Multi-layer Multicast Congestion Control in Satellite Environments

Jun Peng and Biplab Sikdar
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute (RPI), Troy, NY 12180
{pengj2, sikdab}@rpi.edu

Abstract*— It is well known that long and variable link delays, link errors, and hand-offs in satellite environments seriously interfere with TCP's congestion control mechanisms. These channel characteristics also adversely affect existing multi-layer multicast congestion control schemes when they are used in satellite environments. In these schemes, multicast pruning is usually used for dropping a layer when receivers detect severe losses. As in the TCP case, link errors and variable link delays may cause unnecessary rate reduction for these schemes because they may wrongly interpret some events like link errors as congestion events. Another problem specific to these multi-layer schemes arises from the long delay of satellite links. While the delay in dropping a layer is already a serious problem in wireline networks for existing multi-layer multicast congestion control schemes, long link delays in satellite environments will further increase the delay in dropping a layer, so the problem is exacerbated. In addition, almost all existing schemes still have problems with fairly sharing bandwidth with TCP flows, controlling the overhead of frequent grafting and pruning, and handling misbehaving receivers. In this paper, we present a new multi-layer multicast congestion control scheme that is suitable for satellite environments and overcomes most of the disadvantages of existing schemes. Our scheme is not affected by the long and variable delays of satellite links. Link errors also do not decrease the performance of our scheme. Further, our scheme has very limited control overhead. In addition to these advantages specific to satellite environments, our scheme achieves good fairness in sharing bandwidth with TCP sessions and is not sensitive to misbehaving receivers.

Index: congestion control, multicast, and satellite

I. Introduction

With their incomparable coverage capability, satellite networks will play an essential role in the future global digital personal communication system. However, satellite networking poses many challenges for protocol designers. The propagation delay of a satellite link is long and may be highly variable. Hand-offs may also happen from time to time to a satellite link. In addition, interference and fading may render frequent or bursty errors on satellite links. Finally, when a satellite link is shared, the bandwidth of the link may become scarce. All of these and other characteristics of satellite links can adversely affect transport, routing and MAC protocols that are specifically designed for wireline networks. For example, link errors and long and variable link delays in satellite environments may cause TCP congestion control to cut rate unnecessarily, which may considerably decrease its performance. In this paper, we investigate the effects of satellite links on existing multi-layer multicast congestion control schemes and present a new scheme that is immune to these adverse effects and also has better performance than existing schemes in other aspects such as fairness in sharing bandwidth among competing sessions.

Existing multi-layer multicast congestion control schemes, such as [1] [2] [3] [4] [5] [6], usually depend on receivers for adjusting their receiving rates according to detected network conditions. Specifically, the source encodes data into several layers [1] and sends each layer to a separate multicast group. A receiver then adds some layers by subscribing to more groups when it "senses" free bandwidth. When a receiver "senses" congestion, it drops some layers by unsubscribing from some groups. However, these "senses" may be wrong sometimes, especially in a satellite environment. This is not the only problem for multi-layer schemes. Because of the design of the Internet Group Management Protocol (IGMP), there is a significant delay in pruning a multicast branch, so dropping a layer usually needs a considerable amount of time. The layer-drop delay is already a serious problem in wireline networks [4] [6]. To deal with this problem, some schemes like [4] [6] schedule the transmission of the data in each layer according to some special patterns at the source. Although these schemes can alleviate the layer-drop delay problem, they usually introduce significant control overhead by frequently adding and dropping layers. Furthermore, it is hard to apply these schemes to some applications such as streaming multimedia, where data can hardly be freely scheduled for transmission at the source.

If existing multi-layer multicast congestion control schemes are adopted in satellite environments, the problems caused by their disadvantages mentioned above will become more severe. As in the TCP case, errors on satellite links can cause these schemes to drop layers unnecessarily because they may mistake link-error losses as congestion losses. Wrongly dropping layers results in poor performance in session fairness, traffic stability and bandwidth utilization. Another problem specific to these multi-layer schemes arises from the long delays of satellite links. Since pruning information has to traverse satellite links to reach upstream routers, additional delay will be introduced in dropping a layer besides the delay already introduced by the IGMP protocol. Even with special scheduling of data transmission at the source, in addition to

[1] We are interested in cumulative layers in this paper, although the proposed scheme can also be used for non-cumulative layers.

the limitation in applications of this mechanism, the control overhead introduced may become a problem if satellite links are shared among multiple receivers (this is highly possible, since one satellite can not serve an unlimited number of earth stations simultaneously at high speed). Besides these specific disadvantages to satellite environments, usually these schemes still have problems in fairly sharing bandwidth among competing sessions, especially with the existence of TCP sessions [7] [8] [4] [6]. For example, a late TCP session may not be able to grab a right share of bandwidth from existing sessions. In addition, the throughput for a multicast receiver is usually not stable with these schemes, which is a big disadvantage for multimedia applications. Another common disadvantage of existing multi-layer multicast congestion control schemes is their sensitiveness to misbehaving receivers. For example, a misbehaving receiver may prevent a large group of receivers from receiving their data properly by adding layers continuously without considering the network conditions.

Instead of depending on individual receivers in adjusting their receiving rates, we propose a multi-layer multicast congestion control scheme that deals with congestion right at the site where the congestion occurs. Our scheme enriches the abstractions of the multicast routing layer to adjust the rate of the multicast traffic passing through a congested link. When congestion occurs or is about to occur on a link, some layers of multicast application sessions are ”blocked” from entering the congested link; when the link is lightly utilized, some blocked layers are ”released” to pass through the link. More specifically, the state of the output queue of the link is continuously observed. When the number of packets in the queue is above a threshold, some layers are usually blocked; while when the number of packets is below another threshold for some time, some layers are released. When the number of packets is between these two thresholds, usually no adjustment is made for multicast sessions. In steady state, the number of packets in the queue should stay between the two thresholds most of the time. This scheme is based on our previous work in wireline networks [9]. In wireline networks, the bandwidth of a link is usually constant and always available, so it is a good metric to consider while fairly assigning bandwidth to competing sessions. In satellite environments, the available bandwidth of a satellite link is usually not constant and not always available because of interference, fading and access competition. Therefore, in this case, link bandwidth is not a good metric in the fair assignment of bandwidth to competing sessions. Instead of using the link bandwidth, in this scheme we use the queue state of a link as the metric in adjusting the rate of the multicast traffic passing through the link.

Our scheme is immune to the problems that existing schemes may suffer from in satellite environments. Link errors can not cause our scheme to wrongly block a layer, since the queue state at a bottleneck instead of the loss information at receivers is used in adjusting the number of layers passing through the bottleneck. Long link delay also can not cause problems to our scheme because our scheme does not depend on traditional pruning and grafting in adjusting multicast traffic rate. All of these features of our scheme exists because our scheme adjusts multicast layers right at each bottleneck in a multicast tree. For the same reason, our scheme only introduces very limited control overhead. In addition to the above features that enable our scheme to work effectively and efficiently in satellite environments, our scheme achieves good fairness in bandwidth sharing among competing sessions, even with the existence of TCP sessions. Besides, the traffic for multicast receivers is fairly stable with our scheme. Another feature of our scheme is that it is not sensitive to misbehaving receivers. Meanwhile, like existing multi-layer multicast congestion control schemes, our scheme has some communication overhead resulting from grafting and pruning. Also, it introduces the possible overhead of an empty layer for a multicast session for avoiding the explicit interactions among routers and receivers and for reducing control traffic in a congestion affected area.

The rest of the paper is organized as follows. We introduce our scheme in detail in Section II. Section III analyzes our scheme for fairness among competing sessions, effectiveness in satellite environments, and scheme cost. Section IV presents our simulation results. Finally, our conclusions appear in Section V.

II. The Multi-layer Congestion Control Scheme

A. Scheme Overview

The profile of our scheme is shown in Fig. 1. When multicast application sessions pass through a link, our scheme begins to observe the output queue of the link and the traffic passing through the link. When the number of packets in the queue, N_{QuPkt}, goes beyond a threshold, $QuThresh_2$, some layers of multicast application sessions are blocked from entering the link. When N_{QuPkt} is below another threshold, $QuThresh_1$, for some time, some blocked layers of multicast application sessions are released to pass through the link. In other cases, there is usually no layer adjustment. In this way, congestion can be alleviated while free bandwidth can also be claimed. This is only a profile of the scheme. Some important details are missing. For example:

- How is it ensured that the bandwidth of a bottleneck is shared fairly between TCP sessions and multicast application sessions if some TCP sessions are also passing through the link?
- How do multicast application sessions share the bandwidth available to them fairly among themselves?
- How is the layer priority information communicated if the layers of a multicast application session have different priorities?

