Using Tree Topology for Multicast Congestion Control

Srinivasan Jagannathan
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
jsrini@cs.ucsb.edu

Kevin C. Almeroth
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
almeroth@cs.ucsb.edu

Abstract— **Multicast is a promising technique for mass distribution of streaming media. However, the inherent heterogeneity of the Internet poses several challenges. A major challenge is to develop a congestion control mechanism that is *efficient*, *flexible* (to administrative heterogeneity) and *deployable*. Many approaches using a layered encoding scheme have been proposed to address this problem. In parallel, many tools are being developed which provide a snapshot of network internals. Of particular interest are multicast topology discovery tools. The existence of such tools motivates the possibility of using tree topology information for multicast congestion control. In this paper we seek to understand the benefits of such a mechanism and the challenges in its practical implementation. We develop an algorithm, called *TopoSense*, which uses topology information and layered streams to control congestion within an administrative domain. Our algorithm presents a new model for multicast congestion control as it does not involve on-router computation as opposed to other approaches which require router support. We evaluate our algorithm using *ns*, the network simulator. Our results indicate that topology information is very useful in understanding the dynamics of multicast congestion and can be used for efficient traffic management.**

Keywords: Congestion Control, Layered Multicast, Tree Topology

I. Introduction

The Internet is seeing an explosive growth in the popularity of streaming media traffic. Originally designed for data transport, the Internet is severely constrained in handling multimedia traffic. However, IP Multicast alleviates the problem by enabling a source to send a single stream to multiple recipients[1]. This yields many performance improvements over the unicast model and conserves bandwidth end-to-end.

However, the inherent heterogeneity in the structure of the Internet poses a number of challenges that need to be addressed before multicast can be used efficiently. Differences in link bandwidth, temporal and geographical variance in traffic density, and differences in administrative policies are factors which make the Internet heterogeneous. In a multicast scenario, where there are multiple receivers receiving the same stream, these factors exacerbate the issue of congestion control. The multimedia stream must be multicast in such a manner so as to satisfy varying requirements of the receivers and at the same time avoid congesting the network. For instance, consider two receivers, one on the same Ethernet as the source and the other using a 56Kbps modem. A high data rate will congest the bottleneck link of the second receiver while a low data rate would imply a poor quality of reception for the first receiver even if (s)he has the resources to receive better quality. To deal with such cases, use of multiple multicast streams has been suggested[2]. Different receivers with varying capabilities can subscribe to different streams based on their capacity. The streams can either be replicas with differing qualities or they can be partitions of the original stream into "layers". In the layered model, each receiver subscribes to the base layer and some enhancement layers.

Knowledge of multicast tree topology can be very useful in congestion control [3]. We illustrate this using a simple example (Figure 1). Assuming a layered multicast model, receivers subscribe to a base layer and some enhancement layers. Assume that layer 1 requires a bandwidth of 32Kbps and every subsequent layer requires twice the bandwidth required by the previous layer. Using the topology in Figure 1, the receivers at nodes 3 and 4 can hope to receive layers 1 and $1, 2$ respectively. Because, the receivers may not be aware of their bottleneck bandwidth, they may subscribe to more layers. Suppose, the receiver at node 4 tries to subscribe to one more layer. This will result in congestion at node 2 and hence losses for both node 3 and node 4. A congestion control mechanism which is unaware of the topological relationship between nodes 3 and 4, may take incorrect decisions to control losses at node 3.

In this paper, we investigate the utility of topology-information in controlling congestion. The approach we suggest is unique in the sense that it does not build upon any existing protocols. It combines the layered models, which focus on congestion control, and topology discovery tools which focus on network administration in a manner that is flexible and easy to deploy. The most significant aspect of our work is that it is designed for the application layer and does not have special requirements other than the existence of a tool which discovers the multicast tree topology in the local domain. The architecture we describe focuses on *local congestion control* in independent domains.

The rest of the paper is organized as follows. In Section 2, we describe the architectural framework of our algorithm and place it in the context of the current Internet topology. In Section 3, we develop an algorithm we call *TopoSense* and present the intuition behind it. In Section 4, we evaluate this algorithm using the network

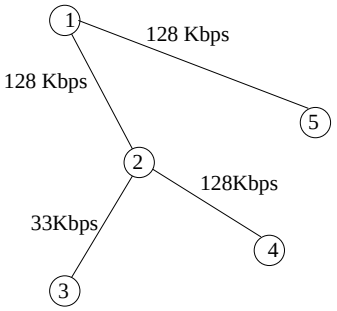

Fig. 1. A simple multicast tree topology.

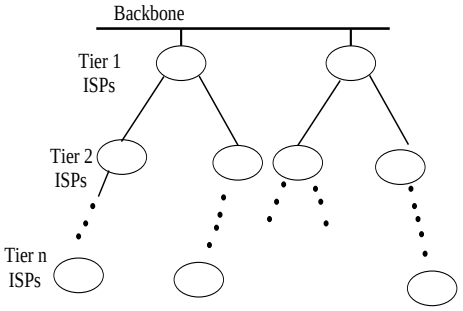

Fig. 2. The Tiered Structure of the Internet.

simulator *ns* and present simulation results. In Section 5, we discuss the challenges in using topology information effectively. We conclude the paper in Section 6.

