# Realtime Congestion Challenges

Randell Jesup, 6/2012

**Abstract:** There are major challenges in addressing realtime flows on the Internet, and tough issues we must discuss and all understand, even if agreement on them may be difficult, such as defining "fairness."

This is partially cribbed from *draft-jesup-rtp-congestion-reqs*. I suggest people also read it, especially for the derived requirements, at: http://tools.ietf.org/html/draft-jesup-rtp-congestion-reqs-00

The traditional TCP congestion control requirements were developed in order to promote efficient use of the Internet for reliable bulk transfer of non-time-critical data, such as transfer of large files. They have also been used successfully to govern the reliable transfer of smaller chunks of data in "as fast as possible" mode, such as when fetching Web pages.

These algorithms have also been used for transfer of media streams that are viewed in a non-interactive manner, such as "streaming" video, where having the data ready when the viewer wants it is important, but the exact timing of the delivery is not.

When doing real time interactive media, the requirements are different; one needs to provide the data continuously, within a very limited time window (no more than 100s of milliseconds end-to-end delay), the sources of data may be able to adapt the amount of data that needs sending within fairly wide margins, and may tolerate some amount of packet loss, but since the data is generated in real time, sending "future" data is impossible, and since it's consumed in real time, data delivered late is useless.

One particular protocol portfolio being developed for this use case is WebRTC, where one envisions sending multiple RTP-based flows between two peers, in conjunction with data flows, all at the same time, without having special arrangements with the intervening service providers.

Other use cases would involve non-media realtime communication, such as telepresence, telerobotics, remote (physical) process control, remote monitoring, realtime games, etc.

There are related uses not strictly falling into the Workshop definition but impacting it, such as background 'scavenger' protocols such as LEDBAT and adaptive streaming and remote DVR type functions where control lag is problematic.

Many applications have (as much as possible) worked around the issues with all sorts of tricks and heuristics - multiple TCP connections (to avoid head-of-queue blocking on loss), LEDBAT (BitTorrent), application-level congestion control and traffic averaging/minimization, etc. Others have implemented non-standard congestion algorithms, usually non-published, and I did this for a former company, using a delay-sensitive algorithm very similar to the WebRTC proposal. Other application algorithms purposely ignore low rates of loss, or adapt very slowly, especially up (such as Skype appears to).

There are many challenges:

- competition with AIMD loss-based algorithms such as TCP (in the many variants it has, and one must test against most than just the standard)

- one-way-delay algorithms such as LEDBAT and the standing queues it tries to provoke
- BufferBloat and all the implications that spring from it, especially when competing with TCP flows
- WiFi routers and aggressive buffering/retransmission
- proper response to current and future AQM deployments
- how to define fairness for a realtime flow versus (say) a webpage load
- how to ensure fairness versus other realtime flows
- interactions with traffic-classification hidden within the network
- how to best handle notifying the network about the traffic in the stream and how to avoid that being misused for extra priority
- and there are more.

Using LEDBAT as an example, if it's widely deployed, especially in a background application such as OS/app updates or for cloud backup, it will induce standing 100ms queues in bottleneck routers, which will be quite problematic for quality realtime communication or remote control.

I won't reiterate the list of requirements from the draft I see as needed for a short-term solution for WebRTC; those will be addressed in the BoF and hopefully-following-on Working Group.

Some tough issues that be dealt with:

- What is fairness in this context?

  Media streams vs 'scavenger' streams vs streaming HTTP flows versus dynamic pageloads.... Browsers typically "warm up" multiple connections (circa 6) and play games (load-balance, etc) to get better loadtimes; and websites "shard" their sites to encourage browsers to open even more connections (images1.blah.com, images2.blah.com, html.blah.com, etc). This leads to extreme bursts of congestion as all these connections pull data at the same time; traditional congestion control doesn't get much opportunity to function, and even if it does the losses may go to the "wrong" flow to help ease the problem.

  How do we avoid realtime flows being hammered down to minimum bandwidth by long-lived competing TCP flows? Can a realtime flow (at cost to itself) cause TCP to back off enough to keep a high quality of experience? (Likely no, but a worthwhile question.)

- Merging of information across multiple flows to the same destination?

  The more data you have about that pathway the better, but classification may play havoc with it.

  Related: If you have N flows, do you get N times the share of bandwidth? With TCP you do.

- How do we transition to an Internet without standing queues? (AQM, ECN, etc). What happens once this is done; how do we build protocols that retain their characteristics when AQM is active? (LEDBAT doesn't)

- How do we handle flow startup, which is a critical time for realtime flows - if the bandwidth/quality to too low at start, the flow will be abandoned. Does some type of "memory" or historical info help here to set starting points, even if it may be wrong for the current flow?

Agreement on these issues may be tough, but to craft solutions we must think about these issues and understand the impact of our proposed solutions.