We will present details of our scheme in the next subsection. In this subsection we used a “multicast application session” to represent an application session whose data needs to be multicast to multiple receivers. For simplicity, we will use “multicast session” equivalently with “multicast application session” in the rest of the paper.

B. The Scheme

1) Retrieving Session Information from Traffic: Our scheme retrieves some information from the passing-through traffic

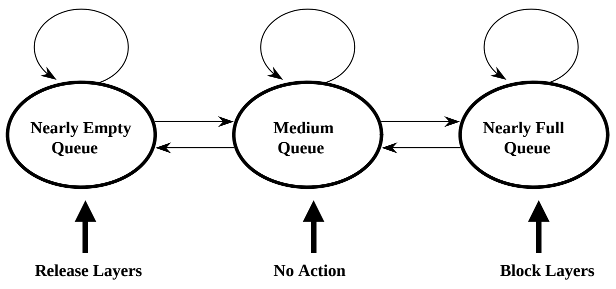

Fig. 1. The Profi le of the Proposed Scheme

on a bottleneck for assisting its operations. Specifically, the number of TCP sessions (N_{TcpSes}), the number of multicast sessions (N_{MctSes}), the number of layers of each multicast session ($N^i_{LiveLayer}$, $0 < i \leq N_{MctSes}$), the relative average session rate of TCP sessions (R_{TcpAvg}) and the relative average session rate of multicast sessions (R_{MctAvg}) are the information retrieved. Usually, the destination address of a packet can indicate whether it belongs to a layer of a multicast session or an independent TCP session. The number of layers, $N^i_{LiveLayer}$, that the i^{th} multicast session has at a bottleneck is indicated by the number of different multicast addresses that the traffic of the session has at the bottleneck. For obtaining the relative average session rates, the total number of passing-through TCP packets (N_{TcpPkt}) and the total number of passing-through multicast packets (N_{MctPkt}) in a period of time are counted, then they are divided by the the number of TCP sessions and the number of multicast sessions passing through the bottleneck, respectively (we assume that the average packet length for multicast sessions is close to that of TCP sessions):

$$R_{TcpAvg} = N_{TcpPkt}/N_{TcpSes}$$
$$R_{MctAvg} = N_{MctPkt}/N_{MctSes}$$

All the information retrieved from traffic is the basis for later congestion control actions. In our scheme, we assume that the number of layers that a multicast session possesses at a bottleneck can reflect its relative data rate among the multicast sessions passing through the link. Specifically, a multicast session with more layers has higher data rate than a multicast session with less layers at the bottleneck. A general case is that all the sessions have the same or similar layer size. For simplicity, we will introduce the scheme in this general case below.

2) Layer Priority Information: For some applications such as streaming multimedia, different layers in an application session usually have different priorities. In general, in the i^{th} session the j^{th} layer has higher priority than the k^{th} layer if j is less than k:

$$P_{L^i_j} > P_{L^i_k} \quad \text{if } j < k$$

A layer with higher priority implies that its data are more useful for a receiver to get the wanted information than the data from a layer with lower priority. For example, when a piece of video is encoded into L layers, usually the data of the m^{th} layer can be useful in decoding only if the data from all the lower layers (1, 2, 3, ..., $m-1$) are available. In this case, the layer with the lowest priority among the living layers of a session should be blocked when a layer needs to be blocked from this session on a bottleneck. The layer priority information in a multicast session must be communicated if our scheme wants to follow the priorities among the layers.

In our scheme, the layer priority information is embedded in the multicast addresses that the layers of a multicast session use. Specifically, the j^{th} layer in session i has higher priority than the k^{th} layer in session i if the address of the j^{th} layer is lower than the address of the k^{th} layer:

$$P_{L^i_j} > P_{L^i_k} \quad \text{if } A_{L^i_j} < A_{L^i_k}$$

In general, when a multicast session applies for multicast addresses, it is assigned a block of addresses. With our scheme, the session lets its lower priority layers use higher addresses and higher priority layers use lower addresses. Specifically, the address of the j^{th} layer in session i is lower than the address of the k^{th} layer in session i if j is less than k:

$$A_{L^i_j} < A_{L^i_k} \quad \text{if } j < k$$

Fig. 2 shows an example of the relationship between the position of an address and the priority of its represented layer. This kind of information embedding is important for our scheme, since it eliminates the need of defining new fields in the packet header to indicate the priority of the packet among the packets from the same source. Otherwise, new fields have to be added to the packet header and the information in these fields has to be retrieved separately upon the arrival of a packet. In that case, forwarding policy has to be changed significantly and extra overhead will be introduced.

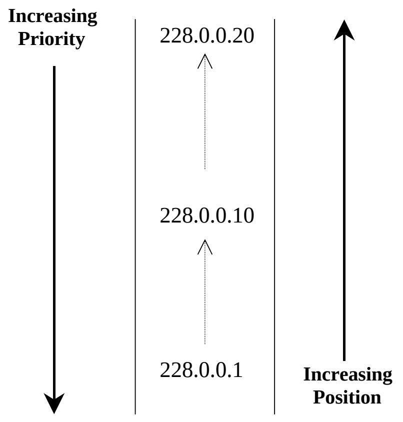

Fig. 2. Address Position and Layer Priority

3) Layer Blocking and Layer Releasing: Instead of using layer-add and layer-drop as in existing multi-layer schemes, our scheme uses layer-block and layer-release to alleviate congestion and to claim bandwidth, respectively. Layer-block is the modification of the multicast routing table to prevent a layer from entering a congested link, while layer-release is the modification of the routing table to re-enable a blocked layer to pass through a link.

When layer-block is necessary among multicast sessions, the multicast session with the greatest number of layers is selected to block a layer. Within this session, the layer with the lowest priority among un-blocked layers are blocked. Specifically:

$$\begin{gathered}\text{If } N^m_{LiveLayer} \geq N^n_{LiveLayer} \text{ for } 0 < n \leq N_{MctSes}, \\ \text{choose the } m^{th} \text{ session.} \\ \text{Then if } P_{L^m_i} \leq P_{L^m_j} \text{ for } 0 < j \leq N^m_{LiveLayer}, \\ \text{choose the } i^{th} \text{ layer.}\end{gathered}$$

When layer-release is required, the multicast session with the least number of layers is selected to release a layer [2]. Within this session, the layer with the highest priority among the blocked layers is released. Specifically:

$$\begin{gathered}\text{If } N^m_{LiveLayer} \leq N^n_{LiveLayer} \text{ for } 0 < n \leq N_{MctSes}, \\ \text{choose the } m^{th} \text{ session.} \\ \text{Then if } P_{L^m_i} \geq P_{L^m_j} \text{ for } N^m_{LiveLayer} < j \leq N^m_{layer}, \\ \text{choose the } i^{th} \text{ layer.}\end{gathered}$$

The above procedures ensure that multicast sessions share the bandwidth available to them fairly among themselves. They also ensure that priorities among layers are followed in an individual session. The following subsections will show how it is ensured that multicast sessions as a whole get and only get a fair share of bandwidth at a bottleneck.

4) Receiver Actions: When a receiver joins a multicast session, it adds layers gradually. After adding a layer, a receiver waits for some time before adding another layer. If the previous layer is not blocked somewhere in the network and the data of the layer are flowing into the receiver, the receiver adds another layer. This process continues until an empty layer is obtained. An empty layer for a receiver is a layer subscribed to but whose data are not flowing into the receiver because of being blocked somewhere in the network.

After its initialization, a receiver is responsible for maintaining a single empty layer. The procedure is as follows:

- If there are more than one empty layer, the receiver drops all but the lowest one.
- If the data of the maintained empty layer are flowing into the receiver, the receiver adds another layer.

Several empty layers may appear when more than one layer is blocked at the bottleneck during severe congestion. Extra empty layers need to be dropped for possibly saving bandwidth above the bottleneck (details will be given in the next section of scheme analysis). When an empty layer becomes not ”empty” because of being released at the bottleneck, a new empty layer needs to be prepared. This is because the bottleneck always needs to know the information of the next possible layer of a multicast session in case free bandwidth will be available later for this session. When free bandwidth is available, the next layer can be released to claim the free bandwidth. The advantages and disadvantages of maintaining an empty layer will be discussed in the next section of scheme analysis.

[2]If this session does not have any more layers, the session with the next least number of layers will be considered.

5) Adjusting Multicast Layers: This subsection introduces the most important part of our scheme: the adjustment of the number of multicast layers (N_{layer}) passing through a bottleneck, where:

$$N_{layer} = \sum_{i=1}^{N_{MctSes}} N^i_{LiveLayer}$$

Appropriate adjustment of the number of layers passing through a link is essential for alleviating congestion and claiming free bandwidth while ensuring fairness in sharing bandwidth among competing sessions.