II. Multicast Topology and TopoSense Architecture: The Tiered Approach

In its current state, the multicast-enabled Internet can be represented as a *tiered* structure (Figure 2). The first tier consists of national Internet Service Providers (ISPs), the second tier of regional ISPs, the third local ISPs and so on. All of the recipients (and possibly the source) are connected to institutional ISPs. These institutional ISPs are in turn served by ISPs higher up in the hierarchy. It is reasonable to assume that owing to the higher volume of traffic, the higher tiers have a larger bandwidth capacity than those of the lower tiers. The link with the smallest bandwidth capacity in the path from the source to a receiver is called the bottleneck link for that receiver. Owing to their smaller bandwidth capacity, the links in the lower tiers are typically the bottlenecks. This is commonly referred to as the last mile problem. Congestion control near the last mile of a low capacity recipient will alleviate the problem locally, while leaving other high capacity recipients unaffected.

Our architecture is based on the challenge of solving the bandwidth problem in the last mile. It exploits the idea that bottlenecks lie deep in the tree and that disjoint subtrees on the multicast tree do not affect each other as long as their common ancestors have a high capacity. For instance, in the example in Figure 1, the subtree rooted at node 5 will be independent of the subtree rooted at node 2. Our algorithm, TopoSense is built around this notion of subtree independence. Our architecture uses multiple controller agents, each concerned with one particular administrative domain. Each domain and controller agent is unaware of the other controller agents' existence. These controller agents are application-level entities — routers in the domain are unaware of their existence. Potential recipients of multicast traffic register themselves with the controller agent managing their domain. The recipients periodically report loss information to the controller agent. The controller agent also uses a topology discovery tool to discover the topology of different multicast sessions in that domain. Since the controller agent is concerned only with the topology in its domain, discovering the local tree topology efficiently may be more tractable than discovering the entire tree topology. We defer a discussion on this subject to later sections. The controller agent uses the topology information along with the loss reports to decide optimal subscription levels for different recipients. It periodically transmits this information to the recipients who then obey the suggestions. Since the network is lossy, it is possible that suggestion packets from the controller agent get lost. The recipients can make unilateral decisions if they do not receive suggestion packets for a long period of time. Our architecture is illustrated in Figure 3. Controller agents can also be very useful for billing customers based on multicast content delivered.

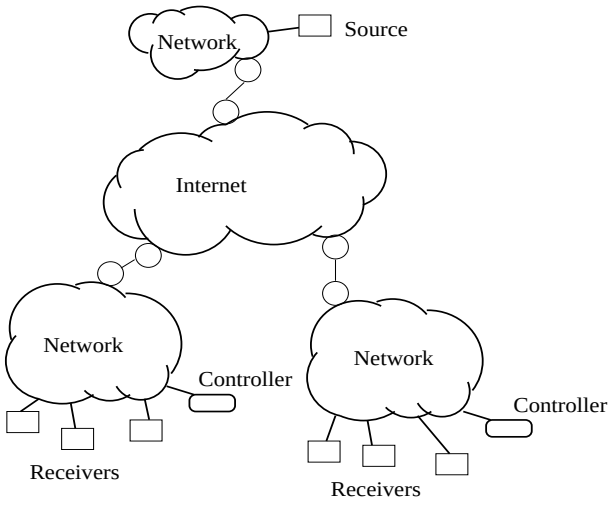

Fig. 3. Scalability using a hierarchical control model.

A very attractive feature of our architecture is that it adds an element of scalability; congestion control can be managed on sub-trees instead of on the whole tree. Secondly, it is highly flexible to administrative heterogeneity. Administrative domains using TopoSense have the incentive of local congestion control, but those choosing otherwise do not have any disincentives either. Moreover, other domains will not be affected by this decision. There is also flexibility in the mechanism used for collecting topology information. Our algorithm concerns itself only with the

information and not how it was acquired. This is of great importance, given that the current Internet is highly heterogeneous. On the negative side, our architecture makes sense only for long-lived multicast sessions. But, since multimedia traffic is typically long-lived, this is a reasonable trade off.

III. TopoSense Algorithm

The TopoSense Algorithm works on a set of graphs representing the multicast tree topologies and layers traversing different parts of the controller agent's domain. In this section:

- A *node* refers to a node in the graphs maintained by the controller agent
- A *link* refers to an edge in the graphs
- A *multicast session* refers to a set of layers being transmitted on different multicast addresses
- A *multicast session topology* is the overlay of the multicast distribution tree for each layer of the session.

Since the layers are cumulative, i.e., a receiver getting layer i must also receive layers 0 ... $i-1$, the multicast session topology will be a tree. The session topologies are constructed using the topology discovery tool and the loss and subscription information reports obtained from recipients of the multicast sessions. All actions performed by TopoSense are on this internal image of the multicast tree topologies. The algorithm assumes that the average bandwidth of each layer is known beforehand. This assumption is reasonable as this can be advertised along with the multicast address of the layer. The TopoSense algorithm uses multicast tree topology information and loss reports from multicast recipients to recognize occurrences of congestion. It allows receivers to explore available bandwidth in a coordinated manner and adapts to transient traffic and competing sessions. TopoSense limits the maximum subscription of layers in a subtree to the maximum bandwidth between any receiver in the subtree and the source.

