Low Latency Requires Smart Queuing: Traditional AQM is not enough!

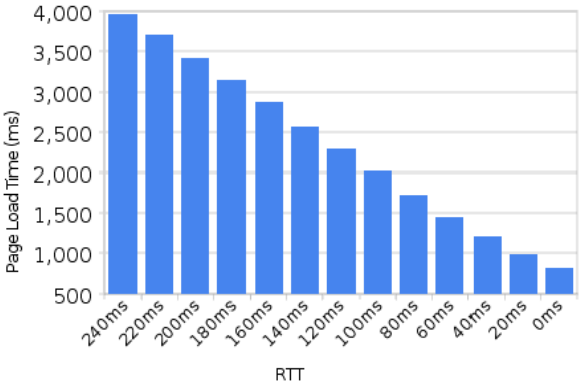
Jim Gettys
Bell Labs
June 23, 2013

Human Perception & Speed of Light

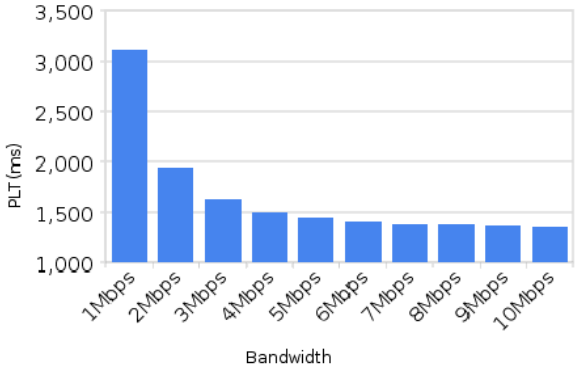
Any unnecessary latency is too much latency.

I've heard many networking engineers and researchers express to me the opinion that 100ms latency is "good enough". If the Internet's worst latency was 100ms today, indeed, we'd be much better off than we are today (and would have space warp technology as well!). But knowledge of the speed of light and human factors can easily demonstrate how far from reality this opinion is.

Performance for most applications is more sensitive to latency than bandwidth: for typical web browsing, today's broadband connections are already past the point of diminishing returns on performance: increasing broadband will not significantly improve performance for anyone, whereas reducing latency will.



Drawing 2: Page Load Time vs. RTT



Drawing 1: Page Load Time vs. Bandwidth

Source: SPDYEssentials, Roberto Peon & William Chan, Google Tech Talk, 12/8/11[1]

Different applications have different latency requirements. These are often set by human perception, which varies by the application; to remind those of us who have not worked in the area of human factors, some numbers I keep in mind include:

- For traditional quality VOIP (or teleconferencing), the acceptable latency is no more than approximately 150ms..

1Page Load Time is sensitive to round-trip latency. Google data shows 14x multiplier. +200ms RTT = +2.8 seconds PLT. Diminishing returns from increased data rate. Page Load Time at 10 Mbps almost indistinguishable from 6 Mbp

- For keyboard key echo to appear "instant", the acceptable latency is around 50ms.
- For "rubber banding" to "feel solidly attached" to your hand, your desirable latency is less than 30ms, or it feels like your feedback is literally connected by a "rubber band".
- Professional musicians playing together may be able to tolerate (but not enjoy) up to 100ms of latency. But for duffers to play music together successfully, the latency may only be 20-40ms.
- Serious gamers (or day traders; is there a difference?) care about even single milliseconds of latency, as they are competing with others and it affects the success of their competition, and strive for under 15ms latency total.
- The lower bound on latency speed of light latency is set by the speed of propagation in the medium and the throughput in that medium: in today's Internet, a trans-continental link across the North America has an inherent latency of approximately 75ms (presuming good routing in the Internet, which is often not available). Speed in glass is about .6C.

