QoS-Based Multicast Routing for Distributing Layered Video to Heterogeneous Receivers in Rate-based Networks

Bin Wang and Jennifer C.Hou
Department of Electrical Engineering
The Ohio State University
{*bwang,jhou*}@*ee.eng.ohio-state.edu*

Abstract

We design an effective multicast routing algorithm for distributing layered multicast video to heterogeneous receivers in networks with rate-based link schedulers. The multicast tree constructed by the algorithm fulfills the QoS requirements imposed by heterogeneous receivers both in terms of bandwidth and delay, and at the same time, consumes as little network resource as possible. For the purpose of scalability, the proposed multicast routing algorithm is decentralized and can adapt to network and membership change. To facilitate construction of such a multicast tree in a distributed manner, we equip each node with an auxiliary routing table, and propose procedures for updating/maintaining the auxiliary routing table. Simulation results indicate that the multicast tree constructed by the proposed algorithm is more resource efficient than those constructed by other existing algorithms, especially when there exist multiple alternative paths to multicast receivers in the network.

1 Introduction

Multicast services have been increasingly used by various continuous media applications. For example, the multicast backbone (Mbone) of the Internet has been used to transport real time audio and video for news, entertainment, and distance learning. Instead of sending a separate copy of the data to each individual group member, a multicast source sends a single copy to all the members. An underlying *multicast routing* algorithm determines, with respect to certain optimization objective, a *multicast tree* connecting the source(s) and the group members. Data generated by the source flows through the multicast tree, traversing each tree edge exactly once. As a result, multicast is more resource efficient, and is well suited for applications such as video distribution.

The main intent of this paper is to design an effective multicast algorithm for distributing layered video to heterogeneous receivers. By effective, we mean that a multicast session should be routed on a multicast tree that consumes as little resource as possible, while satisfying the quality of service (QoS) imposed by heterogeneous receivers. Moreover, the multicast algorithm should adapt to network and membership change.

In the context of video distribution, video quality perceived depends on the bandwidth/buffer capabilities of receivers. For example, users on a local area network with high bandwidth connectivity would prefer to receive high-rate and high-quality video. On the other hand, the fact that high bandwidth access is more expensive may prevent users from choosing to receive high quality video. To accommodate different users' requirements, a multicast source may vary the video transmission rate by using rate-adaptive video [3] or combining a layered compression algorithm with a layered transmission scheme [11, 15]. In this paper, we adopt the latter approach. That is, a signal is encoded into a number of layers that can be incrementally combined to provide progressive refinement. Users with high bandwidth access may receive higher quality video by subscribing to more layers while low bandwidth users can still get a degraded video quality by receiving the basic video layer.

Several researchers have considered different issues in distributing layered video using multicast [7, 8, 11, 12]. They focus mainly on grouping and assigning different video layers to the links of a multicast tree so as to maximize a predefined source gain [11, 12], designing an adaptive congestion control mechanism [8], exploring retransmission techniques [4], or constructing a multicast routing tree that minimizes the total network resources used [7]. What is not addressed is perhaps the issue of constructing a multicast routing tree such that the QoS requirements imposed by heterogeneous receivers both in terms of bandwidth and delay are satisfied, while the amount of total network resources used is minimized.

We consider a network that uses rate-based link scheduling algorithms, such as those proposed in [2, 10, 14], to provide guaranteed QoS. We propose a distributed multicast routing algorithm that explicitly takes consideration of the heterogeneous QoS requirements imposed by receivers as well as their physical constraints, and at the same time, attempts to minimize the total amount of network resources used. For the purpose of scalability, the proposed algorithm is distributed and makes use of the routing information provided by the underlying routing protocol (e.g., OSPF). We also propose procedures for handling dynamic multicast membership changes.

The rest of the paper is organized as follows. We present the background materials in Section 2. We then formulate the multicast routing problem in Section 3. Following that, we elaborate on the proposed distributed algorithm in Section 4. We evaluate the proposed algorithm in Section 5, and summarize related work in Section 6. The paper concludes with Section 7.

2 Background

2.1 Network Model

We consider packet-switched networks in which each link uses a rate-based packet scheduling algorithm to schedule packet transmission. A network is represented by a weighted digraph $G = (V, E)$, where V denotes the set of network nodes and E the set of communication links that connect the nodes. $|V|$ and $|E|$ denote the number of nodes and links in the net-

The work reported in this paper was supported in part by NSF under Grant No. NCR-9625064.

work, respectively. We consider digraphs in which there exists at most one arc between a pair of ordered nodes. Associated with each link are parameters that describe the current status of the link, for example, the average link delay, the link cost, and the bandwidth available on the link. We term these parameters *link state*. Similarly, associated with each node are parameters representing the current status of the node, for example, the available buffer. We term these parameters *node state*. In particular, we define

- A *link-bandwidth* function $b : E \rightarrow R^+$ which assigns a non-negative weight to each link in the network. The value of $b(\ell)$ is a measure of the bandwidth available on link ℓ.
- The constant link delay, d_ℓ, of link ℓ: which depends on the capacity, and the propagation delay, of link ℓ, as well as the maximum packet size of the connection. d_ℓ also takes into account of approximation that results from the practical implementation of an ideal link scheduling algorithm and the non-preemptive effect of packet transmission.

We term as the *global state* the collection of the local node/link state maintained by all the nodes in the network. Every node in the network maintains the global state using either the distance vector protocol (e.g., RIP) or the link state protocol (e.g., OSPF). The cost metric is assumed to be additive.

Let $M \subset V$ be a set of nodes involved in a group communication session. We call the set M a multicast group with each node $v \in M$ as a group member. Packets originating from node v are delivered to a set of receiver nodes $M - \{v\}$. A multicast tree G_T is a subgraph of G that spans all the nodes in M. G_T may include relay nodes, i.e., non-group member nodes along a path in the tree.

We use a *source-receiver delay bound* D to quantify the level of QoS required by a receiver. A *source-receiver delay bound* D represents an upper bound on the acceptable end-to-end delay along any path from a source node v_s to a receiver node $v_d \in M - \{v\}$. This parameter specifies that packets transmitted from node v_s at time t must be received by a receiver node v_d by time $t + D$; otherwise, the late packets are depreciated, and may simply be dropped.

2.2 Rate-based Scheduling Algorithms

There have been a myriad of link scheduling algorithms proposed in recent years, among which rate-based fair queuing scheduling algorithms, e.g., generalized processor sharing [10], worst-case-fair weighted fair queuing [2], leap forward virtual clock [14], have perhaps received the most attention. A rate-based scheduler links the worst case delay of a connection to its requested rate and generally requires a connection be constrained by a traffic model, e.g., a leaky bucket (R, σ). Specifically, let P be denoted as a path traversed by a connection, $|P|$ the number of links on P, and $R(P)$ the maximum bandwidth available along the path P, i.e., $R(P) = \min_{\ell \in P} b(\ell)$. Then, when a connection is routed over path P with the reserved bandwidth $r \leq R(P)$, the end-to-end delay bound, $D(r, P)$, for the connection is [10]:

$$D(r,P) = \frac{\sigma + |P| \cdot c}{r} + \sum_{\ell \in P} d_\ell, \qquad (2.1)$$

where c is the maximum packet size of the connection, and d_ℓ is the constant link delay.

