

# Delay-based Flow Control for Layered Multicast Applications

*Mathias Johanson*

Framkom Research Corporation for  
Media and Communication Technology  
mathias@framkom.se

## **Abstract**

*This paper presents an approach to flow control for real-time, loss-sensitive, layered multicast applications. The fundamentals of flow control for multicast applications are related and a novel delay-based flow control algorithm is introduced. The basic idea of the algorithm is to react to incipient congestion before packet loss occurs by monitoring variations in the one-way packet delay from sender to receivers. By using a hierarchical representation of the real-time data in combination with a layered multicast transmission model the flow control algorithm can be implemented entirely in the receivers. Furthermore, by constraining the bandwidth of the layers to a well-defined rate, the congestion control can be accomplished almost entirely without packet loss. This is particularly suitable for real-time multimedia conferencing applications that are inherently multipoint and loss-sensitive. The performance of the flow control algorithm in terms of link utilization, inter- and intra-protocol fairness, session scalability and loss probability is evaluated through extensive simulation.*

## **1 Introduction**

One of the reasons why the Internet has been so successful in supporting large numbers of simultaneous users is the ability of the network protocols to adapt to changing conditions. Specifically, the transport protocol used for most Internet traffic, TCP, includes a flow control algorithm that adapts the packet transmission pace of the sender so as not to congest the network [1]. The algorithm tries to experimentally find the optimal transmission rate by gradually increasing the rate until packet loss is experienced. However, delay sensitive applications like audio and video conferencing tools do not use TCP because of its poor real-time properties. Rather, these applications use the UDP and RTP protocols, leaving the flow control entirely to the application. In point-to-point configurations, flow control can be implemented by utilizing a rate adaptive coding algorithm, wherein feedback from the receiver is used to periodically adjust the media encoding parameters to match the available bandwidth [2]. For multipoint configurations, where the receivers typically are subject to disparate bandwidth limitations, a more sophisticated arrangement is needed. One approach is to use audio/video gateways that transcode the media to match the available bandwidth of each receiver. This has the drawback of requiring specialized network configurations and is inherently not very scalable. Another approach is to use a layered multicast transmission scheme wherein a hierarchical representation of the data is transmitted to a set of multicast group addresses that can be subscribed to individually by the receivers. The number of groups subscribed to determines the bandwidth utilization for each receiver and consequently the quality of the decoded media. In order for multipoint real-time multimedia applications to be realized on a large scale a flow control algorithm is needed that can adapt the bandwidth of the multicast flows to the network and host resources available for each independent receiver. Since real-time multimedia streams are sensitive to packet loss it is desirable with a flow control algorithm that can detect congestion before packet loss occurs. For ease of deployment in existing network environments the flow control should ideally not be dependent on changes to network routers or switches.

## **2 Flow Control Algorithms for Layered Multicast**

Flow control for layered multicast applications is implemented solely in the receivers. By joining and leaving multicast groups as the network load changes the receivers can dynamically adapt to the available

bandwidth. The decision of when to join groups, leave groups or remain at the same level is the task of the flow control algorithm. Several approaches have been suggested:

A technique generally referred to as *receiver-driven layered multicast* (RLM) was proposed by McCanne, Jacobson and Vetterli [3]. In this scheme the receivers periodically perform what is known as a *join experiment*, wherein a receiver tentatively joins an additional multicast group and monitors packet loss to determine whether the additional bandwidth causes congestion. To avoid the implosion of join experiments that would result if all in a potentially large group of receivers performed their join attempts independently, the experiments are co-ordinated through a procedure called *shared learning*.

Vicisano, Rizzo and Crowcroft elaborated on this scheme by introducing the concept of *synchronization points* [4]. In this model receivers are only allowed to perform join experiments immediately after receiving a synchronization packet from the sender. Synchronization packets are sent periodically as flagged packets in the encoded media stream. This proves to be more scalable than the shared learning algorithm of RLM.

The problem with these algorithms is that they use packet loss as a congestion detection signal. Since there is no corresponding signal when the network gets unloaded the applications must repeatedly perform join experiments to probe for available bandwidth. Packet loss caused by the failed join experiments will negatively impact the quality of the received data, not only for the member performing the experiment, but for each member located behind the same bandwidth bottleneck. The problem is further aggravated by the fact that the pruning of the reverse data path to the sender after a multicast leave operation can take a substantial amount of time (up to a few seconds), which means that the congestion caused can be relatively long-lasting.

