acmqueue

Bufferbloat: Dark Buffers in the Internet

Networks without effective AQM may again be vulnerable to congestion collapse.

Jim Gettys, Bell Labs, Alcatel-Lucent; and Kathleen Nichols, Pollere Inc.

Today's networks are suffering from unnecessary latency and poor system performance. The culprit is bufferbloat, which is the existence of excessively large and frequently full buffers inside the network. Large buffers have been inserted all over the Internet without sufficient thought or testing. They damage or defeat the fundamental congestion-avoidance algorithms of the Internet's most common transport protocol. Long delays from bufferbloat are frequently attributed incorrectly to network congestion, and this misinterpretation of the problem leads to the wrong solutions being proposed.

Congestion is an old problem on the Internet, appearing in various forms with different symptoms and causing major problems. Buffers are essential to the proper functioning of packet networks, but overly large, unmanaged, and uncoordinated buffers create excessive delays that frustrate and baffle end users. Many of the issues that create delay are not new, but their collective impact has not been widely understood. Thus, buffering problems have been accumulating for more than a decade. We strive to present these problems with their impacts so that the community can understand and act upon the problem and, we hope, learn to prevent future problems.

This article does not claim to be the first to identify the problems of excessive buffering, but it is an attempt to create a wider understanding of the pervasive problem and to give a call to action.

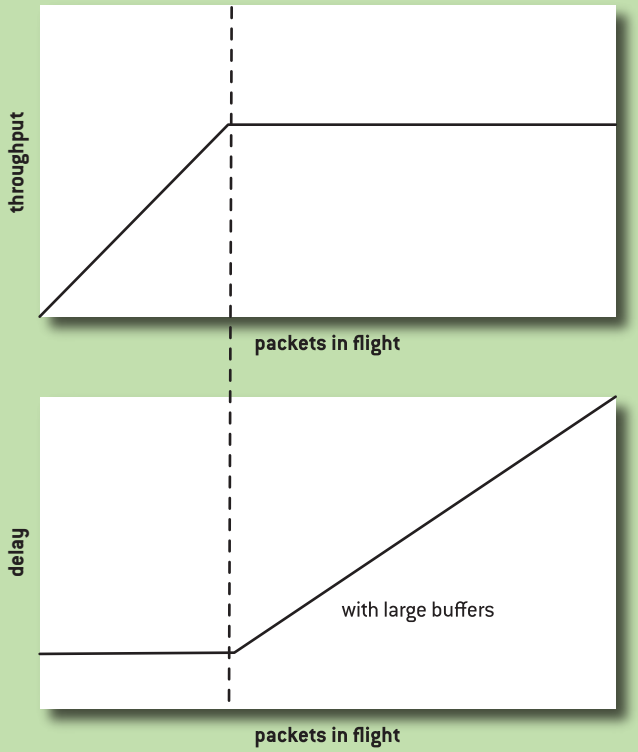
INTERNET BUFFERS AND CONGESTION

The latency a packet experiences in a network is made up of transmission delay (the time it takes to send it across communications links), processing delay (the time each network element spends handling the packet), and queuing delay (the time spent waiting to be processed or transmitted). Paths between communicating endpoints in the Internet are typically made up of many hops with links of different rates or bandwidths; the smallest bandwidth along the path is referred to as the bottleneck bandwidth. Packets cannot arrive at the destination any faster than the time it takes to send a packet at the bottleneck rate; without effective use of the network, the delay can be much worse.

Latency along the path—the time from the start of a packet's transmission by the sender until the packet is received at the destination—can be much longer than the time it takes to send the packet at the bottleneck rate. To maintain a steady flow of packets at the maximum rate, the "packets in flight" must be sufficient to fill the "pipe" of latency between sender and destination. Buffers are placed in front of a communications link in case packets arrive while the link is in use, thus requiring storage while the previous arrivals are serviced. The important location for buffers is at the path bottleneck, but the critical fast-to-slow transition can be different for different paths, different in the reverse path, and with dynamic bandwidths can change along the same path.

Figure 1 shows the relationship between throughput and delay for a packet network. Throughput

FIGURE 1 **Throughput and Delay**



is the fastest rate at which the count of packets transmitted to the destination by the network is equal to the number of packets sent into the network. As the number of packets in flight is increased, the throughput increases until packets are being sent and received at the bottleneck rate. After this, more packets in flight will not increase the received rate. If a network has large buffers along the path, they can fill with these extra packets and increase delay.

A network with no buffers has no place for packets to wait for transmission; thus, extra packets are dropped, creating an increasing loss rate and decreasing throughput, though the received packets would have the same constant delay. To operate without buffers, arrivals must be completely predictable and smooth; thus, global synchronized timing is critical to avoiding loss. Such networks are complex, expensive, and restrictive (i.e., they lack the flexibility of the Internet). A well-known example of a bufferless network is the original telephone network before packet switching took over. Adding buffers to networks and packetizing data into variable-size packets was part of the fundamental advance in communications that led to the Internet. The history of Internet congestion and its solution is the story of trying to find the optimal way to deploy and use buffers in a network. That story is still being written, but some of the lessons of the past are being ignored.

The fundamental transport protocol of the Internet is TCP/IP. TCP's persistence is testimony both to the robust and flexible design of the original algorithm and to the excellent efforts of the many

researchers and engineers who have tuned it over the decades. TCP made use of the idea of pipe size and the knowledge that there was reasonable but not excessive buffering along the data path to send a *window* of packets at a time—originally sending the entire window into the network and waiting for its acknowledgment before sending more data.

Even under moderate loads, the packets in flight of one or more connections could arrive at a bottleneck link in a burst and be dropped because of insufficient bandwidth. This led to heavy losses and the plummeting throughput associated with congestion collapse. Internet researchers and engineers had to advocate for sufficiently large buffers to avoid this poor network utilization. Congestion collapse hit a large part of the Internet in 1986. The network became clogged with retransmitted packets while *goodput* slowed to a trickle. As part of the solution, slow-start and congestion-avoidance algorithms were added to TCP and rapidly deployed throughout the Internet. They enabled the early Internet to recover and set the stage for the Internet's rapid growth in the 1990s with the adoption of World Wide Web applications.

