be successfully achieved by Vern Paxson's PBM technique. Step 3 depends on results from Step 1 and Step 2. So we focus on Step 2 in our paper. By the property of IP multicasting, a loss occuring at one link will be seen by all receivers in the subtree rooted at that link. Thus, loss correlation reflects topological correlation. We therefore choose loss information to infer topologies. Next, we need to determine how to quantitatively measure loss correlation. If we view loss at each receiver as independent Bernouli variables, loss correlation can be measured by covariance between those variables [13]. If we view loss as a loss vector, loss correlation can be measured by vector correlation, i.e., $\frac{\vec{v_1}\cdot\vec{v_2}}{|\vec{v_1}||\vec{v_2}|}$. Those metrics are mathematically meaningful and cannot truely reflect the property of multicasting tree[1]. The metric we choose is the estimated loss rate on shared path between any pair of receivers[2].

2.3 Analysis of the Metric

As a start, we focus our study on binary trees. For any pair of receivers, the path from source to them can be decomposed as two parts: shared path and non-shared path (See Figure 1).

Let the loss probability on shared path by a pair of receivers 1 and 2 be L_s. Let L_{n1} and L_{n2} respectively be the loss probability on nonshared path for receiver 1 and 2. The loss probability seen by receiver 1 is L_1 and 2 is L_2. The loss probability seen by both is L_{12}. By multicasting property, loss on shared path is seen by both receivers and loss on nonshared path is independent. So we have,

$$L_1 = L_s + (1 - L_s)L_{n1} \quad (1)$$
$$L_2 = L_s + (1 - L_s)L_{n2} \quad (2)$$
$$L_{12} = L_s + (1 - L_s)L_{n1}L_{n2} \quad (3)$$

[1]Experimental results also show that either metric measures loss correlation good enough to infer topology information.

[2]In fact, this metric is the same as in [11].

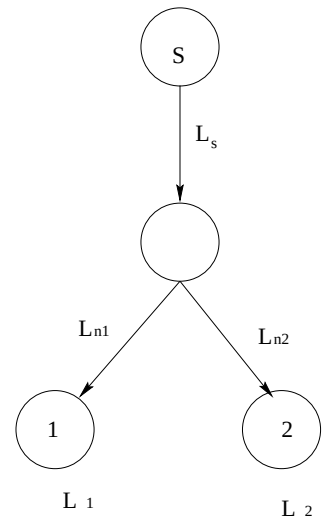

Figure 1: Shared Path Loss Rate

Since we have all loss statistics, we can estimate L_1, L_2 and L_{12}. Let n be the total number of packets sent. Then,

$$L_1 \doteq \frac{\text{\# of losses seen by 1}}{n} \quad (4)$$
$$L_2 \doteq \frac{\text{\# of losses seen by 2}}{n} \quad (5)$$
$$L_{12} \doteq \frac{\text{\# of losses seen by both 1 and 2}}{n} \quad (6)$$

As n increases, L_1, L_2 and L_{12} converge to the real loss probability. From Equations 1, 2 and 3, we can obtain L_s,

$$L_s = \frac{L_{12} - L_1L_2}{1 + L_{12} - L_1 - L_2} \quad (7)$$
$$L_{n1} = \frac{L_1 - L_{12}}{1 - L_2} \quad (8)$$
$$L_{n2} = \frac{L_2 - L_{12}}{1 - L_1} \quad (9)$$

We assume L_s also converges to its true value when n goes to infinity. Our algorithm is based on this assumption[3]. There are two simple facts about L_s.

1. In Figure 2, any pair of receivers in subtree A or B has higher L_s than any pair of receivers across subtree A and B, i.e., one receiver is in A, the other is in B, since the shared path between receiver pairs in subtree A or B includes the shared

[3]However, we do not know how fast it converges. When the number of receivers is increased, it might converge very slow. The number of packets to converge might increase very fast with the number of receivers.

path between receiver pairs across subtree A and B.

