MSB: Media Streaming Booster

Akihiro Nakao, Limin Wang, and Larry Peterson
{nakao,lmwang,llp}@cs.princeton.edu

Department of Computer Science
Princeton University

***Abstract*— The recent trend of constructing application-level overlays makes it increasingly common to have end-to-end paths decomposed into multiple overlay segments (tunnels). This paper investigates the potential benefits of making these intermediate overlay nodes both application- and congestion-aware in the context of streaming media. In particular, we demonstrate that hop-by-hop congestion control improves the quality of TCP-friendly delivery of layered video applications by up to a factor of four in the steady state. Moreover, the resulting system is significantly more responsive to changes in capacity, all the while not negatively impacting competing TCP flows.**

I. Introduction

Recent research advocates the use of rate-based, TCP-friendly congestion control for realtime streaming media applications [1], [2], [3], [4]. The main goal of TCP-friendly congestion control is to provide the relatively smooth short-term response to congestion events required by streaming applications, while guaranteeing fairness with competing TCP flows over the long term. At the application level, layered-encoded video nicely matches such a congestion control scheme since it is easy to add or drop layers of the media stream in response to the available rate [5].

Independent of work on rate-based congestion control and rate-adaptive applications, we are seeing an increase in the use of overlay networks as a mechanism for providing improved performance, reliability, and functionality; examples include RON [6] and End System Multicast [7], [8]. Such application-level overlays typically break the end-to-end path into a sequence of tunnels, with intermediate overlay nodes forwarding data in an application-specific way. Interposing proxies has a similar effect in that one or more intermediate nodes sitting between the source and sink have an opportunity to process and forward packets.

We observe that these intermediate nodes provide an opportunity to run both the rate-based congestion control algorithms and the rate-adaptive video applications mentioned above, but to do so on a hop-by-hop basis rather than strictly end-to-end. We call such a system running on the intermediate nodes a *media streaming booster* since it can be viewed an example of a protocol booster [9].

This paper studies the effectiveness of running TCP-friendly congestion control—in particular, the TFRC algorithm [1]—on a hop-by-hop basis. We demonstrate that the shorter RTT of each hop provides opportunities for (1) fast response to local congestion, (2) higher throughput while remaining TCP-friendly, and (3) better error recovery through the use of localized retransmission. A secondary contribution of this paper is to define an interface between TFRC and layered-encoded video applications.

II. Architecture

A. Overview

We assume a streaming application that traverses a path through an overlay network, where each hop along the path is a tunnel through the underlying Internet. Since the path selection algorithm used by overlay networks often requires at least one intermediate node to reflect traffic, we expect to be able to decompose the end-to-end connection into two (or more) segments at one (or more) intermediate node(s). As shown in Figure 1, we run three components of the application—`MmAppSend`, `MmAppInt`, and `MmAppRecv`—at the sending, intermediate, and receiving nodes, respectively. Underneath these modules, we run TFRC over each segment.

`MmAppSend` and `MmAppInt` both have a transmit queue (`xmitq`) and a retransmit queue (`rexmitq`), while `MmAppRecv` has a receive queue (`recvq`). These queues are implemented as priority queues whose key is the transmission time needed to ensure EDF scheduling. As soon as `MmAppSend` generates a video frame, it puts it into `xmitq` with a pre-calculated *presentation time*. When a packet is transmitted from `xmitq`, it may be cached in `pktbuf` for future retransmission. When a packet is delivered to `MmAppRecv`, it is enqueued in `recvq` until it can be decoded.

`MmAppInt` and `MmAppRecv` send NACKs when packet loss occurs. We use these NACKs to trigger re-

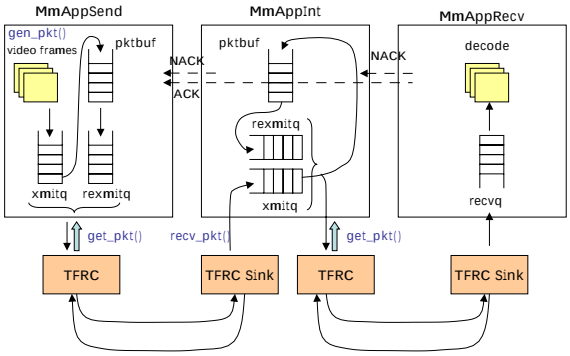


Fig. 1. Decomposition of an end-to-end path into multiple segments.

transmission, based on the observation that video streaming applications usually have playback buffers, which give retransmission some cushion of time. However, this retransmission should not be regarded as a reliable transport like TCP. In response to the NACK, `MmAppSend` and `MmAppInt` look for the missing packet in `pktbuf`, and if they find it, place the packet in `rexmitq`, as long as there is a chance the packet can be retransmitted by its presentation time. `MmAppSend` and `MmAppInt` select a packet for transmission from either `xmitq` or `rexmitq`, as described below. Finally, `MmAppRecv` periodically sends ACKs that are relayed back to `MmAppSend`. This ACK is necessary to update the end-to-end semantics to all the nodes.

B. Application/Transport Interaction

1) Interface: Three operations define the interface between the application and transport layers.

- `Packet* get_pkt(int xrate, int *size, int mode)`
- `void recv_pkt(Packet* pkt)`.
- `void break_ss(void)`.

The application implements the first two operations; the transport layer implements the third. The transport layer calls `get_pkt()` when it decides it is time to transmit another packet. It specifies the transmission rate (`xrate`) and the congestion control mode (`mode`) as arguments; a pointer to the packet is returned. Both the rate and the mode serve as hints to the application, where the mode indicates whether or not the transport layer is operating in slow-start mode (more below). The transport layer also does an upcall to `recv_pkt()` to deliver an incoming packet to the application. The application calls `break_ss()` to force the transport layer out of slow-start mode. This function is necessary for an extension to TFRC that we explain in Section II-B.3.

2) TCP-Friendly Transport Protocol: Although TCP is the dominant transport protocol in the current Internet, it is not well suited for multimedia streaming. One reason is that its Additive Increase/Multiplicative Decrease (AIMD) algorithm [10] halves the sending rate in response to a single congestion indication, and this may have a severe impact on the user-perceived video quality.

Several TCP-friendly congestion control algorithms have recently been proposed in response to this problem [1], [2], [3], [4]. These algorithms have two main goals. One is to slowly adapt the congestion window. This is done by adapting over relatively longer time periods (e.g., an RTT) rather than on a per-packet basis. The second is to be TCP-compatible in the sense of being fair to competing TCP flows. This property is often discussed in terms of an equation-based model of TCP throughput [11]. In particular, the recently proposed TCP-Friendly Rate Control (TFRC) scheme uses equation-based control to explicitly give the maximum acceptable sending rate as a function of the recent loss event rate reported by the receiver [1]. We use TFRC as the transport mechanism in our study.