These algorithms attempt to keep the network operating near the inflection point where throughput is maximized, delay is minimized, and little loss occurs. A sender-destination pair's TCP tries to determine the pipe size between them and to keep exactly that number of packets in flight throughout the data transfer. Since networks are shared and conditions change along the path, the algorithms continually probe the network and adapt the number of packets in flight. The slow-start algorithm (*slow* relative to the algorithm it replaced) attempts to make a first guess as to how fast TCP may operate by an initial exponential-growth phase in transmission rate. When the first packet loss is detected, TCP reduces its sending rate and enters the congestion-avoidance phase.

At the advent of congestion control in TCP, the recommendation for buffer sizing was to have a BDP's (bandwidth-delay product) worth of buffer, where bandwidth is the bottleneck link and delay is the RTT (round-trip time) between sender and destination. The rationale is that such a buffer can hold an entire "flight" of packets should they all arrive at the bottleneck link in a burst. To apply this rule, the bandwidth used in link buffer sizing was that of the immediately outgoing link, since the location of the actual bottleneck is unknown. Similarly, a canonical value was suggested for the RTT: 100 ms, a continental delay for the United States and Europe.

Once adequate buffers became routine, another problem could occur: the buffers were now part of the pipe that TCP is so good at filling. Filling these buffers would cause delay to increase, and persistently full buffers lack the space to absorb the routine *burstiness* of a packet network. John Nagle's seminal work in 1985[8] first drew attention to the consequences of large buffering. While working on TCP congestion-avoidance algorithms, Van Jacobson recognized the "persistently full buffer" issue in 1989, culminating in the development of RED (Random Early Detection) with Sally Floyd in 1993.[5]

A number of implementations, variations, imitations, and reports on RED's use are available in the literature.[12] These are generically termed AQM (active queue management), which attempts to keep the queues at the bottleneck from growing too large by monitoring the growth of the packet queue and signaling the sender's TCP to slow down by dropping (or marking) packets in a timely fashion. Different approaches have been taken to monitoring the packet queue and making the drop (or mark) decision. The IRTF (Internet Research Task Force) urged the deployment of active queue management in the Internet, publishing an RFC in 1998, popularly known as "the RED manifesto."[2]

Note that packet loss for TCP is not in itself a problem but is *essential* for TCP's functioning in the face of congestion. The excessive and consecutive packet losses that come from persistently full

buffers do present a problem, which is what the "warning" drops of AQM prevent (in addition to long delays).

The truth is, AQM is not widely configured and enabled in routers, and it is completely unavailable in many devices. Furthermore, the existence of cheap memory and the misguided desire to avoid packet loss has led to larger and larger buffers being deployed in the hosts, routers, and switches that make up the Internet. It turns out that this is a recipe for bufferbloat. Evidence of bufferbloat has been accumulating over the past decade, but its existence has not yet become a widespread cause for concern. The next section outlines Jim's personal journey of discovery.

(RE)DISCOVERING LATENCY

"The Internet is slow today, Daddy." This had become a frequent refrain in the Gettys household. When I would attempt to debug the problem, like a will-o'-the-wisp, it would usually vanish. On several occasions symptoms occurred long enough for me to waste significant amounts of time on my ISP's support line before they vanished. I attributed the recurring problem to the doubtful quality of the cable to my house or equipment damage from a lightning strike. Since my job is research in immersive teleconferencing, I knew I had to dig into the problem, if only for myself. The intermittent poor network performance was my first puzzle piece.

AN ENLIGHTENING LUNCH

Suspecting features of my cable provider to be part of the problem, I met with Comcast's Rich Woundy, who provided a number of new puzzle pieces to consider:

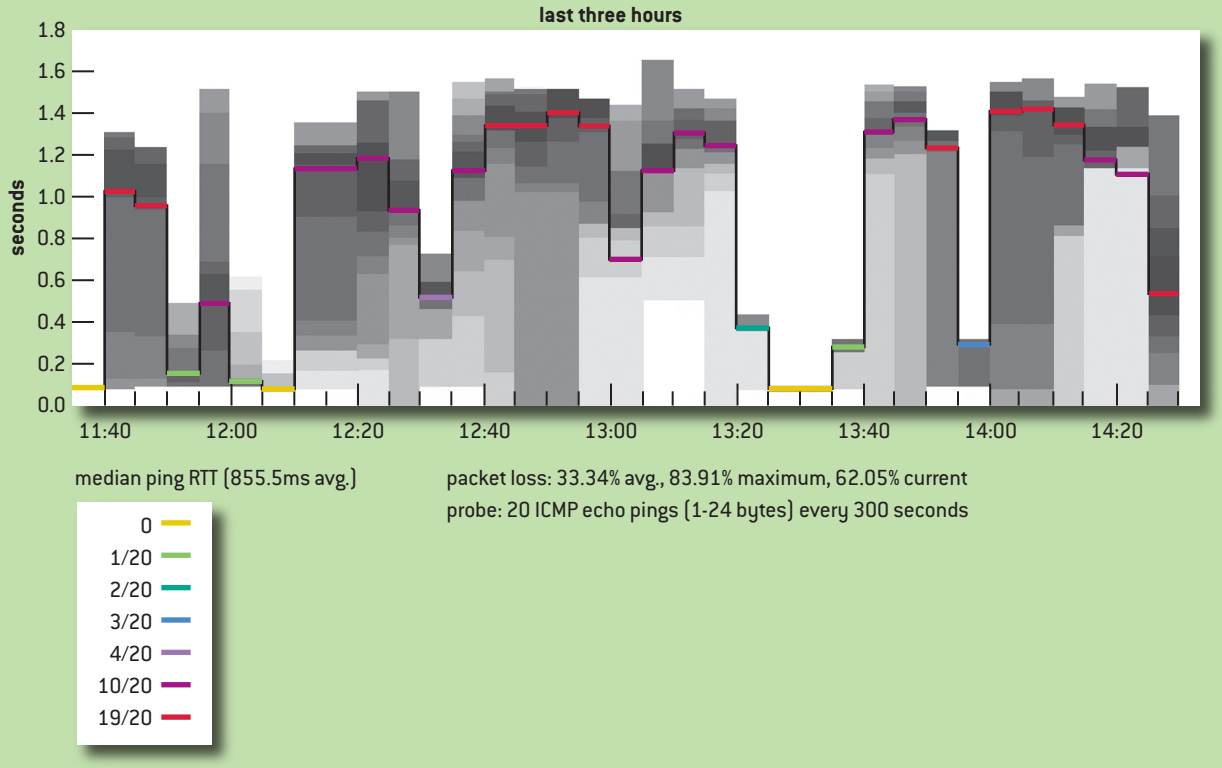
- The "big-buffers" problem, which David Clark [Internet network architect, currently senior research scientist at MIT] had warned about several years earlier.
- Broadband measurement studies have been indicating overly large edge buffers.
- AQM is MIA—many ISPs are running without any AQM even in circumstances where they really should.
- A group at UC Berkeley's ICSI (International Computer Science Institute) had developed a very fine tool for network diagnosis called Netalyzr (http://netalyzr.icsi.berkeley.edu).

