Congestion Control for Interactive Media: Control Loops & APIs

Varun Singh
Aalto University
varun@comnet.tkk.fi

Jörg Ott
Aalto University
jo@comnet.tkk.fi

Colin Perkins
University of Glasgow
csp@csperkins.org

23 June 2012

Abstract

Standardisation of Web Real-Time Communications (WebRTC) and Telepresence will likely lead to greatly increased deployment of interactive multimedia applications on the Internet. The resulting high-rate multimedia traffic may cause congestion, but few congestion control solutions have been proposed. We suggest that the congestion control, codec, and application be coupled and engage in a dialogue on the correct response to congestion; and that congestion control consider semantic feedback, not just transport-layer metrics.

1 Introduction

The development of the WebRTC conferencing standards in the IETF and W3C, and parallel work on telepresence in the IETF CLUE working group, can be expected to kickstart growth of interactive media applications on the Internet. When coupled with the rise in HTTP streaming video and IPTV, we see that multimedia traffic will soon (if not already) comprise the majority of network traffic.

The congestive load caused by non-interactive streaming video and IPTV flows has been widely discussed, and is kept manageable by using TCP-based adaptive bit rate streaming (e.g., MPEG DASH and related systems), or ISP-managed IP multicast delivery. The problem of congestion control for interactive multimedia applications has received much less attention, however, but is now potentially urgent to solve before WebRTC clients become widely available in all web browsers.

In this position paper, we argue that effective congestion control for interactive multimedia applications requires close coupling between the media codec, the network, and the logic driving the congestion control loop. This coupling can only be realised by giving the multimedia application insight into both codec operation and network behaviour, through a cross-layer API that allows the application to make policy choices to direct the operation of the codec based on changing network conditions, and an understanding of both the application requirements, user preferences, and quality of experience (QoE) constraints.

2 Background

Interactive multimedia traffic on the public Internet is subject to the vagaries of a best effort IP network. This includes packet loss, variable queuing delay, and potential packet reordering. Loss is generally, but not always, due to congestion and queue overflow. Buffer bloat [2] and drop-tail routers cause excess delay and bursty loss.

Media codecs have variable rate output. This can be be modelled as a sequence of occasional large I-frames, with many smaller P-frames in between. This comprises a group-of-pictures (GOP). The output rate can be highly bursty. A codec will have maximum and minimum rates it supports, and may only be able to code to a limited set of rates between those extremes. Codecs may also be limited in when they can adjust their rate, for example they may only be able to vary their encoding parameters at a GOP boundary.

Applications use RTP [5] for media transport. RTP provides reception quality feedback every few seconds, but the RTP/AVPF profile [3] can allow near-immediate feedback for most events of interest. Feedback includes NACKs based on packet sequence numbers, but also semantic feedback such as picture/slice loss indication or reference picture selection. Retransmission or forward error correction (FEC) may also be used to repair loss.

3 Control Loops

Observing the operation of rate adaptive interactive multimedia applications, we see three control loops: between the codec and sending process[1]; between the sending process and network; and between the network and decoder. We discuss each in turn, considering the information needed to make the best end-to-end decisions.

3.1 Codec–Sender Control Loop

The codec encodes media at a particular average rate, subject to variation depending on the amount of activity in the media input, and can adapt that average rate based

[1] We use 'process' in the general sense, and do not intend to imply an operating system process.

on feedback from the sending process. Unlike in elastic applications, there are generally constraints on how and when a codec can adapt. The simplest of these relate to possible sending rates: codecs have a limited range of adaptation, and constraints on what rates can be selected between these limits.

When the sender congestion control algorithm calculates a new rate for the codec, it is necessary to consider how quickly the codec can adapt to the new rate. There may be some considerable lag before the codec rate can change. If codec and congestion control are tightly coupled, and the lag is known, then the rate control can compensate; else the slow control loop can cause instability and oscillation.