3) Limitations of TFRC: Although TFRC has desirable features as a TCP-friendly protocol, for streaming media applications to take advantage of it, the following limitations must be addressed: (1) it assumes infinite packet supply from applications, (2) it moves out of slow-start mode only at the first packet loss, and (3) it deals with fixed-size packets. We consider each limitation, in turn.

First, while an infinite packet supply may be appropriate for a pre-existing media file, it is not a realistic assumption for a streaming application like video-conferencing, since such applications have a maximum rate at which they can generate data for transmission, and this rate may be less than the TFRC-calculated available rate. Usually, a real-time application sets the maximum bandwidth `app_max_bw` early on. Adaptive applications may adjust the transmission rate due to congestion by changing transmission layers (short term) or the frame rate (long term) within `app_max_bw`. We assume `app_max_bw` is typically set at a point that is less than capacity of the link, meaning that when the transmission rate allowed by transport layer exceeds `app_max_bw`, `get_pkt()` returns a null packet.

Second, the original TFRC scheme breaks out of slow-start mode only when it experiences packet loss. If there is no congestion, the application keeps sending data in slow-start mode. This leads to the following pathological situation. At the beginning of a connection, if the application does not provide enough packets—due to its rate control algorithm being based on the transmission rate suggested by TFRC—the transmission rate might not

increase, which prevents the application from supplying enough packets to force congestion. TFRC uses feedback every 1.5 RTTs, and if it does not receive feedback within 2 RTTs, it halves the transmission rate. This cycle prevents the initial increase in the transmission rate.

In slow-start mode, the first priority for applications is to gain speed. If the application knows that TFRC is in slow-start mode, it can keep transmitting packets regardless of the TFRC-provided rate hint. Note that this does not break TFRC, since the application transmission rate cannot exceed the TFRC enforced rate. However, this opens up another problem. As we remarked above, if there is no congestion to cause packet losses, TFRC never gets out of slow-start. This means the application keeps sending packets in an uncontrolled manner. Therefore, the application needs to know if the transport is in the slow-start phase, or not, and it also needs to tell TFRC that it should get out of slow-start when the rate reaches `app_max_bw`. This is the reason for the `mode` argument to `get_pkt` and the `break_ss()` operation.

Third, TFRC supports only fixed-size packets. While this simplifies TFRC, it complicates the interaction between the application and the transport protocol. In some real-time applications, one way to control the transmission rate is to change packet size while fixing the transmission interval; many layered-video applications make such assumptions. However, TFRC changes the transmission interval to adjust the transmission rate while using the fixed size of packets. This mismatch has to be resolved at the application level. For example, as we see later in II-D, we circumvent this limitation by modifying a layered video application, WaveVideo, to make each layer approximately the same size.

There have also been other research efforts on TFRC variants that address similar problems. For example, Variable Packet Size Equation Based Congestion Control (VPS-TFRC) [12] deals with the variable size packets in TFRC, but VPS-TFRC has not been extensively studied yet, compared to standard TFRC.

C. Packet Transmission Algorithm

1) Two Queues: We maintain two separate priority queues, one for transmission and one for retransmission. There are two reasons for this. First, retransmission of base layer (important) packets has priority over transmission of higher layer (less important) packets. At the same time, we have a limited bandwidth budget for each frame, so we have to arbitrate between transmitted packets and retransmitted packets. If we maintain both kinds of packets in the same queue, we have to implement complex queue management since we do not want to drop potentially useful packets from the queue. Having two separate queues makes the arbitration logic simpler. Second, if we always put retransmitted packets in a retransmission queue, we can always guarantee the order of packets in the transmission queue, since retransmission does not pollute the transmission queue. This invariant is useful because in the transmission queue, packets are ordered according to the importance of the packets. This makes the packet dropping logic simpler.

2) Arbitration: Arbitration between the transmission and retransmission of packets happens when the underlying transport protocol TFRC calls `get_pkt()`. For the transmission rate hint $X(t)$ at time t given from the underlying TFRC, we assign total budget $B_{total}(t)$ to the current frame f, denoting the available bytes for frame f. Note that to respond to the sudden change in $X(t)$, the budget $B_{total}(t)$ changes over time, even within the same frame f.

$B_{total}(t)$ is divided into the transmission budget $B_{xmit}(t)$ and the retransmission budget $B_{rexmit}(t)$. We define α to be the fraction of budget $B_{total}(t)$ allocated to $B_{rexmit}(t)$, and $(1-\alpha)$ to be the fraction allocated to $B_{xmit}(t)$. We calculate α as follows. For the current frame f, we keep track of how many bytes have been used for transmission U_{xmit} and calculate availability ratio $R(t)$ of U_{xmit} to $B_{total}(t)$. Then, we take the minimum value α between preset value α_{max} and $R(t)$. α_{max} is a maximum ratio of the retransmission budget to the total budget. In our simulation, typical value of α_{max} is 0.2.

$$
\begin{aligned}
B_{total}(t) &= X(t)/fps && (1)\\
B_{rexmit}(t) &= \alpha \cdot B_{total}(t) && (2)\\
B_{xmit}(t) &= (1-\alpha) \cdot B_{total}(t) && (3)\\
R(t) &= 1 - U_{xmit}/B_{total}(t) && (4)\\
\alpha &= \min(\alpha_{max}, R(t)) && (5)\\
\beta &= S(t)/A(t) && (6)
\end{aligned}
$$

Each time `get_pkt()` is called at time t, if there is a packet in the retransmission queue, we update α and transmit it if $U_{rexmit} + s < B_{rexmit}(t)$ where s is the packet size, and if the deadline of the packet will be met. After transmitting it, we update $U_{rexmit} += s$.

If there is no packet in the retransmission queue, or the budget for retransmission is short, we check the transmission queue. We transmit a packet in transmission queue if $U_{xmit} + s < B_{xmit}(t)$, the packet belongs to the current frame f, and the deadline of the packet will be met. Otherwise, we increment the current frame f and reset U_{xmit} and U_{rexmit} so subsequent packets belonging to the old

Parameter	Description
t	time when `get_pkt()` gets called
s	packet size
$B_{total}(t)$	total budget per frame at time t
$B_{xmit}(t)$	budget for transmission at time t
$B_{rexmit}(t)$	budget for retransmission at time t
f	current frame in transmission
U_{xmit}	transmitted bytes of frame f
U_{rexmit}	retransmitted bytes of frame f
$R(t)$	budget for retransmission
α	ratio of B_{rexmit} to B_{total}
α_{max}	preset maximum value of α
$A(t)$	average maximum layers in transmission
$S(t)$	secure level
β	constant ratio of $S(t)$ to $A(t)$

TABLE I
ARBITRATION PARAMETERS

frames get dropped from then on. As described briefly in the previous section, in transmission, we cache packets into `pktbuf` for possible future retransmissions. In order to do this, we keep track of the long term average $A(t)$ (exponentially weighted mean average) of the maximum layers in transmission. We define *secure level* $S(t)$ as β portion of $A(t)$. We try to guarantee the delivery of the packets of the layer less than (or more important than) $S(t)$. Accordingly, we cache packets containing layers less than $S(t)$ in the packet buffer for future retransmission. Typical value of β in our experiments is 0.8.