Our scheme blocks and releases multicast layers on a link according to the state of the output queue of the link. The queue is classified into three phases: phase 1, phase 2 and phase 3. The queue phase is decided by the number of packets in the queue (N_{QuPkt}) and the two specified thresholds ($QuThresh_1$ and $QuThresh_2$). The two thresholds depends two scheme parameters ($F_{QuThresh1}$ and $F_{QuThresh2}$):

$$\begin{gathered}QuThresh_1 = QuSize \times F_{QuThresh1} \\ QuThresh_2 = QuSize \times F_{QuThresh2}\end{gathered}$$

where $QuSize$ denotes the size of the queue. The phases are classified in the following way:

$$\text{QuPhase} = \begin{cases} 1 & \text{if } N_{QuPkt} \leq QuThresh_1 \\ 2 & \text{if } QuThresh_1 < N_{QuPkt} < QuThresh_2 \\ 3 & \text{if } N_{QuPkt} \geq QuThresh_2 \end{cases}$$

A phase 1 queue may indicate free bandwidth on the link, while a phase 3 queue may imply link congestion or potential link congestion. To fully utilize available bandwidth and effectively deal with congestion, different actions are required in different queue phases.

When the queue is in phase 1, multicast layers are not necessarily released. This is because a phase 1 queue does not necessarily mean that free bandwidth is available on the corresponding link. A TCP session has its famous fluctuation pattern in rate from its AIMD congestion control scheme. Upon congestion, a TCP session usually cuts its rate multiplicatively to relax the congested link, then it additively builds up its rate to probe for free bandwidth. So a phase 1 queue may just mean that all TCP sessions passing through the link cut their rates multiplicatively a moment ago and are building up their rates now. In this case, no multicast layer should be released, or TCP sessions may be deprived of their share of bandwidth. However, if the queue stays in phase 1 for a relatively long time, it is almost certain that some free bandwidth is available on the link. Therefore, multicast layers are released in phase 1 with our scheme only if the queue can stay in phase 1 for a specific amount of time ($T_{observe}$).

When the queue is in phase 2, usually no action is taken for multicast sessions. A kind of balance is achieved when the queue stays in phase 2 most of the time, but this does not necessarily mean that good fairness is also achieved between TCP sessions and multicast sessions. For example, if a multicast session leaves, TCP sessions may increase their rates quickly and keep the queue in phase 2 most of the time (i.e., to multicast sessions, there is no free bandwidth on the link). In this case, other multicast sessions cannot get a share of the bandwidth spared by the multicast session that has departed. This is because without additional precautions, no multicast layer will be released unless the queue stays in phase 1 for a duration longer than $T_{observe}$. To avoid this kind of unfairness situations, the average session rate of TCP sessions (R_{TcpAvg}) and the average session rate of multicast sessions (R_{MctAvg}) are checked in phase 2. If R_{MctAvg} is less than R_{TcpAvg}, a multicast layer is released. Otherwise, no action is taken.

When the queue is in phase 3, multicast layers may be blocked instantly. This is because TCP sessions can be throttled from frequent losses if the queue stays in phase 3 for a long time. However, there is a special situation that needs to be considered. If the traffic fluctuation of TCP sessions causes the queue to visit phase 3 from time to time, multicast layers may be blocked frequently. Although the blocked layers may be released a moment later after TCP sessions back off for congestion, the number of layers of multicast sessions fluctuates in this case. To avoid this problem, the average session rate of TCP sessions (R_{TcpAvg}) and the average session rate of multicast sessions (R_{MctAvg}) are also checked in phase 3. Only if R_{MctAvg} is higher than R_{TcpAvg}, a multicast layer is blocked in phase 3. This procedure can stabilize multicast traffic while ensuring fairness among participating sessions.

To finish this subsection, in Algorithm 1 we give the pseudocode for adjusting the number of multicast layers passing through a link. The pseudocode shows a simple view of the logic of the layer adjustment but without going into details.

Algorithm 1: Algorithm for layer adjustment

```
repeat
    switch phase do
        case phase1
            if phase1-timer times out then
                release a multicast layer;
            end
            if phase1-timer is idle then
                start phase1-timer;
            end
        case phase2
            if phase1-timer is running then
                cancel phase1-timer;
            end
            if $R_{MctAvg} < R_{TcpAvg}$ then
                release a multicast layer;
            end
        case phase3
            if phase1-timer is running then
                cancel phase1-timer;
            end
            if $R_{MctAvg} > R_{TcpAvg}$ then
                block a multicast layer;
            end
    end
until forever ;
```

6) Scheme Adaptation: If the scheme introduced above is followed, our general design goal can be achieved: a scheme suitable for satellite environments and with good fairness in sharing bandwidth among competing sessions. However, the scheme can be improved. Usually, multicast traffic should be as stable as possible, which is good for some specific applications such as streaming media and also good for bandwidth utilization. When queue balance is achieved, usually multicast sessions have a stable number of layers. In this case, the multicast traffic has low fluctuation, but multicast traffic is also under the influence of TCP traffic. If TCP traffic fluctuates severely, the queuing delay for other traffic will also fluctuate in a similar way. Highly variable path delay can cause traffic fluctuation to receivers. This kind of traffic fluctuation is unavoidable when sharing bottlenecks with TCP sessions. Meanwhile, there is also another kind of traffic fluctuation that can be prevented, which is from the constant layer adjustment of multicast sessions. Constant layer adjustment may happen even if the queue is in balance. For example, if the average session rate of multicast sessions (R_{MctAvg}) is lower than the average session rate of TCP sessions (R_{TcpAvg}) when the queue is in phase 2, a layer is released for multicast sessions ($N_{layer} \leftarrow N_{layer} + 1$). The queue may then enter phase 3 because of the increased traffic rate, but in phase 3 it is possible R_{MctAvg} is higher than R_{TcpAvg} now. So a layer may be blocked in phase 3 ($N_{layer} \leftarrow N_{layer} - 1$). Then, the queue falls back to phase 2 and multicast sessions release a layer again ($N_{layer} \leftarrow N_{layer}+1$). After that, the queue enters phase 3 and a layer is blocked again ($N_{layer} \leftarrow N_{layer} - 1$). This process repeats forever if no precautions are taken. This is not the only situation where frequent layer adjustment occurs. For example, if multicast sessions increase their rate too fast upon detecting free bandwidth, layer fluctuation may also occur. To avoid various kinds of layer fluctuation, we add the following procedures to adapt our scheme to various situations:

1) When multicast sessions need to increase their traffic rate continuously, the rate of increase is decreased each time when a layer is released. Specifically, the observation time ($T_{observe}$) for the next layer release is increased by a factor ($F_{SlowDown} > 1$):
$$T_{observe} \longleftarrow T_{observe} \times F_{SlowDown}$$
2) When a layer is blocked right after a layer is released, the observation time ($T_{observe}$) for the next layer release is increased by another factor ($F_{BackOff} > 1$):
$$T_{observe} \longleftarrow T_{observe} \times F_{BackOff}$$
3) A layer is blocked in phase 3 only if the average session rate of multicast sessions is larger than the average session rate of TCP sessions by a ratio threshold (R_{Block}):
$$(R_{MctAvg} - R_{TcpAvg})/R_{TcpAvg} \ > \ R_{Block}$$

The last procedure may result in a little higher priority for multicast sessions, but arguably it does not hurt fairness in general. A multicast session usually has a large number of receivers downstream from a bottleneck while a TCP session only has a single receiver. Therefore, assigning a little more bandwidth to multicast sessions on a bottleneck is good for the utilization of the bandwidth of the bottleneck from a global point of view.

7) Scheme Parameters: In our scheme, there are two parameters for defining the two queue thresholds. They are $F_{QuThresh1}$ and $F_{QuThresh2}$. The two queue thresholds are obtained by multiplying the two factors with the queue size, respectively. In general, $F_{QuThresh1}$ should be close to 0 while $F_{QuThresh2}$ should be close 1. In addition, there are three parameters in scheme adaptation: the slow-down factor ($F_{SlowDown}$) in continuous layer adding, the back-off factor ($F_{BackOff}$) upon congestion, and the rate difference ratio threshold (R_{Block}) in blocking a layer. As introduced above, the first two parameters are used for increasing the observation time ($T_{observe}$) in releasing a layer, while the last parameter is for comparing the average session rate of TCP sessions with the average session rate of multicast sessions before blocking a layer. Two other parameters of our scheme are the interval in checking queue state (I_{ChekQu}) and the initial observation time ($T_{observe}$) in releasing a layer. The former controls the frequency in checking queue state, while the latter decides the initial observation time before a layer can be released in phase 1.

