IMPLEMENTING CONGESTION CONTROL IN THE REAL WORLD

Ladan Gharai   *Colin Perkins*

University of Southern California
Information Sciences Institute

ABSTRACT

It is well known that congestion control is a key issue for the safe deployment of multimedia applications over IP. We describe our initial experiences implementing TCP-friendly congestion control in a system designed to deliver HDTV content over IP. In particular we discuss the effects of packet reordering on the calculated throughput, and highlight the problems this can pose for high-rate applications.

1. INTRODUCTION

Given the proliferation of high speed networks and multimedia applications, it is becoming increasingly important to consider congestion control. This is especially critical for applications with unusual bandwidth requirements, due to their potential to disrupt existing network traffic.

An example of the emerging class of ultra-high rate multimedia applications might be delivery of gigabit rate high definition television (HDTV) signals over IP networks. We have implemented such a system [7], at a constant data rate of 850 Mbps, and have experience of the problems such high rate traffic can cause. To make this application safe for use outside carefully controlled testbeds, we desired to implement congestion control. This paper describes our initial experiences with TCP-friendly rate control of this application.

The paper is organized as follows. Section 2 describes the demonstrator system, and outlines algorithms for multimedia congestion control. Section 3 describes our implementation, while Sections 4 and 5 discuss experimental setup and results. The lessons learnt from our experiment are described in section 6, along with directions for further work. Finally, Section 7 concludes the paper.

2. BACKGROUND

In previous work, we developed a prototype telepresence system that uses HDTV equipment to provide very high quality telepresence over IP networks [7]. The system runs at rates of approximately 850 Mbps, delivering 1280x720 pixel video at 60 frames-per-second in 24-bit YUV color. It is implemented with off-the-shelf components: a PC-based server running Linux, with HDTV I/O and gigabit Ethernet cards. It uses standard RTP over UDP/IP network transfer protocols [8, 4].

Our wide area tests with this system proved the viability of transporting high bandwidth video streams over IP. However, they also highlighted a severe limitation: due to the lack of congestion control our tests could only be conducted with permission, and careful monitoring, from the network operations staff, so as to ensure that such a high-rate non-congestion controlled stream did not adversely affect other traffic on the network.

In order for multimedia traffic and TCP/IP flows to co-exist and receive a fair share of available bandwidth, the non-TCP traffic must be TCP friendly. A TCP friendly flow will fairly share bandwidth with other flows, while judiciously seeking free bandwidth. It has been shown that, for a saturated steady state TCP sender, throughput is proportional to inverse of the square root of the packet loss rate, p [5]. This is known as the TCP-friendly equation, and it provides an upper bound on the steady state throughout T, for packet size S, round trip time R, retransmission timeout $t_{RTO} \approx 4R$ and the steady state loss event rate p, such that:

$$T = \frac{S}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \qquad (1)$$

Utilizing the TCP-friendly equation has resulted in a class of equation based congestion control schemes, such as the TCP friendly rate control (TFRC) protocol [3]. The basic concept is to regulate throughout using equation 1, guaranteeing that the flow is TCP-friendly. Once a sender is aware of the loss event rate p and the round trip time R, it can compute its fair share of bandwidth and adjust its sending rate accordingly. Damping is applied, to ensure that the rate of adaptation is smoother than TCP, while maintaining long-term fairness. The dynamics of TFRC, and its interaction with TCP, are described in [3].

3. DESIGN AND IMPLEMENTATION

TCP friendly rate control relies on the sender being able to adjust its sending rate according to the amount of loss the flow is experiencing. In TFRC, loss is measured as a *loss event fraction* by the receiver. TFRC distinguishes between loss fraction and loss event fraction, to better emulate TCP. Loss event fraction measures the fraction of loss occurring more than one round trip time (RTT) apart. In other words, once an initial loss occurs, any other following loss within a RTT is ignored. This closely mimics most TCP variants.

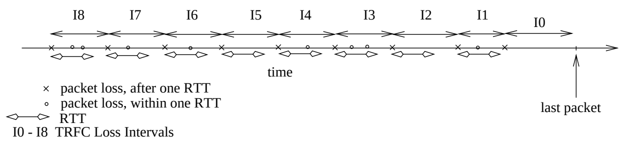
I8
I7
I6
I5
I4
I3
I2
I1
I0
time
× packet loss, after one RTT
∘ packet loss, within one RTT
RTT
I0 - I8 TRFC Loss Intervals
last packet

Figure 1: TFRC Loss Intervals.