Note that in the transmission queue, packets are always ordered according to their importance. Therefore, less important packets are naturally tail-dropped

3) Deadline Consideration: We also consider the deadline of the packets during the course of the arbitration algorithm. Let us define the following parameters

Parameter	Description
G	packet generation time at source
S_i	packet send time at node i
D	packet presentation time (deadline)
P	probation period ($P = D - G$)
$RTT_{a,b}$	estimated RTT between a and b

TABLE II
TRANSMISSION PARAMETERS

At node i, we take a packet from the transmission queue, and as a general rule, if the following condition is true,

$$(S_i + RTT_{i,n}/2) < D \tag{7}$$

we (re)transmit this packet. In the retransmission case, the number of possible retransmission k is given by the following condition.

$$RTT_{1,n}/2 + k \cdot RTT_{i,i+1} \quad < \quad P = (D - G) \tag{8}$$

Cache entries will be evicted after the period $k \cdot RTT_{i,i+1}$ has passed, so the buffer size required at the node i is at most $k \cdot RTT_{i,i+1} \cdot app\_max\_bw$. For example, 1.5Mbps video requires 1.5Mbps/8 $\times$ 100msec = 18.75KB memory for retransmission ($k = 1$) over 50 msec link.

Usually, unless the link is heavily congested, k is typically small and so is the required buffer size, since (1) the packet loss is randomized so the same packet rarely gets lost, and (2) due to the deadline limitation, large k does not satisfy Equation 8.

D. Layered Video

Among many possible coding schemes, we use a wavelet-based system called WaveVideo [5] as our example encoding. WaveVideo spatially decomposes each video frame into 33 layers of subbands for adaptive transmission. WaveVideo iteratively applies two-dimensional discrete wavelet transform (2D-DWT) to each of luminance (Y) and color difference (Cb, Cr) channels of the image. 2-D DWT steps from one depth to the next includes a horizontal and vertical 1-D DWT performed in a series. Since the 1-D DWT produces low- and high-frequency subbands, the 2-D DWT at each depth produces four (LL,LH,HL,HH) frequency subbands of a quarter of the original size. LL-subbands are recursively passed to the next higher depth for further decomposition. The luminance (Y) channel is often transformed one depth deeper than the color channels (Cb, Cr). Wavevideo uses 3 (4 for Y channel) depths of transformation for QCIF size of the image, which allows 33 layers of subsampling of the image. (See Figure 2.) It does further spatial compression within subbands, and also temporal compression between frames.

The general idea of adaptive layered video is to give priority to coarse video channels (low frequency subbands) over detail video channels (high frequency subbands). Figure 2 illustrates the priority of each subband used in WaveVideo, where 1 is the most important (coarse) layer and 33 is the least important (fine-grain) layer.

We measured the average size of the subband for two example video streams (Akiyoqcif.avi, Foremanqcif.avi) used in WaveVideo. Figure 3 shows the average data size

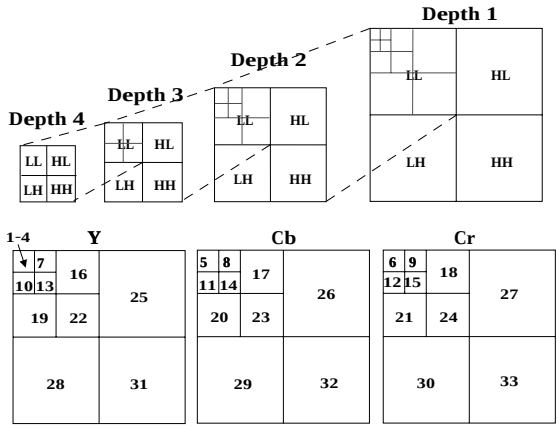
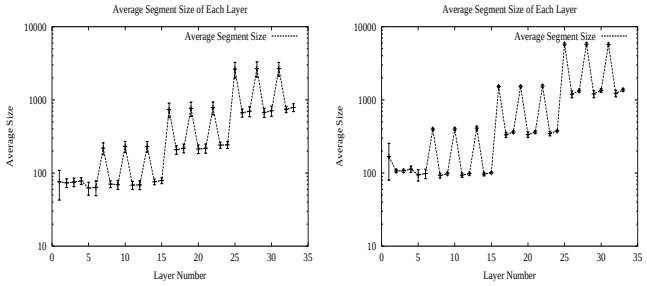
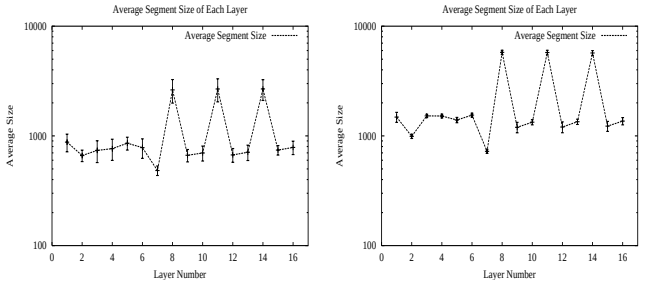


Fig. 2. Wavevideo Layering



(a) Akiyoqcif.avi (b) Foremanqcif.avi

Fig. 3. Average Segment Size of Layer



(a) Akiyoqcif.avi (b) Foremanqcif.avi

Fig. 4. Average Segment Size of New Layer

distribution among layers in the example video streams. (We set I-frame frequency to 1/25, so the majority of the frames are Δ-frames.) The Figure 3 suggests the following observations. First, sizes range over almost two orders of magnitude among layers. Note that the graph shown in 3 has logarithmic scale along y-axis. Theoretically, depth n subbands are 4 times larger than depth $n+1$ subbands, without considering inter- or intra-band compression. Second, it does not seem to make sense to subsample into as many as 33 layers. Loss of layers less than 10 results in unacceptably degraded images, while the size of these low (important) layers are often very small. In terms of network adaptation, it would make more sense to combine these low layers into one (base) layer. Third, the Y layers always have larger size than Cb, Cr layers.