2.3 Traffic Model for Layered Video

A simple solution to distributing video to heterogeneous receivers is to distribute a uniform representation of the signal to all interested receivers. However, this naive approach could lead to low-capacity regions of the network suffer from congestion and some receivers' QoS cannot be satisfied even though network resources exist to provide better QoS for these receivers. An alternative approach is to combine a layered compression algorithm with a layered transmission scheme [11, 15]. Several layered coding techniques are summarized and analyzed in terms of their suitability for multicast transmission in [11]. In these approaches, videos are encoded into a number of layers that can be incrementally combined to provide progressive refinement. Lower layers encode coarse video information while high layers encode details. A video stream with increasing number of layers has better visual quality and at the same time requires more network bandwidth to transmit as well as faster receivers to decode and display. Only layers that a given link can manage are selectively forwarded.

We adopt this layered video approach in the paper. Each layer of an encoded video signal is constrained by a leaky bucket (R, σ), where R is the bandwidth requirement and σ is the maximum burst size. A video signal is then encoded into m layers, each of which is characterized by a two-tuple $(R_i, \sigma_i), 1 \leq i \leq m$. Due to the additive property of the leaky bucket model, it can be shown that $R^k = \sum_{j=1}^{k} R_j$ and $\sigma^k = \sum_{j=1}^{k} \sigma_j$ are, respectively, the aggregated (up to layer k) bandwidth requirement and burst size of the video.

3 Problem Formulation

We consider the problem of distributing layered video via multicast in a network that uses rate-based link schedulers. We assume that videos are encoded into different layers and are distributed from a sender to a set of receivers with heterogeneous receiving capabilities and heterogeneous delay requirements. Specifically, a one-to-many multicast video distribution session is characterized by the following variables:

- s is the source.
- $\mathbf{d} = \{j | 1 \leq j \leq n\}$ is the set of receivers.
- $\{D_j | 1 \leq j \leq n\}$ is the set of delay requirements, where D_j is the delay requirement of receiver d_j.
- A video is encoded into m layers, and characterized by a set of leaky bucket parameters (R_1, σ_1), (R_2, σ_2), ..., (R_m, σ_m). Let $R^k = \sum_{j=1}^{k} R_j$ and $\sigma^k = \sum_{j=1}^{k} \sigma_j$ be, respectively, the aggregated (up to layer k) rate and burst size of the video.
- $\{R_j^r | 1 \leq j \leq n\}$ is the set of maximum rates at which receivers in $\mathbf{d}$ can receive, where R_j^r is the maximum rate at which receiver d_j can receive. If $R^k \leq R_j^r < R^{k+1}$, then receiver j is capable of receiving up to layer k of the video. We term such a receiver a layer-k receiver.
- c is the maximum size of packets sent by the source s.

Problem 1: Consider a network $G = (V, E)$. For each link $\ell \in E$, let $b(\ell)$ be the maximum available bandwidth on link ℓ and d_ℓ be the constant delay. A multicast session is characterized by the source s, the set of receivers $\mathbf{d}$, the delay requirements $\{(D_j | 1 \leq j \leq n\}$, and the maximum receiving capabilities $\{R_j^r | 1 \leq j \leq n\}$, of the receivers, the maximum

R^r	r	k	cost	OnTreeNode	tag	path

Figure 1: Fields defined in $RECEIVER[i]$.

packet size c, and the video traffic $\{(R_i, \sigma_i), 1 \leq i \leq m\}$. The problem is to find a multicast tree T rooted at the source s and spanning the set of receivers $\mathbf{d}$ such that (i) the delay requirements of all receivers are met; (ii) each receiver obtains as high visual quality as possible; and (iii) the total bandwidth used by the session is minimized, subject to the constraints of bandwidth availability on physical links and the receivers' maximum receiving capabilities. □

4 Proposed Algorithm

4.1 Overview

The tree construction starts with a tree that contains only the source s. Starting from the receiver that requires the highest video quality, the source s (i) calculates (with Eq. (2.1)) for each receiver i the minimum bandwidth, r_i, that must be available on a path to the receiver in order to satisfy the delay requirement and (ii) locates for each receiver i a least hop path, P, with sufficient bandwidth to satisfy (i), if such a path exists. These two steps are facilitated with an auxiliary routing table, T, that is maintained/updated independently of the proposed multicast routing algorithm. The auxiliary routing table records, for every node in the network, an h-hop maximum bandwidth path, where $1 \leq h \leq H$, and H is the maximum possible hop count.[1] Specifically, T is a $|V| \times H$ matrix, the $(u; h)$-th entry of which consists of four fields:

- bw: the maximum bandwidth available on a path of h hops between the current node and receiver node u;
- P: the h-hop path P with available bandwidth bw from the current node to node u;
- *neighbor*: the next hop neighbor on the h-hop path P from the current node to node u;
- $d^h = \sum_{\ell \in P} d_\ell$: the cumulative delay constant on path P.

Each node updates its link state at regular time intervals (with a default value of 30 seconds) and sends it to all its neighbors. The link state advertisement includes, in addition to the topology information, the dynamic residual bandwidth over each link. Upon receipt of a link-state advertisement, a node updates its database and forwards the advertisement to all its neighbors, except the one from which the advertisement was received. Periodically each node invokes a routing table update procedure (to be presented in Section 4.4) to pre-compute h-hop maximum bandwidth paths from a node to all other nodes, based on the collected link state information.

The calculated value of r_i and least-hop path, P, located for receiver i are stored in the ith-entry of a data structure $RECEIVER$. $RECEIVER[i]$ contains the following fields (Fig. 1):

- $RECEIVER[i].OnTreeNode$: the on-tree node at which a tree branch is grafted to receiver i. It is initialized to the source node s.
- $RECEIVER[i].path$: the least hop path, P, from the OnTreeNode to receiver i with sufficient bandwidth selected from the auxiliary routing table.
- $RECEIVER[i].r$: the minimum bandwidth, r_i, available on the path, *path*, in order to satisfy receiver i's delay requirement.
- $RECEIVER[i].cost$: the total bandwidth incurred in adding receiver i to the multicast tree. It is initialized to infinity.
- $RECEIVER[i].R^r$: the maximum rate, R_i^r, at which receiver i can receive.
- $RECEIVER[i].level$: the number of video layers up to which receiver i can receive.
- $RECEIVER[i].tag$: the indicator of whether or not receiver i is on-tree. It is initialized to false.

The source s then composes a *setup* message that carries both the $RECEIVER$ and the cumulative delay from node s, D (which is initialized to 0), and sends the message to the receiver with the smallest cost among receivers that can receive the largest number of layers (let the receiver be receiver j). The message is sent, hop by hop, along the path $RECEIVER[j].path$, until it reaches receiver j.

Upon receipt of a *setup* message, an intermediate node u updates and remembers the parameter D. In addition, node u calculates for each currently off-tree receiver, i, the minimum bandwidth r_i required if a route is grafted from itself to the receiver, and checks if a route with bandwidth $\geq r_i$ exists and has a smaller (total bandwidth) cost than the one currently recorded in $RECEIVER$. This is again facilitated by using Eq. (2.1) and the auxiliary routing table. If so, for each such receiver i, node u updates $RECEIVER[i]$ to reflect that a tree branch to receiver i should be grafted from itself (rather than from source s).

After the *setup* message reaches receiver j, receiver j selects the next receiver to join the multicast tree. The process then repeats until all the receivers are on the multicast tree or no appropriate routes can be located for remaining receivers, whichever occurs first.

