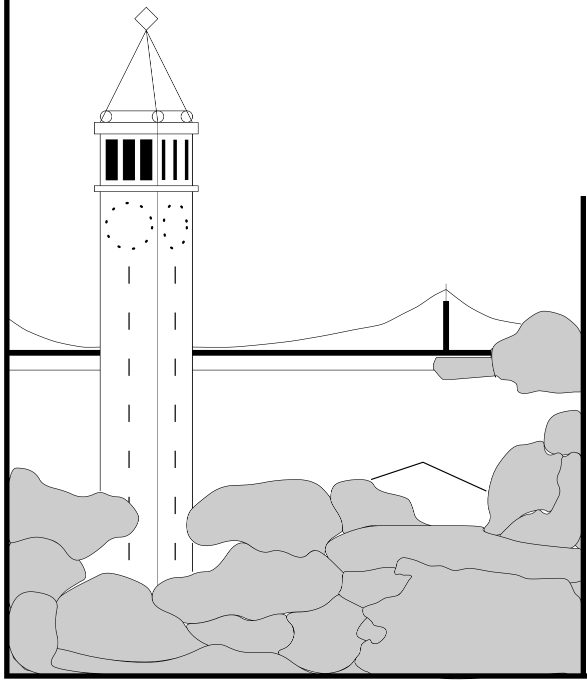
Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements

Sylvia Ratnasamy and Steven McCanne



Report No. UCB/CSD-98-1019

October 1998

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Work supported by DARPA Grant
66001-96-C-8508 and the
California State MICRO Program.

Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements

Sylvia Ratnasamy and Steven McCanne

October 1998

Abstract

The efficacy of end-to-end multicast transport protocols depends critically upon their ability to scale efficiently to a large number of receivers. Several research multicast protocols attempt to achieve this high scalability by identifying sets of co-located receivers in order to enhance loss recovery, congestion control and so forth. A number of these schemes could be enhanced and simplified by some level of explicit knowledge of the topology of the multicast distribution tree, the value of the bottleneck bandwidth along the path between the source and each individual receiver and the approximate location of the bottlenecks in the tree. In this paper, we explore the problem of inferring the internal structure of a multicast distribution tree using only observations made at the end hosts. By noting correlations of loss patterns across the receiver set and by measuring how the network perturbs the fine-grained timing structure of the packets sent from the source, we can determine both the underlying multicast tree structure as well as the bottleneck bandwidths. Our simulations show that the algorithm is robust and appears to converge to the correct tree with high probability.

1 Introduction and Motivation

The IP Multicast service provides for efficient one-to-many packet transmission. A single packet transmitted by the source is delivered to an arbitrary number of receivers by replicating the packet within the network at fan-out points along a distribution tree rooted at the traffic's source [11]. The IP Multicast service model provides a best-effort service, yet a number of emerging applications, such as shared white-boards, software updates, news articles etc require reliable packet delivery. To meet this requirement, reliable multi-cast protocols such as SRM [4], RMTP [9], and TMTP [20] build reliability on top of this unreliable service.

A key challenge in the design of a reliable multicast protocol is its loss recovery algorithm, which has proven difficult to scale to a large number of receivers. For example, the global loss recovery component of SRM multicasts retransmission requests and replies to the entire group and thus scales poorly [16] as the entire tree participates in the recovery process and even a single lossy receiver can significantly degrade the overall session performance. To solve this problem a number of schemes have been proposed that try to restrict error recovery traffic to the required scope, i.e., these schemes attempt to achieve *local recovery*. The key idea behind local recovery is to identify loss neighborhoods of receivers that share similar loss patterns and confine error recovery to this neighborhood without disturbing the rest of the tree. Schemes based on this approach include:

- the use of hop-scoping to control the distance travelled by retransmission requests and replies [10];
- the use of separate local multicast groups for error recovery [10];
- replier-based schemes, based on a new set of router forwarding services such as directed multicast and subcast forwarding [13]; and,
- the use of a new "randomcast" forwarding service to form "search parties" of loss affected members searching for lost data [3].

All these schemes essentially require that the receiver discover the loss recovery group to which it belongs and search for potential candidates to retransmit lost packets.

The Reliable Multicast Transport Protocol [9] attempts to solve the loss recovery problem by organizing members into a hierarchy. Acknowledgments are sent not to the source, but to the parent member in the tree. Internal nodes in the hierarchy called Designated Receivers (DRs) cache data packets for later retransmission of lost packets. RMTP therefore provides both implosion avoidance and local recovery. However for RMTP to perform well, the hierarchy of members must be very closely correlated to the underlying multicast distribution tree and DRs need to be optimally and dynamically distributed over the tree. How this can be achieved is still an open research problem.

The recent work on Self-Organized Transcoding (SOT) [7] tries to adapt continuous-media applications to varying network conditions through the use of self organized transcoding. In SOT, when a group of co-located receivers detects loss caused by a congested link, an upstream receiver with better reception at the far end of the bottleneck acts as a transcoder and provides a customized version of the stream. A new stream is multicast to a new address and receivers adversely affected by the bottleneck switch to the new group. Receivers use the observed loss patterns to decide when to switch groups. Since it is crucial to the stability of the protocol that all receivers within the same loss subtree switch groups together, decision errors regarding joining and leaving groups need to be minimized. The problem of knowing when to join and leave groups and knowing which group to join is equivalent to the problem of knowing which loss neighborhood a receiver belongs to. The problem of optimally placing a transcoder or a designated receiver is essentially the problem of determining which receiver would be an ideal candidate for retransmitting lost packets.

