Scheduling Uplink Bandwidth in Application-layer Multicast Trees

Sridhar Srinivasan and Ellen Zegura

Networking and Telecommunications Group
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332, USA
{sridhar,ewz}@cc.gatech.edu

Abstract. Many applications can benefit from the use of multicast to distribute content efficiently. Due to the limited deployment of network-layer multicast, several application-layer multicast schemes have been proposed. In these schemes, the nodes in the multicast tree are end systems which are typically connected to the network by a single access link. Transmissions to the children of a node in the multicast tree have to share this single uplink, a factor largely ignored by previous work.In this work, we examine the effect of access link scheduling on the latency of content delivery in a multicast tree. Specifically, we examine the general case where multiple packets (comprising a *block* of data) are sent to each child in turn. We provide an analytical relation to compute the latency at a node in the multicast tree and show the relationship to the packet size and block size used to transfer data. We propose heuristics for tree construction which take link serialization into account. We evaluate this effect using simulations and show that using larger block sizes to transfer data can reduce the average finish time of the nodes in the multicast tree at the expense of slightly increased variance.
Keywords: Peer-to-peer networks, Multicast

1 Introduction

Application-layer multicast [6, 10, 1, 9, 5] has been proposed as a viable method for deploying large-scale multicast on the Internet. Unlike network-layer multicast, in most proposals for application-layer multicast, the nodes that form the multicast tree are end hosts. These end hosts are responsible for creating and maintaining the multicast tree and also forwarding the data to their children in the tree. Applications that benefi t from the use of application-layer multicast include media streaming, multi-player games, conferencing and fi le or content distribution.

Important metrics for these applications are the delay and jitter experienced during data transfer. In the case of fi le distribution applications, the average time to obtain the fi le is also an important requirement. For this reason most application-layer multicast schemes concentrate on creating multicast trees with low latency paths.

The end hosts that form the multicast tree are often connected to the rest of the Internet using a single access link such as a DSL or cable modem line [14]. The single access link is a shared resource that must be scheduled among the children of the node. This scheduling can affect the time taken to transfer data to the child nodes. As a simple

example, consider a case in which a source node r with a single access link to the rest of the network, transfers a block of data to its k children. Let us further assume that this block is composed of several packets. Consider two different simple means of scheduling access to the link: in the first scheme, the block is sent a packet-at-a-time to each child in turn and in the second, the entire block is sent to a single child at a time. The methods of delivery are illustrated in Figures 1 and 2 respectively. The figures show the delivery of a block of data composed of four packets from a source node to its k child nodes. In the packet-at-a-time case, the finish times of all the children are nearly equal, while in the block-at-a-time method, the 1st child has a finish time of T, the second a finish time of $2 * T$ and so on. From the figures, we see that in the packet-at-a-time case, all children get the block at almost the same time while in the block-at-a-time case, some of the children get the block much earlier. It can be shown that the average finish time in the latter case is lower.

To the best of our knowledge, there has been no work examining the effect of this link sharing on the data delivery of application-layer multicast trees.[1] In this work, we analyze the effect of this link sharing and we demonstrate a simple technique, called Time Division Streaming, to exploit this sharing to reduce the average time to transfer data. We provide analysis of TDS using a simple network model. We then show how the construction of multicast trees can take advantage of TDS and propose heuristics for tree construction.Our results show that sending larger blocks of data in multicast trees constructed using our heuristic can provide a substantial improvement in the average finish time of the nodes at the expense of some increase in the maximum finish times of some nodes. We examine the tradeoff in our evaluation of TDS.

The rest of the paper is organized as follows: In Section 2, we describe the link sharing in detail. In Section 3, we develop the relation to compute the latency to any host in an end-system multicast tree. We then present algorithms and heuristics to create TDS trees in Section 4. We evaluate these TDS trees in Section 5 and discuss related work in Section 6. We conclude with some discussion of the results in Section 7.