4.2 Detailed Description of the Proposed Multicast Routing Algorithm

The pseudo code of the proposed multicast routing algorithm is listed in Fig. 2. First, receivers are grouped into m groups with G_k being the set of receivers that can receive up to k video layers, $1 \leq k \leq m$. Starting with receivers in G_m, the source node s locates for each receiver i the least hop path P with sufficient bandwidth to satisfy the delay requirement of receiver i. Specifically, the delay requirement of receiver i (Eq.(2.1)) enforces

$$D_i \geq \frac{\sigma^k + |P| \cdot c}{r_i} + \sum_{\ell \in P} d_\ell, \quad \text{or} \quad r_i \geq \frac{\sigma^k + |P| \cdot c}{D_i - \sum_{\ell \in P} d_\ell}. \tag{4.1}$$

To determine r_i and P, node s looks up the row $T(i, k), 1 \leq k \leq H$ in T for the path P that has the minimum hop count among all the paths that (i) satisfy Eq. (4.1) (i.e., the *bw* field is larger than or equal to $\max(r_i, R^k)$) and (ii) do not intersect any on-tree node. The latter condition ((ii)) is verified by checking if for every node $v \in P$ (other than the source s) $v \notin RECEIVER[i].path$ for any node i with $RECEIVER[i].tag = true$. This guarantees that the multicast tree constructed will not form a loop.

[1] The value of H can be set implicitly as the diameter of the network, or explicitly as some predetermined value to limit the path length to at most H hops. In the latter case, H should be assigned a value larger than the length of the minimum hop-count path to any node in the network.

If no such path exists or r_i is larger than the maximum receiving capability, R_i^r, of receiver i, the delay requirement of receiver i cannot be met at the current visual quality (i.e., layer k), and receiver i is degraded to be a layer-$(k-1)$ receiver and will be considered with other layer-$(k-1)$ receivers for route establishment. Otherwise ($r_i \leq R_i^r$), if $r_i < R^k$, then the bandwidth reserved r_i is increased to R^k to accommodate the source's transmission rate. Source s fills out the entry $RECEIVER[i]$ to record the route selected for receiver i. Next, node s selects the receiver that consumes the minimum total bandwidth among those that can receive the largest number of layers, i.e., if $G_k \neq \emptyset$ and $G_{k+1} = \ldots = G_m = \emptyset$, then the receiver selected is the receiver that renders min ${}_{i \in G_k}(|P| \cdot r_i)$. (Let the receiver selected be denoted as receiver j.) Node s then marks the *tag* field of $RECEIVER[j]$ as true, and sends a *setup* message to receiver j along the path $RECEIVER[j].path$.

When a *setup* message arrives at an intermediate node u, on the way to receiver j, node u records the cumulative delay on the on-tree path, D, and the number of maximum video layers received, *level*. For each currently off-tree receiver i, node u tests whether or not there exists a (less costly) path P that emanates from node u and has sufficient bandwidth to meet the delay requirement of receiver i. Specifically, suppose receiver i may receive up to k layers, node u finds the first entry, $T(i,h)$ in row $T(i,k), 1 \leq k \leq H$ in T such that (i) $T(i,h).bw \geq \max(r_i, R^k)$, where r_i satisfies

$$D + \frac{\sigma^k + |P| \cdot c}{r_i} + \sum_{\ell \in P} d_\ell \leq D_i; \qquad (4.2)$$

and (ii) P does not intersect any on-tree node. If such a path P is located, node u further checks if $|P| \cdot r_i < RECEIVER[i].cost$. If so, node u replaces the source node to become the on-tree node from which a tree branch, P, is grafted to receiver i, i.e., $RECEIVER[i].r = r_i$, $RECEIVER[i].cost = |P| \cdot r_i$, $RECEIVER[i].OnTreeNode = u$, and $RECEIVER[i]$ $.path = P$.

After the *setup* message reaches receiver j, receiver j selects the next receiver to join the multicast tree as the receiver, say k, with the smallest cost among receivers that can receive the largest number of layers. Receiver j then sends a *fork* message (that contains $RECEIVER$) to the on-tree node, $n \triangleq RECEIVER[k].OnTreeNode$. When node n receives the *fork* message, it sends a *setup* message to receiver k along the path $RECEIVER[k].path$. The procedure repeats as the multicast tree is extended to include receivers one by one, until either all receivers in $\mathbf{d}$ are on the tree or the multicast tree cannot grow further. A *finish* message is then sent to the source s.

4.3 Handling of Dynamic Receiver Join/Leave

A multicast group member may join or leave a multicast session dynamically. It is thus important to ensure that member join/leave will not disrupt the on-going multicast, and the multicast tree after member join/leave still satisfies the delay requirements of all the on-tree receivers and remains near-optimal. If a multicast tree is re-constructed from the source each time a member joins or leaves, on-tree nodes may not switch to the new tree simultaneously and the packets that are originally

```
Procedure QoS_Multicast_Routing()
{
  while receiving a message do
  switch(message.type) {
      case setup: setup(); break;
      case fork: fork(); break;
      case finish: finish(); break;
      case join: join(); break;
      case leave: leave();
  }/* switch */
}
Procedure setup ()
{
  if (self is the source node) { /* source initiates route setup */
      D = 0; initialize RECEIVER;
      for (k = 1; k ≤ m; k++)
          G_k = {i ∈ d | R^k ≤ R_i^r < R^{k+1}};
      for (k = m; k ≥ 1; k−−) {
          foreach (i ∈ G_k) {
              /* Find from row T(i,j), 1 ≤ j ≤ H, the first entry,
              T(i,h) such that T(i,h).bw ≥ max(r_i, R^k) and
              T(i,h).path does not intersect any on-tree node; h = 0
              if no such entry can be found */
              h ← search_path(T, i, D_i, k, &r_i, &P);
              if ((h == 0) || (r_i > R_i^r)) { /* degrade receiver i
              to level k − 1 */
                  if (k − 1 ≠ 0) { G_k ← G_k \ {i}; G_{k−1} ← G_{k−1} ∪ {i}; }
                  continue;
              } /* if */
              else if (r_i ≤ R^k) r_i ← R^k;
              RECEIVER[i].R^r = R_i^r; RECEIVER[i].r = r_i;
              RECEIVER[i].level = k; RECEIVER[i].path = P;
              RECEIVER[i].cost = r_i × |P|;
          } /* foreach */
      } /* for */
      for (k = m; k ≥ 1; k−−) {
          if (G_k ≠ ∅) {
              select j ∈ G_k with the smallest RECEIVER[j].cost;
              RECEIVER[j].tag = true; message.type = setup;
              message.RECEIVER = RECEIVER;
              message.dest = j;
              send message to the first node in RECEIVER[j].path;
              break;
          } /* if */
      } /* for */
  } /* if (self is the source node) */
  else { /* self is not a source node */
          RECEIVER = message.RECEIVER;
          update and record D at the current node;
          foreach (j ∈ G_k such that RECEIVER[j].tag == false,
          k = m downto 1) {
              h ← search_path(T, j, D_j − D, k, &r_j, &P);
              if ((h == 0) || (r_j > R_j^r)) continue;
              else if (r_j ≤ R^k) r_j ← R^k;
              if (r_j × |P| < RECEIVER[j].cost) {
                 RECEIVER[j].cost = r_j × |P|;
                 RECEIVER[j].r_j = r_j;
                 RECEIVER[j].OnTreeNode = current node;
                 RECEIVER[j].path = P;
              } /* if */
          } /* foreach */
  } /* else */
```