Using the topology discovery tool, the controller agent obtains regular updates of the tree topology of each multicast session in its domain and the layers traversing each part of that tree. This is the only phase where information needs to be obtained directly from routers. However, this is handled by the topology discovery tool and the controller agent does not concern itself with how this information is obtained. Every receiver is assumed to have registered itself with the controller agent when it starts subscribing to a multicast session. The controller uses the topology discovery tool to construct the tree topology for each session. In addition, the agent gathers packet loss information and the number of bytes received at each receiver. Loss information can be obtained using mechanisms like (but not limited to) Realtime Transport Control Protocol (RTCP[4]). Gathering loss and subscription information involves the flow of control traffic and is done over a period of time.

TopoSense Algorithm

Input:
Multicast session topologies in the domain
Receiver packet loss rates
Number of bytes received at leaf nodes.
Output: Subscription level for each leaf node in each session
Algorithm:
For each session do
Compute congestion state for each node in the session topology
Estimate link bandwidths for all shared links
For each session do
Find the bottleneck bandwidths for each node in the session.
Estimate the fair share of BW on shared links based on the bottlenecks.
For each session do
compute the subscription level for each leaf node in the session.

Fig. 4. Stages of the TopoSense Algorithm

The controller agent runs the TopoSense algorithm periodically with the above information as input. As output, the TopoSense algorithm produces the number of layers each receiver should subscribe. The controller agent then sends control messages to the receivers with suggestions for their subscription level. The main stages of the TopoSense algorithm are described in Figure 4.

A preliminary version of the algorithm for a single multicast session focusing on intra-session fairness was presented in our previous work[5]. In this paper, we address the more general case of multiple multicast sessions competing for bandwidth. Sarkar and Tassiulas[6], [7] have discussed the intractability of this problem. We evaluate TopoSense vis a vis. inter-session fairness, stability of subscription and the impact of stale topology information. We now briefly describe the various stages of the TopoSense algorithm.

Computing Congestion States: The algorithm assumes that a leaf node is congested if the packet loss rate at that node (for a particular session) is higher than a threshold. Since, packet loss rate is known only for leaf nodes in the tree, the packet loss rate at an internal node (for a particular session) is assumed to be the minimum of the packet loss rates at the children of that node. The intuition behind this assumption is that if all the children of a node are congested, then all the children will have to reduce their bandwidth demands. This in effect means that the parent node reduces its demand to a level equal to the maximum demand of all its children, even if it is not congested. Packet loss rates for internal nodes are therefore computed in a bottom-up fashion. An internal node is considered congested if:

- its parent is congested, or
- all its children have a packet loss rate greater than a threshold value $p_{threshold}$ and more than $\eta_{similar}$ percent of its children have packet loss rates which are close to the average packet loss rate of all the children, the deviation in packet loss rates of the children is negligible.

At the end of this stage each node is labeled as CONGESTED or NOT−CONGESTED. In addition, the maximum number of bytes received by any receiver in the sub-tree rooted at each node is recorded. This is useful for estimating the capacities of shared links.

Estimating Link Capacities: In the absence of knowledge of link capacities, the algorithm estimates link capacities. An estimate of the link capacity is needed to share the bandwidth among competing sessions. The links are assumed to be of infinite capacity until 1) the overall loss of packets on that link (i.e. the destination node of that link) is greater than a threshold, and 2) all the sessions sharing that link have a packet loss greater than a threshold. The second condition is required because, loss at an internal node for a particular session is, in reality, the minimum packet loss of any receiver in the subtree rooted at that node. It is possible therefore that the estimated loss for one session is much higher than the estimated loss for another session. This may be because some downstream node for the first session is congested while no downstream nodes for the second session are congested. However, if all the sessions have high losses, then the probability that the link that they share is congested is much higher. If both these conditions are satisfied, then the bandwidth of the link is computed to be the average number of bits transmitted per second, in that interval. Once the bandwidth is computed, the estimate is increased every interval by a small amount. This is done because, the number of bytes transmitted over that link may not equal the number of bytes reported received by the receivers. This is possible when packets are still remaining in the network and a receiver sends a report. Since transient non-conforming flows, as well as bottleneck capacities downstream can lead to wrong estimates of bandwidth, the capacity is reset to infinity at periodic intervals and recomputed.

Finding Bottleneck Bandwidths: Once the shared link capacities are known, we would like to know how much bandwidth each session can possibly use on a shared links without causing congestion downstream. Bottleneck bandwidths for each node in each session are found by propagating the minimum link capacity from source to every node in the tree. This can be done in a single top-down breadth-first pass over the tree. The maximum bandwidth that each node can handle in some session is set to be the maximum bottleneck of all the children in that session. This can be done using a breadth-first, bottom-up pass.

Bandwidth Sharing: In this stage, the available bandwidth on a shared link is shared among competing sessions. Since each session may have a different bottleneck capacity based on downstream link capacities and different bandwidth requirements based on session characteristics, an equal share would be inappropriate. Sarkar and Tassiulas have shown that min-max fair allocations may not exist for the layered model. We therefore use an intuitive model of fairness. This is a reasonable approach because we do not even know the exact link capacities in the network. Every session must get as much bandwidth at each link as can possibly be used without congesting the network. For example, if there are two sessions sharing a link, one with a downstream bottleneck of 250Kbps and the other with a downstream bottleneck of 1Mbps, then we would expect the first session to get 250Kbps and the second session to get 1Mbps. TopoSense computes a fair share as follows. It is assumed the every session gets at least as much bandwidth as that of the base layer. Hence, at each shared link, TopoSense computes the maximum number of layers that each session can receive if all other sessions were to receive only the base layer. This is done top-down breadth first for each session. Thus, at the end of this pass, the maximum possible layers that can be received by each leaf node is known. Now, there is a bottom-up pass, where the maximum possible demand of internal nodes is set to the maximum possible demand of any of its children. Suppose there are n sessions and the maximum possible demand so computed is x_i for each session i. Then the fair share of session is computed to be: $\frac{x_i \times B}{\Sigma_{j=1}^{n} x_j}$ where B is the estimated link capacity. One can easily construct examples where this algorithm may allocate more bandwidth than needed for some session and thereby deprive some other session. However, no session is under-allocated. Moreover, the system will stabilize when the link capacity is re-estimated. At the end of this stage, the following are known:

- the congestion state of every node (for each session),
- the capacity of shared links and
- the "fair" maximum bandwidth that each session can utilize at shared links without causing congestion downstream.