2 Time Division Streaming

We present our work in the context of a data delivery application such as an audio or video stream that requires a minimum data rate or bandwidth of b bits per second (bps). We note that this data rate can be achieved either by small packets delivered at regular intervals or by sending a larger block of data, such as a block containing a few seconds of content, at the beginning of a time period such that the effective data rate over the time period is greater than or equal to b. Thus, in our discussion, we use the term "block" of data to imply the transfer of one or more back-to-back packets of data between nodes in the multicast tree.

We now present a scheme to exploit the serialization of packets at the access link. We call this mode of transfer of data Time Division Streaming or TDS. The idea behind this mode of data transfer is essentially to use the available upload bandwidth of a

[1] The authors in [3] point out that the single access link imposes the constraint that the data intended for children of this node must be serialized at the link though they do not consider the problem of scheduling the access to the link.

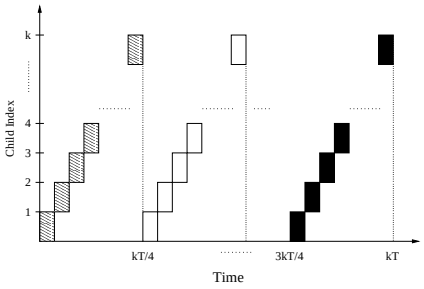

Fig. 1. A block comprising of four packets being sent to k children a packet at a time

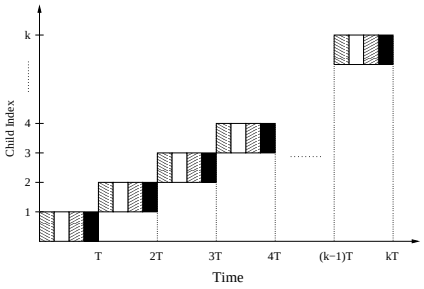

Fig. 2. A block comprising of four packets being sent to k children a block at a time

node in a manner similar to Time Division Multiplexing (TDM) of a communication channel, hence the name. In this mode, a node will send a block of data, composed of several packets, to only one of its children, utilizing all of its upload bandwidth for that transfer[2]. Once the transfer is complete, the node sends the block to the next child in order and so on.

Effect of TDS on a single node To illustrate the effect of link scheduling, we revisit our initial example shown in Figures 1 and 2. The figures show a scenario in which we do not consider the propagation delay to the nodes. Figure 1 illustrates the case where the block of four packets is sent as a packet to each child in turn. The average time to finish receiving the entire block for the children is $3kT/4 + (T/4)(k+1)/2$. Figure 2 shows the case in which the entire block is sent to each child. The average finish time in this case is $T(k+1)/2$. The gain in the average finish time for this works out to $(k-1)3T/8$ which is greater than zero for more than one child, i.e., $k > 1$. More generally, if we let n be the number of packets in a block, s be the size of a packet in bits and B be the uplink bandwidth at the node r, the average finish time, while sending a packet to each child in turn, is $(n-1)ks/B + (s/B)(k+1)/2$ whereas it is $(ns/B)(k+1)/2$ when sending a block at a time. The gain in average finish time over all the nodes is $(s/B)(n-1)(k-1)/2$. It is simple to conclude from this analysis that larger blocks are better at reducing the average finish time.

If we incorporate the propagation delay of the links connecting the nodes, the analysis remains unaffected as the propagation delay remains unchanged in both cases with only the transmission delay changing due to the use of larger blocks.

Effect of TDS in a multicast tree To evaluate the effect of using TDS in a multicast tree, consider an interior node r in the tree. If the node has k children, each the root of a subtree, and the children are numbered from 1 to k, consider the finish time of the first child. If this child is the first to receive a block from r, its finish time is ns/B when n packets are sent as a block as opposed to $(n-1)ks/B + s/B$ when a single packet is sent to each child in turn. Irrespective of the scheduling in any other node in

[2] For this work, we assume that the transport protocol used by the application allows for blocks of data to be sent in packets that are back-to-back on the connection.

this subtree, the finish times of the nodes in this subtree are reduced by this factor of $(n-1)(k-1)s/B$. The finish time of the last child child remain unaffected by this and hence, the nodes in its subtree remain unaffected.