We give the definitions for the parameters of our scheme and the values used in our experiments for these parameters in Table I.

III. Analysis of the Scheme

The previous section described our scheme in detail. In this section we analyze our scheme for fairness among competing sessions, effectiveness in satellite environments, and scheme cost. In our analysis we assume that multicast sessions as a whole always need more bandwidth when bandwidth is available.

A. Fairness Among Competing Sessions

1) General Analysis: We begin by showing that our scheme is able to achieve balance (i.e., the queue stays in phase 2 most of the time). From the scheme description in the previous section, the queue can only be in one of the 3 phases: phase 1, phase 2 and phase 3. So can the queue stay in phase 1 or phase 3 for a long time? If the queue stays in phase 1, TCP sessions will increase their congestion windows. The rates of TCP sessions will then increase. So the queue will leave phase 1 eventually. Even if TCP sessions can not increase their rates because of the limitation of the receiver's advertised window, multicast sessions will release a layer if the queue stays in phase 1 for a period of time longer than $T_{observe}$. Consequently, the queue will also eventually leave phase 1. Therefore, the queue can not stay in phase 1 for a long time with our scheme. On the other hand, the queue can not stay in phase 3 for a long time either. Multicast sessions will block a layer instantly when the queue enters phase 3 if the average session rate of multicast sessions (R_{MctAvg}) is significantly higher than the average session rate of TCP sessions (R_{TcpAvg}). Even if no multicast layer is blocked, TCP sessions will back off upon queue overflow. Therefore, the queue can not stay in phase 3 for a long time.

The only left phase now is phase 2. Can the queue stay in phase 2 for a long time? The answer is yes. Except in a temporary state where multicast sessions grab a share of bandwidth from TCP sessions to achieve fairness, multicast sessions take no action in phase 2. Therefore, the behavior of TCP sessions usually decides where the queue will go in phase 2. In general, TCP sessions increase their rates slower and slower after the queue enters phase 2. This is because when the queue is building up, the RTT will keep increasing from increasing packet-queuing delays. With increasing RTT, TCP sessions slow down in increasing their rates. Now there are two cases. The first case is that the TCP receiver's advertised window is not large enough. In this case, TCP sessions may stop increasing their rates in phase 2. The queue will then stay in phase 2. The second case is that the TCP receiver's advertised window does not limit the rate increase of TCP sessions. In this case, TCP sessions will continuously increase their rates, although slower and slower, until the queue overflows. The queue then returns to phase 2 or temporarily visit phase 1 after TCP sessions cut their rates upon queue overflow. Therefore, in both cases the queue can stay in phase 2 for a relatively long time.

Fig. 3 shows the queue trend patterns when the queue is in balance. In general, there are four patterns: staying in phase 2, staying in phase 2 but visiting phase 3 and phase 1 from time to time, staying in phase 2 but visiting phase 3 from time to time, and staying in phase 2 but visiting phase 1 from time to time.

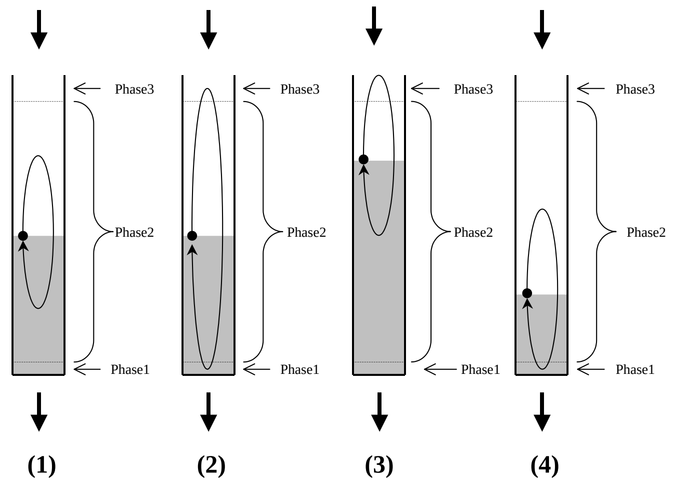

Fig. 3. Queue Trend Patterns When the Queue is in Balance

We now show that fairness is indeed achieved when the queue is in balance. There are two cases of unfairness. The first case is that the average session rate of TCP sessions is much higher than the average session rate of multicast sessions ($R_{TcpAvg} \gg R_{MctAvg}$). This kind of unfairness does not happen when the queue is in balance. This is because multicast sessions will release layers in phase 2 if their average session rate is lower than the average session rate of TCP sessions ($R_{MctAvg} < R_{TcpAvg}$). The second case of unfairness is that the average session rate of multicast sessions

TABLE I

SCHEME PARAMETERS

Parameter	Definition	Value Used
$F_{QuThresh1}$	the factor for the first queue threshold	0.0
$F_{QuThresh2}$	the factor for the second queue threshold	0.8
$F_{SlowDown}$	the factor in increasing the observation time in continuous layer releasing	1.5
$F_{BackOff}$	the factor in increasing the observation time upon congestion	4.0
R_{Block}	the rate difference ratio used in blocking a layer	0.2
I_{ChekQu}	the interval in checking queue state (seconds)	0.02
$T_{observe}$	the initial observation time for releasing a layer (seconds)	1.0

is much higher than the average session rate of TCP sessions ($R_{MctAvg} \gg R_{TcpAvg}$). This situation does not happen either if TCP sessions are not limited in their rate increase by their receiver's advertised window. With large enough receiver's advertised window, TCP sessions increase their rates continuously until the queue overflows. After the queue enters phase 3 in this process, multicast sessions will block some layers because their average session rate is significantly higher than the average session rate of TCP sessions. In this way, the average session rate of multicast sessions will finally decrease to become close to the average session rate of TCP sessions. In summary, when the queue is in balance, the average session rate of multicast sessions stays close to the average session rate of TCP sessions.

2) Proof of the fairness: In this section we analytically prove that the proposed scheme shares bandwidth fairly with competing TCP flows. An intuitive interpretation of fairness is that irrespective of the initial window values of the flows (i.e. their transmission rates) all flows sharing the bottleneck link must eventually end with identical window sizes or transmission rates when the flows reach steady state. Mathematically, we use Jain's fairness index [10] to quantify a notion of fairness. Consider a set of n flows where the window (or equivalently the rate) of the i^{th} flow is given by x_i. The Jain's fairness index F is then given by

$$F = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2} \tag{1}$$

which attains the value of 1 when the allocation is totally fair ($x_1 = x_2 = \cdots = x_n$). For ease of illustration, we consider a two flow case where a TCP session shares a bottleneck with a multicast session and note that the proof can be easily extended for multiple flow cases. The action of both these protocols can be described in terms of their response to congestion or the absence thereof, as dictated by their increase ($\mathcal{I}$) and decrease ($\mathcal{D}$) policies. Let each application of the increase or decrease policy be separated by R time units (we assume that these are the same for both TCP and the multicast session for ease of analysis). The behavior of the multicast flow with rate $x_1(t)$ at time t, with the corresponding rate of the TCP flow being $x_2(t)$ is then given by

$$\mathcal{I}: \quad x_1(t+R) \longleftarrow \begin{cases} x_1(t) + g & \text{if } x_1(t) < x_2(t) \\ x_1(t) & \text{otherwise} \end{cases} \tag{2}$$

$$\mathcal{D}: \quad x_1(t+R) \longleftarrow \begin{cases} x_1(t) & \text{if } x_1(t) < x_2(t) \\ x_1(t) - g & \text{otherwise} \end{cases} \tag{3}$$

where g is the size of a layer. Similarly, the behavior of the TCP flow can be modeled as

$$\mathcal{I}: \quad x_2(t+R) \longleftarrow x_2(t) + \alpha \tag{4}$$

$$\mathcal{D}: \quad x_2(t+R) \longleftarrow x_2(t) - \beta x_2(t) \tag{5}$$

with standard values of α and β being 1 and $1/2$ respectively.

To prove the fairness of our scheme, we use the following theorem from [11] which provides a general framework for proving the fairness of a class of congestion control protocols: *Two flows with window sizes (or equivalently rates) x_1 and x_2, $x_1 < x_2$, sharing a bottleneck link will eventually converge to and maintain a totally fair allocation of bottleneck link bandwidth if the following condition is satisfied after each application of an increase or decrease policy over a reasonable amount of time:*

$$\frac{\Delta x_1}{x_1} \geq \frac{\Delta x_2}{x_2} \tag{6}$$

where Δx_1 and Δx_2 represent the changes in the window or rate at the application of the increase and decrease policy for flow 1 and 2 respectively. An intuitive interpretation of the proof is that when the above condition is satisfied, the change in F corresponding to changes in x_1 and x_2 is positive regardless of its initial value and the system moves to a fair allocation.