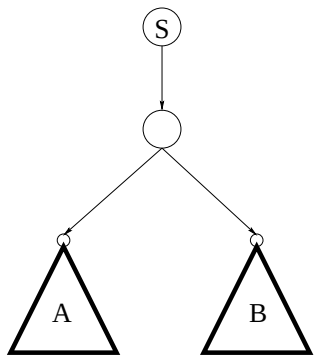

Figure 2: Top Down View

2. In Figure 3, the receiver 1 has highest L_s with receiver 2, since 1 shares the longest path with 2. So does 2.

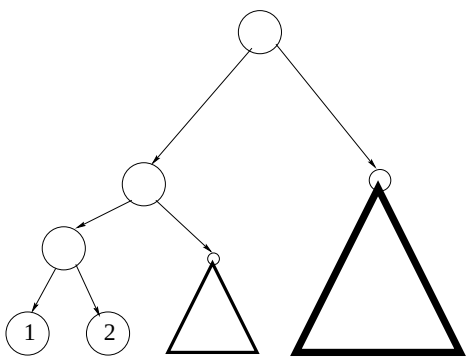

Figure 3: Bottom Up View

These two facts lead to two different inferring algorithm. We talk about them in next section.

3 Algorithms

Let $L_s(i,j)$ be the shared path loss probabililty between receivers i and j. We construct a weighted complete graph $G =< V, E >$ among all receivers. Each pair of receivers has an edge connecting them. The edge wight is computed by:

$$w(i,j) = -L_s(i,j) \tag{10}$$

We run the standard Minimum Spanning Tree(MST) algorithm on G and construct a MST.

Starting from the MST, we develop two inferring algorithms.

1. Top Down Clustering algorithm. Based on the top down view in Figure 2, the MST tree edge connecting receivers between subtree A and B must have the higher wight (means lower L_s) than any edge connecting receivers within subtree A or B. Since any edge connecting receivers from subtree A to B or vice versa has higher weight, so does the MST edge. Suppose subtree A and B splits at the top level in the multicasting tree, i.e., they split at the source. Deleting this edge will split the MST into two sub MSTs. One contains all receivers in A and the other contains all receivers in B. Thus, top-level tree topology is successfully discovered. Repeating the same procedure respectively on the two sub MSTs will reconstruct the logical topology of the original multicasting tree. The algorithm can be summarized as:
   (a) Procedure Name: InferTree(MST)
   (b) On input MST, delete the maximum weight edge and split MST into MST1 and MST2.
   (c) Create a node P as the parent.
   (d) Set the left child of P to be InferTree(MST1); the right child to be InferTree(MST2).
   (e) Return P
2. Bottom Up Clustering Algorithm[4]. Based on Fact 2 in Figure 3, the minimum weight edge (largest L_s) in the MST connects two closest receivers. We make a subtree from the two nodes and replace them with their parent in the MST by combining the loss of the two nodes into their parent and recomputing edge weight connecting the parent to the rest of MST. Recursively doing so until we reach the root of the inferred logical tree.

Theoretically, if the estimated $L_s(i,j)$ converges to the true loss probability very well, both algorithms will give good inferring results. However, as we show in Section 4, Bottom-Up algorithm performs more robust than the Top-Down algorithm.

4 Testing Results

4.1 Topology Inference

We are currently implementing our scheme using the simulation environment developed by Ratnasamy

[4]This agorithm has the same idea as in [11]. Our MST implementation saves computation time.

in [11]. The tree generator and tree comparator were borrowed directly from Ratnasamy. We implemented the inferring algorithm in C.

Our main goal is to evaluate how fast top-down and bottom-up approaches converge to the true topology. We increase the number of sample size (i.e., the number of packets sent) by power of two each time. For each sample size, we test our algorithm for 100 different random topologies. We also test the bottom-up approach using the same 100 topologies. The probability of correct inference for each sample size is then the total number of correctly inferred results divided by 100.

Each binary tree is generated randomly based on a specified max tree level/height and max number of vertices. Only leaves of the binary tree can be receivers. Each link is given a loss rate, randomly picked from a uniform distribution of a specified range.

The first test result is shown in Figure 4.