What is needed is a way of telling that the network is becoming congested before packet loss is experienced. At the onset of congestion queues start to build up at network routers leading to an increased end-to-end delay. Several congestion avoidance algorithms for TCP (most notably *TCP Vegas* [5]) have been proposed based on reacting to changes in the round-trip time (RTT) from a segment of data is sent until it is acknowledged by the receiver [5, 6, 7]. Wu et al. proposed a layered multicast transmission architecture called *ThinStreams* that, in the spirit of TCP Vegas, uses the difference between the expected throughput and the actual throughput as a means to detect congestion [8]. To calculate the expected throughput the ThinStreams algorithm requires a constant bitrate for each multicast layer. This paper suggests an approach to layered multicast congestion avoidance based on direct measurements of packet delay variations. Unlike the ThinStreams approach it does not require a constant bitrate for the layers and hence imposes less restrictions on the layered media encoding.

### **3 Delay-based Layered Multicast Flow Control**

In order for the flow control algorithm to be able to respond to congestion before packet loss occurs, the variations in packet transmission delay can be used to detect congestion. An increasing delay indicates that router buffers are filling up and must be responded to by lowering the effective bandwidth. Similarly, a delay that has decreased below some threshold indicates that it might be possible to increase the bandwidth. To avoid packet loss the increase in bandwidth resulting from joining an additional group must be small enough for the network to buffer the excessive packets for the time it takes the receivers to detect the congestion and respond to it by leaving the group. This time is prolonged by the fact that the packet forwarding will proceed at multicast routers until the prune message of the leave operation is propagated back through the reverse multicast path. By carefully assigning an upper limit to the bandwidth of each layer (corresponding to a multicast group), packet loss as the result of joining an additional group can be avoided. To compute this bandwidth limit, assume that  $Q$  is the minimum queue size in use on the network and that  $L$  is the leave latency. Then the bandwidth limit  $B$  is

$$B \leq \frac{Q}{L}.$$

If we conservatively assume  $Q$  to be 5 Kbytes and  $L$  to be 2 seconds we get a bandwidth limit of 20 kilobits per second (kbps). The organization of data into layers at the transmitter should thus be made with a granularity of approximately 20 kbps. For real-time multimedia data this granularity is probably sufficiently small since the improvement in perceived quality by a refinement signal in the magnitude of 20 kbps is likely to be rather moderate (at least for video).

Given the above data organization and the layered multicast transmission architecture, what we now need is a way to monitor variations in packet delay. Recall that in TCP Vegas the round trip time is used to measure the variations in throughput. For multicast transmission, however, a round trip delay cannot be computed since the network path from sender to a receiver is not generally the same as the path from the receiver to the sender and thus cannot give a reliable measure of the buffering in the multicast data path. Nevertheless, the variations in transmission delay can be measured by a scheme involving timestamping the packets at the transmitter and clocking the arrival times of packets at the receivers.

### 3.1 Variable Transmission Delay Estimation

The one-way transmission delay from a source to a receiver can be seen as consisting of two parts; the fixed propagation delay and the variable delay due to buffering. The variable delay that interests us can be determined in the following way.

Let the source put a timestamp in every packet that reflects the *sending time* of that packet. Then the variable delay for packet  $i$ ,  $\delta_i$ , is

$$\delta_i = (r_i - r_0) - (t_i - t_0),$$

where  $r_i$  and  $t_i$  are the arrival and sending times of packet  $i$  respectively. Note that the delay calculations are performed only by the receiver and that the values of  $t_i$  are determined from the timestamp in packet  $i$ . For the algorithm to give a reliable estimation of the variable delay the first (reference) packet must be transmitted when the network is uncongested, that is  $\delta_0 = 0$ . A reference packet with a non-zero variable delay will result in negative variable delays once the network gets uncongested. This is an indication that the values of  $t_0$  and  $r_0$  must be reassigned (i.e. a new reference packet is chosen).