In each of the schemes outlined above (i.e., local recovery, RMTP, and SOT), the receiver's protocol could be enhanced and potentially simplified with explicit knowledge of the underlying multicast distribution tree. Unfortunately, the IP service model deliberately hides this information in favor of a universal packet service that is easily ported across diverse technologies and environments. To overcome this, protocols like TCP adapt to physical path characteristics through end-to-end adaptation (e.g., searching for the bottleneck bandwidth with *slow start* and adapting to changes in available capacity with its *congestion avoidance* mode). But unlike unicast TCP, multicast communication creates many paths between a source and its receivers with potentially heterogeneous characteristics. Consequently, researchers have devised schemes like local recovery and SOT to discover the homogenous sub-regions of a heterogeneous multicast distribution tree, and exploit this knowledge in the adaptation processes.

In this paper, we propose a scheme for deriving a fairly accurate picture of the topology of a multicast distribution tree strictly from end-to-end observations. Our approach relies upon complete information of loss statistics at every receiver and thus is not a practical protocol building block in its own right. However, we believe the process of exploring this extreme point sheds light on the difficulty of the problem and forms the foundation for follow-on work that could exploit variations in the basic approach to trade off computational overhead for topological accuracy. Even so, our scheme could be of potential use in its current form for network monitoring, debugging, and performance characterization using off-line processing.

Our approach to this topology discovery problem consists of two core pieces: a tree inference algorithm and bottleneck bandwidth estimator. The tree inference engine clusters nodes according to shared loss and estimates the tree according to a probabilistic model that eliminates "false sharing". The estimate converges to the true tree as more loss statistics are collected. We combine this topological information with a bottleneck bandwidth estimation technique in order to approximate the location of the bottlenecks in the tree. The result is a model that faithfully captures the link capacities and multicast topology of the underlying physical tree even though our algorithms require information that is easily available at the end hosts and work with the existing multicast routing service.

In the next section, we describe the bottleneck bandwidth estimation technique. Section 3 describes the tree inference algorithm. In Section 4,

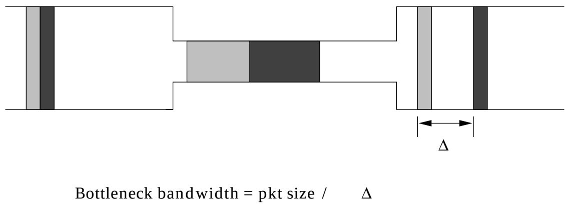


Figure 1: *Packet pairs flowing through a bottleneck link. The vertical dimension is bandwidth, horizontal dimension is time*

the two algorithms are combined into a comprehensive algorithm that approximates link capacities from the bottleneck measurements. Implementation details and preliminary test results are in Section 5. Finally, we describe related work on bottleneck bandwidth estimation and path inference techniques, and conclude.

2 Bottleneck Bandwidth Estimation

Transmission of a packet from a source to a receiver involves forwarding the packet along a series of consecutive links. Each link has a maximum rate at which it can forward packets. The maximum rate of the slowest link along the chain determines the maximum rate at which data can be transmitted between the source and receiver. In other words, the slowest link sets the bottleneck bandwidth along a given path. The ability to measure this bottleneck bandwidth value stems from the observation that as a packet is transmitted along a link, it is "spaced" out in time depending on the transmission rate of the link with the amount of spacing being inversely proportional to the capacity of the link [5]. The basic idea behind the packet-pair mechanism is as follows: if two probe packets travel together such that they are adjacent at the bottleneck link, with no packets intervening between them then, on emerging from the bottleneck link the inter-packet spacing will be proportional to the transmission time of the first packet over the bottleneck. This can be seen in Figure 1.

Let Q_b seconds be the time required to forward a packet of length P bytes through the bottleneck link. If the bottleneck bandwidth is $B(bytes/s)$ then $Q_b = P/B$. Q_b can be approximated at the receiver's end. The problem that then arises is that queuing elements beyond the bottleneck can distort the spacing between the probe packets. Either the first or the second packet can be randomly delayed thus randomly increasing or decreasing the calculated estimate of the bottleneck bandwidth. These random variations can be viewed as noise affecting the consistent inter-packet spacing caused by the bottleneck. Filtering mechanisms are thus needed to extract the desired measurements.

2.1 Filtering algorithm for robust bottleneck bandwidth estimation

In [15], Paxson develops a robust algorithm called Packet Bunch Mode (PBM) that estimates the bottleneck bandwidth along a unicast path. Our filtering algorithm is adopted from Paxson's work on PBM. In this section, we briefly review our filtering techniques. A more in-depth description of the details of PBM and the selection of the appropriate values for the required parameters can be found in [15].

Probe packets transmitted by the sender include a sequence number, and a time-stamp indicating the transmission time. The packet's arrival time is noted at the receiving end. Inter-packet spacing measurements are made by recording the difference in arrival times ΔT_r between consecutive packets. The difference in transmission times, ΔT_s, is calculated from the packet time-stamps. The criteria used to select valid sample measurements are:

- We define an expansion factor ξ which measures the factor by which the packets were spread out by the network as:

  $$\xi = \Delta T_r / \Delta T_s$$

  If $\xi < 1.0$, then the packets were not spread out by the network and hence not shaped by the bottleneck. Thus, calculations based on their arrival times should not be used in estimating the bottleneck and are not accepted as valid samples.