Figure 2: The QoS-based multicast routing algorithm for transporting layered video.

```
if (current node ≠ message.dest) { /* keep forwarding */
        message.type = setup;
        message.RECEIVER = RECEIVER;
        extract next_node from RECEIVER[message.dest].
            path;
        forward message to next_node;
} /* if */
else /* current node == message.dest */
    if (∀i ∈ d either (RECEIVER[i].tag == true) or
    ((RECEIVER[i].path == null) &&
      (RECEIVER[i].tag == false)) )
        send finish to s;
    else {
        for (k = m; k ≥ 1; k − −)
            if (G_k ≠ ∅) {
                select j ∈ G_k with RECEIVER[j].tag == false
                  and RECEIVER[j].cost being the smallest;
                send a fork message to RECEIVER[i].
                  OnTreeNode;
                break;
            }
    } /* else */
} /* setup() */
Procedure fork ()
{
  RECEIVER = message.RECEIVER;
  for (k = m; k ≥ 1; k − −)
    if (G_k ≠ ∅) {
        select i ∈ G_k with RECEIVER[i].tag == false and
        RECEIVER[i].cost being the smallest;
        break;
    }
  RECEIVER[i].tag = true; message.dest= i; message.
  RECEIVER = RECEIVER; message.type = setup;
  send message to the first node in RECEIVER[i].path;
}
```

Figure 2: The QoS-based multicast routing algorithm for transporting layered video (Cont'd). Procedures *finish*, *join*, and *leave* are not listed due to their simplicity.

routed on the old multicast tree may be lost and have to be retransmitted. A seamless transition may thus not be achieved. We adopt an approach that incrementally changes the multicast tree in the case of member join/leave. When a receiver leaves a multicast session, if the receiver is a leaf node of a multicast tree, it sends a *leave* message upstream. The *leave* message travels upstream along the on-tree branch until it reaches a fork node (i.e., a node with more than one downstream on-tree interface and/or with receivers on its directly attached subnet). Upon receipt of a *leave* message, an intermediate node releases the network resources. The rest of the multicast tree remains unchanged. On the other hand, if the leaving receiver is not a leaf node, it is not disconnected from the multicast tree. From this point onwards, this node simply relays incoming messages to the outgoing downstream interfaces. The ongoing session will not be affected.

When a receiver intends to join a multicast session, it sends a *join* request to the source of the session with the information of its delay requirements and the maximum receiving capability. The source in turn multicasts a *join* message to the current on-tree receivers. The *join* message contains the cumulative delay D (initialized to 0), the id of the joining receiver, and $RECEIVER[id]$. Upon receipt of a *join* message, an on-tree node updates the value of D. In addition, if the node can locate (using the procedure listed in Fig. 2) a less costly path that emanates from itself to the joining receiver, it updates the *cost*, *OnTreeNode*, *path* fields of $RECEIVER[id]$. When a *join* message reaches a leaf receiver on the tree, the receiver sends $RECEIVER[id]$ back to the source. After receiving the replies from all the leaf nodes, the source then selects a fork node which connects the new receiver least costly to the routing tree. Join of the new receiver then proceeds in the same manner as was discussed in Section 4.2.

4.4 The Auxiliary Routing Table Update Procedure

The routing table update procedure is listed in Fig. 3. The procedure pre-computes h-hop maximum bandwidth paths from the current node to all the other nodes. The procedure iterates H times. At the k-th iteration, the k-hop maximum bandwidth path from the current node to every receiver (if it ever exists) is located and recorded.

The auxiliary routing table T is first initialized with all *bw* fields set to 0, d^h fields set to 0, and *neighbor* and P fields cleared (lines 3–8 in Fig. 3). In the first iteration, the entries in the first column (which correspond to the information of one-hop paths) are modified as follows (lines 9–13 in Fig. 3): if node i ($1 \leq i \leq |V|$) is a neighbor of the current node s, then the *bw* field of $T(i, 1)$ is set to the available bandwidth on link (s, i), the d^h field is set to the d_ℓ value on link (s, i), the *neighbor* field is set to i, and the P field is updated as $\{s\}$.

In the h-th iteration, entry $T(j, h)$ is updated, $1 \leq j \leq |V|$. The procedure inspects each of node j's neighbors, u, that has available bandwidth $b(u, j) \neq 0$ and lies on a path of $(h-1)$ hops from the current node. If node j lies on the path of $(h-1)$ hops from the current node to node u (i.e., $j \in T(u, h-1).P$), then a loop is formed (line 18 in Fig. 3). The procedure then recursively calculates a new $(u; h-1)$-th entry (procedure *find_loop-free_entry()*, line 19 in Fig. 3). This is done by eliminating node j from being considered as part of an $(h-1)$-hop path from node s to node u.

The procedure then proceeds to locate the h-hop maximum bandwidth path from node s to node j whose final hop is link (u, j) (lines 22–25 in Fig. 3). This amounts to taking the minimum of $T(u, h-1).bw$ and the bandwidth, $b(u, j)$, available on link (u, j). If this value is larger than the present value in the *bw* field of entry $T(j, h)$, a better (larger-bandwidth) h-hop path has been located. This process repeats for every neighbor node, u, of node j's, and the best h-hop path ever located is recorded in entry $T(j, h)$ (lines 27–30 in Fig. 3).

We illustrate the auxiliary routing table update procedure by the following example.

Example 1: Consider the network in Fig. 5. The links and their bandwidth values are shown in the figure. Suppose node z is the current node. The auxiliary routing table (maintained by node z) in the 1st – 4th iteration of the proposed algorithm is given in Fig. 5. (For clarity of presentation, both the *neighbor* and d_ℓ fields are not shown in Fig. 5.) In the first iteration (Fig. 5(a)), the links from node z to its adjacent node u and x are recorded as the 1-hop maximum bandwidth path to node u and x, respectively. There is no 1-hop path from node z to node v or node y. In the second iteration (Fig. 5(b)), $T(v, 2).bw = \max\{\min\{T(u, 1).bw, b(u, v)\}, \min\{T(x, 1).bw, b(x, v)\}\} = \max\{\min\{6, 5\}, \min\{7, 3\}\} = 5$, and $T(v, 2).P = T(u, 1).P \cup \{u\} = \{z, u\}$. (Other entries can be similarly updated.)