We have other latency "costs" we must budget for: it is unacceptable for the network to use all available time allowing no time for anything else in the end to end system. These include:

- Access to the medium: both wireless and broadband technologies often have built in latency to access the shared medium, measured in milliseconds. In your home, you may have two such access latencies: WiFi and broadband. These access latencies go up when the medium is loaded. I measure 230 ms today on my hspa+ cellular service, and 11ms to MIT from home (40km + 9ms for cable + around 1ms for 802.11 latency)
- Encoding (in particular) and decoding audio and video packets requires time
- Screen refresh is beginning to increase: but 60hz is commonplace, which is 16ms
- Input devices also typically insert latency between the user and the application, often in the 10 or more millisecond range.
- Device driver rings for audio and video; there is latency driving multimedia hardware.
- And most insidious today: head of line blocking in "stupid bloated FIFO queues" in our operating systems, and worst yet, all through the path from end to end.
- Mixing audio & video also requires time.

Latency must be regarded as a "budget", and you spend down the tolerable latency at each "hop". Sometimes you can hide latency behind other operations, but sometimes not. You can never get spent latency back.

Given the speed of light is something we find it difficult to change, and impossible to change even by a factor of two, this means that for most applications, adding *any* additional latency limits how satisfactory the user's experience will be in may circumstances.

You can view latency as a radius over which applications will be perceived as working fast and well.

Latency variance is also key: if occasionally your latency spikes, applications that cannot tolerate these spikes are forced to operate at "worst case" observed latency, that may be many times worse than "steady state".

You can see that even a simple a goal of enabling people to play music together reliably in a metropolitan area requires care.

Bloated Stupid FIFO Queues Must Die

Today's use of TCP is radically different from when (W)RED and similar algorithms were developed: TCP's initial window was but one or two packets (versus today's 4 packets, on its way to 10 packets). In the 1990's, most web browsers and web servers used but two TCP connections between them. Memory is cheap, processors are fast, and web servers never go to disk for nearly anything.

With the increase in the initial window, increase in the # of parallel TCP connections, and web site "sharding", we've reached a crisis. Many web sites induce web browsers to open tens of TCP connections simultaneously, and it is easy for there to be flights of packets that can easily reach more than a hundred packets in length, leaving data centers at high speed, cross the Internet, and arrive at the bottleneck link, usually your broadband, cellular or WiFi links. I have measured hundreds of milliseconds of transient latency on 50Mbps broadband connections when visiting image heavy web sites, implying packet trains of a megabyte or more! Given the Mlabs and netalyzr data, we can easily understand that a substantial fraction of the broadband edge has so much buffering that there has be little observed packet loss and little to push back on web site developers to avoid this pathology.

But these packet bursts fill FIFO buffers in either your broadband equipment or in your home router, blocking any other application from timely transmission to your screen or audio equipment. Most of the benefits of packet switching have, in fact, been lost; statistically multiplexing is no longer occurring.

A single 1500 byte packet represents around 13ms @ 1Mbps; it doesn't take many packets to have a profound effect on anything sharing bottleneck links. We must be able to isolate one flow from another, so that a new flow does not suffer intermittent catastrophic latency spikes. We must be smart about scheduling each packet: FIFO's cannot suffice.

"Traditional" AQM algorithms only attack part of the problem: that of TCP filling buffers. This is necessary, but far from sufficient as the above shows. It is necessary, as TCP cannot share efficiently (nor individual applications themselves have good latency) in the face of excessive buffering. Existing algorithms such as (W)RED for this are inadequate as that class of algorithm requires careful tuning, which, in practice, has caused these algorithms to often go unused, but which has become impossible in the face of the highly (and almost instantaneous) variable throughput found at the edge of today's Internet. Only a rapidly adaptive mark/drop algorithm requiring no tuning can suffice.

Combining flow queuing along with signaling TCP in a timely fashion with a better algorithm is therefore not only "nice to have", but essential. This is why we are so excited about the fq_codel algorithm, which combines flow queuing with CoDel, and scales easily to any bandwidth we see in the edge of the Internet even in software only implementations. Other algorithms and combinations are certainly possible: but both controlling TCP and flow queuing are necessary in today's Internet, and result in radically better latency under all tested loads, even significantly improving web browsing with no competing traffic.