In this technique, we are delaying the beginning of transmission to the children that come later in the TDS order to finish transmission of data to the children earlier in the TDS order much sooner than otherwise. This observation shows that if the subtrees of node r are all of equal size, the nodes in child 1's subtree would finish much earlier than nodes in child k's subtree. This can be mitigated if the subtrees of the child nodes were distributed unequally, i.e., if the subtree of the first child were larger than that of the kth child. We build on this intuition in Section 4 where we propose tree construction algorithms that take TDS into account.

3 Analysis of TDS

Until now we have been considering the effect of using TDS to deliver data from a node to its children. We now develop a relation to calculate the finish time of an arbitrary packet in a data stream at any node in a TDS tree. For our model, we assume that the data to be transfered is composed of packets of size s. Several packets are aggregated to form blocks. We denote the number of packets in a block by n. Let m be the packet index in a stream of data that is being transfered to the clients.

The end hosts that are the targeted by application-layer multicast trees have diverse access links connecting them to the Internet [14], varying from dial-up links and cable and DSL modems to Ethernet. A characteristic of most of these types of access links is that they offer much larger download bandwidth to the node than upload bandwidth from the node (a factor of 10 with some ISPs using cable modems). Let the upload bandwidth available to each node a participating in the application-layer multicast tree be denoted by B_a. The bandwidth requirements of the application creates an upper bound on the number of children that a node can support. This upper bound can be given in terms of the number of children that a node a can support in an application-layer multicast tree and is given by $d(a) = \lfloor B_a/b \rfloor$. We make the assumption that in the tree, nodes with higher upload bandwidth are closer to the source, specifically for a node a with k children $c_0, c_1, \ldots, c_{k-1}$, $B_a \geq max(B_{c_0}, \ldots, B_{c_{k-1}})$. This assumption is reasonable as it has been shown in [7] that to obtain short trees with minimum propagation delay, the nodes with largest degrees should be closest to the source. Given our interest in improving the average latency to transfer data to the clients, any tree considered would have this property.

We begin by considering the simple case with a single source node r with k children. If $0 \leq m < n$ and node i is the j^{th} child of the source, the finish time $t_{i,m}$ of packet m at node i can be written as $t_{i,m} = nj(s/B_r) + (m+1)(s/B_r)$ where the first term represents the time to send the block to the j children before i and the second term represents the time to send the m packets to node i. In general, for $m > 0$, the finish time a packet can be broken up into three parts,
– time to transmit the packets of all previous blocks,
– time to transmit current block to the $(j-1)$ children preceding i,
– time required to transmit the packets of the current block to child i.

Symbol	Definition
B_a	Upload bandwidth of node a
n	Number of packets in a block
s	Size of a packet in bits
b	Bandwidth required by data stream
m	Packet index in a stream of packets
i	Index of node in a global numbering scheme
$p(i)$	Parent index of node i
$c(i)$	Number of child nodes of node i
$I(i)$	Index of node i in its parent's TDS schedule
$d(i)$	Maximum degree bound of node i

Table 1. Summary of Notation used

This can be represented by $t_{i,m} = (m/n)k(s/B_r) + nj(s/B_r) + (m+1)(s/B_r)$.

For a general application-layer multicast tree, we begin by making the observation that once the first packet of a block arrives at a node, the subsequent packets of that block arrive back-to-back. From our previous assumption about the upload bandwidth of a node being greater than or equal to that of its children, we know that the time to transfer a block to a node is less than or equal to the time that node takes to transfer the block to a child. In other words, once a node begins receiving a block from its parent, it can retransmit the block to its child without waiting for a packet to arrive. Therefore, the time a packet arrives at a node depends on the packet's arrival at the node's parent. We assume that once a node receives the first packet of a block, it immediately begins transmitting the packet to its children. We ignore the propagation delay of links in this analysis to make the exposition clearer but incorporating the delays is straightforward.