How entry $T(u, 3)$ is updated in the third iteration warrants some explanation. Node u lies on the 2-hop maximum bandwidth paths from node z to nodes v (node u in the P

```
Procedure aux_routing_info()
{
/* V: the set of vertices labeled as 1 to N */
/* E: the set of links labeled by ordered pairs (u, v) */
/* s: the current node at which the procedure is executed */
/* b(u, v): the available bandwidth on link (u, v) */
/* d_ℓ(u, v): the d_ℓ value on link (u, v) */
/* T: the 2-D (node;hop) auxiliary routing table */
/* T(i, j): the (i; j)-th entry in table T; it has 4 fields bw,
neighbor, P, d^h */
/* H: the maximum hop count */
/* initialization */
  for (i = 1; i ≤ N; i++)
  for (j = 1; j ≤ H; j++) {
       T(i, j).bw = 0; T(i, j).neighbor = ∅;
       T(i, j).P = ∅; T(i, j).d^h = 0;
       if (i == s) T(i, j).bw = ∞;
  } /* for loop */
/* first iteration */
  for (i = 1; i ≤ N; i++)
       if (i is a neighbor of s) {
            T(i, 1).bw = b(s, i); T(i, 1).neighbor = i;
            T(i, 1).P = T(i, 1).P ∪ {s};
            T(i, 1).d^h = T(i, 1).d^h + d_ℓ(s, i);
       } /* if */
/* 2nd to H-th iteration */
 for (i = 2; i ≤ H; i++)
 for (j = 1; j ≤ N; j++) {
       tmp_bw = T(j, i).bw;
       foreach (u such that (T(u, i − 1).bw ≠ 0 && b(u, j) ≠ 0)) {
            if (j ∈ T(u, i − 1).P ) /* a loop is formed */
            /* new_entry is the new (u, i − 1)th entry obtained
                 by eliminating node j from being considered as
                 part of an (i − 1)-hop path to node u */
                 new_entry = find_loop_free_entry(T, i − 1, {j}, u);
            else
                 new_entry = T(u, i − 1);
            if (min{new_entry.bw, b(u, j)} > tmp_bw){
                 tmp_bw = min{new_entry.bw, b(u, j)};
                 tmp_neighbor = new_entry.neighbor;
                 tmp_node = u; tmp_d_ℓ = d_ℓ(u, j);
            } /* if */
       } /* foreach */
       if (tmp_bw ≠ 0) {
            T(j, i).bw = tmp_bw; T(j, i).neighbor = tmp_neighbor;
            T(j, i).P = T(tmp_node, i − 1).P ∪ {tmp_node};
            T(j, i).d^h = T(tmp_node, i − 1).d^h + tmp_d_ℓ;
       } /* if */
       else {
            T(j, i).neighbor = ∅; T(j, i).P = ∅; T(j, i).d^h = 0;
       } /* else */
 } /* for loop */
}
```

Figure 3: The auxiliary routing table update procedure.

```
Procedure find_loop_free_entry(T, h, S, u)
{
/* T: the auxiliary routing table */
/* new_entry: the temporary routing table entry that holds
the new calculated values of T(u, h)
     such that ∀n ∈ S, n ∉ new_entry.P */
  if (h == 2){
       tmp_bw = 0;
       for (i = 1; i ≤ |V|; i++)
            if( i ∉ S && b(i, u) ≠ 0 )
                 if( min{T(i, 1).bw, b(i, u)} > tmp_bw ) {
                      tmp_bw = min{T(i, 1).bw, b(i, u)};
                      tmp_neighbor = T(i, 1).neighbor;
                      tmp_node = i; tmp_d_ℓ = d_ℓ(i, u); }
       if (tmp_bw == 0)
            new_entry.bw = 0;
       else {
            new_entry.bw = tmp_bw; new_entry.neighbor =
            tmp_neighbor;
            new_entry.P = T(tmp_node, 1).P ∪ {tmp_node};
            new_entry.d^h = T(tmp_node, 1).d^h + tmp_d_ℓ;
       } /* else */
       return new_entry;
  } /* if */
  tmp_bw = 0;
  for (i = 1; i ≤ |V|; i++) {
       if (b(i, u) ≠ 0) {
            if ((∃n ∈ S s.t. n ∈ T(i, h − 1).P ∪ {i, u}) ||
            (u ∈ T(i, h − 1).P ∪ {i}))
                 tmp_entry = find_loop-free_entry(T, h − 1, S ∪ {u}, i);
            else
                 tmp_entry = T(i, h − 1);
            if(min{tmp_entry.bw, b(i, u)} > tmp_bw) {
                 tmp_bw = min {tmp_entry.bw, b(i, u)}; tmp_node = i;
                 tmp_neighbor = tmp_entry.neighbor;
                 tmp_d_ℓ = d_ℓ(i, u); } /* if */
       } /* if (b(i, u) ≠ 0) */
  } /* for loop */
  if (tmp_bw == 0) new_entry.bw = 0;
  else {
       new_entry.bw = tmp_bw; new_entry.neighbor = tmp_neighbor;
       new_entry.P = T(tmp_node, h − 1).P ∪ {tmp_node};
       new_entry.d^h = T(tmp_node, h − 1).d^h + tmp_d_ℓ;
  } /* else */
  return new_entry;
}
```

Figure 3: The auxiliary routing table update procedure (Cont'd).

fields of entries $(v; 2)$). If a 3-hop path to node u uses a 2-hop maximum bandwidth path which ends at node v, a loop is formed. However, a loop-free 3-hop path $(zxvu)$ with bandwidth 2 does exist. This loop-free 3-hop path can be located by constructing a new $(v; 2)$-th entry whose P field does not include node u. The *bw* field of the new entry $(v; 2) = \max\{\min\{T(x, 1).bw, b(x, v)\}\} = \max\{\min\{7, 3\}\} = 3$ and the P field is $(z\ \ x)$. Then, the update procedure can be applied to entry $T(u, 3)$ to obtain the 3-hop loop-free path to node u with bandwidth 2. The auxiliary routing table after the 4th iteration is shown in Fig. 5(d). □

Theorem 1: The routing table update procedure in Fig. 3 generates, for each receiver node, an h-hop maximum bandwidth path if such a path exists.

Proof: Omitted due to the page limit. Interested readers are referred to [17] for details. □

4.5 Complexity Analysis

The complexity of the QoS-based multicast routing algorithm is evaluated in terms of the number of messages needed to construct a multicast tree. Specifically, it takes the following messages to construct a tree: (1) $|\mathbf{d}|$ *setup* messages from source s or fork nodes to the $|\mathbf{d}|$ receivers; (2) at most $|\mathbf{d}| - 1$ *fork* messages from $|\mathbf{d}| - 1$ receivers to fork nodes; and (3) one *finish* message to inform the source node of the completion of the tree construction process. Thus, it takes $O(2 \cdot |\mathbf{d}|)$ messages to construct a multicast tree to $|\mathbf{d}|$ receivers. The complexity of the auxiliary routing table update procedure is exponential in the worst case due to the recursive fashion in which a new

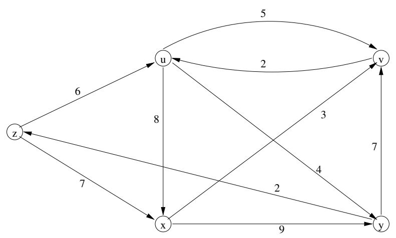


Figure 4: An example network.

loop-free h-hop path is recalculated in *find_loop_free_entry()*. If we modify the procedure to bypass the recursive search then the complexity can be significantly reduced. For example, we may modify the procedure as follows: if a loop is detected ($j \in T(u, i-1).P$, line 18 in Fig. 3), then the corresponding neighbor node u is not considered (i.e., line 19 in Fig. 3 is replaced by *continue*). This trades the routing performance for efficiency. Or, instead of recursive search, we can limit the scope of recursion to some predetermined integer.