These points are not favorable to the use of TFRC as the underlying transport layer protocol. Ideally, we want each layer to be of nearly the same size, since TFRC is limited to fixed size packets. For this purpose, we define a new layering scheme, denoted ($L'_1 \sim L'_{16}$) out of the existing WaveVideo layering scheme ($L_1 \sim L_{33}$).

$$
\begin{aligned}
L'_1 &= \{L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_{10}\},\\
L'_2 &= \{L_8, L_9, L_{11}, L_{12}, L_{13}, L_{14}, L_{15}\},\\
L'_3 &= L_{16},\\
L'_4 &= L_{19},\\
L'_5 &= \{L_{17}, L_{18}, L_{20}, L_{21}\},\\
L'_6 &= L_{22},\\
L'_7 &= \{L_{23}, L_{24}\}\\
L'_i &= L_{(17+i)} \quad (8 \le i \le 16)
\end{aligned}
$$

Figure 4 shows the distribution of average size of the new layers. Except for the new layers 8, 11, and 14 (Y channel, LH, HL, HH subbands in depth 1), the size is almost evenly distributed over the layers. We can packetize the data in the new layers 8, 11, and 14 into segments of a size comparable to the other layers. In this new layering scheme, we have the following desirable features: (1) every layer has almost the same size (except for a few layers that need to be further segmented into the comparable size); (2) base layer 1, by itself, contains an acceptable image; and (3) we still preserve the original characteristics: the higher the layer, the more detail of the image it includes.

III. Evaluation

Our goal is to determine if a layered video application benefits from taking advantage of intermediate nodes in an overlay network, that is, by running the congestion control algorithm plus the adaptive application on a hop-by-hop basis rather than end-to-end.

We conducted a set of experiments using the ns-2 network simulator [13]. In lieu of actual video data, we modeled the WaveVideo as described in the previous section. Throughout the simulations, we use the following parameters: 16 layers, 1000-byte packets, and a frame rate of 10 fps. We also limit our study to a single intermediate node, resulting in two segments.

We evaluate our scheme in three scenarios. The first quantifies the improvement in video quality—specifically, the number of layers successfully delivered—of the hop-by-hop case verses the end-to-end case, when only one overlay segment is congested. It also allows us to investigate the impact of this improvement on layers' smoothness, latency and loss recovery. In the second scenario, we study the dynamic behavior of decomposed TFRC control, for example, its responsiveness and aggressiveness when congestion conditions change. The last scenario repeats similar tests as in the first one, but with both overlay segments congested.

A. Scenario I: Localized Congestion

We first examine the scenario in which the decomposition localizes the congestion to either the first or the second half of the connection. If the first half is congested and the second half is not, the behavior of decomposed congestion control is equivalent to the end-to-end case, except with a shorter latency. The more interesting case is when the second segment is congested while the first is not, due to a bandwidth gap at the intermediate node. We consider this latter case.

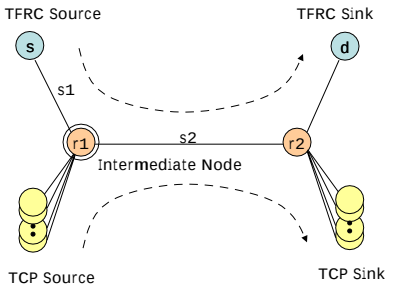


Fig. 5. Topology in Scenario I

For this experiment we use a single-bottleneck topology with RED queue management at the bottleneck router, as shown in Figure 5. We set the bandwidth of the bottleneck to 5Mbps. We decompose the end-to-end connection into two segments $s1$ and $s2$ at the bottleneck router (overlay node) $r1$. Although we treat $r1$ as both a router and an overlay node in the simulation, in practice they would likely be separate nodes located close to each other (e.g., at the same site). We change the location of the intermediate node so that the latency of the first segment $s1$ ranges—at 10ms intervals—from 10msec to 90msec. Our single TFRC flow competes with 15 end-to-end TCP flows with the same 100msec latency. Note that regardless of the location of the intermediate node, and whether or not we have an intermediate node, we get almost the same packet loss rate (about 2%) mainly caused by competing TCP flows. We set the retransmission parameters α_{max} to 0.2, β to 0.8, and the probation time to 500msec. A typical experiment runs for 600 seconds of simulated time.

1) Layers: The quality of video is primarily determined by how many layers are delivered within the predetermined delay (probation time). We define the *minimum layer* for each frame to be the largest layer number received, *consecutively*, starting at the base layer.

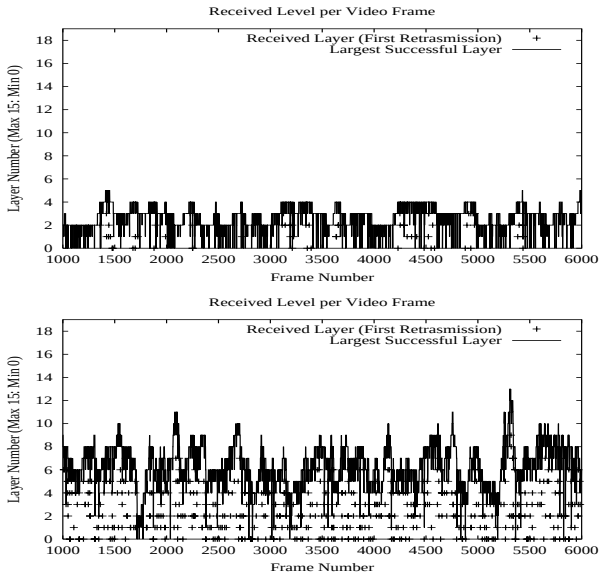


Fig. 6. Received Minimum Layers (top: end-to-end , bottom: with intermediate node (first segment = 50ms))

We change the location of the intermediate node and measured the number of layers the receiver gets to see how the location of intermediate node affects behavior. Figure 6 shows the traces of the number of successfully received minimum layers for each frame, with and without the intermediate node (only the case with the first segment delay of 50 msec is shown), respectively. It's clear the hop-by-hop case successfully delivers more layers; further analysis shows precisely how much better.

Figure 7 compares the average number of layers over frames in the end-to-end case without an intermediate node, and in cases with an intermediate node at various locations. Note that in the end-to-end case, the curves of enabling or disabling retransmission are overlapped. This is because the average number of minimum layers is too small to produce a significant difference with retransmission.

We see from the graph that a shorter RTT on the congested segment leads to a better number of layers being successfully delivered to the receiver. Comparing to the

end-to-end case, having an intermediate node at 50 msec and 90msec improves the average number of layers by a factor of 2 and 4, respectively. Enabling retransmission adds slight improvement to the factors. The result in 50msec is consistent with the goodput of TFRC, since the goodput T is inversely proportional to the RTT of the segment as in the TFRC equation [1], whereas the 90msec case does not perform as well as the equation predicts, perhaps because the number of layers tends to saturate at the maximum number of layers in this case.

