One control to rule them all

Michael Welzl

With the RTCWEB protocol suite, real-time video and audio will be delivered using (S)RTP and UDP, and there is an envisioned "data channel" using SCTP over UDP; all of this traffic is going to use the same port/destination pair, i.e. it will be multiplexed over what will look like one "channel" to intermediate systems.

What form of congestion control to use is currently under discussion. Requirements for RTCWEB congestion control include: minimizing delay, the ability to cope with senders that transmit less data than the network allows, and the ability to adjust priorities between the RTCWEB transfers.

A proposal has been put forward [1]. It realizes congestion control for one of the RTP-based transfers, using RTCP messages as feedback. Since the amount of such feedback is limited by RTP rules, calculations are partially carried out by the receiver and much less feedback is sent than with TCP, for example. Similar to TCP, if the sender does not get feedback for a while, it reduces its rate.

The intention of this paper is to explain why this design is suboptimal, and sketch what the author believes to be a better approach.

Criticism of the existing proposal

Issue 1: a larger amount of feedback may be needed, and should be acceptable

The interval after which a congestion controller can see the effect of its actions is a Round-Trip Time (RTT), and delaying the controller by more than that can obviously harm its performance. A "heartbeat" schedule, determined by a minimum and maximum interval, and bounded by the session's RTCP bandwidth and its rtcp_frr setting, is recommended in [1]. These parameters are not necessarily related to the RTT.

A larger amount of feedback is acceptable with, e.g., TCP. It is also acceptable to use RTP over TCP. It therefore seems illogical to require that the feedback for congestion control is limited by RTP's rules and not the RTT when RTP is used over UDP. Notably, in case of RTP over TCP, the RTP rule limits the amount of RTCP messages that can be sent *in addition to* TCP's ACKs.

Issue 2: feedback will sometimes be available on the data channel

When a data channel is used in addition to e.g. an audio conversation, as in a multiparty on-line game with voice communication, SCTP is expected to regularly generate transport-layer feedback messages (SACK chunks). These messages may or may not be generated more frequently than the RTCP messages from [1], but anyhow they are assumed to capture the congestion state of the same network path. Ignoring them seems to be wasteful and inefficient – but the split sender/receiver design of [1] requires feedback messages to include a bit-rate estimate, which is not available in SCTP SACK chunks. This will make it hard to make the mechanism additionally exploit the information that SCTP SACK chunks provide.

Issue 3: low delay is a common goal

… which can only be ensured if all participating senders follow the same rules. With the RTCP-based congestion control in [1], it seems obvious that at least the data channel will have to use its own separate congestion controller, and these congestion controls will compete for available bandwidth, contributing to queuing delay. Even if some of the involved transfers are application-limited, the congestion controllers would have to be aware of the total amount of traffic that they all produce together in order to correctly control the impact that this traffic has on the queue at the bottleneck.

Reason 4: controlling priorities
As long as some transfers, such as the data channel, use a separate congestion controller, assigning priorities to traffic translates into adjusting the aggressiveness with which the controllers seek to use the available bandwidth. This is a way to influence the competition for resources at the bottleneck, which is not under control of the host where the data streams originate. As such, the precision with which priorities can be applied is limited.

An alternative design

Part 1: a single congestion controller

Many of the problems listed above disappear when a *single congestion control* instance is used. For instance, with only one congestion control instance, controlling priorities among data streams becomes a scheduling function on the sender side – which can be much more precise than trying to steer competition at the bottleneck, and is fully under control of the sender. If that congestion control instance is designed to minimize delay, it can play out its benefits without RTCWEB "shooting itself in the foot" by causing traffic type #2 that pushes up the bottleneck queue while traffic type #1 tries to keep it small.

The fairness that is achieved by using only a single congestion control instance can translate into massive subjective gains for users. Consider the example illustrated in Figure 1 below, based on TCP: if the congestion window (cwnd) of a transfer is large, a benefit can be attained from making a later, shorter transfer share the same cwnd – otherwise, the shorter transfer can unnecessarily spend many RTTs before it reaches a reasonable send rate. Note that the same logic holds for other types of congestion control, as they all need to – more or less cautiously – probe for the available bandwidth. Also note that this would not only be beneficial for a later, shorter transfer, but also for an application-limited transfer, much in the same way.