In case that recursive search is totally bypassed, at the h-th iteration, the procedure checks, for each node i, whether or not each of its neighbor nodes, u, lies on a h-hop path from the current node to node i with the last hop (u, i) (line 17, which takes in the worst case $O(V)$), whether or not the h-hop path is loop free (line 18, which takes $O(H)$), and whether or not the h-hop path yields a smaller total bandwidth (line 27, which takes $O(1)$). This takes $O(|V|) \cdot (O(|H|) + 1)$ comparisons. Then, if such a loop-free path does exist (which takes another comparison in line 27), the procedure updates the information in $T(j, i)$. As a result, the total number of comparisons needed in the modified procedure is

$$((O(H) + 1) \cdot O(|V|) + 1) \cdot |V| \cdot H = O(H^2 \cdot |V|^2).$$

The auxiliary routing table procedure should be run off-line and much less frequently than the frequency of multicast session setup, i.e., not every time when a setup/join message arrives.

dest	1
u	6;(z)
v	0;$\emptyset$
x	7;(z)
y	0;$\emptyset$

(a)

dest	1	2
u	6;(z)	0;$\emptyset$
v	0;$\emptyset$	5;(zu)
x	7;(z)	6;(zu)
y	0;$\emptyset$	7;(zx)

(b)

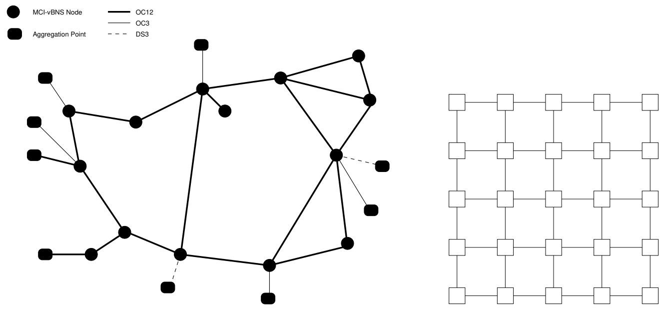
dest	1	2	3
u	6;(z)	0;$\emptyset$	2;(zxv)
v	0;$\emptyset$	5;(zu)	7;(zxy)
x	7;(z)	6;(zu)	0;$\emptyset$
y	0;$\emptyset$	7;(zx)	6;(zux)

(c)

dest	1	2	3	4
u	6;(z)	0;$\emptyset$	2;(zxv)	2;$(zxyv)$
v	0;$\emptyset$	5;(zu)	7;(zxy)	6;$(zuxy)$
x	7;(z)	6;(zu)	0;$\emptyset$	0;$\emptyset$
y	0;$\emptyset$	7;(zx)	6;(zux)	2;$(zxvu)$

(d)

Figure 5: An example that demonstrates how the auxiliary routing table is updated/maintained. (a), (b), (c), and (d) are the routing table (fields *bw* and *P*) in the 1st – 4th iteration, respectively.



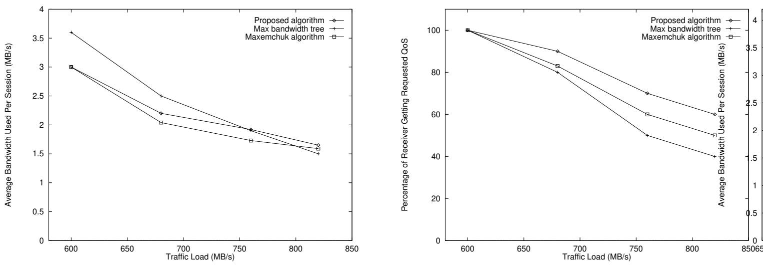
(a) (b)

Figure 6: Network topologies used in the simulation. (a) the vBNS multicast network topology. (b) a switched cluster topology.

5 Simulation Results

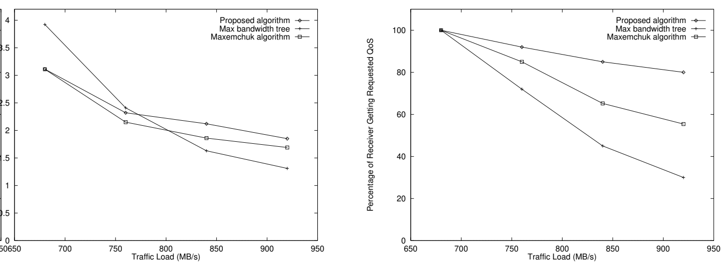
Simulation environment: The simulation study is conducted using a customized, Java-based network simulator, *NetSim*^{Q} [16]. *NetSim*^{Q} allows users to input (1) the network topology, the link capacity and the node buffer size; (2) the traffic characteristics, and the QoS requirements of, sessions; (3) the resource reservation protocol (if any); (4) the link scheduling algorithms used, and (5) the routing algorithms used. With all the inputs, the underlying simulation engine then emulates, for each session, the session establishment phase, the data transfer phase, and the session termination phase, and keeps track of the network states and simulation statistics. A new protocol/algorithm can be easily implemented in *NetSim*^{Q} by virtue of its objective-oriented, layered, modularized software architecture. Generic classes such as network, link, node, and routing are first defined. To simulate a specific network architecture (e.g., point-to-point networks), one simply subclasses a network class from the generic network class and re-defines the new attributes and methods. Similarly, to implement a specific routing algorithm, one simply subclasses a routing class from the generic routing class. We take advantage of the simulation environment provided in *NetSim*^{Q}, and implement the proposed multicast routing algorithm, along with two most relevant algorithms: the maximum bandwidth tree algorithm [11] and Maxemchuk's algorithm [7].

The topologies used in the simulation study are the MCI-vBNS Internet topology (Figure 6 (a)), a switch-based cluster topology that has approximately the same number of nodes as the MCI-vBNS Internet (Figure 6 (b)), and large random networks generated using the method proposed by Waxman [18]. The parameters α and β of range $(0, 1]$ in Waxman's method can be used to obtain certain desirable characteristics in the topology. With parameters appropriately selected, this method has been shown in [18] to render networks that resemble "real world" networks. We fine tune α and β to generate sparse networks with desirable node degrees (which range from 3 to 6). Each link is assumed to be equipped with a rate-based link scheduler (e.g., PGPS [10], WF^{2}Q+ [2], and LFVC [14], just to name a few). The link capacity for the OC12, OC3, and DS3 links is 622 Mbps, 155.5 Mbps, and 45 Mbps, respectively, in the MCI-vBNS Internet topology, and is 622 Mbps uniformly for the links in the switch-based cluster topology. The link de-



(a) (b)

Figure 7: Performance comparison in the vBNS topology: (a) the average values of the total bandwidth used by a session and (b) the percentage of receivers in a session that attain their requested QoS, under the proposed routing algorithm, the maximum bandwidth tree algorithm, and Maxemchuk's algorithm.



(a) (b)

Figure 8: Performance comparison in the switched cluster topology: (a) the average values of the total bandwidth used by a session and (b) the percentage of receivers in a session that attain their requested QoS, under the proposed routing algorithm the maximum bandwidth tree algorithm, and Maxemchuk's algorithm.

lay constant, d_ℓ, over each link ℓ follows a normal distribution with mean 1 ms.

