DOI:10.1145/2209249.2209264

Article development led by acmqueue
queue.acm.org

A modern AQM is just one piece of the solution to bufferbloat.

BY KATHLEEN NICHOLS AND VAN JACOBSON

Controlling Queue Delay

NEARLY THREE DECADES after it was first diagnosed, the "persistently full buffer problem," recently exposed as part of *bufferbloat*,[6,7] is still with us and made increasingly critical by two trends. First, cheap memory and a "more is better" mentality have led to the inflation and proliferation of buffers. Second, dynamically varying path characteristics are much more common today and are the norm at the consumer Internet edge. Reasonably sized buffers become extremely oversized when link rates and path delays fall below nominal values.

The solution for persistently full buffers, active queue management (AQM), has been known for two decades but has not been widely deployed because of implementation difficulties and general misunderstanding about Internet packet loss and queue dynamics. Unmanaged buffers are more critical today since buffer sizes are larger, delay-sensitive applications are more prevalent, and large (streaming) downloads common. The continued existence of extreme delays at the Internet's edge can impact its usefulness and hamper the growth of new applications.

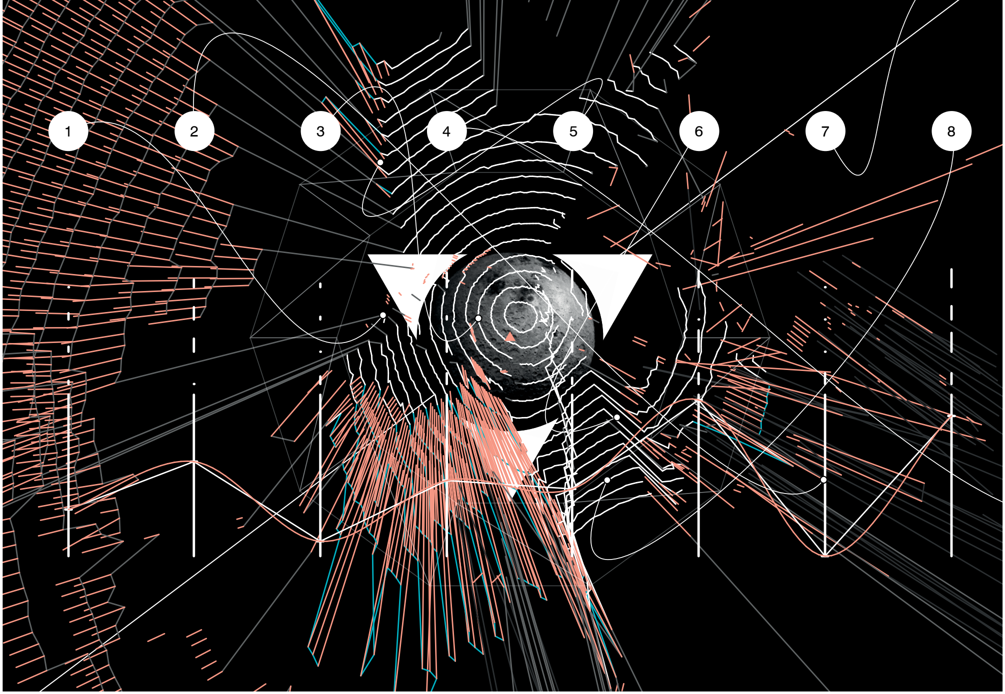
This article aims to provide part of the bufferbloat solution, proposing an innovative approach to AQM suitable for today's Internet called CoDel (for Controlled Delay, pronounced like "coddle"). This is a "no-knobs" AQM that adapts to changing link rates and is suitable for deployment and experimentation in Linux-based routers (as well as silicon).