The RTP protocol that is used to fragment audio and video into UDP packets defines a packet header that includes a timestamp field, primarily intended to be used for things like playout scheduling and cross-media synchronization. The recommended clock frequency of the RTP timestamps is 90 kHz for video content and 8 kHz for audio [9]. The variations in transmission delay are typically in the order of 10 to 100 ms, so both clock frequencies are sufficiently high resolution for the delay estimation. (For example, a 10 Kbytes router buffer and a wire speed of 1 Mbps gives a maximum delay of 80 ms.)

To prevent measurement noise from impacting the join/leave decision algorithm, the packet delay estimation should be calculated as a running average over a number of measurements. That is, the delay estimation for the  $i$ :th packet,  $\hat{\delta}_i$ , is given by

$$\hat{\delta}_i = \frac{1}{N} \sum_{k=0}^{N-1} \delta_{i-k},$$

where  $N$  is the number of delay measurements used to compute the average. In the simulations presented in this paper, a value of  $N=20$  was used.

Note that the algorithm relies heavily on the fact that the sender's and receiver's system clocks are isochronous (that is, that they tick at the same speed). This could potentially be a serious deployment problem, since workstation clocks are frequently badly tuned. Note also that the algorithm does not require the clocks to have the same conception of absolute time. The issue of clock synchronization is beyond the scope of this paper, but techniques exist to synchronize clocks (both in terms of absolute time and clock frequency) down to microsecond precision [10].

### 3.2 Fairness

In order for layered multicast applications to be successfully realized in existing network environments it is important that the flow control algorithm adjusts the rate of the traffic so that the application competes in a fair way for bandwidth with other applications. To this end one can distinguish three different fairness issues that can be considered crucial:

1. fairness among members of the same layered multicast session,
2. fairness among different sessions of the same layered multicast application,
3. fairness to TCP.

Fairness among members of the same session and among members of different layered multicast sessions can be realized by adjusting the threshold delay values used in the algorithm to decide whether to join multicast groups, leave multicast groups or remain at the same subscription level. By decreasing the leave threshold and the join threshold with increasing layer numbers, receivers at lower subscription levels will be more inclined to joining new layers and less inclined to dropping layers compared to receivers at higher subscription levels. This means that on a heavily loaded network connection with many competing sessions the receivers subscribed to more layers will be more responsive to increased packet delays and hence will make leave decision sooner than receivers at lower subscription levels. Similarly, at decreasing network load receivers at lower subscription levels will join groups before receivers at higher levels. This will lead to a fair sharing of the available bandwidth between the members of a session and between the sessions, provided that all sessions use the same flow control algorithm.

Fairness to TCP's flow control is important since the bulk of network applications in use on the Internet are based on TCP. The throughput of a TCP session can be shown to be inversely proportional to the product of the round trip time (RTT) and the square root of the packet loss rate [11]. Since the throughput of the layered multicast flow control presented in this paper is independent of the packet loss rate and the round trip time the concept of fairness to TCP is not well-defined. Furthermore, since the types of applications targeted by the multicast flow control is very different from the "typical" TCP application, the relative performance exhibited by competing TCP sessions is not immediately appropriate. For instance, two TCP sessions with different RTTs will allocate the bandwidth of a shared bottleneck unevenly. While this "unfairness" can be motivated in the TCP case it does not make much sense for two participants of a multicast videoconference, sharing a bandwidth bottleneck, to receive the video at different rates depending on the distance to the sender. The important point to be stressed is that real-time multimedia data need to be rate-controlled in some way in order to coexist with TCP on congested links. This behavior is sometimes referred to as *TCP-friendliness*.

### 3.3 The Join/Leave Decision Algorithm

The flow control algorithm implemented by each receiver of a layered multicast session uses the measured queuing delay,  $\hat{\delta}_i$ , as indication to whether the layer subscription level should be increased or decreased. By considering not only the magnitude of the delay but also the rate of change, the algorithm can respond

earlier to impending congestion. Since the algorithm responds to congestion by leaving a multicast group, the effect of lowered bandwidth is not manifested until the multicast delivery-tree is pruned back to the sender. Thus, in order to be able to respond in time, the algorithm needs to predict the congestion level at some time ahead determined by the leave latency. If  $y(t)$  is the queuing delay at time  $t$  and  $L$  is the leave latency then the queuing delay at time  $t + L$  can be predicted by