Generation of traffic loads: The bandwidth requirements differ from one video stream to another as the bit rate depends on the amount of movement in the stream and the actual content. The resolution of a Common Intermediate Format (CIF) quality video stream is 352×288 pixels. When it is represented in 8 bits quantization of each R, G, and B component at a frame rate of 30 fps, it results in approximately 73 Mbps of uncompressed video. When the stream is further compressed using today's compression techniques with rates at $30 : 1$, the required bandwidth is 2.43 Mbps [5]. On the other hand, the bit rate of a high quality MPEG video can be as high as 8 Mbps. In our simulation study, we vary the video bit rate uniformly from 2 Mbps to 8 Mbps. A video stream is encoded into three layers: a base layer, a most significant layer, and an enhancement layer [5] with their bit rates being 30%, 50%, and 20% of the total video bit rate, respectively. The burstiness, σ_i $(i = 1, 2, 3)$, allowed for each layer is set to 2 maximum-size packets. The maximum packet size is set to 500 bytes. Sessions arrive according to a Poisson process with the mean arrival rate $\lambda = \frac{\text{network load}}{\text{mean bandwidth required per session}}$. Both the sender and the receivers (not shown in the figure) are randomly generated and connected to the vBNS nodes or the aggregation nodes. The delay requirement, D_j, and the receiving capability, R_j^r, of each receiver j are also randomly generated. In particular, D_j varies from 20ms to 100ms for all j and R_j^r is set to a value such that receiver j can receive up to 1, 2, or 3 layers with 1/3 probability, respectively.

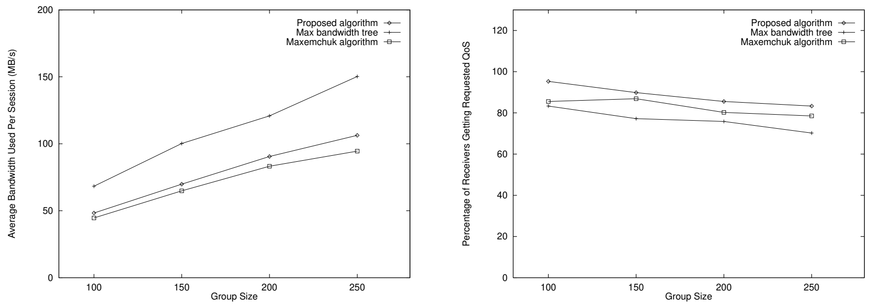
Parameters varied and performance metrics used: Each simulation run is designed to vary one or more of the following parameters: the session arrival rate λ, the number of receivers $|\mathbf{d}|$, and the delay requirement D_j of each receiver. For each given set of parameters, 100 experiments are performed, and each data point is obtained by averaging statistics gathered from these 100 experiments. Due to the space limitation, in what follows we only report the simulation results for the case $|\mathbf{d}|$ $= 10$, and $D_j = 50$ ms, unless otherwise specified. In spite of numerous system parameters involved, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a representative set of parameter values (reported below) is valid over a wide range of parameter values.

The performance metrics used are: (i) the total bandwidth required; and (ii) the percentage of receivers that attain their requested QoS in a multicast session. The first metric quantifies the cost of constructing a multicast tree, while the second metric quantifies how well a multicast tree is constructed with respect to QoS as viewed by receivers.

Performance evaluation: Figs. 7-8 give the simulations results of (i) the average value of the total bandwidth used by a session and (ii) the percentage of receivers in a session that attain their requested QoS, under the proposed routing algorithm, the maximum bandwidth tree algorithm, and Maxemchuk's algorithm in the vBNS topology and the cluster topology, respectively. Maxemchuk's algorithm aims at minimizing the total bandwidth resource used without considering the QoS requirement. As shown in Fig. 7 (a), the multicast tree constructed under the proposed algorithm is more resource efficient as compared to that under the maximum bandwidth tree algorithm, and at the same time, achieves resource usage close to that under Maxemchuk's algorithm. As the network load increases, the percentage of receivers that are not able to attain their requested QoS increases more significantly under the maximum bandwidth tree algorithm and Maxemchuk's algorithm (Fig. 7 (b)). (As a result, the average value of the total bandwidth used under the proposed algorithm is higher in the case of high traffic load.) Also, as shown in Fig. 8, since the cluster topology provides more rich interconnections and thus more alternate paths, the percentage of receivers successfully joining the multicast group using the proposed algorithm is much higher than that under the maximum bandwidth tree algorithm or Maxemchuk's algorithm. This suggests that the proposed algorithm is more effective in networks with multiple alternate paths.

Fig. 9 depicts the performance for the three algorithms under the the case of $D_j = 200$ ms for all j as the group size varies from 100 to 250. The proposed algorithm is more resource efficient and the performance gain is more pronounced as the group size increases.

As discussed in Section 4.5, it takes $O(2 \cdot |\mathbf{d}|)$ messages to construct a multicast tree to $|\mathbf{d}|$ receivers, but the computational complexity to update the auxiliary routing table is theoretically exponential. (We also discussed modified versions that trade the performance for efficiency in Section 4.5.) Fig. 10 further evaluates the average computational complexity of the auxiliary routing table update procedure. The complexity is evaluated as the ratio of the average number of comparisons executed in each invocation of the procedure to the theoretical best case complexity $O(H^2|V|^2)$. As shown in Fig. 10, the number of comparisons is about an order of magnitude larger than



(a) (b)

Figure 9: Performance with respect to group size in the random topology: (a) the average values of the total bandwidth used by a session and (b) the percentage of receivers in a session that attain their requested QoS under the proposed routing algorithm the maximum bandwidth tree algorithm, and Maxemchuk's algorithm.

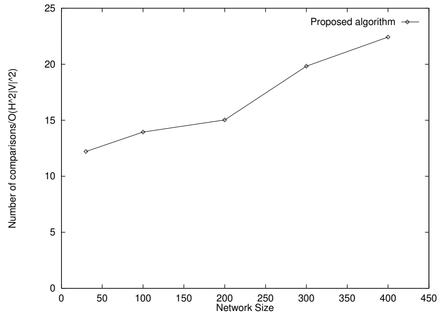


Figure 10: The ratio of the number of comparisons executed in each invocation of the auxiliary routing table update procedure to the theoretical best case complexity $O(H^2 \cdot |V|^2)$.

$O(H^2|V|^2)$, but does not exponentially increase with the network size in large networks.

6 Related Work

Max bandwidth tree algorithm: Shacham [11] studied the one-to-many multicast communication for hierarchically-encoded data. Their approach works as follows. First, they compute the maximum available bandwidth tree to connect all receivers from the source. Second, receivers are classified into different categories according to their receiving capabilities. A quality value is assigned for each layer of data. The satisfaction level of a receiver is measured by summing up the quality value over all the intended layers that are received. The rate at which each receiver will receive is then determined to maximize the sum of all the satisfaction level of individual receivers. The link bandwidth will then be allocated appropriately on the maximum bandwidth tree.

There are two major deficiencies associated with this algorithm: (1) it does not guarantee shortest paths, and (2) it may include large bandwidth links that lead only to receivers with lower bandwidth capability. To eliminate the first deficiency, an improvement was then proposed during the tree construction phase: shorter paths can be obtained by selecting a node with the shortest distance to the source during the execution of the maximum bandwidth tree algorithm. To deal with the second deficiency, when the tree construction algorithm is terminated and the link bandwidth is assigned, the bandwidth on the maximum bandwidth tree can be reduced by allocating the bandwidth starting from the receivers. If a receiver is selected that has an assigned bandwidth lower than the maximum bandwidth, the link bandwidth over the path from the receiver to the source is reduced (to the receiver's assigned bandwidth), until the path meets a node with an outgoing link of higher bandwidth than that of the selected destination. At that point, another receiver is selected and the procedure continues. Since the paths are assigned on the maximum bandwidth tree, this "bandwidth trimming" procedure may still render longer paths.