One must also consider how a codec can adapt to match the desired rate. This can be done by changing the frame size, by changing the frame rate, or by changing the quality. Any rate change will be noisy due to the variable rate nature of many codecs and changes in the input signal. The application should be able to control, or at least suggest preferences, on what it is appropriate, and should be consistent in its decision since alternating between strategies may produce suboptimal results.

Effective operation requires a dialogue between the sender process, with a desired sending rate based on network conditions, and the media codec which has constraints on what rates it can adopt, how quickly it can change rates, and the effect of rate changes on the user experience.

As an example of how this dialogue might affect the operation of the system, many codecs produce I-frames on a fixed schedule, irrespective of how the traffic bursts they cause fit into the available network capacity. A smart media codec might cooperate with the network to delay sending I-frames in some cases, when capacity is unavailable. Similarly, a congestion control algorithm that is forced, due to codec constraints, to send at a lower rate than TCP would achieve over the same path, might defend that lower share of the capacity aggressively, to avoid being beaten down by competing flows.

3.2 Sender–Network Control Loop

The sender process frames data for transmission over the network, and transmits RTP packets based on the rate determined by the congestion control algorithm. It collects quality feedback, in the form of RTCP reports, to determine the future sending rate.

The receiver can accurately monitor packet reception, but has comparatively limited scope to feed information on network condition back to the sender (a correctly configured RTP/AVPF stack will allow *most* events to be reported, but not fine-grained per-packet feedback). The limited bandwidth of the feedback channel can be important when it comes to distinguishing between congestive and non-congestive loss, for example when using wireless networks, or if an application wants to detect the onset of congestion by monitoring changes in packet timing before they lead to packet loss.

For example, congestive packet loss in drop-tail queues is preceded by increased end-to-end delay and discards due to late arrival of packets at the receiver. This suggests that it is important to give frequent feedback on RTP packet timings, but there is insufficient RTCP bandwidth to do so on a per-packet basis (like delay-based TCP variants) within the constraints of RTCP. In other environments, however, such frequent feedback is not useful. If a router implements an active queue management scheme like, for example WRED, then excess queuing delay and discards due to late arrival should not occur. This makes per-packet timing feedback unnecessary, but complicates distinguishing between congestion and bit-error losses.

The difference between testing in real-world deployments and in simulations is also important to consider. This affects the accuracy of RTT measurements, impacting delay based algorithm (e.g., TFRC) [4]. Time slot driven simulation systems, such as *ns2*, have accurate timing that is not representative of real-world systems.

A challenge with the sender–network control loop is adapting given limited feedback, and distinguishing different network conditions and environments. *Semantic information* becomes critical: when there is insufficient feedback bandwidth to give accurate per-packet feedback, the sender must rely on higher-level summary feedback of the *meaning* of network events, both for their cause and the correct application-level response. Trust in the receiver feedback accuracy is also important.

3.3 Network–Receiver Control Loop

The receiver has a playout buffer of data waiting to be decoded and rendered. Given a rapid reaction, and a sufficiently low RTT, it is possible to correct or conceal missing packets. This can be effective at improving QoE.

A receiver can send packet-level feedback based solely on sequence numbers and arrival times. Alternatively, if the playout and codec decode buffers are combined, it may infer the effect of loss on the codec state, and send picture/slice-loss feedback to inform the sender of the impact on the decoded video. This is more meaningful, and requires fewer feedback packets. Separate jitter, decoding, and playout buffering also result in excess delay. A single buffer is desirable, taking into account network jitter, decoding look-ahead, and repair time.

4 APIs for Adaptive Media

The discussion in Section 3 suggests that significant performance penalties can arise from the mismatch between codec expectations and the demands of the congestion control algorithm, and because the network APIs do not expose sufficient information to allow effective rate adaptation for multimedia flows. To rectify this, we suggest that interactive multimedia applications need new APIs that expose details of the delivery process. These APIs will allow applications and codecs to engage in a much closer dialogue with the transport layer, and better manage the delivery process.

