On Fairness, Delay and Signaling of Different Approaches to Real-time Congestion Control

Stefan Holmer, Google, holmer@google.com

Introduction

This paper touches three different topics regarding congestion control for real-time flows. First the difference between loss-based, receive-side delay-based and send-side delay-based congestion control algorithms is discussed. Next the difficulties with self-fairness is discussed, focusing on a particular scenario where, for example, Receive-side Real-time Congestion Control (RRTCC) [1] is shown to have problems.

Differences in Delay and Signaling

To be able to build a good congestion control algorithm for streams which are to be consumed in real-time, it's important to have clear requirements. Typical requirements can be; avoid congestion, fair usage of the bottleneck link, low end-to-end delay, low packet loss, full utilization, etc. Obviously these would have to be defined in more detail, and in the end some of them may end up being conflicting.

For conversational real-time flows I'm of the opinion that the end-to-end delay should be kept reasonable whenever possible, where reasonable is something around 150 ms (acceptable for most user applications according to [2]) from microphone to speaker. Considering that other parts of the end-to-end chain, such as capture, encoding, dejittering, decoding and rendering also need their share of processing time, it is therefore necessary that a good algorithm tries to keep the network delay as low as possible whenever it can. Given the definitions of congestion and contention in [3], such an algorithm should aim to minimize contention rather than congestion, which allows it to avoid the degenerate state of buffers becoming full, defined as congestion. It must at the same time allow for some delay variations required to get good, smooth media quality. Assuming a lot of routers are using tail-drop queueing, this requirement is not always fulfilled using algorithms such as TFRC which are solely based on packet loss event rates [4]. A TFRC flow will effectively increase its bitrate until a loss event is identified, which only happens when the bottleneck queue is full if that particular queue employs a tail-drop queue.

Assuming that there is still some time until most routers are using active queue management, the best approach is to employ an algorithm which, besides triggering on packet loss events, also is delay-sensitive. There are different approaches to delay sensitive congestion control, for example the detection and control can be done at either the send-side or the receive-side. Typical approaches to send-side algorithms try to estimate a base one-way delay and compares that with an estimate of current one-way delay. Receive-side algorithms typically observe the one-way delay differential, and therefore don't need to estimate the base one-way delay. Estimating the base one-way delay is prone to issues with bias when different endpoints have different estimates of it, which leads to known problems with late newcomer flows [5]. This is not an issue when observing one-way delay differentials. It should be noted that it is also possible for a send-side algorithm to observe the differential instead of absolute delay.

These two types of algorithms also differ in the amount of necessary feedback messages. A receive-side based algorithm typically only needs to feed back a suggested target bitrate to the sender, while a send-side based algorithm needs information about each packet arrival time at the receiver to be able to estimate the one-way delays. For the send-side approach protocol changes are also often necessary to employ new signals, such as ECN, which the receive-side approach would be able to use without changes in the feedback messages.

Self-fairness

Apart from keeping the delay low, it is also reasonable to require that flows using the algorithm are fair. There are several definitions of fair, and here I will focus on self-fairness. An informal definition of self-fairness could be:

“N flows controlled by the same congestion control algorithm are self-fair if they converge to using the same amount of bandwidth.”

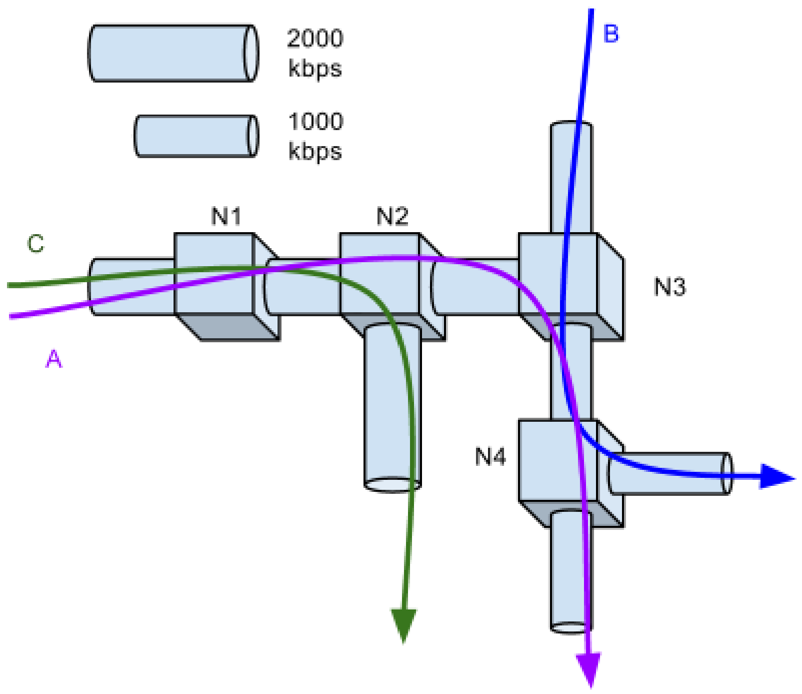

Figure 1. Example of a network which may cause self-fairness issues for delay-based congestion control algorithms.