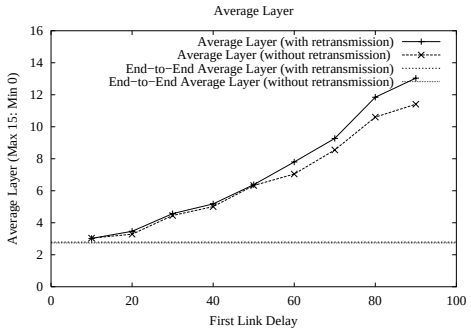


Fig. 7. Average Layers

While we want to have a larger number of layers, a large deviation in the number of layers is undesirable. To see how well the two cases performed, we also measured the standard deviation of the average layers over frames in Figure 8. We see the shorter RTT the congested segment has, and the great the average number of layers, the more deviation we tend to see. The ratio of the standard deviation to the average layer—i.e., the coefficient of variation (COV)—is shown in Figure 9. Without retransmission, this graph shows that COV stays flat around (40%) regardless of the average number of layers. However, with retransmission, COV decreases as the first segment delay increases. At 50msec and 90msec, COV is 29% and 23%, respectively.

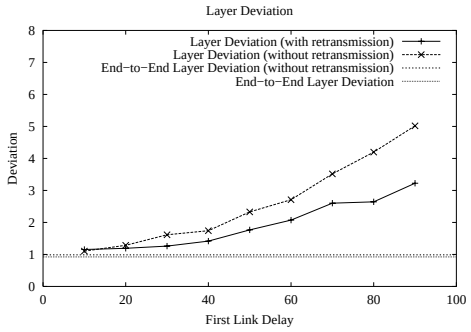


Fig. 8. Standard Deviation of the Average Layers

To see the distribution in the average number of layers successfully received, we also plotted the histogram of layers in Figure 10. As we see in the figure, the cases with the first segment having shorter RTTs have a wider distribution. The less congested cases add more randomness in the distribution.

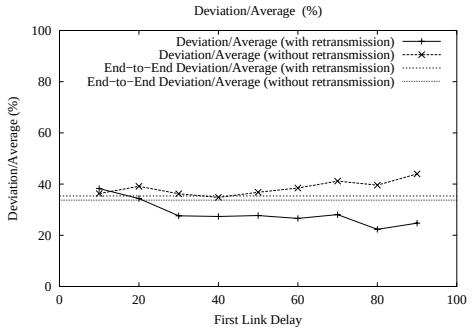


Fig. 9. Coefficient of Variation (COV)

2) Smoothness: In [14], *smoothness* is defined to be the largest reduction of the sending rate in one RTT in the deterministic steady-state scenario. In order to see smoothness from the application's point of view, we exploit a similar idea to that stated in [14]. Specifically, we use *layer ratio* $s_i = L_i/L_{i+1}$ as a metric, where L_i is the number of layers of the frame i. Figure 11 shows the histogram of layer ratio s_i. Although the trace fluctuates somewhat over frames, the figure shows that at a frame level, the layer ratio is mostly distributed close to 1. We see that we tend to lose smoothness with shorter RTT. Note that the 90msec case has better smoothness than the 80msec case, probably because in the 90msec case, the received layer tends to saturate at the maximum layer.

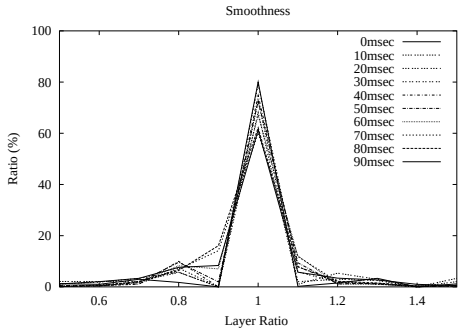
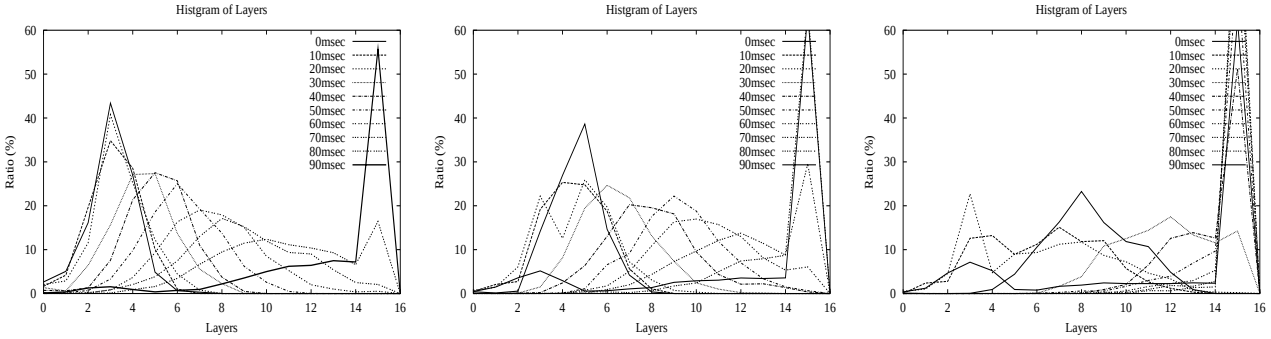


Fig. 11. Smoothness

3) Latency: We compared the average end-to-end delay with various intermediate node locations, with retransmission enabled. Since we use real-time video applications, latency is measured between the time a packet is generated at the sender, and the time it is delivered to the destination. Figure 12 plots the average latency of packets successfully received in the first transmission and that



(a) 15 TCP flows (b) 10 TCP flows (c) 5 TCP flows

Fig. 10. Histogram of Layers with different competing TCP flows (Retransmission Enabled)

of those received in the single retransmission. Note that the average latency values are plotted as straight lines in the end-to-end case. Also the standard deviation values are plotted around the average latency values as error bars in the hop-by-hop cases and as separate dashed lines in the end-to-end cases. As we can see in the figure, the transmission case incurs only a slight additional delay ($<$ 10msec) on average, compared to the end-to-end latency of 160msec, but the intermediate node can significantly reduce retransmission delay as the second-segment delay decreases. For example, with an intermediate node at 90 msec, we have only 250msec latency for retransmitted packets, whereas the end-to-end case has 418 msec. The improvement by 168msec may be accounted as the time difference in retransmission paths which is 180msec.