$$y(t + L) = y(t) + Ly'(t).$$

Now, in order to prevent loss,

$$y + Ly' < M,$$

where  $M$  is the maximum queuing delay in the network. The value of  $M$  can be experimentally learned by initializing it to a conservatively small value and adjusting it whenever a larger delay is experienced. The leave latency can also be found experimentally by using the algorithm described in [8]. Alternatively, a preconfigured upper limit can be used. The algorithm continually computes  $y(t+L)$  and whenever the value is above a certain limit (the leave threshold) a layer is dropped. To decide when to join an additional layer the algorithm uses the value of  $y(t)$  directly, instead of the predicted  $y(t+L)$ . This asymmetry is due to the fact that the join decisions should not be made in a way that keeps the network in a constantly congested state. Whenever the value of  $y(t)$  is below the join threshold an additional multicast group is subscribed to.

In order to ensure inter- and intra-session fairness, as discussed in section 3.2, the join and leave thresholds should depend on the layer subscription level. The threshold functions are designed in a way that assures that all members of the same session sharing a bandwidth bottleneck eventually converge to roughly the same number of layers. Following the discussion in section 3.2 it is clear that both the join and leave thresholds should decrease with increasing layer subscription level. In the current implementation the join and leave threshold values are calculated using functions that decrease quadratically with the number of layers joined. The range of the join threshold function is from zero to 75 percent of the maximum delay whereas the range of the leave threshold is from 65 to 100 percent. The appropriateness of using these functions and parameter values were determined experimentally from simulation results.

### 3.4 Scheduling the Join/Leave Operations

Since the flow control algorithm is designed to detect and respond to congestion before packet loss occurs there is no need to synchronize the join operations from different receivers of the same session. The situation is different for algorithms that detect congestion from packet loss, since uncoordinated join attempts in this case will lead to constant congestion and packet loss. However, members of different sessions sharing the same bottleneck link can cause packet loss if they join new groups simultaneously. This is because the aggregate bandwidth change can be larger than what the network can buffer if many receivers join layers at the same instant in time. To prevent this from happening the join operations performed by receivers from different sessions need to be decorrelated. Since a strict decorrelation is hard to realize without negatively affecting scalability and convergence time, a reasonable approximation can be achieved by scheduling the join operations using a pseudo-randomized timer. By having the timer interval increase as more layers are joined, the applications are allowed to converge relatively fast to a reasonable quality level.

Since the leave operation does not have the desired effect (of lowered congestion) until all members subscribed to the same layer leave, the leave operations for members of the same session at the same subscription level should ideally be synchronized. However, all members sharing a bandwidth bottleneck will experience the same variations in packet delay and therefore the leave operations will be reasonably synchronized automatically if only the leave decisions are scheduled frequently enough. Given that the

effect of the leave is not manifested in a lowered packet delay until the multicast tree is pruned back to the sender, the receivers must defer their next leave decision for a time equal to the leave latency to avoid dropping more than one layer in response to the same congestion signal. This is easily implemented with a hold-down timer after a leave.

### 3.5 The Delay-based Layered Multicast Flow Control Algorithm

The algorithm lined out above can be described by the following pseudo-code segment.

```

y = current queuing delay
y' = rate of change of y
M = maximum delay
n = number of layers joined
N = maximum number of layers
t = current time
L = leave latency

join_threshold := 0.75*M*(1 - sqrt(n/N))
leave_threshold := M*(0.65 + 0.35*(1 - sqrt(n/N)))
if ( y + L*y' > leave_threshold and t > leave_timeout )
    drop_layer(n)
    n := n-1
    leave_timeout := t + L
if ( y < join_threshold and t > join_timeout )
    add_layer(n+1)
    n := n+1
    join_timeout := t + (n/N + random(0, 0.5))*L

```

The procedures `add_layer(n)` and `drop_layer(n)` are assumed to implement the joining and leaving of multicast groups corresponding to layer  $n$ . The `random(x, y)` function is assumed to return a random value between  $x$  and  $y$ .