Giving the application visibility into packet timing and delivery is needed to allow application-level feedback. Accordingly, multimedia transport APIs should be structured to promote coupling between the application, codec, and network. This suggests exposing transport-layer (e.g., UDP) packet timing and loss events in the RTP encoding and de-jitter/playout buffers, rather than trying to abstract away these details. These processes may be exposed via triggers and callbacks to application code, or by making the entire transmit and receive paths visible to the application. Use of triggers and callbacks allows applications to provide policy input and a smart response to congestion, while avoiding the complexity of exposing all the details of the media transport path.

As an example, congestion control algorithms that monitor variation in one-way packet delay should be able to reveal the estimated available network queuing capacity. With appropriate callbacks when capacity is available, this should allow cooperation between applications and codecs to schedule traffic bursts at non-disruptive times. For example, I-frames can be sent to refresh decoder state, but there is little point in doing this if the traffic burst due to the I-frame will cause loss that will disrupt decoder state.

Visibility into connection dynamics will also allow applications to chose how they allocate bandwidth between flows. For example, if a sender has multiple video cameras, it might have application-specific reasons to give priority to one flow over another at certain times, which cannot be known by the transport without cooperation. The application should also be able to monitor video and network statistics to adapt application behaviour: an applications may want to define a minimum video quality below which it may consider switching to voice only or terminating the call.

The transport layer API should allow an application and codec to proactively manage the repair process, with late data modifications at the sender side via a cooperative API to ensure appropriate repair response (e.g., to allow replacement of a queued but not yet sent segment in response to feedback from the receiver). Receivers should be aware of the state of the playout buffer and expected network RTT, allowing them to make appropriate requests for repair. For example, a receiver should be able to determine that a retransmission will not arrive before playout is due, and so be able to send a semantic slice loss indication instead, allowing the sender to adapt the media encoding to compensate for the loss. Alternatively, applications should be able to response to picture/slice loss indications by adapting media coding, as well as by retransmitting lost packets.

A tightly coupled transport interface of this form will give application flexibility to adapt to the network, by engaging in a dialogue between the codec and the transport layer over available capacity, traffic scheduling, and repair, while still ensuring network safety. Cross-layer signalling also allows for passing triggers from the network layer to signal known bandwidth and/or latency changes on edge links, or link addition or removal.

The two principles of application-layer framing and integrated layer processing [1] were fundamental to the design of RTP. They should be applied to congestion control as much as to robustness and the design of RTP payload formats.

5 Conclusions

We have discussed the tension that arises between the congestion control algorithm's expectation of the media codec, the capabilities of the media codec to adapt, and the limits of the feedback available via RTP/AVPF. This tension may lead to lower quality of experience if the media codec, network layer, and application logic do not collaborate with each other to adapt the media to the network. This requires expressive APIs that expose the semantics of loss and delay events, rather than raw metrics; Section 4 describes some features of such APIs.

To date, there has been considerable focus on packet-level dynamics in the discussion of congestion control for RTP sessions. This is expected, since existing congestion control algorithms use this information directly. However, it is not clear that this type of feedback is most suitable for interactive multimedia applications that use RTP/AVPF. We urge designers of congestion control algorithms for interactive multimedia to consider semantic feedback as a lower bandwidth alternative, and to develop APIs that couple media coding, congestion control, and repair feedback loops in a way that allows meaningful policy input from applications and codecs.

References

[1] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. SIGCOMM*, Philadelphia, PA, USA, September 1990. ACM.

[2] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the Internet. *Communications of the ACM*, 55(1):57–65, January 2012.

[3] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP profile for real-time transport control protocol (RTCP)-based feedback (RTP/AVPF). IETF, July 2006. RFC 4585.

[4] A. Saurin. Congestion control for video-conferencing applications. Master's thesis, University of Glasgow, December 2006.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF, July 2003. RFC 3550.