We first consider the case when the rate of the multicast flow is less than that of the TCP flow, i.e., $x_1 < x_2$. Under these conditions, the application of the increase policy, we have $\Delta x_1 = g$ and $\Delta x_2 = 1$. Thus Equation 6 is satisfied since $g/x_1 > 1/x_2$ where inequality results from the fact that $g \geq 1$ and $x_1 < x_2$. For the decrease policy, $\Delta x_1 = 0$ while $\Delta x_2 = -x_2/2$. Thus Equation 6 is satisfied since $0 > -1/2$. The conditions for fairness are thus satisfied for both the increase and the decrease policy when $x_1 < x_2$.

We now consider the case where the rate of the multicast flow is greater than that of the TCP flow ($x_1 > x_2$). On the application of the increase policy, $\Delta x_1 = 0$ while $\Delta x_2 = 1$. Equation 6 is then satisfied since we now need $\Delta x_1/x_1 \leq \Delta x_2/x_2$ which in this case corresponds to $0 < 1/x_2$. The case with the application of the decrease policy is a bit more involved since an isolated application of the decrease policy worsens the fairness. Since each application of the decrease policy is followed by at least one increase policy (which, as we have already shown, increase the fairness), it suffices to show that over the subsequence of events between the two applications of the decrease policy, the fairness increases. In the general case, two applications of the decrease policy are spaced by a number of increase instances. Let the rates of the multicast and the TCP flows at the instant of the first of the two decrease instants be x_1 and x_2 and at the instant immediately following it be x_1' and x_2' respectively. Thus

$$x_1' = x_1 - g \tag{7}$$
$$x_2' = \frac{x_2}{2} \tag{8}$$

It is easy to verify that $x_1' \geq x_2'$ (where we do not allow the case $x_1' = 0$ to happen) and thus till the next application of the decrease policy, there is no change in x_1' (from Equation 2) resulting in $\Delta x_1 = -g$. However, the rate of the TCP flows increases so that it claims the freed rate g from the multicast session as well as the $x_2/2$ which was freed as a result of its own decrease policy. At this point, when the next decrease in rate occurs, the current rate is given by $x_2' + g + x_2/2 = x_2 + g$. Thus, Δx_2 is given by

$$\Delta x_2 = (x_2 + g) - x_2 = g \tag{9}$$

Intuitively, the above expression means that between each step taken towards fairness in this case, the multicast session reduces its rate by g while the TCP flow increases its rate by g thereby moving towards fairness. Thus we have $\Delta x_1/x_1 = -g/x_1$ and $\Delta x_2 = g/x_2$ and we again have $\Delta x_1/x_1 \leq \Delta x_2/x_2$ as required. This proves the fairness of the proposed multicast congestion control scheme. •

In the above analysis of fairness, we generally assumed that TCP sessions are not limited in their rate increase by their receiver's advertised window. Our conclusion is that the average session rate of multicast sessions is close to the average session rate of TCP sessions when the queue is in balance. What happens if TCP sessions have small receiver's advertised window or have poor throughput because of various reasons such as high link error rate? In this case, the average session rate of multicast sessions may be significantly greater than the average session rate of TCP sessions. In other words, each multicast session may use more bandwidth than each TCP session on average. This happens because the restriction of rate increase on TCP sessions causes them unable to use up their share of bandwidth. Consequently, the link is sparsely utilized and the queue stays in phase 1 before multicast sessions claim the free bandwidth. Each time when the queue stays in phase 1 for a period of time larger than $T_{observe}$, multicast sessions release a layer to use some of the free bandwidth. Eventually the free bandwidth will be utilized.

One thing we need to mention is that TCP is also in evolution. Many efforts have been made in improving the performance of TCP in different environments. Highly variable delays, link errors and hand-offs in satellite environments may decrease the performance of TCP significantly. Some work has been done to deal with these problems [12] [13] [14] [15] [16]. The TCP issue in satellite environments is beyond the scope of this paper; in this paper we only concentrate on the possible impacts of satellite environments on existing multicast congestion control schemes and our efforts in dealing with related problems. As shown above, with our scheme, multicast sessions will claim the free bandwidth spared by TCP sessions if TCP sessions can not use up their share of bandwidth on a bottleneck. Therefore, the performance of TCP in satellite environments, in general, will not affect the performance of our scheme.

B. Effectiveness in Satellite Environments

The effectiveness of our scheme in satellite environments stems from its following features. First, instead of depending on receivers in detecting congestion and adjusting their receiving rates, our scheme adjusts multicast traffic rate (i.e., blocks or releases multicast layers) right on the link where congestion occurs. Therefore, our scheme is not affected adversely by either the long link delays in satellite environments or the long pruning delay of the IGMP protocol. Second, our scheme uses the queue state on a bottleneck instead of the loss information at receivers in adjusting the number of multicast layers passing through the bottleneck. Therefore, link errors in satellite environments also can not decrease the performance of our scheme. Third, our scheme only has very limited control traffic overhead. In existing schemes, either poor coordination among receivers or the design of the scheme itself results in frequent grafting and pruning. Frequent grafting and pruning may produce a significant amount of control traffic overhead. In our scheme, although the receivers need to adjust the number of empty layers (i.e., to maintain a single empty layer), the adjustment is few because our scheme does not have frequent layer adjustment on a bottleneck. Furthermore, all receivers downstream from a bottleneck are well coordinated in our scheme by the multicast traffic, which is effectively controlled on the bottleneck. No layer-drop delay, limited control traffic overhead, and no penalty from link errors enable our scheme to work effectively and efficiently in satellite environments.

Another feature of our scheme is that a misbehaving receiver can neither benefit itself nor hurt other receivers, since the number of layers a receiver can receive with income data is solely decided by the bottleneck along the path from the multicast source to the receiver. In fact, if a receiver intentionally or accidently subscribes to too many layers, the number of layers that have data flowing into the receiver will not change; the bottleneck will block the excessive layers automatically. Furthermore, other receivers downstream from the bottleneck are not affected. The only consequence is that some limited bandwidth above the bottleneck is possibly wasted (see the next subsection for details).

C. Scheme Cost

In this subsection we analyze the cost of our scheme. The main cost of our scheme comes from retrieving some information from the passing-through traffic on a bottleneck. Specifically, the number of TCP sessions (N_{TcpSes}), the number of multicast sessions (N_{MctSes}), the number of layers of each multicast session ($N^i_{LiveLayer},\ 0 < i \leq N_{MctSes}$), the average session rate of TCP sessions (R_{TcpAvg}) and the average session rate of multicast sessions (R_{MctAvg}) are the information our scheme needs in its operation. All the information can be obtained by analyzing the addresses of passing-through packets. Since addresses have to be analyzed anyway in packet forwarding, the overhead introduced by our scheme in retrieving the above information is arguably not high. In fact, the forwarding process only needs to put the retrieved addresses of packets into a buffer and another separate process can analyze them to retrieve the needed information. One thing we need to mention is that in retrieving the information, some additional states are maintained in the router just above the bottleneck. These additional states are mainly the session addresses and counting variables. Among them, only the number of session addresses increases with the increase of the number of sessions passing through the bottleneck. Furthermore, these additional states only need to be maintained in the router just above the bottleneck.

Another type of cost for a network-assisted scheme may come from the interaction between network elements or between a network element and a host. With our scheme, there is no explicit interaction between network elements except those required for standard multicast routing. When a router blocks or releases a multicast layer, it does not notify any other router or any receiver explicitly. Instead, the receivers downstream from the bottleneck will detect the changes of the incoming traffic (i.e., layers being blocked or released) and drops or adds layers appropriately. Also because of this mechanism of coordination by traffic change, receivers do not interact with each other or with routers explicitly. All receivers downstream from a bottleneck are implicitly coordinated by the traffic coming from the bottleneck, while the traffic is effectively controlled by the router just above the bottleneck. Receivers use standard pruning and grafting to cooperate with routers, while routers block or release layers to adjust multicast traffic rate and to signal receivers for proper actions. Although this mechanism of coordination by traffic change eliminates the explicit interactions among routers and receivers, some overhead may be introduced by the empty layer maintained by a multicast session. We discuss this kind of overhead below.