Handling of loss intervals in TFRC is shown in Figure 1. TFRC recommends the use of $N = 8$ intervals, however as seen in Figure 1, $N+1$ intervals are actually maintained. To compute the average loss interval, TFRC chooses the maximum of the values of $\sum_{n=0}^{7} I_n$ and $\sum_{n=1}^{8} I_n$. Therefore, if the interval since the last packet loss event, I_0, is large, it is accounted for in the computation of the loss event rate, helping TFRC increase its sending rate in the absence of loss.

To implement TFRC, the following two feedback loops are needed: first, the sender must periodically send perceived RTT to the receiver, thereby allowing the receiver to compute the loss event rate, p. Secondly, the receiver must send the computed lose event rate, p, back to the sender. Figure 2 illustrates the process.

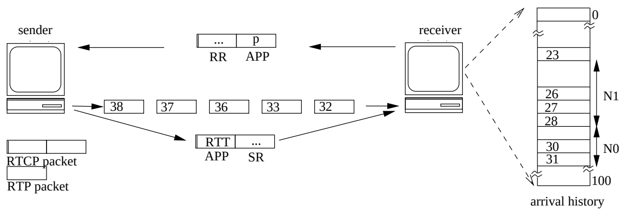

Figure 2: TFRC feedback loops implemented in RTCP.

Our implementation uses RTP over UDP/IP transport. RTP provides feedback using the RTP Control Protocol, RTCP. At regular intervals, implementations generate Receiver Report (RR) or Sender Report (SR) packets, providing reception quality feedback and support for lip-synchronization. Application specific feedback is supported using APP packets, that are piggy-backed at regular intervals with RR or SR packets. In our implementation, each time the sender generates a sender report it also sends the RTT to the receiver in an APP packet. Likewise, when the receiver sends back a receiver report it also includes an APP packet with the latest computation on the loss event rate, p.

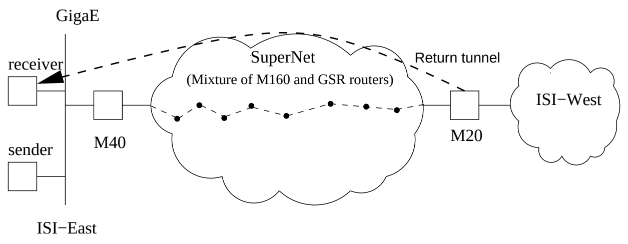

Figure 3: The network used in our tests.

4. EXPERIMENTAL SETUP

To test our system, we need a wide-area network capable of supporting high rate UDP flows. Several such networks have become available recently, including Internet2 and the DARPA SuperNet testbed. We report on tests conducted using SuperNet (previous experiments have used Internet2).

The SuperNet testbed comprises several research networks, connected using a cross-country overlay on a commercial ISP network. The individual research networks are multi-gigabit capacity, and the overlay is intended to support gigabit rate applications. In practice, the capacity of the overlay network varies with the load on the underlying network.

The network path we tested is shown in Figure 3. The wide area path from ISI East in Arlington, VA, to ISI West in Los Angeles is nine IP hops. We configured a tunnel to return traffic from the router in LA, looping traffic back to our laboratory. This allows us to display the results, and gives a network path with 10 logical – 18 actual – hops and a 132ms round trip time.

The sender and receiver are Dell PowerEdge 2500 servers with dual 1.2GHz Pentium III processors, running Linux 2.4.2. They are equiped with 3Com 3c985 gigabit Ethernet and DVS HDstationOEM HDTV interface cards. We capture live HDTV content, packetize and transmit RTP packets destined for the tunnel interface of the receiver. The routing is such that the packets traverse the network before returning though the tunnel to the receiver, where they are depacketized and displayed. The full rate of the system is 850 Mbps, although it can adapt by sending at reduced frame rate.