## 4 Simulation Results

The behavior of the flow control algorithm described in chapter 3 has been simulated using the network simulator *ns* [12]. The topologies used for the simulations are depicted in Figure 1. Each simulation used a packet queue length of 20 packets and a dense multicast routing protocol. The transmission delays on the links were 10 ms unless otherwise noted.

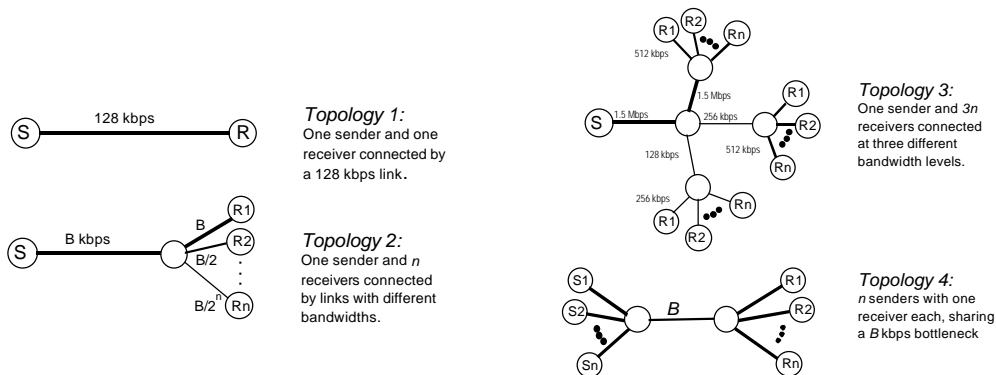
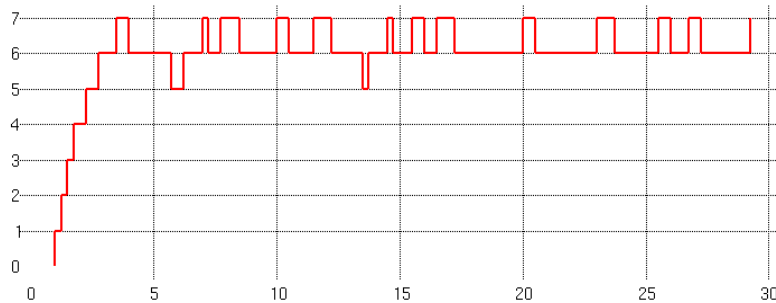


Figure 1: Topologies used in simulations

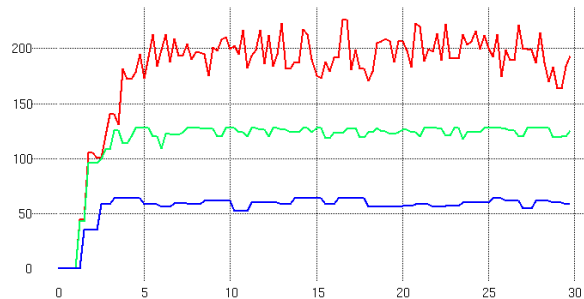
#### 4.1 Link Utilization and Intra-session Fairness

The first simulation was performed using the simplest possible topology; one layered multicast sender and one receiver connected with a point-to-point link (topology 1 in Figure 1). The aim of the simulation was to test the link utilization on a network connection with no intervening traffic. The sender transmits ten layers of approximately 20 kbps each, resulting in a total bandwidth requirement of 200 kbps. The link bandwidth is 128 kbps, so theoretically the receiver should be able to receive six layers ( $6 \cdot 20 = 120$  kbps) without congesting the network. Figure 2 shows how the algorithm quickly joins seven layers before the network becomes congested. Then two layers are dropped in response to increased packet delay and throughout the simulation the receiver oscillates between the sixth and seventh layer. This shows that the algorithm indeed utilizes the available bandwidth as expected. The simulation was conducted without packet loss.