- If the last packet pair we inspected yielded a valid sample and spanned an interval of $\Delta T'_r$ then we perform a heuristic test: If $\Delta T_r / \Delta T'_r > 2$ then the current pair was spaced out more than twice as much as the previous pair and we skip the current pair as it is likely to reflect sporadic arrivals.

- Pairs that include out of order arrivals or lost packets are rejected.

Samples meeting the above criteria are used to calculate a set of bottleneck estimates. Let N_b be the number of estimates obtained. If N is the total number of packets sent, then $N/2$ is the maximum number of possible estimates using packet pairs. If N_b is less than 70% of $N/2$ then we reject further analysis of the set of estimates as it consists of too few estimates. Otherwise, we turn to the problem of extracting the best estimate from the set. The set of estimates is first sorted in decreasing order of the frequency of their occurrence. Let X be the estimate that occurs with maximum frequency. We then search the set for values that fall within ± 5% of X and combine them as a single entry with value X and frequency equal to the sum total of the individual frequencies. The set of estimates is thus narrowed down to a set of disjoint ranges. The estimate that occurs with the maximum frequency is then selected as the bottleneck bandwidth provided it occurs with a frequency that exceeds all other estimates by at least 60%. If not, the results obtained from the set of samples is ambiguous and no estimate of the bottleneck bandwidth is made.

The above bottleneck estimation algorithm can be extended to step through an increasing series of packet bunch sizes as outlined in [15] in order to detect multi-channel bottlenecks and changing bottleneck bandwidths.

2.2 Estimation of the bottleneck bandwidth in a multicast tree

To apply the techniques described in Section 2.1 , the traffic source in the multicast tree transmits a stream of back-to-back probe pairs. Each receiver measures the arrival times of packets at its end and uses the filtering algorithm outlined in Section 2.1 to infer the bottleneck bandwidth of the path between itself and the source. Note that this method does not, by itself, in any way indicate where along the path the bottleneck is located.

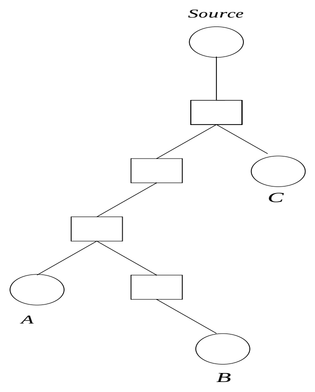


Figure 2: *Original tree topology*

3 Tree Inference Algorithm

This section describes our tree inference algorithm which reconstructs a logical representation of the multicast distribution tree using information obtained from the losses seen by the receivers.

Multicast packets flow along a distribution tree rooted at the source. The receivers form the leaves of the tree, the routers are the internal nodes in the tree and the links form the edges of the tree. A packet that is dropped along any link of the distribution tree, is lost by all the downstream receivers in the subtree rooted at the link. The tree structured delivery model thus introduces correlations in the packet losses seen by the different receivers. This loss correlation between receivers can be exploited to infer the topology of the tree that caused the observed loss patterns.

Our algorithm reconstructs a 'logical' representation of the multicast tree. A logical representation of a multicast tree is one in which each interior node is merely the closest common ancestor of all downstream receivers in the tree [18]. In reality each branch of the logical tree could consist of a series of links. In order to learn the exact topology of the tree we would have to enlist the help of each intermediate router along the path as is done in the traceroute and mtrace tools. Our algo-

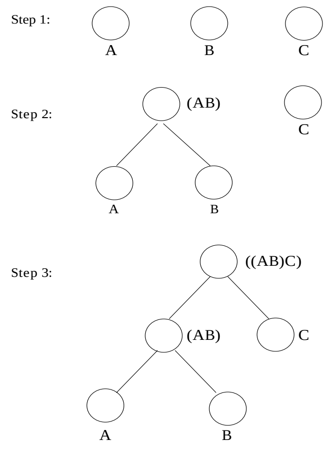


Figure 3: *Inference of the logical tree*

rithm is based only on end-to-end measurements using only information that is readily available at the end hosts and requires no special router support, as such, reconstructing a logical tree is as accurate as we can get. Knowledge of the logical tree is however sufficient for our purpose because all the receivers downstream of a given logical branch will see the same path characteristics such as the bottleneck bandwidth and loss rate irrespective of which component link of the logical branch caused the observed characteristics.

The tree inference algorithm described in the following sections attempts to reconstruct this logical tree in a bottom-up fashion using information regarding the loss patterns of the different receivers. Receivers having similar loss patterns are aggregated together and represented by a single node one level higher in the tree.The aggregated nodes can then be regarded as a single node for further aggregation. The entire tree has been reconstructed when all the receivers have been coalesced in this manner into a single tree. For example: In order to rebuild the tree shown in figure 2, the algorithm initially begins with a set of individual receivers A, B and C. Information obtained from the loss patterns of the three receivers indicates that A and B are more closely located than A and C or B and C. We thus aggregate A and B into a single "macro-node" (AB). Next, (AB) and C are aggregated to yield the logical tree $((AB)C)$.