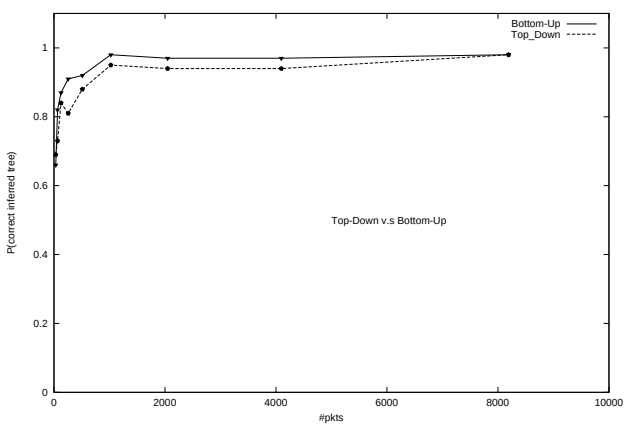

Figure 4: # of Avg Nodes = 10.9; # of Avg Recvrs = 4.3; Loss rate for each link is in [0, 10%]

Random trees were generated with a max depth/height of 10 and a max vertices of 50. By limiting the max height of the tree to 10, the random tree generator yields an average number of receivers of 4.3. As the number of transmitted packets increased, the probability of correct inference approaches 1. Both top-down and bottom-up algorithms converge very well.

We then increased the possible max depth of the tree to 50. The result is shown in Figure 5.

We note that both bottom-up and top-down approaches converge more slowly to the correct result when the number of receivers are large. However, the top-down approach performs particularly poorly when the number of receivers increases.

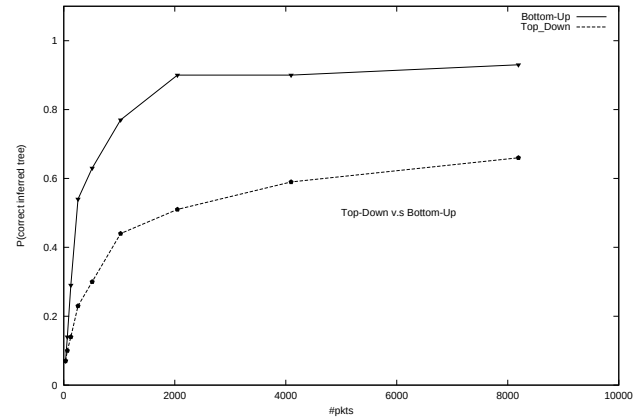

Figure 5: # of Avg Nodes = 46.3; # of Avg Recvrs = 16.2; Loss rate for each link is in [0, 5%]

To explain this result, we note that there are two implications to increasing the max tree height from 10 to 50. First, the number of receivers in a tree increases. Second, as the max allowable level increases, receivers tend to see a much wider range of different loss rates. If the link loss rates are randomly picked from the range of $[p1, p2]$, then the min possible loss rate observed by a receiver is $\sum_{i=0}^{k}(1-p1)^i p1$ and max possible loss rate of a receiver at level k is $\sum_{i=0}^{k}(1-p2)^i p2$ (Recursively derived from Equation 1). In short, as we increase the height of the tree, there will be more receivers and receivers in general will see higher loss rates.

We would like to answer two questions. First, why do both approaches perform worse when more receivers have higher loss rates? Second, why do top-down perform so much worse than bottom-up in this situation?

In order to answer the first question, we first note that even when node 1 and 2 are totally independent, the estimated L_s can be deceivingly high with small sample space, even though it will eventually converge to zero. In other words, th bad performance is due to the fact that the estimated L_s is not converging to the true shared path loss probability very well.

To answer the second question, we note that the bottom up algorithm works much better because when we coalesce two nodes into one, we also combine their loss into the new node. Suppose the new node is k, the combined two nodes are i and j. We then recompute $L_s(k, l)$ (l: the left receiver). For binary trees, the shared path between (k, l) is the same as that of (i, l) and (j, l). So $L_s(k, l) = L_s(i, l) = L_s(j, l)$. However, the estimated $L_s(k, l)$ usually converges better

than $L_s(i,l)$ or $L_s(j,l)$ since it has the information of estimated loss probability seen by the internal node k, which is $L_k = L_{ij}$.