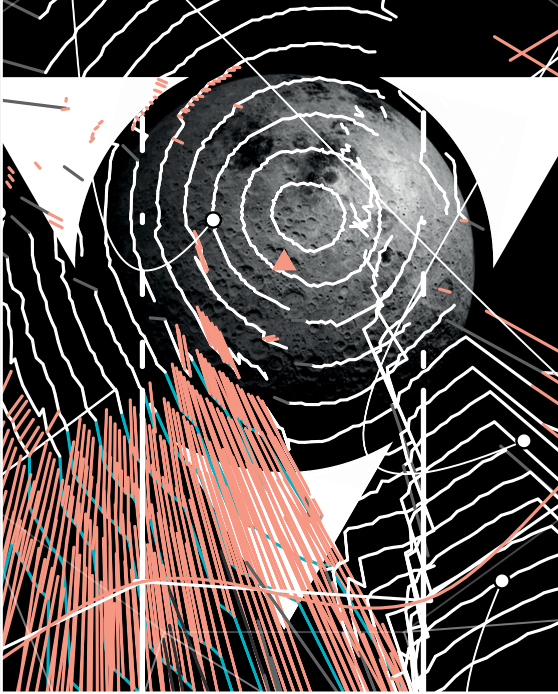
Packet networks require buffers to absorb short-term arrival rate fluctuations. Although essential to the operation of packet networks, buffers tend to fill up and remain full at congested links, contributing to excessive traffic delay and losing the ability to perform their intended function of absorbing bursts. The "full buffer problem" was recognized in the early days of the Internet, and mitigations were explored then.[9,15,17]

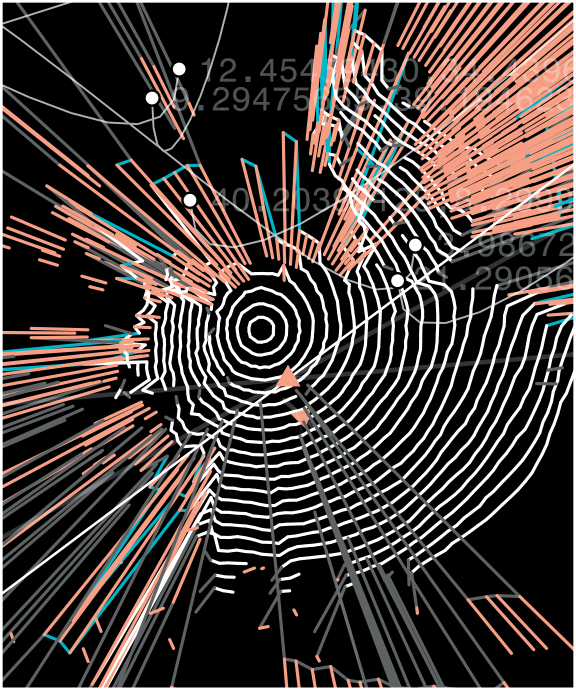
In 1998, the Internet Research Task Force urged the deployment of active queue management in the Internet,[1] specifically recommending Random Early Detection (RED).[5] Although RED was simple and can be effective at reducing persistent queues, little guidance was available to set its configuration parameters and it functioned poorly in a number of cases. This led to a general reluctance to use it. As RED's problems became apparent, research documented those issues and proposed new AQMs, adding more configuration and complexity to the original RED. Though Feng et al.[22] pointed out in 2002 that queue length is not a good predictor of congestion, it continued to be used. Although research continued, deployment did not.