Computing Subscription Levels: In this stage, the available bandwidth for each session is used to prescribe subscription levels for each node. This stage of the algorithm has been presented in greater detail in our previous work[5]. This stage can be subdivided in two parts: 1) computing demand for each node and 2) allocating supply. The demand computation process can be represented as a table lookup indexed by the congestion state history (congestion states computed over the previous intervals) and total advertised bandwidth subscribed in the past two intervals. The decision table is presented in Table I. The congestion state history is presented as a 3 bit integer in the table where CONGESTED=1 and NOT−CONGESTED=0.

TopoSense is run in intervals. Suppose that the i^{th} interval starts at time T_i and we are computing demand for the interval starting at time T_2. The congestion states at times T_0, T_1 and T_2 are represented at bit positions 2, 1, and 0 respectively. The column, "BW Equality", represents the relationship of the total bandwidth received in interval $T_0 - T_1$ with respect to total bandwidth received in interval $T_1 - T_2$. The demand computation proceeds in a bottom-up, breadth-first manner. The demand at an internal node is the aggregation of the demands of the children. If a parent node is congested, the children assume that they are congested because the parent is congested and defer action to the parent. If a node is congested but its parent is not, then it reduces the overall demand. In general, in case of congestion in a sub-tree, action is taken by the root of that sub-tree. This node also sets a back-off timer for the highest layer being dropped so that this layer is not subscribed to by another receiver in the near future. Coordination among receivers is thus achieved. Allocation of supply is simply a top-down pass which sets the supply according to the demand.

Leaf/Internal Node	Congestion State History	BW Equality	Action
Leaf	0	Lesser	Add next layer, if not backing off.
Leaf	1	Lesser	If loss rate is high, drop layer, set backoff timer
Leaf	2,4,5,6	Lesser	Maintain Demand
Leaf	3	Lesser	Reduce demand to supply in $T_0 - T_n$
Leaf	7	Lesser	Reduce Demand to half the supply in $T_0 - T_n$ Set the backoff timer
Leaf	0,4	Equal	Add next layer, if not backing off.
Leaf	1,2,5,6	Equal	Maintain Demand
Leaf	3,7	Equal	Reduce Demand to half the supply in $T_0 - T_n$ Set the backoff timer.
Leaf	0	Greater	Add next layer, if not backing off.
Leaf	1,2,4,5,6	Greater	Maintain Demand
Leaf	3,7	Greater	If loss is very high, then reduce demand to half the supply in $T_0 - T_n$
Internal Node	0,4	All Cases	Accept all demands of the child nodes
Internal Node	1,5,7	Greater	Reduce Demand to half the supply in $T_n - T_{2n}$
Internal Node	1,5,7	Equal, Lesser	Reduce Demand to half the supply in $T_0 - T_n$
Internal Node	2,3,6	All Cases	Maintain Demand

TABLE I

DECISION TABLE FOR COMPUTING DEMAND AT EACH NODE AT TIME T_2.

IV. EVALUATION OF TOPOSENSE

We implemented the TopoSense algorithm described above in the network simulator *ns*. Our work uses a hierarchical source model[8]. Sources transmit a layered video session consisting of 6 layers. The base layer is sent at a rate of 32Kbps, with the rate doubling for each subsequent layer. We consider constant bit rate as well as variable bit rate sources. We used the same method to generate variable bit-rate (VBR) traffic as proposed by Gopalakrishnan et al.[9]. For the base layer, n packets are transmitted in 1 second intervals. n has the value 1 with probability $1 - \frac{1}{P}$ and the value $PA + 1 - P$ with probability $\frac{1}{P}$. A is the average number of packets per interval and P is the peak-to-mean ratio. Peak-to-mean ratios in the range of 2 to 10 have been observed for VBR traffic[9]. The packet size is chosen to be 1000 bytes. A drop-tail policy was used at all nodes and link latency was chosen to be 200ms for all links. The link capacities were varied.

In our previous work[5], we presented results to illustrate that TopoSense converged to optimal subscription of layers in a heterogeneous environment. These results also showed that TopoSense imposed intra-session fairness for a single multicast session. In this paper, we evaluate TopoSense on the following:

- **Stability:** There must not be frequent changes in bandwidth received. Ideally, layers should only be removed when congestion occurs and layers should only be added when capacity exists.
- **Inter-session fairness:** When there are multiple sessions traversing the same set of links, bandwidth should be fairly and fully utilized.
- **Robustness to Stale Information:** We study the impact of old topology and loss information on the performance of TopoSense. Ideally, we would like TopoSense to work well even with old information.