Let η represent the block index, i.e., the integer value m/n and let η_1 be the first packet of block η. Let the function $p(i)$ denote the parent of node i and the function $I(i)$, the index of i in its parent's TDS schedule. The time of arrival of a packet m at a node i can be computed as follows:

$$t_{i,m} = t_{p(i),\eta_1} + I(i)n(s/B_{p(i)}) + (m \bmod n + 1)(s/B_{p(i)}).$$

From the above relations, we note that the latency to any node is dependent on the size of the packet and the number of packets in a block. We evaluate the effect of these factors in Section 5.

4 Tree Construction

We now consider the problem of constructing trees that take into account TDS. Current algorithms for constructing application-layer multicast trees optimize for delay or bandwidth without considering the transmission delay at interior nodes. We wish to create trees that not only take into account the transmission delay but also optimize for the block size being used in TDS. We begin by stating the objective for our tree construction algorithm.

The optimization problem can be stated as follows: Let $G = (r, N, E)$ be a complete graph with a source node r and end hosts N. Let the degree constraints of the nodes be given by the function d. Let E be the set of edges between the nodes. Our objective is to find the tree T with minimum average finish time to transfer the block

```
Tree T = createTree(Nodes N, Source s, DegreeConstraints d,
                    Blocksize B)
  Sort the nodes of N in non-increasing order of degree
    constraints into n_1,n_2,...,n_|N|.
  T = {s}
  Compute t(s) as the time to transmit B to first available
    child of s.
  Insert s into MinHeap with value t(s).
  i = 1
  while i <= |N|
    Get next node a from MinHeap.
    Attach n_i as child of a.
    T = T ⋃ {n_i}
    if c(a) < d(a)
      Recompute t(a) and insert a into MinHeap with
        value t(a).
    if d(n_i) > 0
      Compute t(n_i) and insert n_i into MinHeap with
        value t(n_i).
    i++
  done
  return T
```

Fig. 3. Tree Construction Algorithm for TDS ignoring propagation delays

B and satisfies the degree constraints. We first consider the case where the end-to-end delay between any pair of nodes to be the same (in this case zero), i.e., we ignore the propagation delay but not the transmission delay caused by the link scheduling. We present a centralized algorithm in Figure 3 that constructs an optimal tree for this case. The algorithm is run by a designated node such as the source in the following manner: First, the nodes are sorted in non-increasing order of their degree constraints. In the main loop, the next node from the sorted list is selected and attached to the tree at the position with the minimum finish time until all nodes are attached. If no attachment points exist due to the degree constraints of the nodes, the tree returned is empty.

Theorem 1. *The algorithm createTree generates a tree such that the nodes have the minimum finish times given the degree constraints d.*

The proof relies on the following lemmas.

Lemma 1. *For any node in the optimal tree, the size of its childrens' subtrees are in the order in which data is sent to the children, i.e., if data is sent to child i before child j, the size of i's subtree is greater than or equal to j's subtree.*

Lemma 2. *For any node in the optimal tree, the child with the larger degree bound is sent data before a child with a lesser degree bound.*

A corollary to lemma 2 is that for any node, the child with the largest degree bound is the first to which data is sent.

Lemma 3. *Given a set of N nodes with degree constraints, the optimal tree has the node with the maximum degree constraint as the root.*

The sketch of the proof of lemma 3 is as follows: We assume, for contradiction, that the optimal tree does not have the node with the maximum degree constraint at the root. It follows that for some subtree in the optimal tree, the node a with the maximum degree constraint is the child of a node with a smaller degree constraint. By lemma 2, a is the first child to be sent data by its parent. We show that exchanging a with its parent and rearranging the subtrees of a and the subtrees of the parent such that the larger subtrees are attached to a leads to a tree with a lower finish time, violating our assumption that this tree was optimal.

The proof of lemma 2 is very similar. The complete proofs can be found in [13].