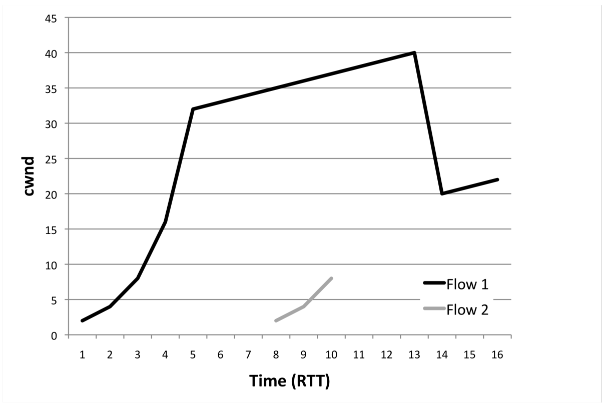

Figure 1: The second, short, TCP transfer is much worse off than the first one.

Figure 2, taken from [2], shows an example of such cwnd sharing. Here, it was attained by mapping TCP transfers onto streams of a single SCTP association. A file transfer of 64 Mbytes (file 1) was joined by a transfer of 1 Mbyte (file 2) after 2.5 seconds in a local testbed (details are given in [2]). The duration of file 2 without mapping was surprisingly long – in a separate test, it took only 0.689 seconds to transfer this file in isolation. This extremely unfair TCP rate allocation does not match the common user expectation of a download's duration to be roughly proportional to the file size. It was caused by the very different cwnd values at the time of starting the second transfer, combined with the exceedingly long default queue used in Linux for the outgoing network interface (the *txqueuelen* parameter of the *eth* device) – the RTT grew from about 60.2 ms to 711 ms when the second transfer began. Note that the benefits in this scenario were obtained despite the fact that this is a very prototypical implementation that uses a proxying approach for simplicity, creating a lot of unnecessary overhead.

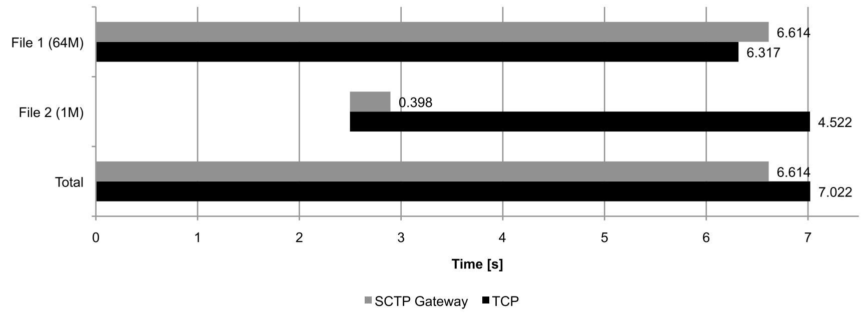

Figure 2: Time required for transferring two files when the longer transfer (file1) is joined by a shorter one (file2) after 2.5 seconds. The grey ("SCTP Gateway") bars show the case where the two TCP flows were mapped onto two streams of a single SCTP association by a proxy.

Part 2: SCTP to the rescue

SCTP can handle multiple independent data streams, and it controls their sending rate with a single congestion control instance. These streams can be reliable or unreliable; they can also have partial reliability, where packets are only retransmitted within a certain time frame – a function that can be very useful for media streams (cf. [3]). SCTP has its own feedback format – SACK chunks – which is independent of RTP; if RTP is used over SCTP, SCTP is just another transport, and the number of these transport-related feedback messages is not limited by RTP rules, similar to using RTP over TCP. If there is still a demand to reduce the total amount of signaling, this can be done in a more appropriate, i.e., dynamic and transport-specific way, cf. ACK-CC [RFC5690].

In conclusion, to resolve all the issues listed above and more, it seems that the best design would be to multiplex all RTCWEB streams – audio, video and data – over a single SCTP/UDP association, and let this association be controlled by a single delay-minimizing congestion controller.

References

[1] H. Lundin, S. Holmer, H. Alvestrand, "A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web", Internet-draft (work in progress) draft-alvestrand-rtcweb-congestion-02, 15 April 2012.

[2] Michael Welzl, Florian Niederbacher, Stein Gjessing: "Beneficial Transparent Deployment of SCTP: the Missing Pieces", IEEE GlobeCom 2011, 5-9 December 2011, Houston, Texas.

[3] Michael Schier, Michael Welzl: "Optimizing Selective ARQ for H.264 Live Streaming: A Novel Method for Predicting Loss-Impact in Realtime", IEEE Transactions on Multimedia 14(2), April 2012.

[RFC5690] Sally Floyd, Andrés Arcia, David Ros and Janardhan R. Iyengar, "Adding Acknowledgement Congestion Control to TCP". Informational RFC 5690, IETF, February 2010.