Application of the aggregation techniques outlined above requires knowledge of two things: first, we need a selection criteria that is indicative of how closely located receivers are in the tree and second we need to know how many receivers are to be aggregated together into a single representative macro-node at each step of the tree building process. We first develop the principles behind identifying a pair of receivers to be coalesced at each step of the selection process, thus yielding a binary tree and then generalize the principles to reconstruct trees with arbitrary fan-out at each interior node.

3.1 Selection Criteria

We associate with each receiver X, a lossprint L_x which is an ordered listing of the sequence numbers of packets lost by the receiver X. In a tree any two receivers A and B see losses as described by their lossprints L_a and L_b respectively. These lossprints could potentially have a certain number of losses in common. We call these common losses the shared losses between receivers A and B.

3.1.1 Selection criteria: Shared losses

At a first glance, the shared losses between a pair of receivers appears to be an ideal indicator of how closely located the receivers are in the underlying tree. Net shared losses can however be misleading. For example, in figure 4 consider the case where the link $R_2 - A$ has a high loss rate. A could then have a high number of losses in common with every other receiver. In particular, if the link R_1-R_2 has a low loss rate then, it is possible that the shared losses between A and C exceed those between A and B which could result in the wrong nodes being coalesced. The flaw in the use of net shared losses as selection criteria is easily understood if we look at the ways in which shared losses occur.

Shared losses arise in two ways. A pair of receivers A and B share the path from the root to their closest common ancestor. Any packets lost along this shared path will appear in the lossprints of both A and B. These losses are caused by the tree structure and are truly indicative of the underlying tree structure. We call these true shared losses. In addition to these true shared losses, each receiver's

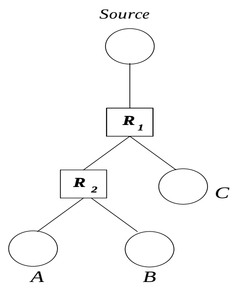


Figure 4: *Selection Criteria*

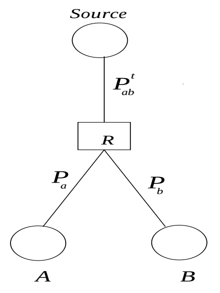


Figure 5: *Loss model*

lossprint will also include the packets that are lost along the separate paths from the closest common ancestor to each receiver. It is possible that two copies of the same packet are lost independently along these distinct paths on account of which a portion of the shared losses between A and B are not caused by the shared path between A and B. These shared losses are random and are not caused by the underlying tree structure. We call these false shared losses. The failure modes that arise in the use of the net shared losses as selection criteria are due to this "false sharing".

3.1.2 Selection criteria: True shared losses

The greater the extent of the shared path between a pair of receivers, the greater is the probability of their seeing true shared losses. The probability of seeing true shared losses between a pair of receivers is thus a good measure of how closely located receivers are in the tree and we use this probability as the selection criteria in order to identify the pair of receivers to be coalesced.

At the end host, there is nothing that distinguishes a true shared packet loss from a false one. The receiver merely sees the net shared loss. In order to allow the receiver to estimate the approximate number of true shared losses from the total shared losses, we apply the following loss model:

- A and B are arbitrary receivers from the set of all receivers with lossprints L_a and L_b respectively. Let n be the total number of packets transmitted at the source.

- A and B share a certain extent of the path from the source. Let P^t_{ab} be the probability of seeing losses along this path. i.e. P^t_{ab} is the probability of seeing true shared losses between A and B.

- Any losses seen by receiver A but not by B occur along the path from the closest common ancestor of A and B and the receiver A. Let P_a be the loss probability along this path. Similarly let P_b be the loss probability along the path from the closest common ancestor of A and B to B.

The above model is represented by figure 5. Using the above model, we can derive the following equations:

- Let the probability of seeing a shared loss (whether true or false) between A and B be P_{ab}. Then,

$$P_{ab} = P^t_{ab} + (1 - P^t_{ab})P_a P_b \qquad (1)$$

  where the first term on the right hand side is the probability of seeing true shared losses. The second term is the probability of seeing false shared losses.

- Let the probability of seeing a loss at A but not at B be $P_{a\bar{b}}$. Then,

$$P_{a\bar{b}} = (1 - P^t_{ab})P_a(1 - P_b) \qquad (2)$$

  Similarly, if the probability of seeing a loss at B but not at A is $P_{b\bar{a}}$, then,

$$P_{b\bar{a}} = (1 - P^t_{ab})(1 - P_a)P_b \qquad (3)$$

- Solving equations (1) to (3) yields the following solutions:

$$
\begin{aligned}
P^t_{ab} &= \frac{P_{ab}P_{b\bar{a}} + P_{b\bar{a}}P_{a\bar{b}} + P_{a\bar{b}}P_{ab} + P^2_{ab} - P_{ab}}{P_{ab} + P_{b\bar{a}} + P_{a\bar{b}} - 1} \\
P_a &= \frac{P_{a\bar{b}}}{1 - (P_{b\bar{a}} + P_{ab})} \\
P_b &= \frac{P_{b\bar{a}}}{1 - (P_{a\bar{b}} + P_{ab})}
\end{aligned}
$$

- Let the number of measured shared losses between A and B be $|L_{ab}|$ where $L_{ab} = L_a \cap L_b$. We approximate P_{ab} as $|L_{ab}|/n$.