The next day, I recorded my "smoking gun," a smokeping (http://oss.oetiker.ch/smokeping/) plot, while moving 20 GB of data from my house to MIT (see figure 2). The uncongested RTT of this path is less than 10 milliseconds, yet I was experiencing more than *1.2 seconds* of latency, significant packet loss, and painful Web browsing. I suspended the rsync a few times to read my e-mail, and, as can be seen on the plot, this would almost instantly "fix" my home network. This was the "Daddy, the Internet is slow today" phenomenon: the fact that the delays went away when I suspended my large data transfer explained why the problem disappeared when I went looking for it; in order to debug the network, I was stopping the work that was inducing the problem.

PACKET CAPTURES AT HOME AND "ABROAD"

I took a capture of a large file transfer over the same path. Scrolling through this capture with Wireshark (http://www.wireshark.org) showed peculiar behavior: obvious bursts of terrible behavior containing hundreds of duplicate ACKs, multiple retransmits, out of order packets, etc., for about 10-second periods followed by long periods of what looked like normal behavior. A plot of the data revealed 500 KB in flight over a 10-ms path. My uplink bandwidth was 2 Mbps, so the true BDP was

FIGURE 2 **Smokeping from Jim's Home to MIT**



2.5 KB. I could expect use of a 100-ms RTT for buffer sizing to result in 25 KB, but 500 KB was an order of magnitude larger. The one-second RTT is consistent with emptying a 500-KB buffer at 2 Mbps. To remove uncertainty, I repeated the experiment directly plugged into the cable modem and saw the same results.

I repeated my tests over my in-laws' fiber broadband service in New Jersey. Again, the results showed much more data in flight and much larger RTT times than expected: 250 KB outstanding on a 20-ms path and 200-ms latency with almost the same shape as on my cable. Over subsequent weeks, I added to my data sets by visiting local libraries and other targets of opportunity; the pattern was the same wherever I went. Finally, I had collected enough disturbing data to be consistent with the big-buffers problem, and I suspected that the problem was endemic among all technologies and providers.

CALLING THE EXPERTS

I posted the packet traces to a group of TCP experts: Dave Clark, Dave Reed, Scott Bradner, Greg Chesson, Van Jacobson, and Vint Cerf. Their feedback revealed that extreme buffering created an artificially large pipe size and that packet discards occur according to *tail drop*—that is, when a packet arrives to a full buffer, it is dropped. The packet's destination is unaware of the dropped packet until

FIGURE 3A

Packet RTT and Window Size: Over Five-Minute Trace

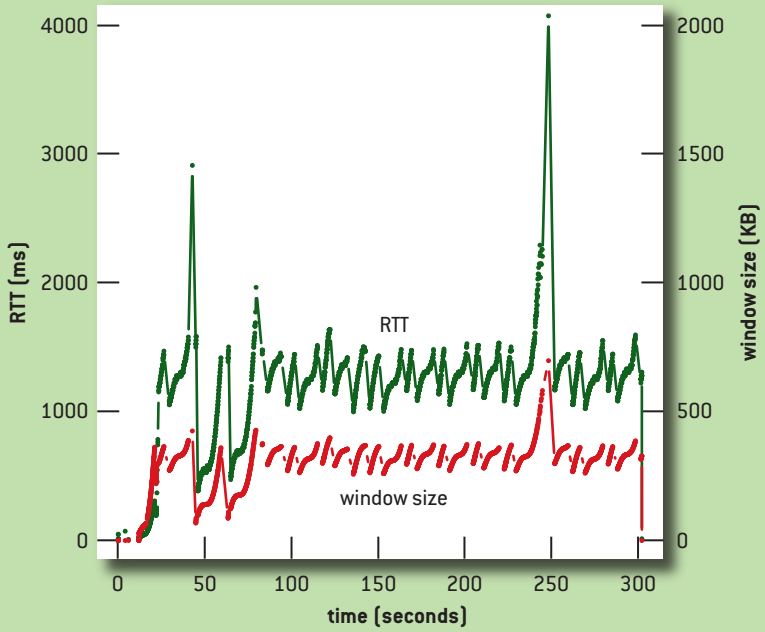
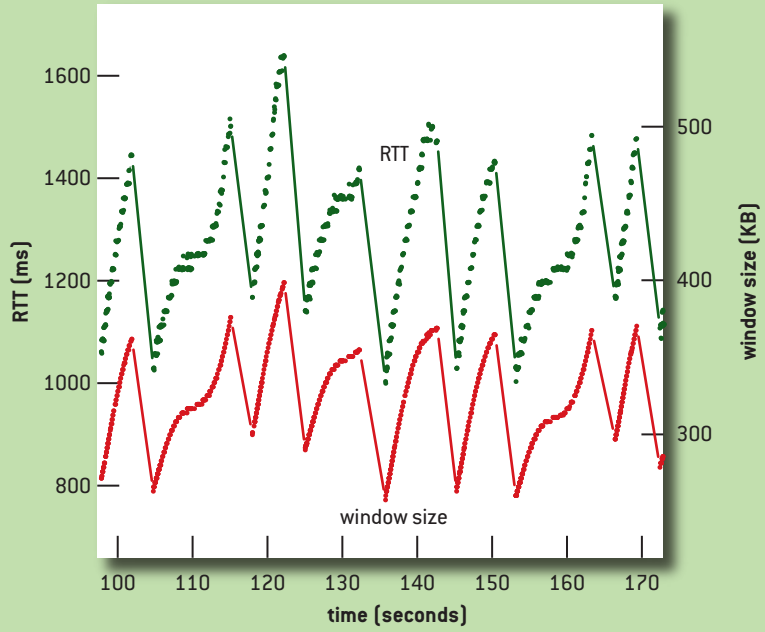


FIGURE 3B

Packet RTT and Window Size: Detail Over 70 Seconds



the entire bloated buffer has been transmitted, which can take many times the uncongested RTT. TCP expects timely notification of packet loss for correct operation. With such large buffers, TCP's slow-start algorithm doesn't see any drops and thus greatly overestimates the correct pipe size and requires multiple packet drops before TCP can enter its congestion-avoidance phase.