When a receiver maintains an empty layer, some bandwidth above the bottleneck along the path reaching the receiver may be wasted if no other receiver above the bottleneck needs that layer. The waste of bandwidth can be significant if the sizes of the layers are exponentially spaced. Existing multi-layer multicast congestion control schemes usually use exponentially increasing sizes. This size planning gives a scheme the ability to cut the traffic rate multiplicatively by dropping a single layer. This is a conservative strategy adopted by end-to-end schemes to facilitate a quick alleviation of congestion without detailed knowledge of the severity of congestion and queue lengths at the bottleneck. However, our scheme facilitates the use of equal-sized layers instead of exponentially increasing size layers (which also allows for finer granularity of control on data rates) which reduces the penalty incurred from a blocked layer. This is possible because our scheme can detect congestion promptly and can also block several layers timely, since it deals with congestion right on each bottleneck of a multicast tree.

Although with its possible overhead, maintaining an empty layer is necessary and has its advantages. First, the bottleneck needs empty layers to increase the multicast traffic rate when free bandwidth is available. Second, grafting above the bottleneck is not required when an empty layer is released, so the delay in increasing multicast traffic rate is significantly reduced with maintained empty layers. Third, pruning and grafting overhead is reduced because of the reduced frequency of pruning and grafting. If the distribution of receivers is even and dense in a multicast tree, the bandwidth used by the empty layer of the multicast session above a bottleneck is, in general, efficiently utilized.

Another important point is that the amount of bandwidth that may be wasted by a multicast session is usually very limited on a link. With our scheme, a multicast session can not use an amount of bandwidth beyond its share on a link (assuming no free bandwidth on the link). Therefore, only the unused bandwidth inside the share of bandwidth for a session on a link can be wasted by the session. Specifically, the maximum bandwidth that may be wasted by session m at link i is limited to:

$$Max(BW^{m}_{WastedLink_i}) \leq Min\{(BW_{link_i}/N_{SesLink_i} - AvgLayerSize \times N^{m}_{LiveLayer}),\ \ Size^{m}_{EmptyLayer}\}$$

where:
BW_{link_i}: the bandwidth of link i
$N_{Seslink_i}$: the number of sessions passing through link i
$AvgLayerSize$: the average layer size of session m
$N^{m}_{LiveLayer}$: the number of living layers of session m at the bottleneck
$Size^{m}_{EmptyLayer}$: the size of the empty layer of session m at the bottleneck

Furthermore, even if a small amount of bandwidth is wasted above the bottleneck along the path reaching the receiver, the receivers downstream from the bottleneck are not affected. The situation is much different in other existing schemes. The amount of bandwidth that can be wasted by their control traffic has no theoretical limit. Furthermore, their control traffic directly reduces the bandwidth available to receivers downstream from the corresponding bottleneck.

The last thing is that all the operations of our scheme in retrieving session information, in general, do not affect the queuing, scheduling, or forwarding policy of existing networks, so our scheme will not affect the structure of existing networks and the applications on them if our scheme is deployed. One possible exception is for identifying a TCP session. Since port numbers together with addresses identify a TCP session, the TCP header has to be checked if the real number of TCP sessions passing through a link is wanted. However, there is a possible alternative: only using address information in identifying a TCP session. In this way, all TCP sessions with the same source address and the same destination address will be regarded as one "big" TCP session. It needs further investigation to ascertain if this kind of aggregation is better for fairness from a global point of view, but one thing is worth mentioning: this kind of aggregation can prevent a pair of hosts from stealing bandwidth from multicast sessions on a bottleneck by initializing a large number of TCP sessions.

IV. Simulation Results

In this section we present our simulation results. The topology for our simulations is shown in Fig. 4. One of the example satellite systems given in NS2 is used in our simulations. It is a broadband LEO constellation with orbital configuration similar to that of Iridium.

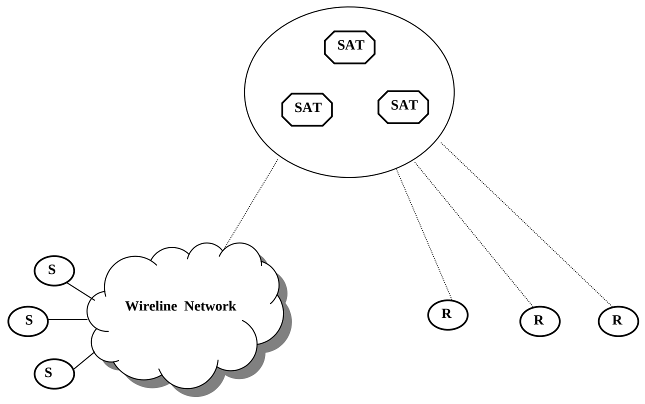

Fig. 4. The Simulation Topology

We chose this wireline-cum-satellite topology because it is rare that all links of a computer network are purely satellite links in reality. In addition, this topology enables us to let all traffic of test sessions go through a common satellite link. Therefore, the behavior of our scheme can be easily observed. In the topology, each link in the wireline network is configured to have a bandwidth of 5Mb/s. For test traffic, there are 5 test sessions: 2 multicast sessions and 3 TCP sessions. The source of each session is in the wireline network, while the destination of each session is in the purely satellite-connected network. Each multicast session has 15 layers and each layer has a rate of 50Kb/s. With these configurations, the bottleneck in our simulations is the upstream satellite link between the wireline network and the satellite-connected network.

We considered 3 scenarios in testing our scheme. In the first scenario, all sessions start and stop at the same time. In the second scenario, two TCP sessions start later than the other sessions. In the third scenario, two multicast sessions start later than the TCP sessions. In addition, we also provide a reference scenario in which a well-known multi-layer multicast congestion control scheme, RLM [1], is adopted for multicast congestion control. Finally, we give some sample queue trend patterns for our scheme. The values used in our experiments for the parameters of our scheme are shown in Table I.

A. Scenario 1: All Sessions Starting and Stopping at the Same Time

In this scenario, all sessions start at the beginning of the simulation and stop at the 1500th second, so all of them compete for bandwidth all the time. The simulation results are shown in Fig. 5 and Fig. 6. The former shows the number of layers and the throughput of each multicast session, while

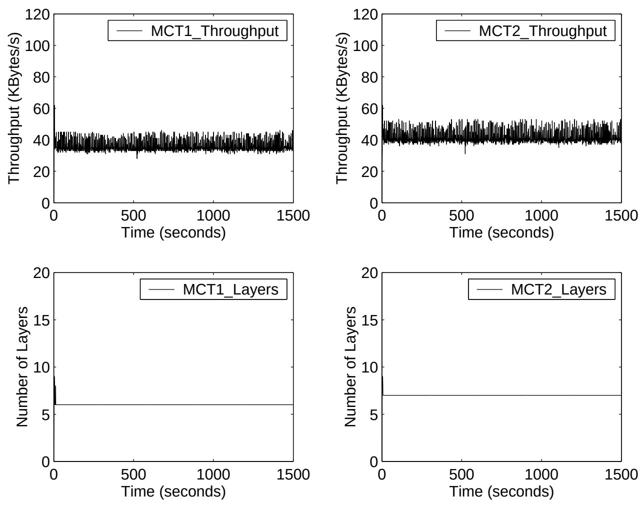

Fig. 5. Throughput and Number of Layers of Multicast Sessions

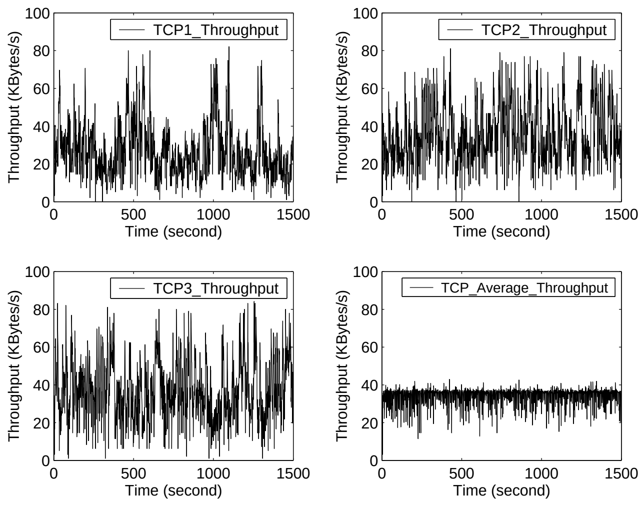

Fig. 6. Average Session Throughput and Individual Session Throughput of TCP

the latter gives the throughput of each individual TCP session and the average session throughput of TCP sessions.

As shown in these figures, both the average session throughput of TCP sessions and the average session throughput of multicast sessions are close to 40 KBytes/s all the time, so fairness among competing sessions, in general, is achieved. If we take a closer look at these figures, we can find two things. First, the average session throughput of TCP sessions is a little lower than the average session throughput of multicast sessions. This is because of the scheme design. For the consideration of stabilizing multicast traffic, our scheme blocks multicast layers only if the average session rate of multicast sessions is higher than the average session rate of TCP sessions by a ratio threshold. The second observation is that there is a small difference between the throughput of the two multicast sessions. A difference of a layer between the throughput of multicast sessions sharing a bottleneck is possible with multi-layer multicast congestion control schemes, since their unit in adjusting multicast traffic rate is a layer. For simplicity, we will not mention these two phenomena again in the following scenarios.