- Similarly, if the number of measured losses seen by A but not by B is $|L_{a\bar{b}}|$, we approximate $P_{a\bar{b}}$ as $|L_{a\bar{b}}|/n$. We approximate $P_{b\bar{a}}$ as $|L_{b\bar{a}}|/n$. As n increases, these approximations should converge to the true value of the defined probabilities.

3.2 Binary Trees

A binary tree is one in which every interior node in the tree has at most two children. As we coalesce a pair of receivers together at every step our algorithm reconstructs a logical binary tree in which every interior node has exactly two children.

Using the selection criteria defined in Section 3.1, the tree inference algorithm works as follows:
Input: A set of receivers $S = \{1, 2, ..., N\}$ with lossprints $L_1, L_2, ..., L_n$.

1. Compute the probability of seeing true shared losses between all pairs of receivers from the set S.

2. The pair of receivers, A and B, with the maximum probability of seeing true shared losses are combined together into a single macro-node (AB). Set $L_{(ab)} = L_a \cap L_b$ and replace A and B by (AB) in S.

3. Repeat the above steps until all the receivers in S have been fused into the tree.

Our tree inference algorithm employs a greedy strategy of making the most likely merger at every step. Our results indicate that such a strategy works well in practice. Future work could look into algorithms that consider correlations across multiple nodes. In recent work, [19] compare the performance of top-down and bottom-up clustering algorithms for the reconstruction of the logical tree topology. Their results indicate that a bottom-up approach yields better results.

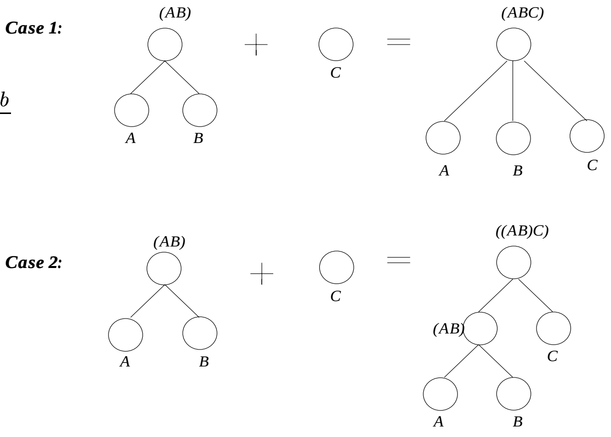


Figure 6: *Possible relationships between a pair of nodes to be coalesced for arbitrary trees*

3.3 Arbitrary tree topologies

In the binary trees reconstructed in the previous section, each interior node has a fan-out of exactly 2. As such, the pair of nodes yielded by the selection criteria are always aggregated as sibling nodes and represented by their parent node for further aggregation. In an arbitrary tree topology, interior nodes have a fan-out of two or more. The selected pair of nodes can thus be aggregated either as sibling nodes as in the case of binary trees, or one of the selected nodes could be the parent node of the other. The two alternatives can be seen in the aggregation of node C and macro-node (AB) in figure 6.

The ability to distinguish between these two cases stems from the observation that in case 1 the probability of seeing true shared losses between A and B should, under ideal circumstances, equal the probability of seeing true shared losses between macro-node (AB) and node C i.e $P^t_{(ab)c} = P^t_{ab}$. In case 2 the probability P^t_{ab} will be greater than $P^t_{(ab)c}$ because A and B share an additional link not shared by C. This added link adds to the true shared losses between A and B on account of which $P^t_{ab} > P^t_{(ab)c}$. We could thus distinguish between the two subtrees by making the following check :

If $P^t_{(ab)c} = P^t_{ab}$ then the nodes are coalesced as in case 1 else the nodes are coalesced as in case 2.

In reality, since we use the measured losses in order to approximate the probabilities P_{ab}, $P_{a\bar{b}}$ and $P_{b\bar{a}}$, the equality criteria for case 1 are too rigid. Strict adherence to the above rules would result in incorrect aggregations. In order to accommodate a certain amount of variation, we would like to identify situations in which $P^t_{(ab)c}$ "almost" equals P^t_{ab}. We thus define an error margin α and modify the above rules to :

If $P^t_{(ab)c}$ is within $\alpha\%$ of P^t_{ab} then the nodes are coalesced as in case 1 else the nodes are coalesced as in case 2.

This decision rule could result in incorrect aggregations being made for subtrees as in case 2 if the additional true shared losses between A and B are responsible for less than $\alpha\%$ of the probability of seeing true shared losses between A and B. As α will typically be low, such errors will only occur if the loss rate along a shared link is very low. As the purpose of these aggregations is to identify nodes that can be grouped together for the purpose of local loss recovery etc, such aggregations although not exact are actually acceptable because the low loss rate link is not the bottleneck causing loss, the problem links, if any, are further upstream and shared by receiver C i.e. for the purpose of local recovery A,B and C should be aggregated together and treated as belonging to the same loss recovery group.

4 Locating the bottlenecks in a multicast tree

Combining the information obtained by the bottleneck estimation algorithm (Section 2) and the tree inference algorithm (Section 3) it is possible to narrow down the possible locations of the bottlenecks in the multicast tree.