We used the topologies in Figure 5 to evaluate TopoSense. Topology A is used to evaluate TopoSense behavior in a heterogeneous environment. Topology A consists of a single multicast session with two sets of receivers, each having different bandwidth constraints. The number of receivers in each set is increased and their subscription level monitored. Topology B consists of a number of different multicast sessions, each with one receiver, sharing the same bottleneck link. The link capacity of the shared link is varied with the number of sessions being simulated, so th at each session can ideally receive 500Kbps (4 layers). Since, we know the optimal solutions for our topologies, we evaluate the performance of TopoSense by comparing its behavior with that of the optimal. To do so, we define the following metric. Let $x_i(\Delta t)$ be the subscription level of receiver i in the interval Δt and y_i be the optimal subscription. Let $\parallel \Delta t \parallel$ be the length of the interval Δt. We define the relative deviation over the entire interval $\sum_{\Delta t}$ to be:

$$\frac{\sum_{\Delta t} \mid (x_i(\Delta t) - y_i) \times \parallel \Delta t \parallel) \mid}{\sum_{\Delta t} y_i \times \parallel \Delta t \parallel}$$

Intuitively, the smaller the relative deviation, the better the performance. It can be noted that once a stable state has been achieved, this metric can be made arbitrarily good by increasing the time interval. However, this is contingent on the system achieving a stable state. This metric is sufficient for our purposes as we can compute the relative deviation for time intervals of arbitrary length. We ran numerous simulations based on the parameters outlined above. All the simulations were run for a period of 1200 simulated seconds. The controller agent was stationed at one of the source nodes in both the topologies. This made the simulations more realisitic as control messages could be lost due to congestion.

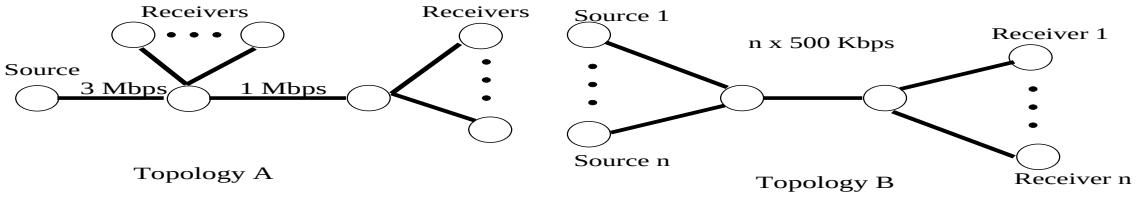

Fig. 5. Simulation topologies.

Stability. We counted the number of times layers were added or dropped by each receiver over the period of 1200 seconds, for Topologies A and B. For Topology A, we plot the maximum number of changes in subscription by any receiver over the entire 1200 seconds. We also plot the mean time elapsed between successive changes in the subscription level for that receiver (Figure 6). Similarly, for Topology B, we plot the maximum number of changes in the subscription level in any session over the entire 1200 seconds and the mean time elapsed between successive changes for that session. The results are plotted for CBR, VBR (P=3) and VBR (P=6) traffic (Figure 7). There appears to be high variability in the number of changes. This is because of the random back-off interval chosen. Most of the changes occur when the receivers explore available bandwidth by adding a new layer. Oversubscribing causes losses and the receivers back-off. These joins are immediately followed by a leave. Hence the subscription consists of long stable spells interspersed with very small intervals of joins/leaves. These results clearly indicate that the subscription level is fairly stable over time and can be controlled using the back-off interval.

Inter-session Fairness. Topology B was used for evaluating inter-session fairness. A common link was shared by different numbers of multicast sessions. The capacity of the link was varied so that each session could ideally receive 4 layers if the bandwidth was shared in a fair manner. Simulations were run with up to 16 competing sessions for CBR, VBR (P=3) and VBR (P=6) traffic. All the sessions started at the same simulated time. The mean relative deviation from optimal subscription is plotted for the time intervals 0-600 seconds and 600-1200 seconds (Figure 8). A small relative deviation in both these intervals indicates that TopoSense imposes fairness among competing sessions irrespective of the time intervals. Figure 9 shows a sample plot illustrating the subscription of layers and loss rate over a 10 seconds window when there are four competing sessions sharing the link. As is seen, some of the sessions over-subscribe to layers 5 and 6 at several points in time. This happens because of two reasons. First, TopoSense recomputes the link capacity at periodic intervals. To do this, it first sets the link capacity to infinity. This allows some sessions to over-subscribe. Second, the traffic is bursty. Therefore the sessions may actually not see losses when they add layer 5. Hence they over-subscribe layer 6. However, heavy losses on adding layer 6, allow TopoSense to compute the link capacity and the system returns to a stable state.