Therefore, fairness among competing sessions, in general, is achieved in this scenario. Another point we can note in Fig. 5 is that the number of layers for each multicast session is fairly stable after the initial adjustment. This indicates that the queue enters balance quickly and stays there for the rest of the time.

B. Scenario 2: TCP Sessions Joining Existing Multicast and TCP Sessions

In this scenario, we test if late TCP sessions can get a fair share of bandwidth with our scheme. One TCP session joins two multicast sessions and one TCP session at the 500th second, and then another TCP session joins them at the 1000th second. The simulation results are shown in Fig. 7 and Fig. 8.

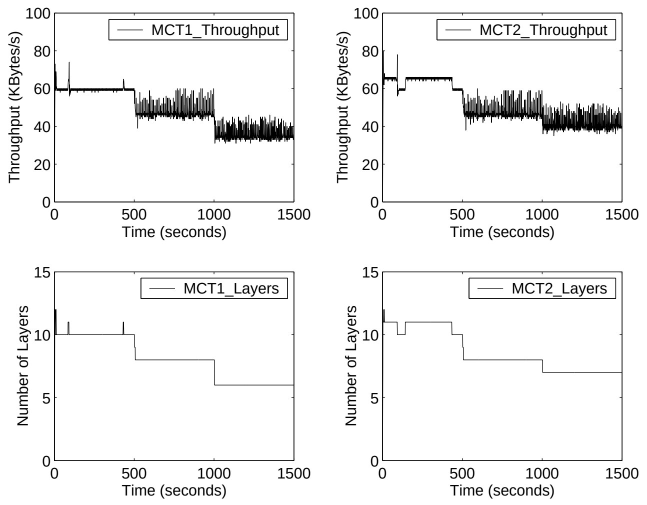

Fig. 7. Throughput and Number of Layers of Multicast Sessions

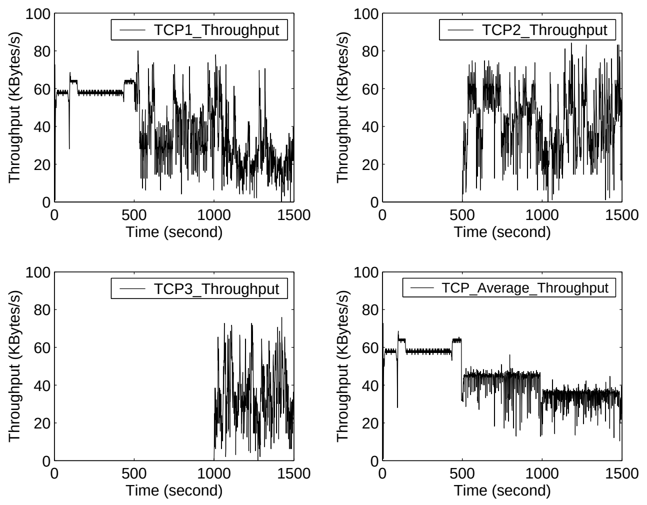

Fig. 8. Average Session Throughput and Individual Session Throughput of TCP

As shown in these figures, in the first 500 seconds there are 3 sessions and each session gets a throughput close to 60 KBytes/s. When a TCP session starts at the 500th second, each existing session releases a right share of bandwidth for the new session. Each session then gets a new throughput close to 45 KBytes/s. This bandwidth sharing pattern persists until another TCP session starts at the 1000th second. When the second TCP session becomes alive, all existing sessions release a right share of bandwidth again. So fairness is still kept: each session has a throughput close to 35 KBytes/s now.

Therefore, late TCP sessions can grab a fair share of bandwidth from existing sessions with our scheme. Furthermore, the number of layers for each multicast session is still fairly stable, although with a little more small adjustments than in the previous scenario. For other existing multi-layer multicast congestion control schemes, most of them are not able to release a right share of bandwidth for late arriving TCP or multicast sessions.

C. Scenario 3: Multicast Sessions Joining Existing TCP Sessions

Lastly, we check if late multicast sessions can get and only get a fair share of bandwidth from existing sessions. In our simulations, one multicast session starts at the 500th second, while the other multicast session starts at the 1000th second. Fig. 9 and Fig. 10 show the simulation results.

As these figures show, before the multicast sessions start, each TCP session has a throughput close to 60 KBytes/s. After the first multicast session starts at the 500th second, it gets a throughput close to 45 KBytes/s. At the same time, the average session throughput of TCP sessions falls to be close to 45 KBytes/s. Then, at the 1000th second the other multicast session starts. The same situation happens: the new multicast session gets a right share of bandwidth and then each session has a throughput close to 35 KBytes/s.

Therefore, late multicast sessions get and only get a fair share of bandwidth with our scheme. In addition, the number of layers for each multicast session is still fairly stable in this scenario.

D. Comparison with existing schemes

For reference, in this section we show the simulation results of RLM [1], which is one of the most well-known multi-layer multicast congestion control schemes in the literature. For multiplicatively reducing traffic rate in congestion, RLM usually uses exponentially increasing sizes for layers. In our simulations, RLM uses 5 layers and the size of each layer is 32, 64, 128, 256 and 1024 kb/s, respectively, from layer 1 to layer 5. In addition, the three TCP sessions and the two multicast sessions start at the beginning of the simulation and stop at the 1500th second. The simulation results are shown in Fig. 11 and Fig. 12.

By comparing Fig. 11 and Fig. 12 with Fig. 5 and Fig. 6, respectively, we can find that the proposed scheme achieves much higher stability in both number of layers for multicast sessions and throughput for all sessions. This demonstrates that the congestion on the bottleneck is more appropriately controlled with our scheme. So our scheme uses the bandwidth of the bottleneck more efficiently than RLM. In addition, these figures show that our scheme achieves much better fairness among the competing sessions than RLM. With our scheme, each session gets a throughput close to 40 KBytes/s all the time. With RLM, the throughput of the two multicast sessions changes severely in the simulation period. Furthermore, the throughput of the two multicast sessions is either much lower or much higher than the throughput of the three TCP sessions.

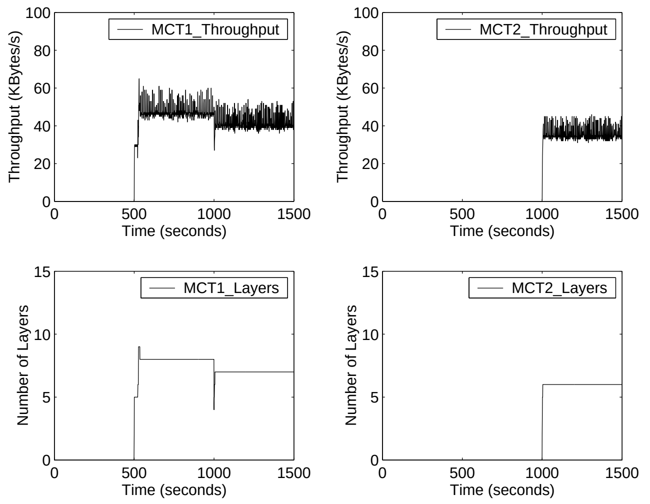

Fig. 9. Throughput and Number of Layers of Multicast Sessions

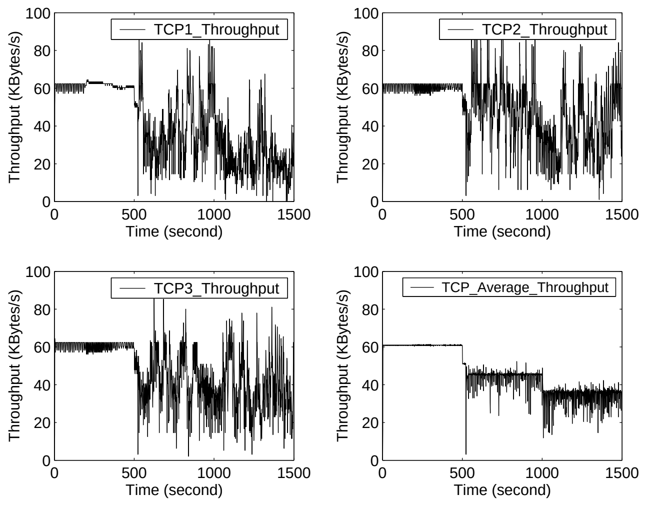

Fig. 10. Average Session Throughput and Individual Session Throughput of TCP

E. Queue Trend Patterns

This subsection presents some queue trend patterns to further support the fairness analysis in Section III. Fig. 13 shows some samples of the queue phases in scenario 1. The upper part of Fig. 13 shows the long-term queue trend patterns, while the lower part gives more details. As shown in this figure, the queue stays in phase 2 most of the time and visits phase 3 and phase 1 from time to time. The reason that the queue can stay in phase 3 for a short time is that our scheme does not