Receivers appear as leaves in the reconstructed logical tree. Section 2 gives us an estimate of the bottleneck bandwidth between the source and each leaf node. The bottleneck bandwidth seen by each

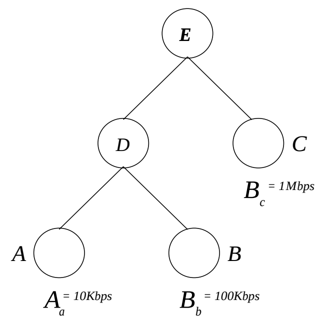


Figure 7: *The bottleneck bandwidth seen by each interior node equals at least the maximum of the estimate of its downstream nodes*

interior node is at least equal to the maximum of the bottleneck bandwidth estimates seen by each of its downstream receiver nodes.

This can be easily understood by the simple example in Figure 7. Node D has to see a maximum rate of at least 100Kbps in order for receiver B to see a bottleneck rate of 100Kbps. This implies that the bottleneck limiting the rate seen by receiver A lies along the branch AD. Similarly node E has to see a rate of at least 1Mbps and hence the bottleneck seen by receiver B lies some where along the path $ED - DB$. We cannot narrow down the location of the 100Kbps bottleneck link any further because having removed link DA from consideration we are left with the same case as the unicast path and hence we cannot obtain a more precise estimate using only information obtained at the end hosts. However, knowing that the bottleneck link lies somewhere along the path from E to B is sufficient for schemes that do not enlist router support because all receivers downstream from B would share the same bottleneck in any case irrespective of which component link along the path constitutes the bottleneck and hence knowing the exact location of the bottleneck does not provide us with any more useful information.

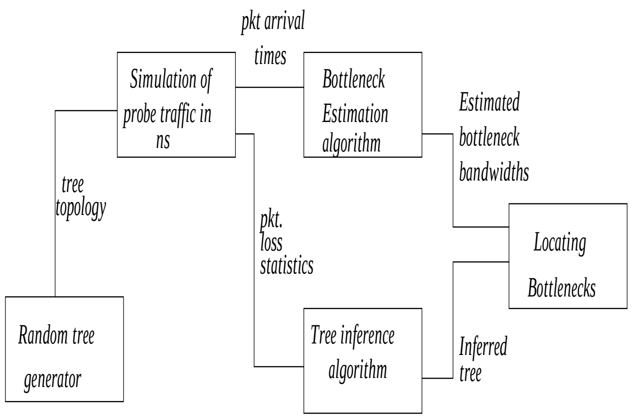


Figure 8: *Implementation modules*

5 Implementation and Testing

5.1 Implementation

The implementation modules for the algorithms described in the previous sections are shown in Figure 8. The topology generated by the random tree generator is constructed in the VINT network simulator, ns [12]. Probe traffic is sent out by the source (root) of the tree. Cross traffic (FTP, Telnet and Constant Bit Rate/UDP) is generated to try and simulate the cross traffic that could cause queuing delays that appear as noise in the measurements made at the receiving end.

5.1.1 Random Tree Generator

The random tree generator module outputs a tree topology which is used to test the bottleneck estimation and tree inference algorithms. Input parameters to this module are: maximum number of nodes ($Nodes_{max}$), maximum number of leaf nodes ($Leaves_{max}$) and the maximum fan-out ($Fanout_{max}$) of every node in the tree. The tree generator algorithm works as follows :

- Initially the tree consists of only the root at level 0. The number of children generated by the root is chosen at random from the range $[1 - Fanout_{max}]$. These child nodes are added to the tree as level 1 nodes.
- Each newly added child node can be either a leaf or an interior (non-leaf) node. A node may be a leaf node with probability $\frac{Leaves_{max} - Leaves_{current}}{Nodes_{max} - Nodes_{current}}$, where $Leaves_{current}$ and $Nodes_{current}$ are the number of leaves and nodes in the tree so far. Thus, as we approach the desired number of nodes in the tree, nodes have a high probability of being leaf nodes. If $Nodes_{max} = Nodes_{current}$ then, a node is a leaf node with probability 1. These heuristic rules ensure that the tree generation process terminates.
- An interior node adds child nodes to the next level in the tree. The number of child nodes generated by an interior node is selected at random in the range $[1, min(Fanout_{max}, Nodes_{max} - Nodes_{current})]$. In this way, starting from the root, child nodes are added to successive levels in the tree. The process terminates when the lowest level in the tree has only leaf nodes.

5.2 Testing

In order to quantify the performance of our bottleneck estimation and tree inference algorithms, we augmented the implementation modules in Figure 8 with two test modules.The tree comparator module compares the original tree topology generated by the random tree generator with the inferred tree topology. The bottleneck comparator module compares the estimated bottleneck bandwidths with the actual ones. We have conducted experiments to test the bottleneck estimation and tree inference algorithms. Our results are described in the following sections.

5.2.1 Bottleneck Estimation

We generated 50 data traces in ns. For each trace the following parameters were varied either singly or in combination with others:

- Topology of the multicast distribution tree.
- Link Bandwidths.