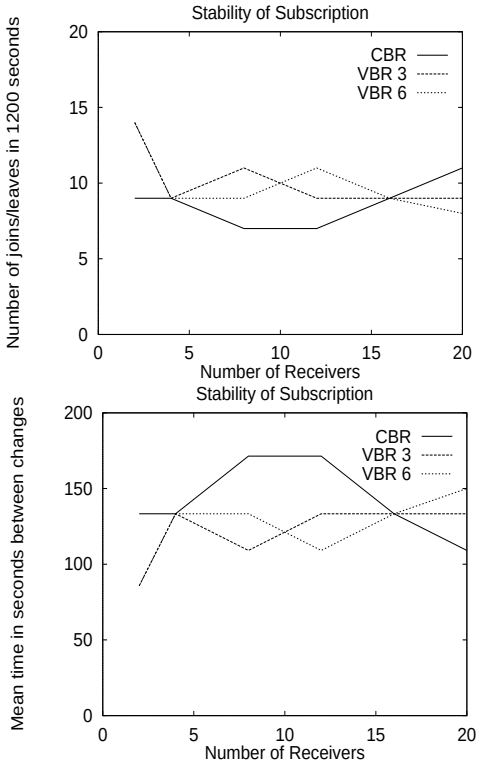

Fig. 6. Stability in Topology A

Impact of Stale Information. The results presented above are based on topology information available instantaneously. This is clearly an unrealistic premise. To see how old information affects results, we ran numerous simulations with old topology information. We varied the staleness of information (from 2 seconds to 18 seconds) and studied its impact on the mean relative deviation from the optimal subscription of layers. We ran these simulations on Topology A. The results for VBR (P=3) traffic, with different number of receivers in the session, are presented in Figure 10. As expected, performance deteriorates with stale information. The session with only 2 receivers appears to be least affected to staleness of information. This can be attributed to minimal control traffic from the controller agent to receivers. For the sessions with more than two receivers, performance deteriorates after 4 seconds and stabilizes at 10 seconds. The maximum path latency between source and receiver in these sessions is 600ms. The maximum latency of any path from source to receiver is important because, discovering the

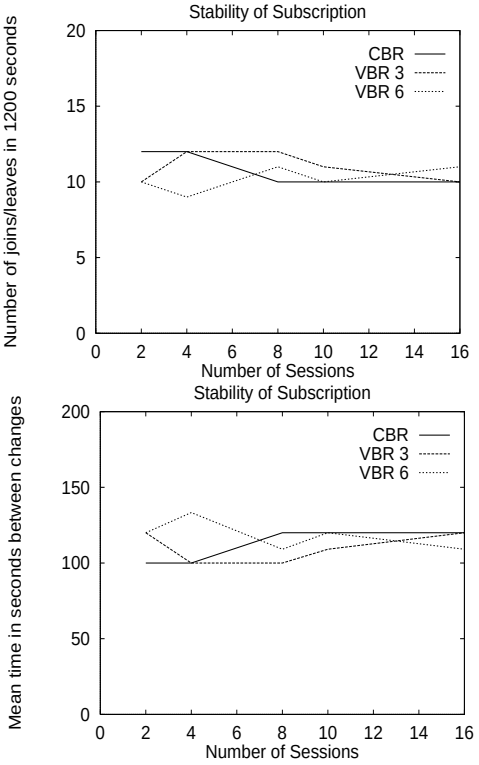

Fig. 7. Stability in Topology B

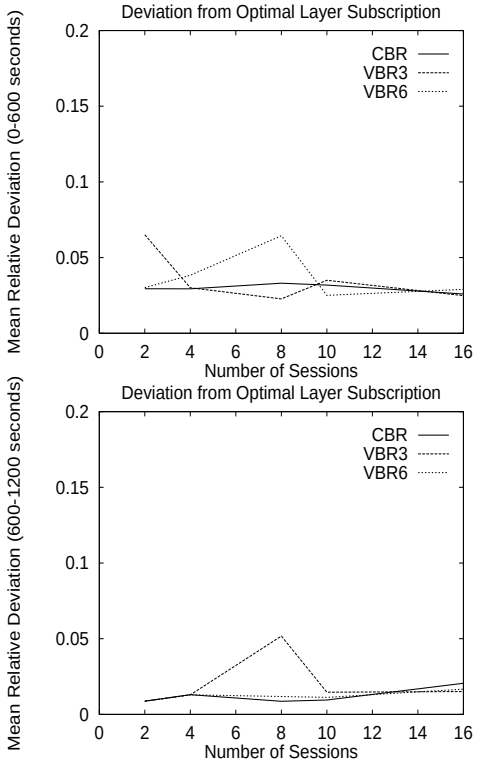

Fig. 8. Fairness in Topology B

tree topology is dependent on this latency. Compared to this latency, TopoSense does appear to perform well even with information as old as 8 seconds. This is long enough for the topology discovery tool to discover the topological information.

V. Challenges in Using Topology

The simulation results presented in the previous section indicate that TopoSense can be valuable for congestion control. However, there are a number of issues to be resolved:

- **Discovering topology efficiently:** Though there are a number of tools which discover tree topology, many of them have scalability problems. However, sub-trees can be discovered with fewer of scalability problems. Tools which allow discovery of multicast sub-trees need to be developed.
- **Group-leave latency and layer granularity:** Leaving a troublesome group may not immediately alleviate congestion because the last hop router must use Internet Group Management Protocol (IGMP[10]) to verify that there are no receivers for that group. The latency in dropping a layer can cause congestion if the layer to be dropped has a very high data rate. A possible remedy to this problem is to have finer granularity in bandwidth requirements of layers. Adding a layer may increase bandwidth demands by smaller amounts thereby limiting the magnitude of possible congestion. However, the granularity of layers is severely constrained by the encoding scheme. Further, a very large number of layers can delay convergence since layers are added one at a time. Some form of interaction between border routers and the controller agent can greatly decrease group leave latency since the controller is aware of all the receivers of a particular layer in the entire session subtree. Expedited group-leaves, where routers keep track of receivers downstream, may also be considered for decreasing group-leave latency.
- **Estimating link capacity:** We have designed an algorithm to estimate link capacities in this paper. However, it is only a first step in this direction[1]. It is constrained by lack of loss information at internal nodes. The algorithm can possibly under-estimate the link capacities. However, this is not a serious problem since the capacities are recomputed at frequent intervals. Nevertheless, there is a need for better mechanisms to estimate link capacities.
- **Dealing with bursty traffic:** Highly bursty traffic can, in some cases, lead to longer convergence time. Burstiness can cause buffer overflows at routers thereby causing packet loss at receivers. When there is more than one session sharing the same link, each with different data rates for different layers, then high burstiness in one of the sessions can impact estimation of the link capacity. A better mechanism is needed to differentiate between bursty losses and sustained congestion.
- **Minimizing control traffic:** Topology discovery, collection of packet loss information and informing