Shacham and Meditch's algorithm: Shacham and Meditch [12] investigated the problem of optimal distribution of multiple streams from a source to a set of destinations, subject to the destinations' requirements and the limitations on the available link capacities. The multiple streams from a source may be unrelated. Users' requests are expressed by means of *bids* and are non-negative values assigned by each receiver to each offered stream. Under the assumption that the source gains an amount equal to a user's bid whenever a stream is delivered to that user, the authors formulated the problem of optimally assigning multiple streams to the links of a multicast tree as an integer programming problem, and sought stream distributions that maximize the sources' gain. They proposed an off-line centralized algorithm for determining the optimal assignment. Succinctly, the algorithm consists of three procedures: listing candidate assignments on a single link, listing candidate assignment on tree paths, and evaluating the gain for each assignment. The algorithm then chooses the one with the maximum gain.

Receiver-driven layered multicast protocol: McCanne, Jacobson and Vetterli [8] described a receiver-driven layered multicast protocol for rate-adaptive video transmission. In their work, the source transmits each layer of its signal on a separate multicast group. Each receiver specifies its level of subscription by joining a subset of multicast groups of the layered video. Each receiver carries out the following control loop: (1) when it perceives there may be surplus bandwidth available, it adds a layer; (2) when it detects congestion, it drops a layer. Under the scheme, a receiver searches for the optimal level of subscription with respect to the network congestion level. Li, Ammar, and Paul [6] also proposed and evaluated a Layered Video Multicast with Retransmissions (LVMR) scheme [4] for distributing video using layered coding over the Internet. Conceptually, they improve the quality of reception within each video layer by retransmitting lost packets given an upper bound on recovery time and applying an adaptive playback point scheme to help achieve more successful retransmission. They also used hierarchical rate control (HRC) mechanisms to adapt to network congestion and heterogeneity.

Maxemchuk's algorithm: Maxemchuk [7] proposed a multicast algorithm for video distribution with the objective of conserving the required network resources. The problem is formulated as a variant of the well known Steiner tree problem. In the conventional Steiner tree problem, the link cost in the network are fixed and a minimum cost tree for a subset of the nodes in a network is constructed. In this problem, the link costs depend upon the set of receivers that share the link. Specifically, the link cost is the basic cost of the link times the rate of the highest rate receiver that uses the link. As a result, the cost of a link will change when a receiver added as a leaf node has a higher resource requirement than any of the other on-tree receivers that use the link.

Maxemchuk proposed a heuristic algorithm that forms a Steiner tree with a set of higher-rate receivers first and then successively adds lower-rate sets of receivers to that tree. The receivers are separated into subsets according to rate. The heuristic then uses existing Steiner tree construction heuristics to add receivers to the multicast tree. Once the tree with the subset of highest-rate receivers has been constructed, the heuristics repeat with this tree as the starting tree for the subset of receivers with next highest rate requirements until all subsets of receivers are connected to the tree. With this procedure, the links that are added to the tree never increase the requirements of links that were previously assigned to the tree.

In all the above work, no explicit QoS in terms of delay bound is considered. Moreover, the proposed algorithms/heuristics are mainly centralized. In contrast, our proposed multicast routing algorithm aims at satisfying the bandwidth and delay requirements imposed by heterogeneous receivers, while minimizing the total amount of network resources. The algorithm is decentralized, takes advantage of the routing information provided by the underlying unicast routing protocol, and adapts to membership changes.

7 Conclusion

We have proposed a QoS based multicast routing algorithm for distributing layered video to heterogeneous receivers in rate-based networks. The proposed algorithm is shown via simulation to be more resource efficient as compared with other existing algorithms. In particular, the proposed algorithm has the advantage of more effectively finding appropriate routes in networks with multiple alternate paths.

We have identified several avenues for future work. The proposed multicast routing algorithm can be extended to networks with other scheduling disciplines. Equations that characterize the relation between the end-to-end delay and the bandwidth allocated (e.g., Eq. (2.1)) as well as procedure *search_path* in Fig. 2 (line 21) have to be modified. On the other hand, the quality of a multicast tree maintained may degrade over time as group members dynamically join/leave the tree, and a new multicast tree may have to be re-constructed if the existing tree cannot make efficient use of network resources. Bauer *et al.* [1], Sriram *et al.* [13], and Narvaez *et al.* [9] proposed several heuristics for locally rearranging part of an existing multicast tree so as to strike a balance between the multicast service disruption and the quality of a multicast tree. The suitability of using these approaches in the proposed algorithm is an area for further study. We plan to study the tradeoff between the reduction of auxiliary routing table updating complexity and the accuracy of finding the maximum h-hop path as well as the impact of such complexity reduction to the effectiveness of our proposed routing algorithm.

References

[1] F. Bauer and A. Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. *IEEE JSAC*, pages 382—397, April 1997.

[2] J. C. R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. *IEEE INFOCOM'96*, March 1996.

[3] J. C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the Iternet. *Proceedings of ACM SIGCOMM'94*, September 1994.

[4] X. Li et al. Layered video multicast with retransmission (LVMR): Evaluation of error recovery. *Proc. of NOSSDAV (St. Louis, MO)*, May 1997.

[5] M. V. Jensen and B. Nielsen. Design and implementation of an efficient, layered video codec for heterogeneous networks. *Technical Report R-98-5008, Department of Computer Science, Aalborg University*, November 1998.

[6] X. Li, M. H. Ammar, and S. Paul. Video multicast over the Internet. *IEEE Network Magazine*, pages 46—60, March/April 1999.

[7] N. F. Maxemchuk. Video distribution on multicast networks. *IEEE Journal on Selected Areas in Communications*, pages 357—372, April 1997.

[8] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *Proceedings of ACM SIGCOMM'96*, 1996.

[9] P. Narvaez, K. Siu, and H. Tzeng. New dynamic SPT algorithm based on a ball-and-string model. *IEEE INFOCOM'99, New York*, March 1999.

[10] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, June 1993.

[11] N. Shacham. Multipoint communication by hierarchically encoded data. *Proceedings IEEE INFOCOM'92*, pages 2107—2114, May 1992.

[12] N. Shacham and J. S. Meditch. An algorithm for optimal multicast of multimedia streams. *Proceedings IEEE INFOCOM'94*, pages 856—864, 1994.

[13] R. Sriram, G. Manimaran, and C. Murthy. A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees. *IEEE INFOCOM'99, New York*, March 1999.

[14] S. Suri, G. Varghese, and G. Chandranmenon. Leap forward virtual clock: A new fair queuing scheme with guaranteed delays and throughput fairness. *IEEE Proc. INFOCOM'97*, 1997.

[15] D. Taubman and A. Zakhor. Multi-rate 3-D subband coding of video. *IEEE Transactions on Image Processing*, pages 572—588, Sept 1994.

[16] H.-Y. Tyan, B. Wang, and C.-J. Hou. NetSim^Q: A Java-integrated network simulation tool for QoS control in point-to-point high speed networks. *3rd NASA Research and Education Network Workshop, Moffett Field, CA*, August 1998.

[17] B. Wang and C.-J. Hou. Qos-based multicast routing for distributing layered video to heterogeneous receivers in rate-based networks. Technical report, Dept. of Electrical Eng., Ohio State University, June 1999.

[18] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, pages 1617–1622, December 1988.