Jacobson provided plots of the data, reproduced in figures 3 and 4. The shape of the window-size evolution is characteristic of the CUBIC implementation of TCP,[11] which is the default in Linux. CUBIC's initial window growth is like Jacobson's original algorithm, but it "flattens out" at an apparently stable pipe-size estimate for a while. [13] If it doesn't detect a congestion event (i.e., a packet drop), then it ramps the window up quickly. In the trace in figure 3, about five seconds pass without a drop, whereupon TCP ramps up the window in an attempt to find a new operating point; after 10 seconds the buffer is full, and packet loss occurs. (Because of the offload engine, the RTTs are from the last packet of the jumbogram it receives.) Notice the RTT ramps up quickly to about 1.2 seconds and mostly stays there. The three-second RTT spikes show where massive dropping took place when the buffer became full. These are followed by a drop in window size and RTT. This, in turn, causes TCP to shut down the window size. The window-size curve shows that sometimes the algorithm gets a drop before it goes into CUBIC's second probing stage.

Figure 4 shows the goodput (determined from the ACKs) versus time. The initial brief period of nearly 10 Mbps is a result of Comcast's PowerBoost feature and is followed by a steady 2 Mbps, showing that I was getting all the bandwidth I expected. Figure 4b shows the RTT seen at each window size. The lower data set clearly results from the PowerBoost phase and the upper data set is the subsequent 2-Mbps phase. These points show exactly the situation shown abstractly in figure 1:

FIGURE 4A

Uplink Bandwidth During Trace

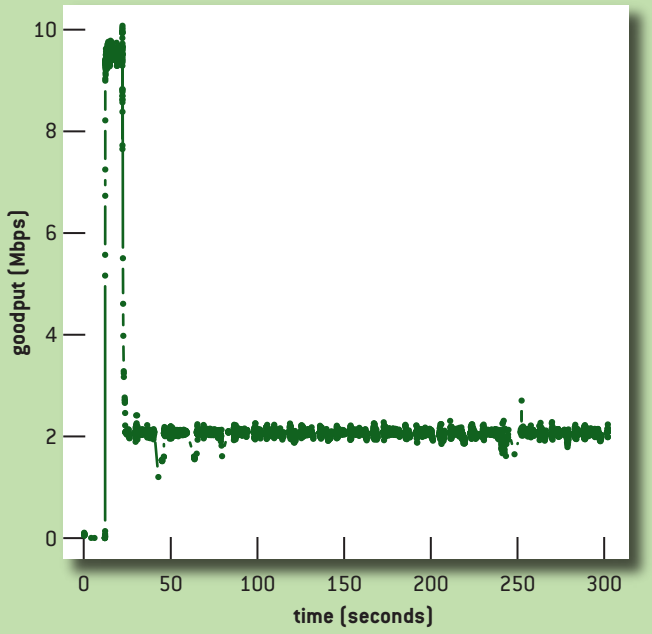
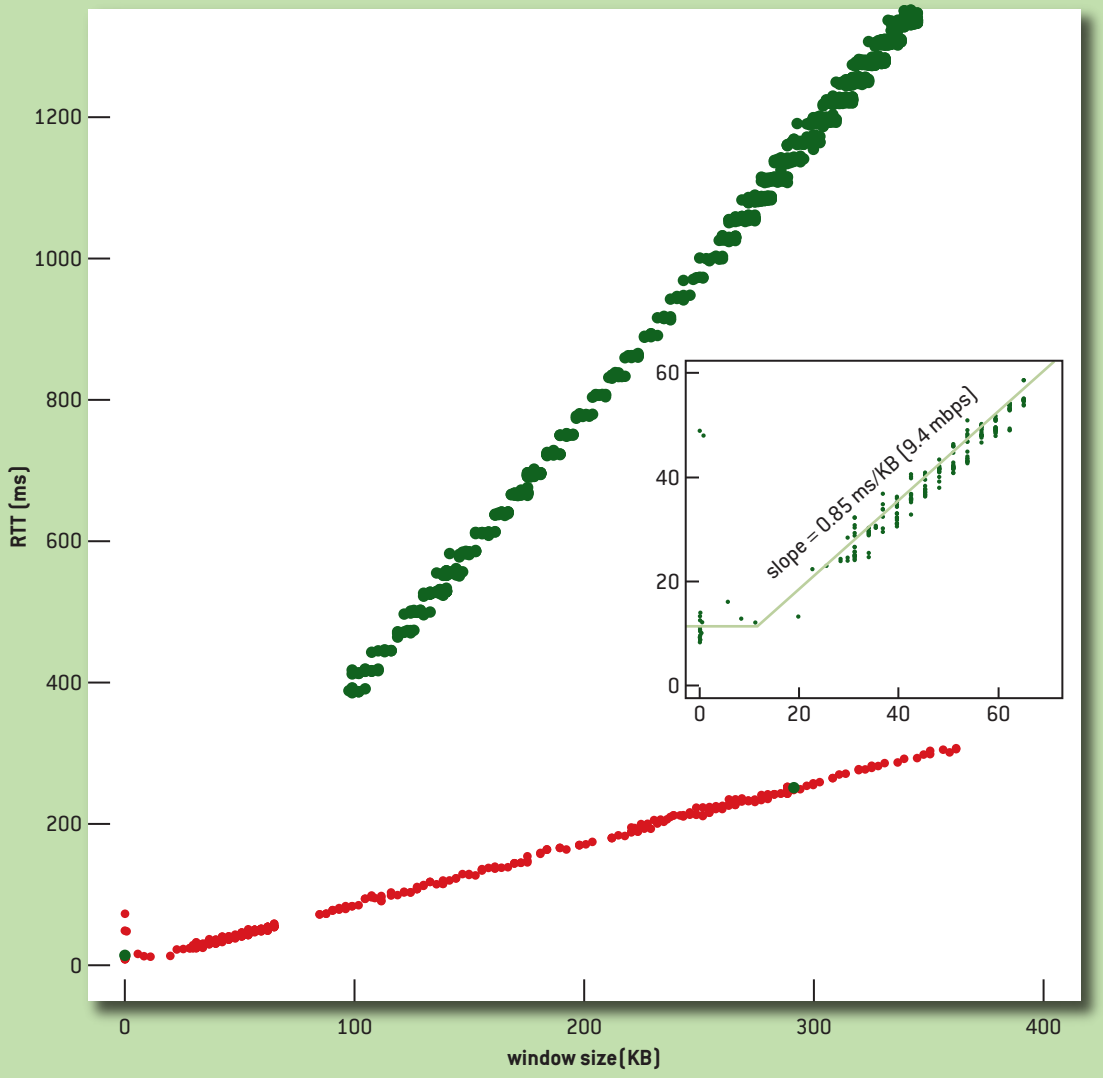