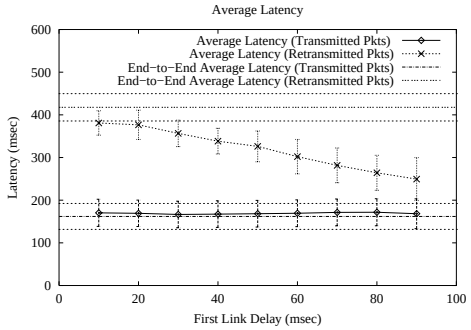


Fig. 12. Average Latency

4) Error Recovery: There are several recovery techniques from errors caused by packet losses, such as retransmission. Retransmission is a general technique that is independent of coding schemes. As we have seen in Figure 12, by decomposing the end-to-end congestion control, we have better opportunity to localize the retransmission delay. From Figure 7, we see that the packet loss recovery by retransmission improves both the average number of layers received and its deviation.

Although we have not fully explored the effect of α_{max} and β to the average number of layers and its deviation, our preliminary experiment shows that small $\alpha_{max} < 20\%$ tends to add long latency in retransmission in our simulation environment with relatively small packet loss rate ($\sim 2\%$).

B. Scenario II: Responsiveness

We next examine the dynamic behavior of our decomposed congestion control. We are especially interested in responsiveness and aggressiveness of the congestion control algorithm in the light of sudden changes in network capacity.

Similar to the first scenario, we assume that only the second segment of the end-to-end connection is congested. Also, we disable retransmission to avoid introducing unnecessary complexity. The topology for the setting is simple: we have two end points and at most one intermediate node. We compare two cases, **Case (1):** end-to-end congestion control without intermediate nodes, and **Case (2):** decomposed congestion control with an intermediate node placed in the middle. Both the first half and the second half delay are set to 50msec, while end-to-end latency is fixed at 100 msec. In this scenario, we simulate the sudden introduction or removal of heavy congestion during the course of the experiment, which be caused by the introduction/termination of a CBR flow, or by a flash crowd.

1) Responsiveness: The *responsiveness* of a congestion control mechanism has been defined in [14] as the number of RTTs of persistent congestion until the sender halves its sending rate, where *persistent congestion* is defined as the loss of one packet per RTT. Analysis and simulation in [1] shows that the responsiveness of TFRC is 4 to 6.

Similar to the experiments in [14], we have a single TFRC flow with every k_d-th packet being dropped at the bottleneck router while in steady state. At the simulation time 150sec, we start dropping every other packet to simulate the heavy congestion. Figure 13 shows the traces of the *minimum* and *maximum* layers. As stated in the previous section, *minimum layer* is defined per frame, as the largest number of layers *consecutively* received, starting at the base layer. *Maximum layer* is simply defined per frame as the largest layer received. We also plotted framewise packet loss shown as a circle at the bottom of the graph. For the sake of visibility, we set $k_d = 700$ for the case (1) and $k_d = 200$ for the case (2) in order to adjust the goodput in the steady (less congested) state to comparable value for both cases (about 14 layers of reception). Setting k_d to the same value would result in very different throughput in the steady state between the case (1) and the case (2), since decomposition gives us better throughput as seen in the previous scenario. The choice of k_d affects the starting throughput but not the responsiveness.

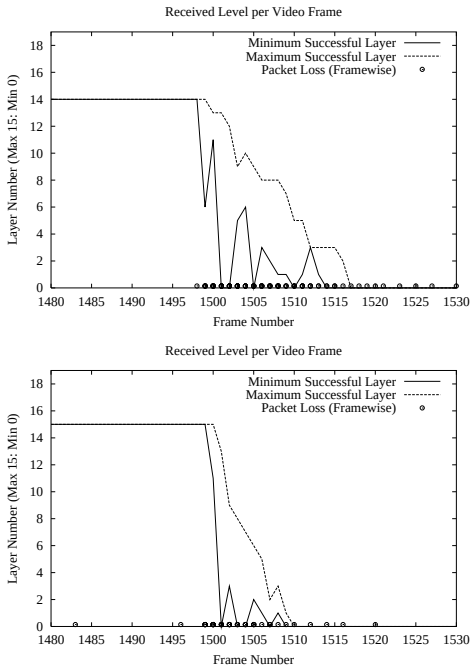


Fig. 13. Responsiveness (top: end-to-end ($k_d = 700$), bottom: with intermediate node ($k_d = 200$). Heavy congestion is introduced at the simulation time 150 seconds (around 1500-th frame)

Figure 13 shows that after congestion is introduced at simulation time 150 seconds (around the 1500-th frame), case (1) takes 8 frames to halve the maximum number of layers, whereas case (2) takes only 5 frames. Since we are sending layered video at 10 fps, this result roughly corresponds to (1) 0.8 seconds and (2) 0.5 seconds of response time. Therefore, by decomposing end-to-end connection by half, we can reduce the response time to about 60% of the end-to-end case.

Note that we just looked at the maximum layer curve to examine responsiveness. However, from the application's point of view, in absence of retransmission, only the minimum layer affects the quality of the video, since layered video has the property that higher layers always depend on lower layers for decoding. Figure 13 shows the minimum layer curve drops sharply (taking only a few frames) right after the congestion begins. This is because we introduce extremely high packet loss rate (50%) to cause congestion, but in general, the maximum layer curve gives an upper bound on the minimum layer curve. This implies that the video quality the application perceives is likely to be degraded faster than the real responsiveness without retransmission. Even though the application is still receiving higher layer packets, they may not contribute to the video quality, since lower layer packets have been lost.

2) Aggressiveness: *Aggressiveness* is defined as the maximum increase in sending rate in one RTT—stated in packets per second—in the absence of congestion. Analysis and simulations in [1] show that the aggressiveness of TFRC is at most 0.14 pkts to 0.28 pkts per RTT. We examined application-level aggressiveness in terms of the number of successfully received layers in the face of sudden absence of congestion. Similar to the experiment conducted in the previous section, we have only a single TFRC flow with every k_d-th packet being dropped at the bottleneck router so that we set equal loss intervals while in congestion. At simulation time 150 seconds, no more packets are dropped, which represents the sudden absence of congestion. Figure 14 shows the traces of the minimum number of successfully received layers per each frame. The definition of minimum layer is the same as in the previous section. We also plotted framewise packet loss at the bottom of the graph.

For the sake of visibility, we set $k_d = 100$ for (1) and $k_d = 30$ for (2) in order to adjust the goodput in the congestion state to comparable value for both cases (about 5 layers of reception).

According to Figure 14, after congestion is removed at the 1500-th frame, case (1) takes 250 frames to ramp up to the maximum reception of layers, whereas case (2) takes only 60 frames. Since we are sending layered video at 10 fps, this roughly corresponds to 25 seconds and 6 seconds of response time, respectively. Therefore, by decomposing end-to-end connection by half, we can reduce the response time by a factor of four.