[1]There has been other work in this area. But they are either not applicable to a multicast scenario or use probe packets which consume bandwidth.

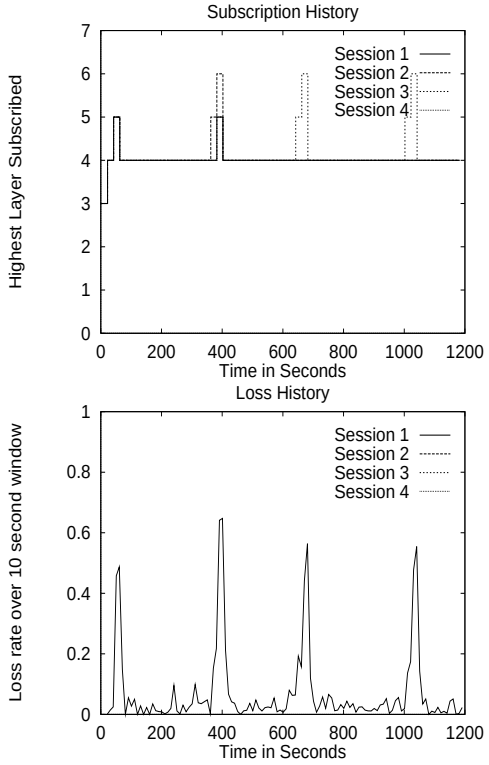

Fig. 9. Layer Subscription and Loss History for 4 competing sessions with VBR traffic

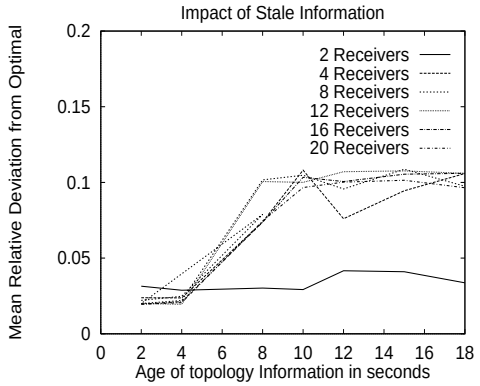

Fig. 10. Impact of stale information on Topology A subscription with VBR traffic

users about subscription level—all of these generate network traffic. TopoSense is designed in such a manner that the number of information packets exchanged in every interval is linear with respect to the number of receivers and sessions. The challenge is in choosing the correct size of domains for the controllers. A very small domain may have the same drawbacks as a receiver-oriented approach while a very large domain may have scalability problems.

- **Interval size.** The frequency with which decisions are made is a very important parameter. It may not be feasible to quickly ascertain the local topology. Burstiness in a short interval may lead to incorrect inferences about congestion. However, a large interval implies slow reaction time to congestion. In addition, information may be "stale" if a very large interval is chosen. Choosing the optimal interval size is thus crucial for the efficient use of topology for congestion control.

VI. Related Work

We can divide the problem domain in two parts: 1) discovering multicast tree topology efficiently and 2) delivering multimedia content efficiently using multicast. Techniques employing the *mtrace* utility[11] as well as SNMP[12] have been explored[3], [13], [14], [15] for multicast topology dicovery. In this paper, we focus more on efficient content delivery rather than on topology discovery. We assume that tree topology is available and assess how it can be put to use. This lets us explore the theoretical limits of this approach. There is copious literature on techniques for multicast distribution of streaming data. Li et al.[2] present a comprehensive history of approaches to multicast-based multimedia content delivery. Other approaches include network based[16] or a hybrid of network and user based[17] priority packet-dropping schemes. These approaches are either router-based or use end-to-end information. Router-based solutions, though effective, may not be easy to deploy while approaches which only use end-to-end information ignore rich sources of useful information. Moreover, coordination among receivers is very difficult to achieve in the latter class of approaches. Our approach is to develop an application-layer algorithm that relies on a topology-constructing tool to make suggestions about optimal subscription levels. An application level approach uses internal state of the network, as well as end-to-end information available from receivers through such mechanisms as RTCP.

Research efforts have also been directed to discover and quantify the TCP approach to congestion control[18], [19], so that it can be applied to UDP-based multicast traffic. These methods rely on an estimate of round trip times from the sender, and on estimates of packet loss rates. These metrics run into an intuitive roadblock when it comes to multicast. Because there are multiple receivers, attempts to define round trip times are nebulous at best. In addition, the characteristics of the streaming media traffic may not be conducive to TCP-like flow control. For instance, additive increase with multiplicative decrease, though practical for TCP, does not make sense for layered streaming media. Any attempts to simulate TCP-like behavior for such traffic requires a fundamental change in encoding techniques. In the absence of such a change, one cannot hope to simulate TCP-like behavior and at the same time ensure "quality" in data reception. Our approach has been to take a liberal view towards TCP friendliness in the light of the fact that HTTP connections, which are typically short lived, form the bulk of Internet traffic[20]. In contrast, we are focusing on multicast sessions which have a much longer duration. Coupled with the fact that multicast prunes may take time on the order of minutes[21], it is a reasonable possibility that individual TCP sessions will transmit their payload and

cease to exist by the time multicast congestion control has an effect.