Proof (Theorem 1). We prove this by induction on i, the number of nodes attached to the tree. If $i = 1$, then there is only one node in the tree, the source s and it is clearly optimal. Assume that the algorithm creates an optimal tree for i nodes. By the algorithm, the degree constraint of the $i + 1$st node is less than or equal to the degree constraint of any node in the tree uptil now. By lemma 3, node $i + 1$ can only be attached as a leaf, and the position that minimizes the finish time of $i+1$ is the position with the minimum transmission delay from the source to $i + 1$ which is node a from the algorithm. Thus, the tree with $i + 1$ nodes is also optimal. □

The general problem of constructing optimal trees with non-uniform propagation delays between nodes has been shown to be NP-Hard in [2]. To handle tree construction for TDS taking propagation delays into account, we propose the following *TDS heuristic* that attempts to balance the degree bound of a node and its propagation delay to the tree. The heuristic iteratively adds nodes to the tree in the following manner: Initially, three sets of nodes are created, N_A consisting of nodes that are attached to the tree, N_{Av} consisting of nodes that are attached and can accept more children and N_U consisting of nodes that are unattached. Let $l(u)$ be the latency of node u to the source along the tree. In the beginning, N_A and N_{Av} contain only the source node and N_U contains all other nodes. At each iteration, the algorithm computes a cost for each $v \in N_U$ as

$$Cost(v) = \min_{u \in N_{Av}} \alpha l(u)/l_{max} + \beta d(v)/d_{max} + (1 - \beta)\delta(u, v)/\delta_{max}$$

where $l_{max} = max_{u \in N_{Av}}\, l(u)$, $d_{max} = max_{u \in N_U}\, d(u)$ and $\delta_{max} = max_{u \in N_U, w \in N_{Av}}\, \delta(u, w)$ are normalization constants.

The variables α and β control the weight of the different factors in the computing the cost of each unattached node. The value of α controls the extent to which the latency in the tree to the attachment point affects the cost, while β determines the relative importance of the degree constraints and the propagation delay of the unattached nodes. From our evaluation, we observed that the heuristic is relatively unaffected by the value of α and so we fixed the value of α to be 1. We explore the effect of the β parameter on the average latency in the next section.

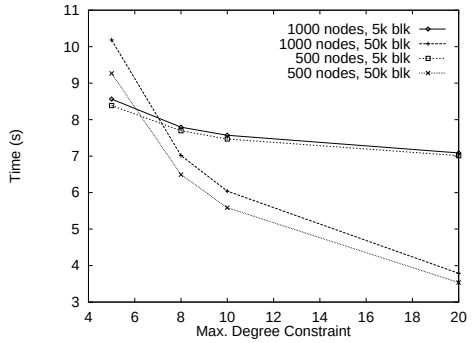

Fig. 4. Varying the maximum degree constraint of the nodes participating in the multicast tree

5 Evaluation

Methodology We used libraries provided by the *p-sim* simulator [8] to write our simulation. For our simulations, we begin by creating a representative Internet topology using GT-ITM [4] comprising of 4050 nodes. We then randomly choose some of the nodes to be the hosts participating in the application-layer multicast tree. We randomly select one of the nodes to be the source of the end-system multicast. The degree constraints for the nodes are assigned from a uniform distribution with the source node being assigned the maximum degree constraint. The link latencies are drawn from uniform distributions with [50ms, 200ms] for the transit links, [25ms, 100ms] for the transit-stub links and [5ms, 50ms] for the stub-stub links. The stream bandwidth requirements are set at 8kBps per child in the multicast tree. We construct the application-layer TDS multicast tree using the algorithm detailed above. For all the experiments we report the average finish time as the time to transfer 50 kB of data from the source to all the nodes in the tree. We chose block sizes of 50kB and 5kB as reasonable bounds on the size of an application's data unit. The packet size used is usually 500 bytes. We also experimented with 1500 byte packets but the results were similar with the 1500 byte packets having a slightly larger average finish time. We begin by investigating the different parameters that affect the TDS scheme.

