Congestion Control for Interactive Real-Time Flows on Today's Internet

Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan[1]
M.I.T. Computer Science and Artificial Intelligence Laboratory
{keithw, anirudh, hari}@mit.edu

Introduction. Many of today's Internet paths that pass through a consumer's connection now include quasi-reliable link-layer protocols and very large buffers. On such paths, traditional loss-triggered congestion control (most recently codified in RFC 5681) performs poorly, by filling up multi-second queues that no other application can leapfrog. In this setting, loss-triggered congestion control can be grossly "unfair" to competing flows that care about latency.

On these challenging (but common) paths, delay appears to be the *only* usable signal of bottleneck link speed and contention. More likely than not, traditional loss-regulated TCP flows will hurt delay-regulated interactive flows by filling up buffers — not the reverse. However, a delay-regulated flow must respond quickly to changes in network conditions in order to avoid filling up buffers itself before it adapts.

The call for papers rightly points out that some video-streaming applications may not be able to adjust their output with a short reaction time. We believe it may be worthwhile to discuss a notion of fairness based on a flow's *agility*, measured by the speed at which an application adapts to variations in network conditions.

We write from the perspective of application developers. We are working to apply the State Synchronization Protocol to mobile videoconferencing. SSP is a new protocol that provides an abstraction of real-time *object synchronization*. It was first used in the remote-shell program Mosh,[2] and contains techniques to choose and time its outgoing packets wisely to minimize the application-experienced latency over real-world networks.

Today's networks are different.

Through much of the Internet's development, the dynamics of IP best-effort delivery generally included:

- A stable mean RTT with lightly-tailed jitter
- A stable bottleneck link speed
- Packet loss generally produced solely by buffer overflow or AQM at a gateway
- Network buffers sized to overflow before queue length reached hundreds of ms

Today, a typical Internet user on a wide-area wireless network will experience:

- Varying RTT with transient latency spikes to hundreds of ms (see Figure 1)
- Link speed that varies according to channel quality
- Packet loss caused by intermittent connectivity
- Buffers that do not drop packets until they contain 5–10 seconds of data at bottleneck link speed

This network reality is present in the latest commercial LTE networks and does not show signs of changing any time soon. RFC 5681 TCP congestion control performs poorly on these networks, both in interaction with itself and with interactive flows that depend on low end-to-end delay.

The unfriendly interaction between loss-regulated flows and interactive (delay-sensitive) flows on such networks has been much discussed, and we don't have a bombshell solution. However, it is still possible to state demands of interactive flows that could form a notion of latency-fairness. We propose two:

- In steady state, an interactive flow should not cause network queues to lengthen.
- If network conditions change (e.g. link speed is reduced, another flow joins), an interactive flow should not cause network queues to lengthen by more than a "certain amount." This limit could be stated in terms of extra bytes sent (relative to the amount that would not lengthen queues) or in terms of reaction time. Specifying a best-practice quantity would give guidance to application developers about the compromise their application must make between continuity of behavior, such as not frequently changing coded bitrates, and network friendliness.

[1] One of us (K.W.) would participate in the workshop.

[2] K. Winstein and H. Balakrishnan, "Mosh: An Interactive Remote Shell for Mobile Clients," *USENIX Annual Technical Conference*, Boston, Mass., June 2012.

Figure 1: One-way latency trace for 100 Hz UDP tinygrams over unloaded Verizon LTE connection

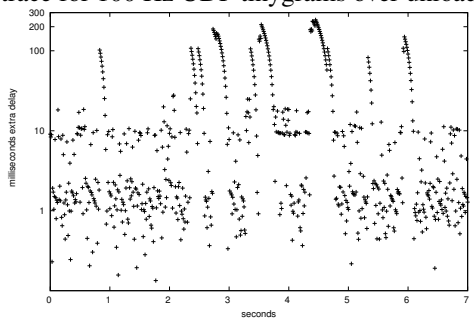
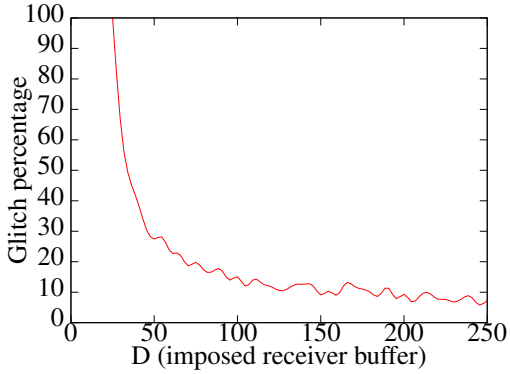


Figure 2: Hypothetical tradeoff curve between glitch proportion and receiver-imposed delay



A utilitarian approach to interactive performance.

We believe that future real-time interactive applications will make explicit performance tradeoffs that depend on their *uncertainty* about future network performance. For example, on a network with constant RTT, a real-time audio receiver may simply play samples as soon as it receives them. But if the receiver fears a possible spike in latency in the future, it would be wise to accumulate a receiver-side buffer, so that it will not cause a "glitch" in the event of a brief network outage.

Figure 1 shows that such "spikes" can be common even on unloaded, high-speed networks. Moreover, spikes are not well-characterized by the RTP/RTCP interarrival jitter estimator specified in RFC 3550. We believe that future applications will need more moments of the distribution of arrival times, and not simply a unitary measure of variance.

In choosing the size of this buffer, the application is making a tradeoff between quality now (low delay) and possible quality later (no "glitch" even if latency increases).

One measure of the tradeoff can be captured in an "expected glitch proportion vs. delay" curve (Figure 2). Assuming an optimal coding scheme, for a particular receiver-induced buffering delay D, causality requires that a "glitch" occur if there is a time T (in the sender's signal) where **no** packets sent at time $x > T$ arrive at the receiver before time $T + D$.

Based on historical packet arrival times, the receiver can calculate an expected percentage of the sender's signal that will not be reconstructed in time by the receiver, for different values of D. Given a utility function that penalizes delay and glitches, the receiver can choose the utility-maximizing value of D for a particular packet-arrival distribution.

Some applications (such as stored video, e.g., Netflix or YouTube), may be willing to accept far larger delays in exchange for a lower glitch probability. Other applications (like Skype) may prioritize low delay above all other considerations. We think the above framework is general enough to express the different tradeoffs made by both types of streaming applications.