When the underlying network is lightly loaded, we have consistently been able to run cross-country HDTV-over-IP

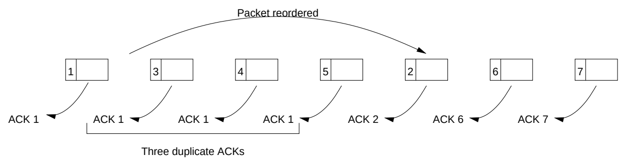

Figure 4: Packet reordering gives the appearance of loss

at 850 Mbps without packet loss. As the network becomes more loaded, typically during business hours, we see packet loss in our application, indicating congestion in the network.

5. EXPERIMENTAL RESULTS

We conducted a number of experiments with our system, both local area and on the wide area network described in Section 4. As expected, the network performance varied: much of the time it was loss free, but there were instances when packet loss was observed, making congestion control necessary.

We are still evaluating the performance of our system in the presence of packet loss, and tuning our congestion control and rate adaptation algorithms. These results are outside the scope of this paper (although we discuss the issues in Section 6). The results we present here reflect our experience when the network was lightly loaded, and loss free.

In the absence of packet loss, we noticed that our congestion control function was suggesting we send at a relatively low rate (and was stable at that rate). This was somewhat unexpected, since TCP performance, and by extension the performance of TFRC congestion control, is driven mostly by packet loss. Indeed, a naive interpretation of equation 1 would say that zero packet loss should result in infinite rate.

That interpretation does not, however, take into account the effects of packet reordering in the network. Experiments showed that some amount, up to 1.3% depending on time of day, of packets were reordered (a value not incompatible with [1, 2, 6]).

Our hypothesis is that reordering causes the congestion control function to return lower-than-expected rates. For example, packets that arrive at least four places out of order would cause TCP to deliver a triple duplicate ACK, giving the appearance of loss (see Figure 4). The analysis behind the TCP-friendly rate control equation [5] reflects this, so TFRC can also be expected to treat reordering as loss.

To validate this hypothesis, we took a closer look at packet reordering and how it effects the computation of the loss event rate. The results shown in Figure 5 plot the evolution

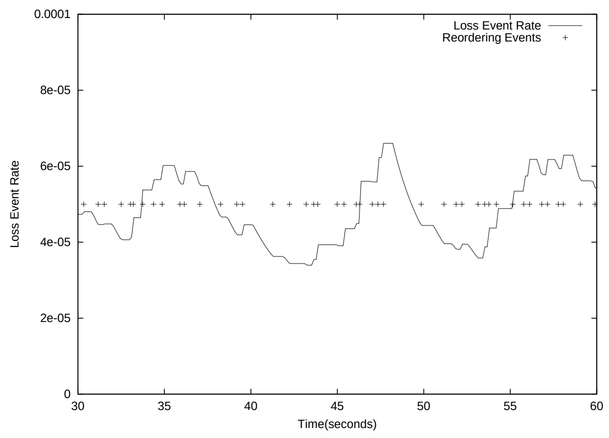

Figure 5: Evolution of Loss Event Rate due to reordering.

of the loss event rate along with reorderings that give the appearance of loss and start a new loss interval, I_n, as defined by TFRC. It is evident that changes in the loss event rate correlate with the new intervals, demonstrating that significant packet reordering causes TFRC to change its transmission rate.

It is also interesting to note that throughout the graph, when new loss intervals are substantially spaced apart, this results in a gradual reduction in the loss event rate. A good example of this occurs at about second 48 in the graph. As discussed in Section 3, TFRC may or may not include the last interval I_0 in its computation of the average loss intervals. Clearly, around point 48 second in the graph, due to lack of loss, I_0 gradually grows, and this growth correlates to the gradual reduction of the loss event rate.

As an additional validation step, we conducted a number of performance tests with TCP traffic. Although there was not an exact match, we found that – after the hosts were tuned for optimal performance – the Linux TCP stack gave comparable throughout to that predicted by our congestion control function. Our results show the Linux TCP achieving throughput on the order of twice that of our TFRC implementation. This is somewhat more than expected, perhaps due to the use of SACK TCP in Linux which is less sensitive to reordering than the Reno TCP used in the derivation of the TCP-friendly equation, but not unreasonable. Detailed comparison of TCP and TFRC throughput in the presence of reordering is ongoing, but omitted here due to lack to space.