**Figure 2: Number of multicast groups joined by the application**

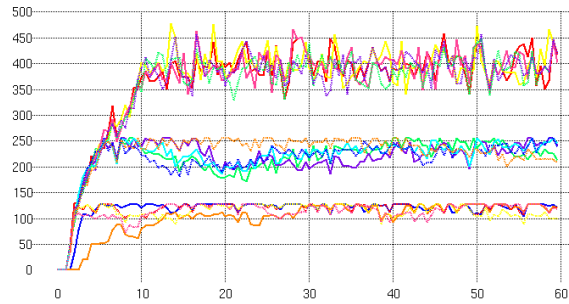
A slightly more complex situation is given by topology 2. Here  $n$  receivers are connected at different link speeds to a sender with the same characteristics as in the previous simulation. This topology was used to test the algorithm's ability to converge to different bandwidths in a heterogeneous network environment. Figure 3 shows the bandwidth allocation resulting from a simulation with three receivers ( $n=3$ ) and a 256 kbps capacity of the shared link ( $B=256$ ). The network path to receivers R1, R2 and R3 were 256, 128 and 64 kbps respectively. The expected result is that R1 should be able to receive all ten layers of the transmission, whereas R2 and R3 should converge to six and three layers respectively. The results of this simulation indicate that different receivers of the same session can converge to different bandwidths. No packet loss was experienced on any of the links.



**Figure 3: Bandwidth consumed by three members of the same session**

A configuration with one sender and three sets of  $n$  receivers located behind bottleneck links is given by topology 3. The resultant bandwidth utilization when the sender transmits 20 layers of 20 kbps each and a value of  $n=5$  is depicted in Figure 4. The receivers can be seen to converge to three distinct bandwidth levels: The five receivers of the uppermost cluster in topology 3 receive the full 400 kbps (all 20 layers),

whereas the rightmost five receivers are limited by the shared 256 kbps bottleneck and the lowermost five receivers are confined to 128 kbps. Again, the simulation was concluded without any packet loss.

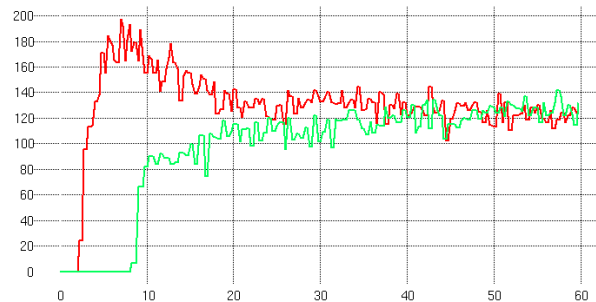


**Figure 4: Bandwidth consumed by the 15 receivers of topology 3**

## 4.2 Inter-session Fairness

To investigate the performance of multiple independent sessions sharing the same bandwidth bottleneck a large number of simulations were conducted using topology 4, with different values for the number of sessions,  $n$ , and the bottleneck bandwidth,  $B$ . Figure 5 shows the result of a configuration with two senders,  $S1$  and  $S2$ , and one receiver for each session,  $R1$  and  $R2$ , with a bottleneck bandwidth of 256 kbps. Both senders transmit ten 20 kbps layers resulting in an aggregate bandwidth requirement of 400 kbps for the shared link. Receiver  $R1$  is started first and initially joins all ten layers resulting in an allocation of 200 kbps out of the available 256 kbps. Then, after approximately ten seconds, receiver  $R2$  is started and the two receivers can be seen in Figure 5 to converge to approximately 128 kbps each. Thus, a fair sharing of the bottleneck bandwidth is achieved.

Similar results were obtained when simulating using topology 3 for many different values of  $n$  and  $B$ . The algorithm approximately allocated the bandwidth  $B/n$  to each session. Hence the algorithm can be seen to share network resources in a fair way among independent sessions.