The work by Sarkar and Tassiulas[6], [7] is very similar to ours. The problem domain they address is the same and the distributed architecture they propose is similar to ours. Their work shows that max-min fairness may not achievable with a layered approach. They also show that lexicographically optimal rate allocation is possible but NP-hard. However, their distributed architecture involves algorithms which will not work with the layered model addressed in this work. Their algorithm, which works for the layered model, has not been elaborated upon for a distributed architecture. A significant difference between their work and ours is that their work assumes computation within the network, or alternately access to information available within the network. For instance, their work assumes knowledge of link capacities. In our work, we assume we do not have access to network internals except for the topology information. We estimate link capacities based on loss rates at the end-receivers.

VII. Conclusions

In this paper, we have presented a case for using multicast tree topology for congestion control. We have identified the key features of a topology sensitive congestion control algorithm and developed an architecture to implement it. The main thrust of the architecture is distributed control in independent administrative domains. We believe this approach is scalable and flexible to administrative heterogeneity. The domain based controller agents described in our paper can also be used for billing customers for multicast content delivered. We have developed an algorithm called *TopoSense* which uses topology information for congestion control. Our simulation results with *ns* indicate that TopoSense is robust, shares bandwidth equitably among competing sessions and is fair to receivers with disparate resources. The results also indicate that TopoSense is fairly stable even with old topology information. Though a number of issues need to be resolved, the results corroborate our intuition that tree topology information is very useful for cognizance and control of congestion in a multicast−enabled Internet.

References

[1] K. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment," *IEEE Network*, January/February 2000.

[2] X. Li, M. Ammar, and S. Paul, "Video multicast over the Internet," *IEEE Network*, April 1999.

[3] D. Makofske and K. Almeroth, "MHealth: A real-time graphical multicast monitoring tool for the MBone," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Basking Ridge, New Jersey, USA), June 1999.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications." Internet Engineering Task Force (IETF), RFC 1889, January 1996.

[5] S. Jagannathan, K. C. Almeroth, and A. Acharya, "Topology sensitive congestion control for real-time multicast," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Chapel Hill, North Carolina, USA), June 2000.

[6] S. Sarkar and L. Tassiulas, "Fair allocation of discrete bandwidth layers in multicast networks," in *IEEE Infocom*, (Tel Aviv, ISRAEL), March 2000.

[7] S. Sarkar and L. Tassiulas, "Distributed algorithms for computation of fair rates in multirate multicast trees," in *IEEE Infocom*, (Tel Aviv, ISRAEL), March 2000.

[8] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *ACM Sigcomm*, (Stanford, California, USA), pp. 117–130, August 1996.

[9] R. Gopalakrishnan, J. Griffioen, G. Hjalmtysson, and C. Sreenan, "Stability and fairness issues in layered multicast," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Basking Ridge, New Jersey, USA), June 1999.

[10] W. Fenner, "Internet group management protocol, version 2." Internet Engineering Task Force (IETF), RFC 2236, November 1997.

[11] W. Fenner and S. Casner, "A 'traceroute' facility for IP multicast." Internet Engineering Task Force (IETF), draft-ietf-idmr-traceroute-ipm-*.txt, August 1998.

[12] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "Protocol operations for version 2 of the simple network management protocol (SNMPv2)." Internet Engineering Task Force (IETF), RFC 1905, January 1996.

[13] D. Makofske and K. Almeroth, "Real-time multicast tree visualization and monitoring," *Software–Practice & Experience*, 2000.

[14] B. Levine, S. Paul, and J. Garcia-Luna, "Organizing multicast receivers deterministically by packet-loss correlation," in *ACM Multimedia*, (Bristol, ENGLAND), September 1998.

[15] D. Thaler and A. Adams, *Mrtree*. Merit Network, Inc. and University of Michigan. Available from http://www.merit.edu/net-research/mbone/mrtree_man.html.

[16] S. Bajaj, L. Breslau, and S. Shenker, "Uniform versus priority dropping for layered video," in *ACM Sigcomm*, (Vancouver, British Columbia, Canada), pp. 131–143, September 1998.

[17] R. Gopalakrishnan, J. Griffioen, G. Hjalmtysson, C. Sreenan, and S. Wen, "A simple loss differentiation approach to layered multicast," in *IEEE Infocom*, (Tel Aviv, ISRAEL), March 2000.

[18] M. Mathis, J. Semke, J. Mahdavi, , and T. Ott, "The macroscopic behaviour of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, July 1997.

[19] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A model based tcp-friendly rate control protocol," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Basking Ridge, New Jersey, USA), June 1999.

[20] S. McCreary and K. Claffy, "Trends in wide area ip traffic patterns, a view from ames internet exchange." http://www.caida.org/outreach/papers/AIX0005.

[21] B. Williamson, *Developing IP Multicast Networks, Volume I (Fundamentals)*. Indianapolis, Indiana, USA: Cisco Press, 1999.