Effect of TDS parameters In Figure 4, we plot the average[3] finish time of the nodes in the TDS tree on the y-axis against the maximum degree constraints allowed for the nodes on the x-axis. Each line represents different size trees with varying block sizes. From the graph, we see that for smaller degree constraints, the smaller block sizes are better for TDS. The small degree constraint results in trees that are tall and narrow, resulting in poor performance of TDS as the difference between the finish times of the first and last child at a node are not significant enough to offset the longer transmission delays. As the maximum degree constraint is increased, the trees created are wider

[3] In all cases, the median was less than the average. We omit plotting the medians for clarity.

Max. Degree Constraint	Block size (kB)	Percentage of nodes in first two subtrees	Depth
10	50	66	8
	5	39	5
8	50	66	8
	5	39	6
5	50	75	9
	5	67	7

Table 2. Node distribution of TDS trees for different block sizes.

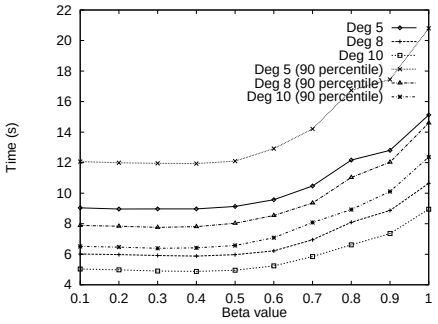

Fig. 5. Varying β with block size of 50kB

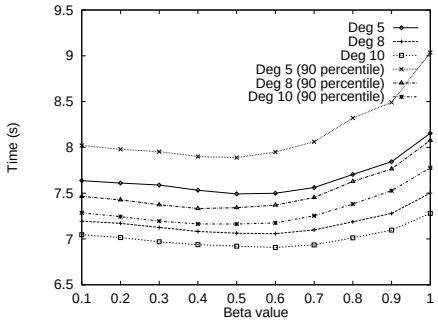

Fig. 6. Varying β with block size of 5kB

and the larger block size has significantly better performance. The trees for the larger block size are more unbalanced with the subtrees of the children that are earlier in the TDS order being much larger that the subtrees of those later in the TDS order. This can be seen in the table in Table 2 in which we show the size of the subtrees in terms of the percentage of the total nodes that the subtree contains. Although the degree bounds for the nodes are assigned from a uniform distribution, the distribution of the degrees of nodes in the final tree is similar to the distribution of degrees observed by Sripanidkulchai et al [14].

Effect of β parameter on the TDS heuristic In Figure 5, we plot the average finish time of the nodes in the TDS tree on the y-axis against various values of β on the x-axis. Each line represents trees constructed with a particular maximum degree constraint and each point is the average of five runs of the simulator with different seeds. We observe that the average finish time is only marginally affected for values of β up to 0.5. Actually the average finish time is reducing in this interval, but as the value of β increases above 0.5, the average finish time increases quickly. This is also seen in Figure 6, in which the curves are plotted for a block size of 5kB. These plots show that selecting the nodes primarily on the basis of the propagation delay to construct TDS trees results in poor performance. The best balance seems to be to equally weight the degree constraints of the nodes and their propagation delay when considering the next node to add to the tree.

We also plot the 90^{th} percentile value of node finish times for each of the degree constraints. We observe that the 90^{th} percentile value of the 50kB block with a degree constraint of 10 has a smaller finish time than the average finish time using 5kB blocks. This indicates that most of the nodes benefit when we use larger blocks. Another ob-

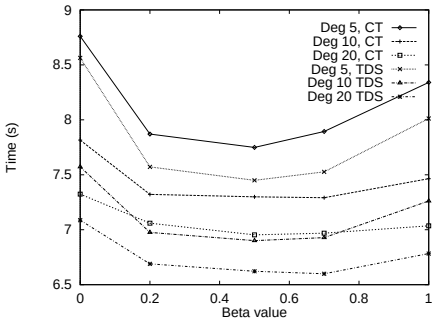

Fig. 7. Varying β with block size of 5kB using CT and TDS heuristics for 1000 nodes