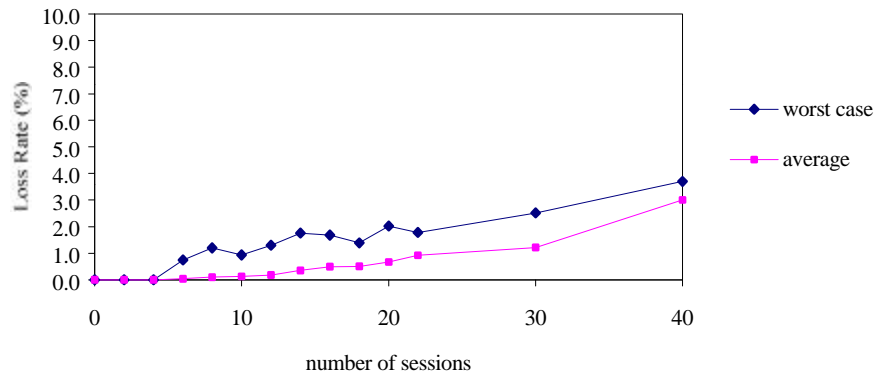


**Figure 5: Bandwidth allocated by two members of different sessions**

## 4.3 Scalability

The primary motivation for delay-based flow control is that the packet loss rate can be reduced compared to loss-based algorithms. The simulations involving only one sender and many receivers can be performed entirely without packet loss resulting from congestion. This is not surprising since the flow control algorithm was designed to predict and react in time to pending congestion, providing that the layers of the encoded media are sufficiently narrowband. If more than one layered multicast session is active simultaneously, however, the increase in bandwidth resulting from two or more receivers of different

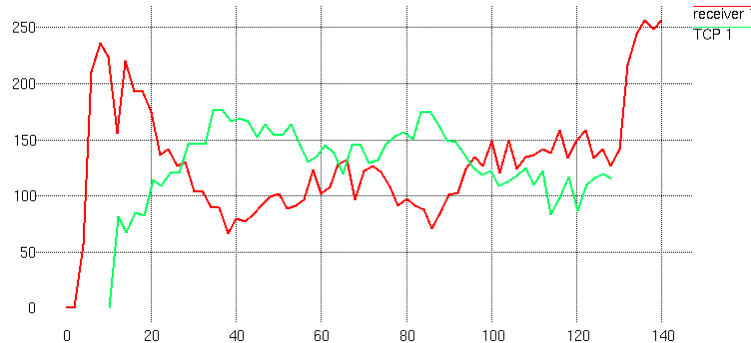
sessions joining simultaneously can be higher than what the router queues can withstand. In order to investigate the scalability of the algorithm when the number of sessions grow, a number of simulations were conducted using topology 4 with increasing values of  $n$ . The bottleneck bandwidth,  $B$ , was scaled in proportion to the number of sessions for each simulation. Figure 6 illustrates the average and worst-case loss rate performance. The loss rates were computed in non-overlapping windows one second wide. As can be seen, the average loss rate is about 1%, whereas the worst-case loss rate is about 2% of the total bandwidth. In comparison, McCanne et al. report a short-term worst-case loss rate of about 10% for RLM and a long-term loss-rate of about 1% [3]. Vicisano et al. report loss rates of about 7-8% for a simulation with 32 senders using their TCP-like congestion control scheme[4].



**Figure 6: Loss rate when superpositioning independent sessions**

#### 4.4 TCP Friendliness

Figure 7 shows the bandwidth of a layered multicast session simulated using topology 1 with the addition of competing TCP traffic. The TCP traffic consisted of one FTP session and ten Telnet sessions. When the simulation is started the layered multicast session can be seen to allocate all the available bandwidth of the link. Then, after 10 seconds the FTP file transfer and the Telnet sessions are started that lasts for approximately two minutes. It is clear that the multicast session yields bandwidth in favor of the TCP sessions. After the file transfer has ended the multicast session regains the full bandwidth.



**Figure 7: Bandwidth consumed by a layered multicast receiver in presence of TCP-traffic**

When simulating this scenario with a 10 ms delay on the shared connection, as in Figure 7, the bandwidth is shared evenly between the multicast traffic and the TCP traffic. However, since the performance of TCP is dependent on the RTT, a less fair sharing will be obtained if the delay is increased. To compare the bandwidth allocation of TCP with that of our multicast flow control we perform the same simulation as

above but with different values of the link delay. Then we calculate the *fairness index* defined as the ratio of the bandwidth allocated by the multicast application and the bandwidth allocated by TCP. The result is plotted in Figure 8 for both droptail and RED routers.

The multicast flow control obviously gets more aggressive in terms of bandwidth allocation compared to TCP when the link delay increases. For a 200 ms link delay the multicast session allocates almost three times as much bandwidth as the TCP sessions when using droptail routers. For RED routers the multicast session is favored even more in terms of bandwidth allocation. This is an expected finding since RED routers will drop packets before router buffers are filled leading to an earlier response from TCP's congestion avoidance, whereas the multicast flow control is unaffected.