We also note that the fraction of reordered packets we observe appears to be somewhat independent of the transfer rate. This can be expected to disrupt the operation of the congestion control algorithm to some degree.

6. LESSONS LEARNT AND FUTURE WORK

First, and foremost, our experience has taught us that packet reordering is not innocuous, even on the scales of 0.2%. The results presented show that TFRC loss events caused by packets arriving too late and out of order can significantly affect throughput in the absence of actual packet loss.

Our implementation utilizes RTCP to provide the feedback loops needed by TFRC. Since feedback timing is important, and directly impacts calculation of the loss event rate, we are investigating the interaction between RTP and the TFRC protocol. In particular, how often loss event and round trip time information can be communicated, and how the transmission rate can be adapted.

As noted in Section 3, we piggyback feedback information into RTCP APP packets. Standard reporting intervals are on the order of seconds, too slow for effective TFRC feedback, but the reduced reporting interval of

$$T_{RTCP} = 360/B_{session} \qquad (2)$$

where $B_{session}$ is the session bandwidth expressed in kilobits per second may be used. For our application, this corresponds to a report every 400μs on average, easily allowing feedback at least once per round trip time (although the processing load may prohibit this).

Processing load is also an issue when implementing the loss interval calculation. We noticed that our implementation observed packet loss at a lower data rate when the calculation of the TFRC parameters was enabled, even if they were not used to control the sending rate. Investigation pointed to the calculation of the average loss interval: performing this computation for every packet is a significant bottleneck, especially for high-rate sources (tests show that the loss event calculation, for a full rate HDTV source, consumes 14% of the CPU on an otherwise unloaded host).

There are also issues with rate adaptation, since the obvious method of changing the transmission rate – adapting the video frame rate – will cause significant step changes in the throughput, and cannot choose any arbitrary rate. TFRC assumes the TCP-friendly rate can be selected, and it is not clear how deviations affect the system behavior. These issues also feed into the human factors of the system: not only must the rate adaptation fit the dictates of TCP-friendly behaviour, it must be chosen to avoid disturbing viewers with sudden quality changes.

7. CONCLUSIONS

When discussing congestion control, it is common to focus on packet loss, since that is the primary driver in TCP, and TCP-friendly, congestion control. There are, however, real-world IP networks in which packet loss is a extremely rare event, but where packet reordering is not infrequent. Our measurements show that this reordering limits the transmission rate of both native TCP flows, and multimedia flows controlled by the TCP friendly rate control protocol.

We understand the desire to be TCP-friendly, but it is not clear that this behavior is appropriate for multimedia applications. Indeed, one of major philosophies in the design of RTP was Application Level Framing, making applications tolerant to packet loss and reordering. We believe that, if the network is not congested, emulation of TCP's response to packet reordering is overly conservative.

To allow the deployment of high-rate multimedia, such as HDTV-over-IP, it is necessary to develop congestion control that is both safe and usable. The TFRC protocol is clearly safe, but we have demonstrated scenarios where its overly conservative nature limits its usefulness. It is desirable to develop modifications to TFRC that decouple its response to congestion and packet reordering, so that reordering without congestion ceases to be a limiting factor.

8. ACKNOWLEDGMENTS

This work is supported by DARPA ITO and by hardware donated by Intel corporation.

9. REFERENCES

[1] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, December 1999.

[2] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, January 2002.

[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation based congestion control for unicast applications. In *SIGCOMM Symposium on Communications Architectures and Protocols*, 2000.

[4] L. Gharai, G. Goncher, C. Perkins, D. Richardson, and A. Mankin. RTP payload format for SMPTE 292M. Internet Draft, Internet Engineering Task Force, February 2002. Work in progress.

[5] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *ACM Computer Communication Review*, 28(4):303–314, September 1998.

[6] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions of Networking*, 7(3), June 1999.

[7] C. S. Perkins, L. Gharai, T. Lehman, and A. Mankin. Experiments with delivery of HDTV over IP networks. In *Proceedings of the 12th International Packet Video Workshop*, Pittsburgh, April 2002.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF Audio/Video Transport Working Group, January 1996. RFC1889.