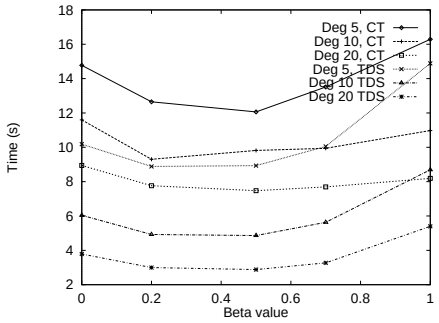

Fig. 8. Varying β with block size of 50kB using CT and TDS heuristics for 1000 nodes

servation we can make from the graph is that the 90^{th} percentile value is closer to the average finish time for the 5kB block size than for the 50kB block size, indicating the increased variance due to the larger block size.

Planet-Lab experiments To evaluate the effects of TDS in a real-world scenario, we developed a small application to run on the PlanetLab network [11]. In a limited experiment using 16 nodes, block sizes of 50kB and 5kB, a packet size of 1000 bytes, we observed that the average finish time of the 50kB block size was 6.32 seconds while for the 5kB block was 8.41 seconds which agrees well with our analysis.

Performance relative to existing heuristics There have been other heuristics proposed to construct degree-bounded trees for application-layer multicast. The heuristics proposed are for minimizing the maximum latency to clients [12] (which we call Compact Tree) and for minimizing the cost of using proxies [7] (which we call Min Cost). The Compact Tree heuristic incrementally constructs a minimum spanning tree from the source s. For each node v not in the tree, it finds the minimum cost edge (u, v) from a node u in the MST. The cost that is minimized is the overlay delay $\delta(s, v)$ from the source to the node v. The Min Cost heuristic is quite similar except for the cost function used to select the next node. The Min Cost heuristic considers the minimum latency edge (u, v) as well as the degree constraint d of the nodes while selecting the best node to attach to the tree. The cost function is $\gamma_\alpha(v) = \alpha d(v)/d_{max} + (1-\alpha)\delta_{min}/\delta(s, v)$. The α parameter plays the same role as the β parameter in the TDS heuristic and d_{max} and δ_{min} are normalization constants. Both heuristics do not consider the cost of transmission of data while constructing the trees.

For our simulations, we implement both heuristics using the same routine. The value of the parameter $\beta = 0$ creates trees based on the Compact Tree heuristic while other values of β create trees based on the Min Cost heuristic. We plot the average finish times for delivering 50kB of data using trees with 1000 nodes constructed by the different heuristics in figures 7 and 8 for block sizes of 5kB and 50kB respectively. In general, the graphs show that the TDS heuristic performs much better for every degree bound that is used as it considers the transmission delays incurred at each node. The magnitude

of the improvement of the TDS heuristic is larger with block sizes of 50kB than with 5kB. The trees created by the TDS heuristic for the 5kB blocks are not very different from the trees created by the other heuristics and so the improvement seen is on the order of a second in the average finish times. On the other hand, the trees for the 50kB block size created by the TDS heuristic exploit the larger block size to create trees that are very different from those created by the other heuristics resulting in significant improvement over the other heuristics. When β is in the range of 0.5 to 0.7, the Min Cost and Compact Tree heuristics perform best as they consider a combination of the degree constraints along with the propagation delay to the nodes in the tree.

6 Related Work

In [12], the authors describe the problem of creating minimum diameter degree bounded spanning trees and show that it is NP-Hard. They propose a greedy heuristic to create trees based on this objective. In [7], the authors define the cost of a tree as the number of special proxy nodes used to create multicast trees. Using this they propose to create trees which satisfy a maximum delay bound while minimizing cost. They provide an optimal solution for graphs with uniform edges and show that this problem is NP-Hard in the general case with non-uniform edges.