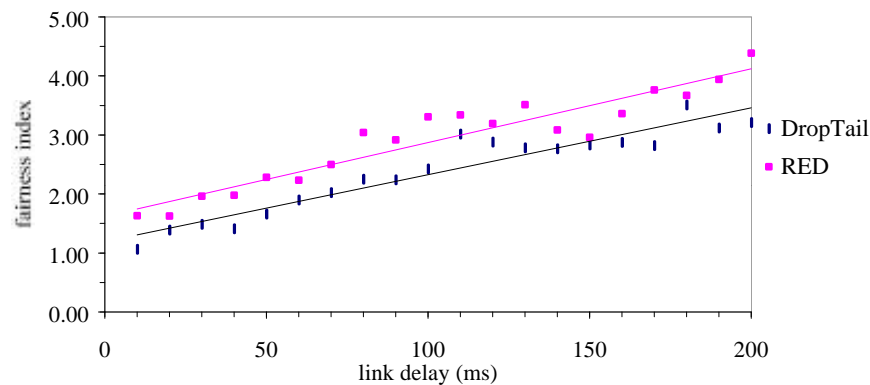


Figure 8: TCP friendliness index

## 5 Summary and Conclusions

Large-scale deployment of multipoint real-time conferencing applications in heterogeneous network environments requires a sophisticated flow control protocol. The protocol must be scalable to a large number of users, efficient in terms of resource utilization, fair to other data streams, adaptive to changing network conditions, and relatively light-weight for ease of implementation. In this paper, an approach to flow control for layered multicast applications was presented that relies on packet delay measurements to detect and avoid congestion. The algorithm was shown by simulation to interoperate in a fair way, in terms of resource allocation, among members of the same session as well as between instances of different sessions. Furthermore, the overall packet loss rate was seen to be very moderate when superpositioning independent sessions. The behavior of the algorithm in the presence of TCP traffic was seen to be TCP-friendly for low delay links and increasingly favorable for the multicast traffic at higher link delays. Further work will be needed to study the behavior of the algorithm in more complex network topologies and with larger sessions.

## References

- [1] V. Jacobson, "Congestion avoidance and control", Proceedings of SIGCOMM '88, Aug. 1988.
- [2] J. C. Bolot, T. Turletti, "A rate control mechanism for packet video in the Internet", Proceedings of IEEE Infocom '94, June 1994.
- [3] S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven layered multicast", Proceedings of ACM SIGCOMM '96, Aug. 1996.
- [4] L. Vicisano, L. Rizzo, J. Crowcroft, "TCP-like congestion control for layered multicast data transfer", Infocom '98, San Francisco, Mar. 1998.
- [5] L. Brakmo, S. O'Malley, L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", Proceedings of ACM SIGCOMM '94, May 1994.

- [6] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks", *ACM Computer Communication Review*, Oct. 1989.
- [7] Z. Wang, J. Crowcroft, "Eliminating periodic packet losses in 4.3 Tahoe BSD TCP Congestion Control Algorithm", *ACM Computer Communication Review*, Apr. 1992.
- [8] L. Wu, R. Sharma, B. Smith, "ThinStreams: An architecture for multicasting layered video", *Proceedings of NOSSDAV'97*, May 1997.
- [9] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control", RFC1890, Jan. 1996.
- [10] D. L. Mills, "Network time protocol (version 3) specification, implementation and analysis", RFC1305, Mar. 1992.
- [11] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The macroscopic behaviour of the TCP congestion avoidance algorithm", *Computer Communications Review*, vol. 27 no. 3, July 1997.
- [12] S. McCanne, S. Floyd, "The LBNL Network Simulator", Software on-line, <http://www-nrg.ee.lbl.gov/ns>.
- [13] T. Turletti, S.F. Parisi, and J. Bolot, "Experiments with a layered transmission scheme over the internet", *IEEE INFOCOM'98*, Feb. 1998.
- [14] T. Turletti and J. C. Bolot, "Issues with multicast distribution in heterogeneous packet networks", 6th international Workshop on Packet Video, Sep. 1994.
- [15] J. C. Bolot, T. Turletti, I. Wakeman, "Scalable feedback control for multicast video distribution in the internet", *ACM SIGCOMM 1994*, Aug. 1994.