FIGURE 4B

Scatterplot of RTT Experience vs. Window Size During Trace



the delay is about 10 ms initially; then the window size increases, the buffer fills up, and the delay (as measured by the RTT) increases linearly.

TCP should approximately share a bottleneck link between competing flows. The impact of bufferbloat on TCP's behavior is subtle and profound in two ways:

- For TCP congestion avoidance to be useful to people using that link, a TCP connection causing congestion must react quickly to changes in demand at the bottleneck link, but TCP's reaction time is quadratic to the amount of overbuffering. A link that is 10 times overbuffered not only imposes 10 times the latency, but also takes 100 times as long to react to the congestion. Your short, interactive TCP connection loses completely to any long-lived flow that has saturated your link.
- The long-lived flow's inability to respond to congestion can cause complete starvation on

competing transfers (yours or anyone who shares the link). A local service may be overbuffered by another factor of 10 times compared to a remote service.

CLOSING THE CASE ON BROADBAND BUFFERBLOAT

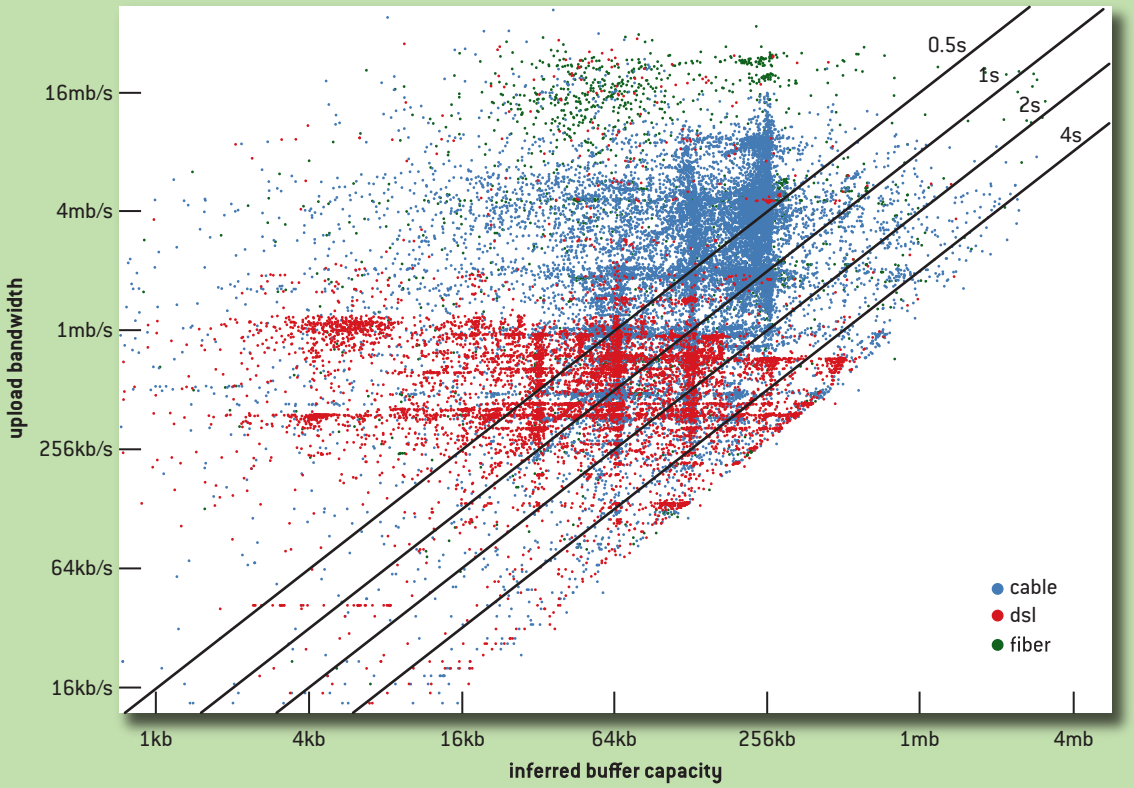
Evidence of excessive buffering accumulated over the course of the past decade is finally sufficient to motivate systematic study.

A 2007 study of nearly 2,000 hosts connected through cable and DSL companies in Europe and North America focused on measuring the residential "last mile." The results showed that upstream queues for DSL were frequently in excess of 600 ms, and those for cable generally exceeded one second.[4]

Netalyzr), a measurement tool for the last mile or access link, has been key to the exposure of bufferbloat. A 2010 study of 130,000 measurement sessions revealed widespread, severe overbuffering in the broadband edge.[7] (The results are used here with permission of the authors.) Figure 5 is a scatterplot of bandwidth plotted against inferred buffer capacity, each point representing a single Netalyzr test session. The solid diagonal lines indicate the latency, in seconds, exposed by Netalyzr's buffer test. The tests show excessive latencies in both downlinks and uplinks in all broadband technologies. Since Netalyzr tops out at 20 Mbps and bounds the test length at five seconds, the situation is clearly worse than shown.