We examined several cases while top down gave wrong inference. We found out tight correlated groups, i.e. groups sharing longer path have higher correct inferring probability. Errors happen very often when a tree is very unbalanced. One branch is short and the other is long. Receivers on the short branch are correlated loosely since they see low shared path loss probability. When there are some receivers at the end of the long brach have high losses, the calculated L_s across the two branches might turn out to be higher than some calculated L_s seen by receivers in the short branch. In this case, the top-down algorithm will make a wrong cut by clustering the receiver in the short branch with the one with high loss rate in the long branch.

Intuitively, when the shared path is comparatively short to the diverged paths, the estimation is harder to converge to the true shared path loss probability than the opposite situation i.e. the shared path is comparatively long to the diverged paths(See Figure 6. Also we assume longer path implies higher loss rate). So the estimated $\max(L_s)$ is more reliable than the estimated $\min(L_s)$. The bottom-up approach chooses to cluster receivers on $\max(L_s)$ while the top-down algorithm choose to split receivers on $\min(L_s)$. So top-down converges much worse than bottom-up with the increase of the tree depth. (For details, see Section 5)

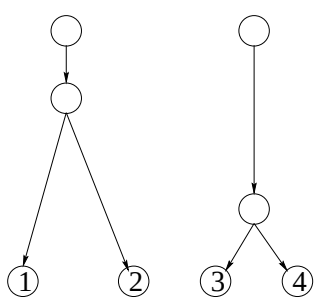

Figure 6: $L_s(1,2)$ is harder to estimate than $L_s(3,4)$

However, the top-down approach has the advantage of controlling granularity. Also, the top-down approach achieves top level information immediately. In the case we are interested in locating the closest bottleneck towards the source quickly, the top-down approach is more efficient. To improve the efficacy of top-down approach, we can either increase the number of transmitted packets, or loosen the comparison criterion. If most of the subtree structure is correct, the inferred result is suffice for most cases. Figure 7 gives the result when we define a correct inferring as 90% of subtrees of the inferred tree are correct. Under this criterion, the top-down approach also converges very well.

Another possible improvement is to construct the right correlation tree. In this paper, we chose MST because for any cut of the graph, an MST edge connecting the cut is the edge which has the minimum weight (i.e., the maximum shared path loss probability) among all edges connecting the cut. We use this estimated loss probability as the shared path loss probability of the cut. Actually, there are also other pairs of receivers who share the same path. Their estimation of the shared path loss probability is also useful. If we could take the average of their estimation as the estimated shared path loss probability, the estimation is expected to converge better. As the correctness of the algorithm totally depends on the convergence of this probability, we expect better inferring results then.

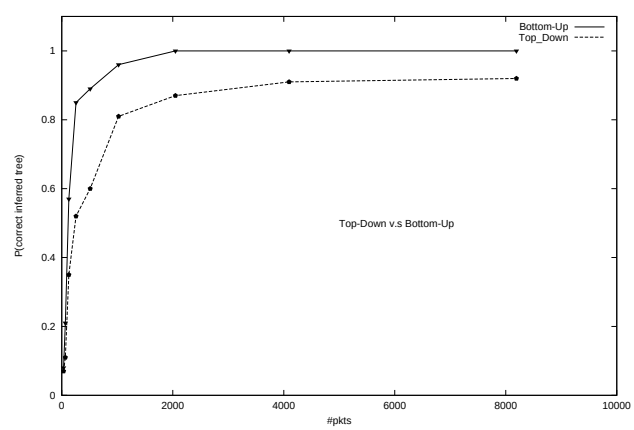

Figure 7: Results of 90% Accuracy. Same testing parameters as in Figure 5

4.2 Estimating Bottleneck Bandwidth and Location

In the previous section, we focus on how loss patterns can help infer the topology of the multicast tree. In this section, we describe how receivers can estimate the bottleneck bandwidth by having the source multicasts probe packets to the group, and how we can combine the clustering algorithm and the bandwidth estimation to infer the bottleneck location.

Bolot, Keshav, and Paxson already noted that by sending two probe packets back to back, receivers