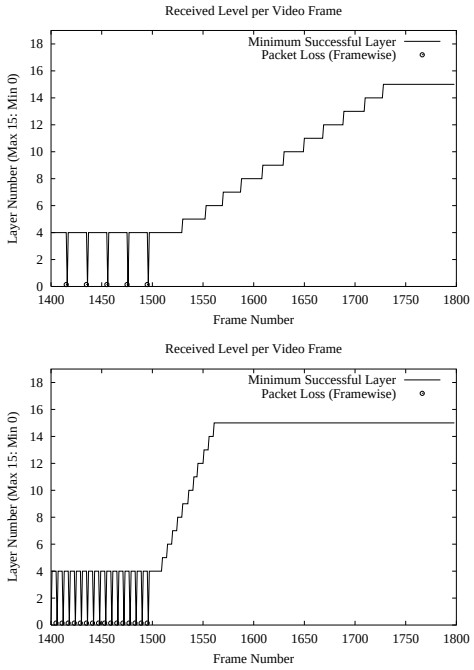


Fig. 14. Aggressiveness (top: end-to-end ($k_d = 100$), bottom: with intermediate node ($k_d = 30$). Congestion is removed at time 150sec (around 1500-th frame)

C. Scenario III: Multiple Congested Segments

The next scenario we examined is similar to the first one, except both the first half and the second half of the connection are *equally* and *independently* congested.

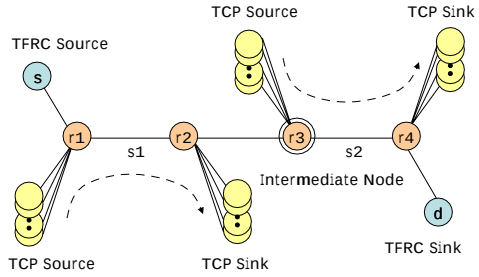


Fig. 15. Average Layers

Figure 15 shows the topology of the experiment. We have two bottleneck routers $r1$ and $r3$ with RED queue management. We set the bandwidth of the bottleneck segment $s1$ and $s2$ to 5Mbps. At the bottleneck router $r3$, we run an intermediate node and decompose the end-to-end connection into two segments. While end-to-end latency is fixed to 100 msec, we change the location of the intermediate node so that the latency of the first half connection $s1$ ranges—at 10ms intervals—from 10msec to 90msec. For the end-to-end case, without an intermediate node, the latency before and after the bottleneck router is 50ms, for a total of 100ms. Our single TFRC flow competes with 20 TCP flows with the same latency, where 10 of them share segment $s1$ with the TFRC, and the other 10 share segment $s2$. This generates two kinds of independent congestion along the TFRC flow. Note that whether or not we have an intermediate node, we get almost the same packet loss rate (about 2.5%) mainly caused by competing TCP flows. [1] Most parameters such as $\alpha_{max} = 0.2$, $\beta = 0.8$ and probation time of 500msec are the same as in the first scenario. Typical experiments run for duration 600 seconds of simulated time.

1) Layers: Figure 16,17, and 18, compares the average number of layers, the deviation, and the ratio of deviation to the average number of layers (COV), respectively, for the end-to-end case and the set of hop-by-hop cases with the intermediate node at various latencies from the source. These figures tell us the following.

First, Figure 16 shows the improvement with an intermediate node. In particular, the number of layers is maximized when the first segment delay is set to 50msec. Compared to the end-to-end case, we see about 4 times (without retransmission) and 5 times (with retransmission) more throughput when we use decomposed congestion control in the very middle of the end-to-end connection.

We try to explain this result relative to the first scenario. In the first scenario, let $f_1(x)$ be a throughput ratio of decomposed congestion control case to end-to-end congestion control case, when the first segment delay is x msec. From Figure 7, $f_1(x)$ is monotonically and slowly increasing as x. Theoretically, this curve $f_1(x)$ is proportional to $(d-x)^{-1}$, where d is end-to-end latency, provided that the packet loss rate is independent of x and throughput strictly follows the TCP equation [11]. Since we have equally congested segments in the third scenario, the throughput ratio function in the third scenario is modeled as $f_3(x) = f_1(x) \cdot f_1(d-x) \propto \{x(d-x)\}^{-1}$. It is obvious that $f_3(x)$ reaches its maximum at $x = d/2$. We could easily imagine that in more realistic and complex scenario where we have unequally congested segments, this maximum would be shifted to either end point.

Second, Figure 17 shows that without retransmission, the deviation tends to increase as the average number of layers. With retransmission, however, the deviation is greatly suppressed, especially for the cases with improved average number of layers. It is apparent from Figure 18

[1] We calculate the *effective* end-to-end packet loss rate p for decomposed congestion control as $p = 1 - (1-p_1)(1-p_2)$, where p_1, p_2, denote the loss rate for the first half and the second half of the end-to-end connection, respectively. This *effective* loss rate turns out to be comparable to that in end-to-end control case.

that COV is minimized when the intermediate node is placed in the middle of the end-to-end connection. For example, at 50msec, COV is 37.8% without retransmission, and is 21.6% with retransmission.

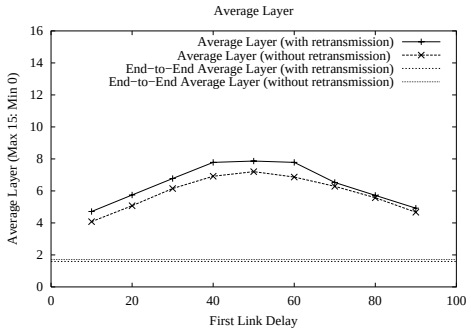


Fig. 16. Average Layers

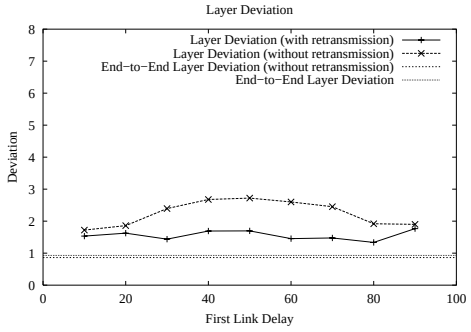


Fig. 17. Deviation of the Average Layers

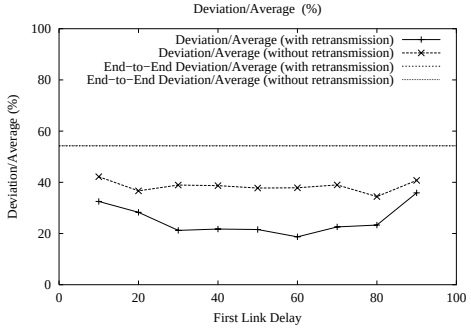


Fig. 18. Coefficient of Variation (COV)

2) Latency: Figure 19 shows the average latency of packets successfully received in the first transmission and that of those received in the single retransmission. We see that having an intermediate node introduces additional delay (<20msec) on average. However, with an intermediate node, we can significantly reduce the average retransmission latency. We have roughly 340msec retransmission latency with an intermediate node as opposed to 441msec in the end-to-end case. We believe this is because we can localize the retransmission by decomposing the congestion control. The improvement of about 100msec is accounted as the average time difference in the retransmission paths.