Results of estimation	No. of estimates
Estimate within 1% of exact value	296
No estimate due to insufficient number of valid samples	12
Incorrect estimate	4
Total number of estimates	312

Table 1: *Results of Bottleneck Estimation*

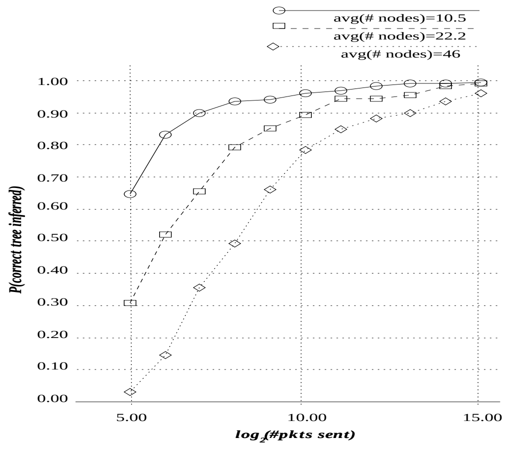


Figure 9: *Binary trees: individual link loss rate is selected at random in the range [0%,10%]*

- Location of bottlenecks (e.g. towards leaves , near the root etc).
- Amount/Type and duration of cross traffic.
- Size of bunches of probe packets (pairs, threes or fours).
- Run time (which affects the number of gathered samples).

For each trace the bottleneck bandwidth seen by each receiver was calculated. The results are tabulated in Table 1. Our tests do not cover the entire range of possible test conditions. Further, the tests are restricted to a simulation environment which differs from actual Internet conditions. Our approach to bottleneck bandwidth estimation needs to be tested on the Internet in order to quantify its performance under realistic network traffic conditions.

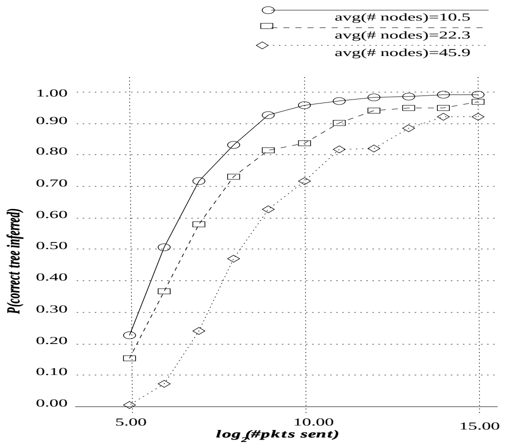


Figure 10: *Binary trees: individual link loss rate is selected at random in the range [0%,5%]*

5.2.2 Tree Inference Algorithm

Figures 9 amd 10 plot our test results for binary trees. We plot the probability of correctly inferring the tree for an increasing number of packets transmitted at the source. As the number of transmitted packets increases, the number of loss samples collected at the receiver's end increases and the approximated probability of seeing true shared losses approaches its true value. We would thus expect the probability of inferring the correct tree to approach unity as the number of collected samples increases. Figures 9 and 10 plot the observed results for different sized trees with the link loss-rates selected as a uniform distribution within a selected range. We see that the observed probability of correctly inferring the tree does in fact converge towards one with an increasing number of transmitted packets.

6 Related Work

Bolot used a stream of packets sent at fixed intervals to probe several Internet paths in order to characterize delay and loss behavior [1]. In [6], the author proposes a "packet-pair" scheme to determine the bottleneck service rate and uses this to develop a rate-based flow control scheme. Keshav's work is in the context of unicast traffic and assumes a round-robin-like queue service discipline. [2] describes the implementation of BPROBE, a tool which provides an estimate of the uncongested bandwidth of a path by sending a series of ICMP echo packets from source to destination and measuring the inter-arrival times between successive packets at the source. [14, 15] displays the fundamental limitations of sender-based packet pair techniques and advocates receiver based techniques. Paxson also points out the failure of packet-pair techniques in the face of multi-channel bottlenecks and generalizes the receiver-based packet pair (RBPP) mechanism to propose a significantly more robust procedure, "packet-bunch modes" (PBM) which is essentially based on sending bunches of probe packets and varying the bunch size keeping in mind the possibility of finding more than one bottleneck value.

[17] proposes a loss-delay based adjustment algorithm for adapting the transmission rate of multimedia applications to the congestion level of the network. The authors estimate the bottleneck bandwidths within the multicast tree in order to dynamically determine the adaptation parameters. Estimation of the bottlenecks is done by enhancing RTP with the packet pair approach. The filtering mechanism used is similar to that adopted in BPROBE.

Route tracing tools developed so far exploit certain features within the routers in order to infer the path from source to destination. The traceroute tool built by Van Jacobson discovers the path between a source and receiver of unicast traffic by using the ttl field of an IP packet header to force intermediate routers to send an error indication (ICMP time exceeded) packet back to the source thus exposing the routers within the network to discover the path between the source and receiver. The pathchar tool, also developed by Jacobson, estimates the bandwidth, delay, average queue and loss rate of every hop between any source and destination on the Internet. Pathchar uses the same basic technique as traceroute and measures the time between the transmission of an IP packet from the source and the return of the corresponding ICMP packet from an intermediate router. Analysis of the timing data reveals the characteristics of each link along the path.