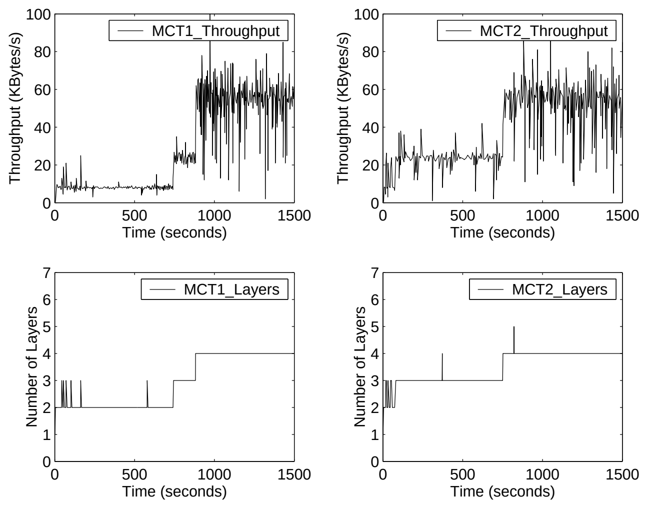

Fig. 11. Throughput and Number of Layers of Multicast Sessions

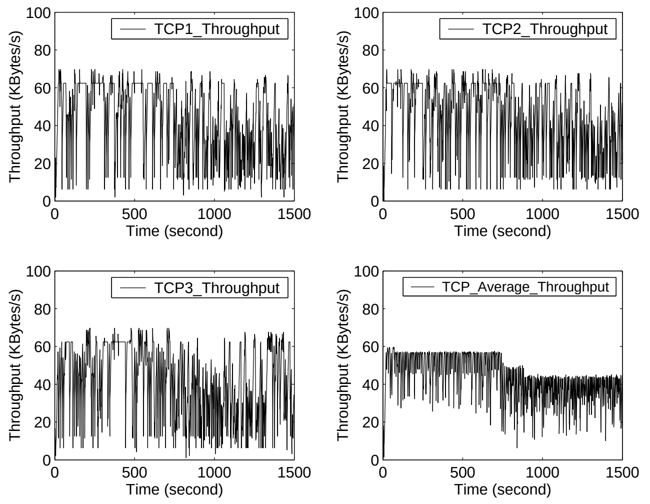

Fig. 12. Average Session Throughput and Individual Session Throughput of TCP

block layers in phase 3 if the average session rate of multicast sessions is not higher than the average session rate of TCP sessions by a ratio threshold (for stabilizing multicast traffic). However, a TCP session needs at least a round trip time to respond to congestion. Therefore, after the queue overflows from the increasing rates of TCP sessions, it takes some time for the TCP sessions to cut their rates and bring the queue back to phase 2.

V. Conclusions

This paper presents a multi-layer multicast congestion control scheme that is suitable for satellite environments and overcomes most of the disadvantages of existing schemes. Link errors in satellite environments can cause existing schemes to wrongly drop some layers, which causes problems in traffic stability, fairness among competing sessions, and bandwidth utilization. While the layer-drop delay problem is already serious in wireline networks, the long delays of satellite links will further increase the delay in dropping a layer for existing

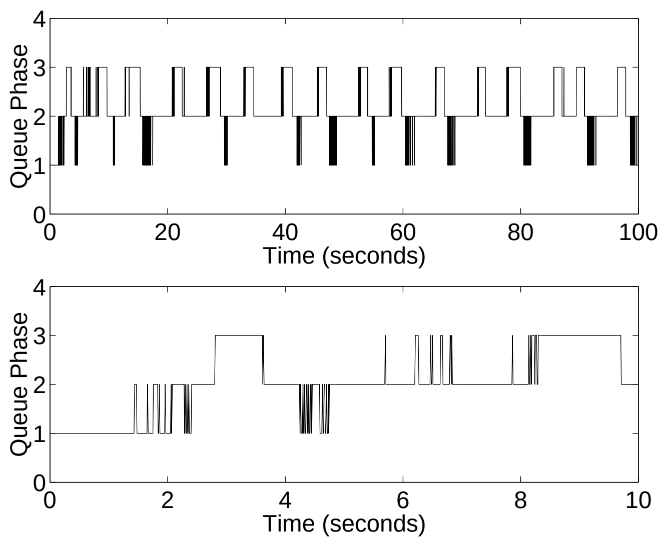

Fig. 13. The Demonstration of Queue Trend Patterns

schemes. Our scheme does not have these two problems. The way in which our scheme overcomes these problems is to deal with congestion right at the site where the congestion occurs. This new mechanism also saves considerable control traffic overhead caused by frequent grafting and pruning in existing multi-layer schemes. Meanwhile, for avoiding the explicit interaction among routers and receivers as in a general network-assisted scheme, the proposed scheme may introduce the overhead of an empty layer for a multicast session. Another feature of our scheme is the good fairness among competing sessions, including TCP sessions while most existing schemes still have problems in fairly sharing bandwidth with TCP sessions. Although some existing schemes have made some progress in this aspect, they may not be suitable for some applications where the scheduling for data transmission is restricted. Along with all of its advantages, our scheme, in general, does not impose any significant change on the queuing, scheduling, or forwarding policy of existing networks.

References

[1] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Proc of ACM SIGCOMM.*, Aug 1996, pp. 117–130.

[2] L. Vicisano, L. Rizzo, and J. Crowcrof, "Tcp-like congestion control for layered multicast data transfer," in *Proc of IEEE INFOCOM.*, San Francisco, March 1998, pp. 996–1003 Vol. 3.

[3] J. Byers, M. Luby, M. Mitzenmacher, and Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proc of ACM SIGCOMM.*, Sept 1998, pp. 56–67.

[4] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver, "Flid-dl: Congestion control for layered multicast," in *Proceedings of NGC 2000*, November 2000, pp. 71–81.

[5] W. Tan and A. Zakhor, "Video multicast using layered fec and scalable compression," *IEEE Trans. on Circuits and Systems for Video Technology*, pp. no. 3, Vol. 11, February 2001.

[6] M. Luby and V. Goyal, "Wave and equation based rate control using multicast round trip time," in *Proc of ACM SIGCOMM.*, Pittsburgh, Pennsylvania, USA., Aug. 2002, pp. 191–204.

[7] R. Gopalakrishnan, J. Griffioen, G. Hjalmtysson, and C. Sreenan, "Stability and fairness issues in layered multicast," in *Proceedings of the NOSSDAV*, June 1999, pp. 31–44.

[8] A. Legout and E. W. Biersack, "Pathological behaviors for rlm and rlc," in *Proceedings of the NOSSDAV*, North Carolina, USA, June 2000, pp. 164–172.

[9] J. Peng and B. Sikdar, "Routing based video multicast congestion control," in *Proceedings of IFIP/IEEE MMNS*, Santa Barbara, CA, Oct. 2002, pp. 328–340.

[10] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," in *DEC Research Report TR-301*, Hudson, MA, September 1984.

[11] N. Sastry and S. Lam, "A theory of window-based unicast congestion control," in *Proceedings of IEEE ICNP*, Paris, France, November 2002, pp. 144–154.

[12] M. Allman, S. Floyd, and C. Patridge, "Increasing tcp's initial window," in *Tech. Rep. RFC 2414 (Experimental)*, Sep. 1998.

[13] M. Allman and D. Glover, "Enhancing tap over satellite channels using standard mechanisms," in *tech. rep., TapsAT Working Group, Internet Engineering Task Force*, Sep. 1998.

[14] R. Durst, a. Miller, and E. Travis, "Tcp extensions for space communications," in *in Proceedings of MOBICOM*, Nov. 1996, pp. 15–26.

[15] K. Scott and S. Czetty, "Improving tcp performance over mobile satellite chennels: The ackprime approach," in *in Proc. of Workshop on Satellite Networks: Architecture, Applications, and Technologies*, June 1998, pp. 509–516.

[16] T.R.Henderson and R. Katz, "Satellite transport protocol (stp): An sscop-based transport protocol for datagram satellite networks," in *2nd International Workshop on Satellite-based Information Services (WOSBIS)*, Oct. 1997, pp. 23–34.