It is interesting to see the deviation in latency of retransmitted packets is minimized when the intermediate node is placed in the very middle of the end-to-end connection. Note that the deviation in the average retransmission latency is high when the intermediate node is close to either end point. For example, at 50msec the standard deviation is only 50msec whereas at 10msec and 90msec it is almost 90msec. Although we can localize the retransmission, when the intermediate node is near either end point, we still have many cases that it takes time close to end-to-end latency to retransmit the packets along the longer segment, whereas it needs less time along the shorter segment.

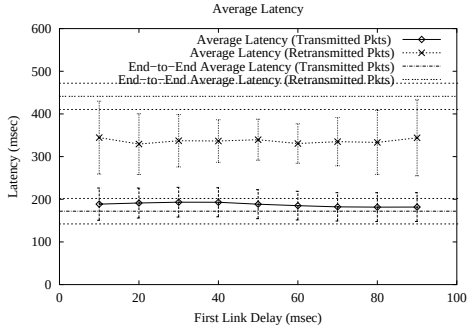


Fig. 19. Average Latency

D. Fairness

Once we accept the possibility of forwarding application data through an overlay network, we have already introduced an element of unfairness into the network. This is because traffic reflected via one or more intermediate nodes effectively traverse multiple short-RTT segments rather than a single long-RTT path, and both TCP and TFRC are biases against long-RTT connections.

This makes it difficult to compare the fairness of flows through an overlay with flows through the Internet. However, we note that on each segement of overlay path, our flows compete with other non-overlay flows in a fair manner, and they do not cause congestion collapse since they employ the TFRC algorithm which guarantees fairness with respect to other flows. Although it is not shown here, we conducted similar experments as in [1], and we observe the fairness guarantee on a hop-by-hop basis.

E. Summary

This section has examined the benefits from decomposing end-to-end congestion control in three scenarios. In scenarios I and III, we investigate the improvement in the behavior of our streaming application in the steady state. The first scenario captures the possible improvement if we are able to localize congestion to just one segment. The third scenario reflects the situation in which the two segments are *equally* and *independently* congested.

Our simulation result shows that when we place an intermediate node in the middle, we can double the reception of layers in the first scenario and improve it by a factor of four in the third scenario. Because TFRC conforms to TCP throughput equation, a shorter RTT implies a better average number of layers. Moreover, we observed a smaller deviation in the average number of received layers results from localization of retransmission. We also examined the improvement in the retransmission latency while adding a little extra delay (10 to 20 msec) in the transmission latency. In both scenarios, retransmission latency was reduced by the time difference in retransmission paths.

In scenario II, we studied the dynamic behavior of the decomposed congestion control in the similar setting to that of the scenario I. The shorter feedbackloop results in improvement in both responsiveness and aggressiveness by roughly a factor of 2 and 4, respectively, when compared to the end-to-end congestion control.

In general, hop-by-hop congestion control is more effective than end-to-end control for short term congestion. Earlier research on ABR in the context of ATM networks came to a similar conclusion [15]. This study goes further, however, in that it shows how a congestion control mechanism designed for realtime applications can benefit from hop-by-hop control, especially when coupled with application-based adaptation. In addition, our approach still conveys end-to-end semantics to the source, which is useful when persistent congestion occurs.

IV. Conclusion

This paper makes two contributions. First, we point out the opportunity to exploit intermediaries to improve the delivery of streaming media in an overlay network. Better performance can be achieved if intermediate nodes are both application-aware and congestion-aware. The decomposition of congestion control from end-to-end to hop-by-hop results in several advantages: responsiveness, aggressiveness while maintaining smoothness in steady state, and improved video quality. This comes at the cost of a small increase in latency and jitter.

Second, we define the interface between layered video application and a particular TCP-friendly transport protocol: TFRC. We identify the limitations of the current TFRC, from the application point of view, and propose a viable way to make the current TFRC work with layered video applications.

References

[1] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer, "Equation-based congestion control for unicast applications," in *SIGCOMM*, Stockholm, Sweden, 2000, pp. 43–56.

[2] Dorgham Sisalem and Henning Schulzrinne, "The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme," in *Proceedings of NOSSDAV*, Cambridge, UK., 1998.

[3] I. Rhee, V. Ozdemir, and Y. Yi, "Tear: Tcp emulation at receivers – flow control for multimedia streaming," 2000.

[4] Reza Rejaie, Mark Handley, and Deborah Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *INFOCOM (3)*, 1999, pp. 1337–1345.

[5] George Fankhauser, Marcel Dasen, Nathalie Weiler, Bernhard Plattner, and Burkhard Stiller, "The WaveVideo System and Network Architecture: Design and Implementation," 1998, http://www.tik.ee.ethz.ch/ wavevideo/Publications.html.

[6] David Andersen, Hari Balakrishnam, Frans Kaashoek, and Robert Morris, "Resilient Overlay Networks," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001, pp. 131–145.

[7] Yang-Hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," in *Proceedings of the ACM SIGCOMM '01 Conference*, August 2001, pp. 1–12.

[8] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, "A Case For End System Multicast," in *Proceedings of the ACM SIGMETRICS '00 Conference*, June 2000, pp. 1–12.

[9] D. Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, and T. Raleigh, "Protocol Boosters," *IEEE Journal on Selected Areas in Communications (Special Issue on Protocol Architectures for 21st Century Applications)*, vol. 16, no. 3, pp. 437–444, April 1998.

[10] Van Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 314–329.

[11] Jitedra Padhye, Victor Firoiu, Don Towsley, and Jim Krusoe, "Modeling TCP throughput: A simple model and its empirical validation," in *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, Vancouver, CA, 1998, pp. 303–314.

[12] Pedro Reviriego Vasallo, "Variable Packet Size Equation-Based Congestion Control," http://www.icsi.berkeley.edu/ftp/pub/techreports/2000/tr-00-008.pdf.

[13] "ns-2 Network Simulator," http://www.isi.edu/nsnam/ns/2001.

[14] S. Floyd, M. Handley, and J. Padhye, "A comparison of equation-based and aimd congestion control," 2000.

[15] Raj Jain, "Congestion control and traffic management in ATM networks: Recent advances and A survey," *Computer Networks and ISDN Systems*, vol. 28, no. 13, pp. 1723–1738, Oct. 1996.