FIGURE 5

Plot Reproduced from ICSI's Netalyzr Studies



Focusing only on cable customers, the same study showed that the equipment had two dominant buffer sizes: 128 KB and 256 KB (for reference, a 3-Mbps uplink would take 340 ms to empty a 128-KB buffer; and a 1-Mbps uplink would take about 1 second). The Netalyzr authors note the difficulty of sizing buffers for the wide range of operational access rates, both from different service levels and from dynamically varying rates. Case closed.

WHERE THERE'S SMOKE, THERE'S USUALLY FIRE

Observation of eight-second latency at my home router sparked installation of OpenWrt (www.openwrt.org) for further investigation. I set the router-transmit queue to zero but saw no effect on latency. The Wi-Fi link from my laptop was of poor quality (resulting in a bandwidth of around 1 Mbps), so the bottleneck link was my Wi-Fi putting the queues on my laptop rather than in the router—and since my test was an upload, the bottleneck was in my laptop! I finally realized that AQM is not just for routers; outbound bottlenecks could easily be at the host's queue, and Wi-Fi is now frequently the bottleneck link.

Manipulating the Linux transmit queue on my laptop reduced latency about 80 percent; clearly, additional buffering was occurring somewhere. "Smart" network interface chips today usually support large (on the order of 256 packets) ring buffers that have been adjusted to maximize throughput over high-bandwidth paths on all operating systems. At the lowest Wi-Fi rate of 1 Mbps, this can add three seconds of delay. Device-driver ring buffers need careful management, as do all other buffers in operating systems. A single packet of 1,500 bytes is 12 ms of latency at 1 Mbps; you can see that the amount of buffering must adjust dynamically very quickly over two orders of magnitude so as not to sacrifice bandwidth *or* latency.

Compounding this problem, modern operating systems adjust socket-buffer sizes in response to observed delay; so operating-system and driver bufferbloat can cause a cascade of excessive buffering higher in the network stack, resulting in still higher latencies in applications.

Bufferbloat is not just in broadband. In 2009, Dave Reed [Internet network architect, now with SAP Labs] reported problems in 3G networks: he saw high RTTs without packet loss and correctly diagnosed the cause.[10] Very high latencies were observed to the point where packets may be delivered but so late that they are seldom useful; people time out before packets.[9]

Broadband and wireless bufferbloat are also the root causes of most of the poor Internet performance seen at many hotels and conferences.

Though the edge is more easily measured, there are some reports of congestion in the core. The RED manifesto has usually been ignored, so there are "dark" buffers hidden all over the Internet.

THE ROAD TO HELL IS PAVED WITH GOOD INTENTIONS

In the past decade, not only was AQM not deployed, but new factors, unknown at the time of the RED manifesto, also exacerbated the problems of full buffers. The early Internet had slow links and a very small number of simultaneous data transfers sharing these links. Wireless did not exist. The first residential Internet systems connected personal computers through low-bandwidth links into an Internet of relatively high-speed links. The Internet has evolved to a very bandwidth-rich core. Today residential and small business Internet connections increasingly connect customers' high-bandwidth stub networks through smaller bandwidth links into this core. Bottlenecks at the Internet's edge can easily move between the wireless access (when its bandwidth is low) and the provider's uplink, both of which can have highly variable bandwidths.

Memory also became cheap; you cannot buy RAM chips small enough for the buffering in edge devices, and these devices have no mechanisms for self-limitation. Commodity network devices now span many downward-compatible generations: Ethernet has gone from 10 Mbps to 10 Gbps; wireless operates from 1 Mbps to 100 or more Mbps; and cable from 10 Mbps to, soon, several hundred Mbps. The result is a single buffer statically sized for larger bandwidths but much too large for lower-bandwidth links. For example, the 256 packets of buffering found in many of today's 802.11 device drivers alone translates to more than three seconds at 1 Mbps, which is all the bandwidth you may have on some wireless networks. Complicating this is that recommendations about the amount of buffering have been influenced by early Internet problems of insufficient buffers, thus erring on the side of larger buffers, perhaps unaware that AQM is rarely used or unavailable.

Wireless links and networks are increasingly part of the edge access and are even more variable than broadband bandwidth: moving a device a few centimeters can change rates by one to two orders of magnitude; and because wireless is a shared medium, this also affects rates. Since bandwidth can vary by a factor of 100 at short time scales, static buffering is never appropriate.

A number of approaches to speeding up Web access contribute to transient access-link bloat by dumping large numbers of packets onto these links simultaneously.

REVISITING THE BANDWIDTH-DELAY PRODUCT

The efficacy of BDP-sized buffers is in question. As pointed out in a presentation at ACM SIGCOMM in 2004, BDP is not appropriate for highly multiplexed core links.[1] The rationale for maintaining a BDP buffer still applies at the network edge where a single flow can congest a link. The problem is in determining that BDP. Bandwidth variations of two or more orders of magnitude clearly play havoc with the bandwidth. At the same time, the 100-ms delay assumption has been weakened by the advent of CDNs (content delivery networks) and other services engineered to bring common RTTs down to 10-30 ms. Thus, even if an access link is a constant bandwidth and its buffer is sized to 100 ms, it may still be 3-10 times too large.

For more than a decade, TCP tuning has been focused on improvements needed for high-BDP environments where large packet windows are required to achieve good throughput. These new algorithms are not in themselves nefarious, focusing on efficiently filling the pipe, but the researchers have unconsciously worked with a model of high bandwidth and AQM-enabled buffers. When the large pipe size comes from buffers rather than bandwidth, the algorithms efficiently fill those buffers, resulting in large delays. Controlling buffers makes it possible for one TCP to work well everywhere, a solution that is preferable to attempting to create a version of TCP specifically for access links.

Clearly there cannot be a "correct" static amount of buffering in the global Internet: a self-adaptive AQM is the only viable long-term solution in *any* device (including your computer or smart phone) with a network buffer.