Estimation of the topology of the multicast tree can be done using the tool "mtrace". mtrace discovers the multicast path from a source to a receiver using an MTRACE tracing feature implemented in multicast routers that is accessed as an extension to the IGMP protocol. A trace query is passed hop-by-hop along the reverse path from the receiver to the source, collecting hop addresses, packet counts and routing error conditions along the path, and returning the response to the requestor as a standard unicast packet. The Tracer protocol [8] uses the same MTRACE router function in order to organize the receivers of a multicast group deterministically into a logical tree structure in order to achieve effective error recovery and congestion control. In Tracer each receiver sends an MTRACE query to the source of the tree. With the existing implementation of MTRACE, this could cause scaling problems due to an implosion of MTRACE queries towards the source. Further this places a heavy load on the source which has to unicast replies back to every receiver. In order to improve the efficiency of tracing in Tracer, Levine et al propose the addition of source-based multicast tracing to IGMP.

7 Conclusions

In this paper, we presented algorithms that allow a receiver to infer the logical topology of the multicast tree, the bottleneck bandwidth of the path between the source and each receiver in the tree and the approximate location of the bottlenecks in the tree. These algorithms attempt to answer the question: how much topological information can a receiver in a multicast tree glean using only information that is readily available at the the end-hosts with the existing IP Multicast service model?

Through the use of an IP group address the IP multicast service provides a "level of indirection" on account of which receivers and senders need not know about each other. While this receiver anonymity allows multicast sessions to scale to large sizes, potentially useful information is lost in the process. Mechanisms that allow group members to reconstruct this lost information are thus useful. The algorithms presented here are a step in this direction.

8 Acknowledgments

We thank Matt Podolosky for his help with the mathematical analysis of our problem, Suchitra Raman, Hari Balakrishnan and Yatin Chawathe for their time and help.

References

[1] BOLOT, J.-C. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of SIGCOMM '93* (San Francisco, CA, Sept. 1993), ACM, pp. 289–298.

[2] CARTER, R., AND CROVELLA, M. Measuring bottleneck link speed in packet-switched networks. Technical Report TR-96-006, Boston University, Boston, MA, Mar. 1996.

[3] COSTELLO, A., AND MCCANNE, S. Search Party: Using Randomcast for Reliable Multicast with Local Recovery. In *Proceedings IEEE Infocom '99* (New York, NY, Mar. 1999).

[4] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of SIGCOMM '95* (Boston, MA, Sept. 1995), ACM, pp. 342–356.

[5] JACOBSON, V. Congestion avoidance and control. In *Proceedings of SIGCOMM '88* (Stanford, CA, Aug. 1988).

[6] KESHAV, S. *Congestion Control in Computer Networks*. PhD thesis, University of California, Berkeley, Sept. 1991.

[7] KOUVELAS, I., HARDMAN, V., AND CROWCROFT, J. Network adaptive continuous-media applications through self organised transcoding. In *Proceedings of the Network and Operating Systems Support for Digital Audio and Video* (Cambridge, U.K., July 1998).

[8] LEVINE, B. N., PAUL, S., AND GARCIA-LUNA-ACEVES, J. Organizing multicast receivers deterministically by packet-loss correlation. In *Proceedings of ACM Multimedia '98* (Bristol, UK, Sept. 1998), ACM.

[9] LIN, J. C., AND PAUL, S. RMTP: A reliable multicast transport protocol. In *Proceedings IEEE Infocom '96* (San Francisco, CA, Mar. 1996), pp. 1414–1424.

[10] LIU, C.-G. Local error recovery in SRM: Comparison of two approaches. Technical Report USC-CS-97-648, University of Southern California, 1997.

[11] MCCANNE, S. Scalable multimedia communication with Internet multicast, light-weight sessions, and the Mbone. Technical Report CSD-98-1002, University of California, Berkeley, CA, Mar. 1998.

[12] MCCANNE, S., AND FLOYD, S. *The LBNL/UCB Network Simulator*. Lawrence Berkeley Laboratory, University of California, Berkeley. Software on-line[1].

[13] PAPADOPOULOS, C., PARULKAR, G., AND VARGHESE, G. An error control scheme for large-scale multicast applications. In *Proceedings IEEE Infocom '98* (San Francisco, CA, Mar. 1998).

[14] PAXSON, V. End-to-end routing behavior in the Internet. In *Proceedings of SIGCOMM '96* (Stanford, CA, Aug. 1996), ACM, pp. 25–38.

[15] PAXSON, V. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, Lawrence Berkeley Laboratory, 1997.

[16] RAMAN, S., MCCANNE, S., AND SHENKER, S. Asymptotic scaling behavior of global recovery in SRM. In *Proceedings of SIGMETRICS '98/PERFORMANCE '98 Joint International Conference on Measurement and Modeling of Computer Systems* (Madison, WI, June 1998).

[17] SISALEM, D., AND SCHULZRINNE, H. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. In *Proceedings of the Eight International Workshop on Network and OS Support for Digital Audio and Video* (Cambridge, U.K., July 1998), ACM.

[18] YAJNIK, M., KUROSE, K., AND TOWSLEY, D. Packet loss correlation in the MBone multicast network. In *Proceedings IEEE Global Internet '96* (London, England, Nov. 1996).

[19] YANG, X., AND WEI LEHMAN, L. Inferring characteristics of multicast trees, Dec 1998. MIT 6.896 Topics in Computer Networks term project and paper.

[20] YAVATKAR, R., GRIFFIOEN, J., AND SUDAN, M. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia '95* (San Francisco, CA, Nov. 1995), ACM.

[1]http://www-nrg.ee.lbl.gov/ns/