In [2], the minimum average-latency degree-bounded directed spanning tree problem is introduced in context of a two-tier infrastructure for implementing large-scale media-streaming applications. The infrastructure, called OMNI (Overlay Multicast Network Infrastructure) consists of a set of Multicast Service Nodes (MSNs) to which end-hosts connect to form the multicast tree. The objective of this work is to reduce the average latency to the end-hosts. This is achieved by arranging the MSNs to create minimum latency trees where each MSN is weighted by the number of clients connected to it. The authors impose a degree bound on each MSN but do not account for the transmission delays at the MSNs which we consider in this work. Also, we focus on application-layer multicast trees without any explicit infrastructure in the network.

In [3], the authors point out that the models used currently to construct these trees neglect to consider the fact that the most nodes in an end-system multicast tree have a single network connection and this connection has to be shared between all the children of the node. The authors propose an overlay network model to account for these costs and propose heuristic algorithms to construct multicast trees that consider the transmission and computation delays at each of the nodes in the multicast tree. The overlay model proposed does not explicitly consider the degree constraints at nodes. The construction of the tree is based on minimizing the delay to hosts but does not consider the effect of the degree constraints imposed by the access link bandwidth of the nodes or the effect of the access link scheduling on the average delay of the nodes in the tree.

7 Discussion and Future Work

Our results show that nodes sending large blocks of data to each child in turn can reduce the average finish time of nodes in the multicast tree. The tradeoff involved in this gain is the increased variance of the actual finish times of nodes. Based on this tradeoff, the

block size and packet size for TDS can be specified to match application requirements. For example, the increased variance with larger block sizes can be used as an incentive mechanism to encourage nodes to dedicate more uplink bandwidth to the application. This in turn would place those nodes in positions where their finish times are earlier.

In this work, we examine the effect of the single access link that many end hosts that participate in application-layer multicast have. We show that the average finish times of nodes in the tree are affected by the way in which this link is used to transfer data to a node's children. We proposed a technique called Time Division Streaming to share this access link such that the average finish times are reduced as compared to previous work. We also provide analytical results based on a limited model of this technique and propose heuristics that take this serialization into account when constructing the tree.

Using the TDS heuristic to construct multicast trees, we show significant reduction in the average finish times of nodes. The heuristic exploits the effect of TDS by creating trees such that the interior nodes have unequal subtrees with the subtrees of children earlier in the TDS schedule being larger. In future work, we plan to create distributed version of the TDS heuristic that can be used to add nodes to trees as they arrive. Another direction of work is to vary the TDS parameters to tailor them for specific applications and to study the effect of these customizations.

References

1. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM*, Aug 2002.
2. S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Infocom*, April 2003.
3. E. Brosh and Y. Shavitt. Approximation and heuristic algorithms for minimum-delay application layer multicast trees. In *Infocom*, March 2004.
4. K. Calvert, E. Zegura, and S. Bhattacharjee. How to model an internetwork. In *Infocomm*, 1996.
5. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *SOSP*, Oct 2003.
6. Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. In *IEEE Journal on Selected Areas in Communication*, Aug 2001.
7. N. M. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A graph theoretic approach to bounding delay in proxy-assisted, end-system multicast. In *IWQoS*, May 2002.
8. S. Merugu, S. Srinivasan, and E. Zegura. p-sim: A simulator for peer-to-peer networks. In *MASCOTS*, October 2003.
9. V. Padmanabhan, H. Wang, and P. Chou. Resilient peer-to-peer streaming. In *ICNP*, 2003.
10. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. In *USITS*, Mar 2001.
11. PlanetLab. http://www.planet-lab.org/, 2004.
12. S. Shi, J. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay network. In *NOSSDAV*, Jun 2001.
13. S. Srinivasan and E. Zegura. Scheduling uplink bandwidth in application-layer multicast trees. Technical Report GIT-CC-04-14, Georgia Tech, 2004.
14. K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *SIGCOMM*, Aug 2004.