AQM FOR THE MODERN WORLD

In early 1998, the second author of this paper discovered flaws in RED and started to work with Jacobson to make improvements. At that time, the main concern was finding an algorithm that could be configured for any link by setting a single-rate parameter, as well as developing a viable approach to tracking persistent queue while ignoring short-term bursts[14] Subsequent research tried

to fix some of the flaws but failed to create an AQM that could keep persistent buffers short without overdropping. Network operators faced only with algorithms requiring expert manual configuration that might hurt them have understandably been unwilling to enable and configure AQM.

In the ensuing decade wireless has been widely deployed, bringing wildly varying bandwidth to many edge links; cable Internet access has become common; and a device's access bandwidth can easily vary by two orders of magnitude. It is now obvious that any AQM algorithm that does not take as an input the rate at which data leaves the buffers cannot work in today's highly variable bandwidth environment. Clearly without such an algorithm, bufferbloat will be hard to defeat.

Surprising to most, AQM is essential for broadband service, home routers, and even operating systems: it isn't just for big Internet routers.

WHEN DOES OVERBUFFERING HURT?

Overbuffering hurts anytime you saturate a link; for example:

- Copying a file over the Internet.
- Running old versions of BitTorrent or other file-sharing applications.
- Sending/receiving e-mails to Grandma with pictures attached.
- Uploading video to YouTube.
- Web browsing, which can hurt you or others momentarily.

The saturated link can be anywhere, in either or both directions in the path: easiest and most common to see are the operating system, wireless link, and broadband service.

WHY IS OVERBUFFERING A PROBLEM?

Oversized buffers fill and cause delay, destroying many uses of the Internet:

- Stock traders and gamers do not want their competition to have even a 1-ms advantage over them.
- To play music, jitter (variation in delay) and latency must be controlled and kept below 100 ms.
- For something to "feel" attached to your hand (perfect rubber banding), latencies need to be below the 30-ms range; for keyboard echoing to be imperceptible, 50 ms.
- Speed-of-light latency dominates VoIP (voice over IP) over long-haul networks, making access latencies critical for keeping end-to-end latency below 150 ms (the longtime telephony metric).
- Excessive packet loss induced by bufferbloat may cause DNS (Domain Name System) lookup failure.
- Essential network protocols such as ARP, DHCP, RA, and ND all presume timely response and can fail without it.
- Web browsing becomes painful as delays go from hundreds of milliseconds to multiple seconds.

Many service providers would like to be able to provide low-latency services in their networks to customers, whether remote gaming, hosted desktop systems, or backup. Solving the bufferbloat problem is necessary for their successful deployment.

THE TIP OF THE ICEBERG

Operating systems and hardware have an amazing number of buffer hiding places. As software and hardware are updated, more sources of bloat can be uncovered. In particular, as older TCPs are replaced with modern ones, users not currently bloating their access buffers may suddenly experience much longer delays (e.g., Windows XP does not enable TCP window scaling, so it never has more than 64 KB in flight at once).

Current commonly used network performance tests fail to test latency simultaneously with bandwidth: a link must become saturated for queuing delays to become obvious. It can take 10 seconds to fill the buffers of a broadband device, home router or operating system, and most consumer broadband tests do not test for that long—thus missing bufferbloat.

Excessive access delays tend to be written off as network congestion. Employing larger backbone pipes and rationing bandwidth use cannot improve performance for the users congesting access uplinks or viewing downloads through bloated buffers at the provider edge.

MITIGATIONS

There are glimmers of hope. DOCSIS (Data over Cable Service Interface Specification) was modified in spring 2011, allowing cable operators to reduce buffering in cable modems. This mitigation will not take effect until 2012 at best and will require cable-modem firmware upgrades or (most likely) modem replacement, as well as motivated and knowledgeable operators.

Proper solutions for Web browsers can improve access-link behavior. These include HTTP/1.1 pipelining and Google's SPDY (http://dev.chromium.org/spdy), both of which can achieve better performance, reduce the total number of packets and bytes transferred, and enable TCP to function better.

Some mitigations are simple and direct for the knowledgeable. A home router or your laptop, for example, almost never operates in the high-bandwidth environment for which the operating system has likely been tuned. Adjusting buffering in the operating system and/or device drivers can make a major improvement over the defaults. Unfortunately, while these adjustments may be accessible in your laptop, they may not be accessible in your home router or handheld devices.

Bandwidth shaping can be used to prevent bottleneck buffers from filling, but at a cost in bandwidth. Contrast the smokeping result in figure 6 with that in figure 2; almost two orders of magnitude improvement isn't bad.

Meanwhile, Nichols and Jacobson have resumed work on a robust, adaptive AQM.

BE PART OF THE SOLUTION

The situation may worsen before it improves, and immediate action is necessary. Potential solutions must be subjected to rigorous testing and analysis before being widely deployed; otherwise, existing problems can be made worse. Unfortunately, today there is a distinct lack of funding for the kinds of performance monitoring, tuning, and improvement that characterized the early Internet.