Being fair according to this definition isn’t as easy as one may first think. For instance consider the network in Figure 1, where two flows, A and B, are competing over the same bottleneck link. At some link, not shared with flow B, flow A is meeting a significant amount of cross-traffic. Although this link is not the bottleneck of flow A, it still affects the arrival time of its packets significantly.

Let’s assume that we compare the estimated delay to a threshold to decide if the path is congested, or if there is contention along the path, and that we are using an additive increase multiplicative decrease approach to control our flow. The noise experienced by flow A will cause it to appear to be more congested than flow B from a delay-sensitivity perspective, even though that is not the case. This effectively makes flow A use less of the bandwidth than flow B, regardless of whether a receive-side or a send-side approach is used. LEDBAT [6] solves this by instead using a control rule which is linear to the error in delay, that is, the difference between the target delay and the estimated delay. This has been shown to have other problems with fairness [5].

This scenario might not be too bad since it’s actually in some sense fair that flow B, which has a better understanding of the bottleneck link (less noisy measurements) gets to use more of its available bandwidth. But it does shed some light on the difficulty of defining fairness, and to actually build an algorithm which is fair in all situations.

It turns out that RRTCC also has problems with self-fairness in these situations. An example of this is shown in Figure 2. Since RRTCC is adaptive to the amount of jitter experienced by a flow [1], the inter-arrival times of flow A will be filtered more than those of flow B. Therefore, it will take longer for flow A to detect contention than it will take flow B to do the same detection, and flow B will therefore have a greater chance of backing off. Simulations show that after approximately 30 seconds the flows have converged, which will happen when either of the following is true:

1. Flow A is limited by another bottleneck.
2. The jitter induced by flow A on flow B is equal to the jitter experienced by flow A.

We can get around this problem by having a constant amount of filtering irrespective of the amount of jitter, which is how many other delay based algorithms solve the problem. But as mentioned earlier, this still doesn't guarantee self-fairness, since flows experiencing more jitter than the amount of filtering can cope with will have more frequent false detections.

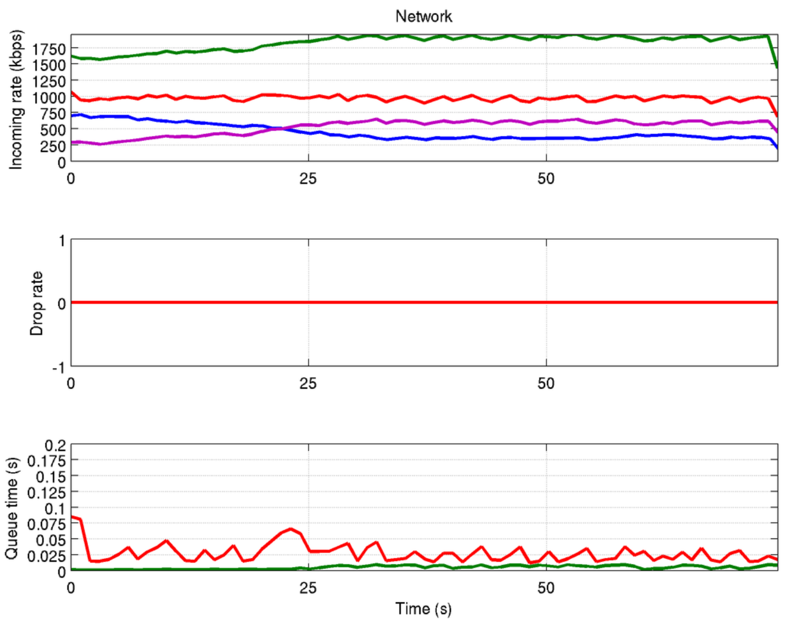

Figure 2. Example of the self-unfairness of RRTCC in a situation as described in Figure 1. In the first plot, the purple line is the rate received by the receiver of the purple flow, and the blue line is the rate received by the receiver of the blue flow. The green line is the total rate received by node N1 and the red line is the total rate received by node N3. In the second and third plots the drop rate and queueing delays of these nodes are shown.

Conclusions

The conclusion to be drawn from this is that, in order to have reasonable conversation quality over the Internet today, a delay-based algorithm is required, but it isn't obvious how to design a delay-based algorithm which guarantees full utilization and self-fairness in all situations.

References

1. H. Lundin, S. Holmer and H. Alvestrand, “A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web,” IETF Informational Draft, April 2012.
2. ITU-T G.114, February 1996.
3. S. Bauer, D. Clark and W. Lehr, “The Evolution of Internet Congestion,” TPRC, August 2009.
4. S. Floyd, M. Handley, J. Padhye and J. Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification”, IETF RFC 5348, September 2008.
5. G. Carofiglio, L. Muscariello, D. Rossi and S. Valenti, “The quest for LEDBAT fairness,” Proceedings of IEEE GLOBECOM 2010, December 2010.
6. S. Shalunov, G. Hazel, J. Iyengar and M. Kuehlewind, “Low Extra Delay Background Transport (LEDBAT),” IETF Experimental Draft, July 2011.