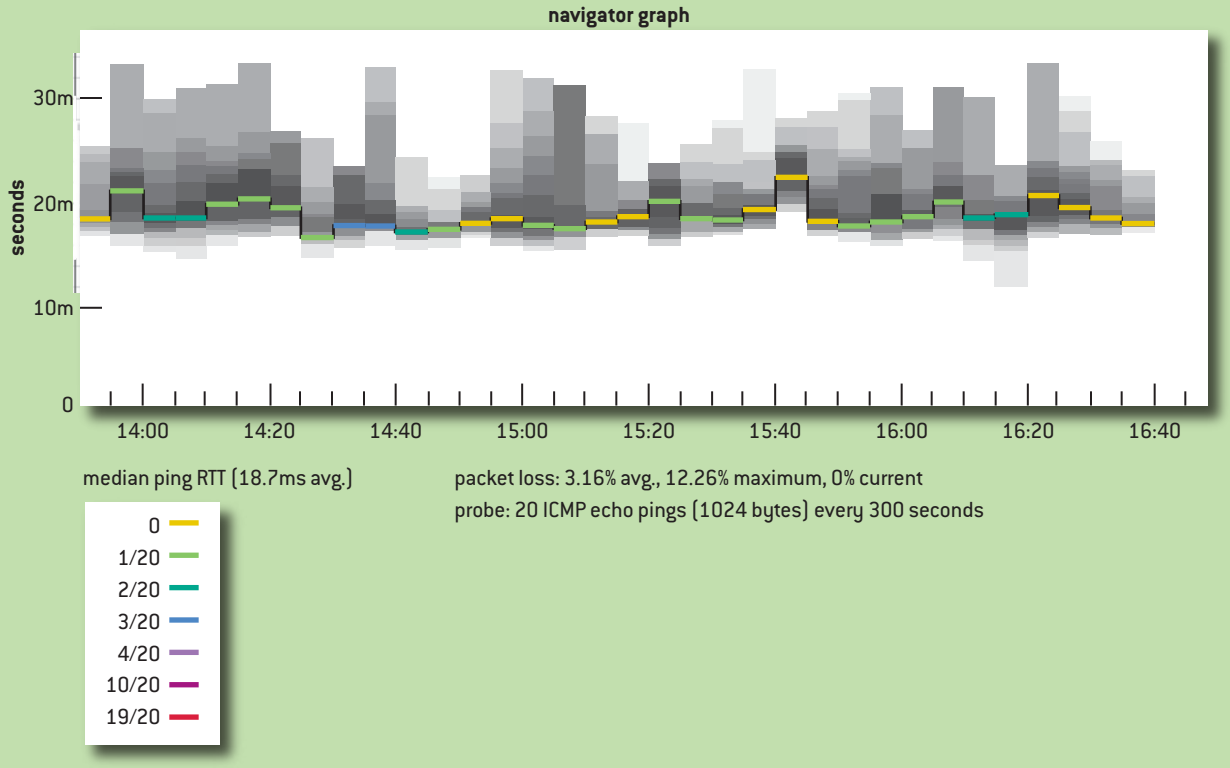
The first step is to make the problem apparent. Consumer tests are important [e.g., Speedtest.net, SamKnows, M-Labs (http://www.measurementlab.net/), Netalyzr], but better tests that point to the correct offender, usable by everyone, are badly needed. Consumer tests often perpetuate the mythology that more bandwidth means higher "speed," and better marketing metrics are essential. Stuart Cheshire's famous "It's the Latency, Stupid" rant should be taken to heart.[3]

An open source project, CeroWrt, is under way at bufferbloat.net using OpenWrt to explore potential solutions, including AQM. Please come help. A wide range of testing is needed for confidence in any algorithm. Since our operating systems are commodities and are used in today's home routers, home-router bufferbloat is a direct result of host bufferbloat. Solve one, and you solve the other.

Unfortunately, since bufferbloat misleads TCP's congestion-avoidance algorithm with respect

FIGURE 6

Smokeping of File Transfer from Jim's House to MIT after Mitigations



to the effective pipe size, modern networks without effective AQM may again be vulnerable to congestion collapse from saturated edge buffers creating packet delays measured in seconds. Congestion collapse has been reported in a large-scale network, requiring complete shutdown and careful restart of the entire network to regain (temporary?) stability.

We are flying on an Internet airplane in which we are constantly swapping the wings, the engines, and the fuselage, with most of the cockpit instruments removed but only a few new instruments reinstalled. It crashed before; will it crash again?

ACKNOWLEDGMENTS

We would like to thank Dave Clark, Dave Reed, Vint Cerf, Van Jacobson, Vern Paxson, Nick Weaver, Scott Bradner, Rich Woundy, Greg Chesson, Dave Täht, and a cast of hundreds.

REFERENCES

1. Appenzeller, G., Keslassy, I., McKeown, N. 2004. Sizing router buffers. ACM SIGCOMM, Portland, OR, (August).
2. Braden, R., et al., 1998. Recommendations on queue management and congestion avoidance in the Internet, RFC2309 (April).

3. Cheshire, S. 1996. It's the latency, stupid; http://rescomp.stanford.edu/~cheshire/rants/Latency.html.
4. Dischinger, M., et al. 2007. Characterizing residential broadband networks. Internet Measurement Conference (IMC), San Diego, CA (October 24-27).
5. Floyd, S., Jacobson, V. 1993. Random Early Detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking (August).
6. Jacobson, V. 1998. Notes on using RED for queue management and congestion avoidance. Talk at NANOG (North American Network Operators' Group) 13; ftp://ftp.ee.lbl.gov/talks/vj-nanog-red.pdf.
7. Kreibich, C., et al. 2010. Netalyzr: illuminating the edge network. Internet Measurement Conference (IMC), Melbourne, Australia (November 1-3).
8. Nagle, J. 1985. On packet switches with infinite storage. Network Working Group RFC 970 (December); http://www.ietf.org/rfc/rfc970.txt.
9. Reed, D. P. 2009. Congestion collapse definition; thread at http://mailman.postel.org/pipermail/end2end-interest/2009-September/007769.html.
10. Reed, D.P. 2009. What's wrong with this picture; thread at http://mailman.postel.org/pipermail/end2end-interest/2009-September/007742.html.
11. Rhee, I., Xu, L. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS* 42 (5).
12. Villamizar, C., Song, C. 1994. High-performance TCP in ANSNET. *Computer Communications Review* 24(5): 45-60.
13. V. Jacobson and M. Karels, Congestion Avoidance and Control, Proceedings of SIGCOMM '88, August 1988
14. V. Jacobson, "Notes on Using RED for Queue Management and Congestion Avoidance", talk at NANOG 13, ftp://ftp.ee.lbl.gov/talks/vj-nanog-red.pdf, see also http://www.nanog.org/mtg-9806/agen0698.html

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

JIM GETTYS is at Alcatel-Lucent Bell Labs, USA, where he is working on bufferbloat, as a properly working low-latency Internet is required for immersive teleconferencing. He was vice president of software at the One Laptop per Child project, and is one of the original developers of the X Window System at MIT. He worked at the World Wide Web Consortium and was the editor of the HTTP/1.1 specification in the Internet Engineering Task Force.

KATHLEEN NICHOLS is the founder and CTO of Pollere Inc., a consulting company working in both government and commercial networking. She has 30 years of experience in networking, including a number of Silicon Valley companies and as a cofounder of Packet Design. Her current interests include seeking the holy grail of active queue management and working on differentiated services for challenging environments.