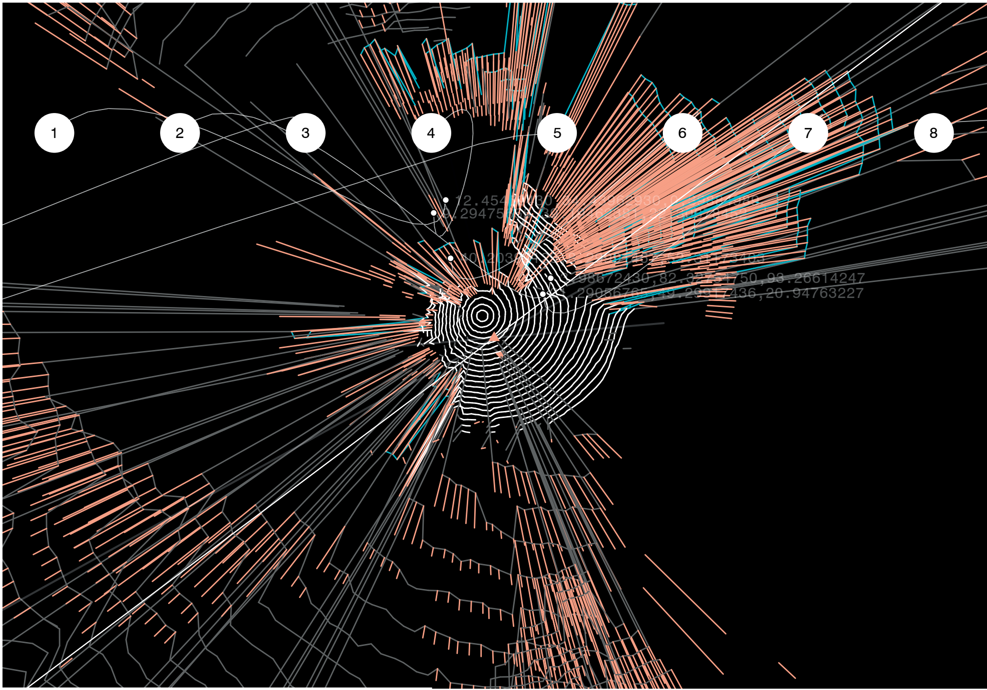
The Internet has been saved from disaster by a constant increase in link rates and by usage patterns. Over the past decade, evidence has accumulated that this whistling in the dark cannot continue without severely impacting Internet usage: independent measurement studies[4,13] have found edge queuing delays from hundreds of milliseconds up to multiple seconds.

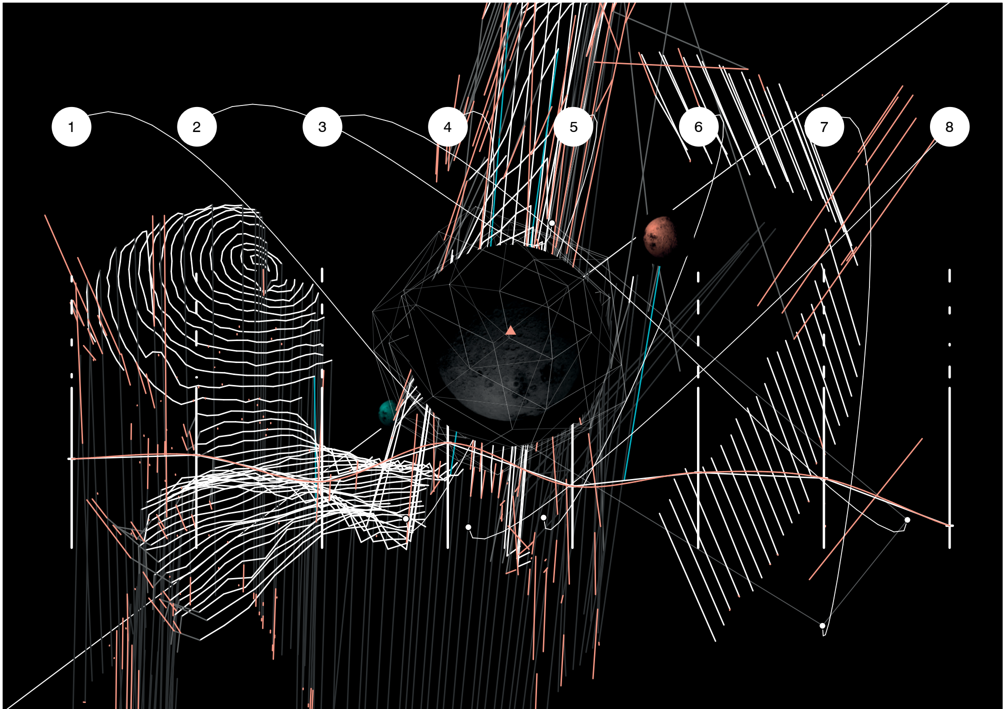
ILLUSTRATION BY ANDREAS KOLLER


1
2
3
4
5
6
7
8




12.454
9.29475


1
2
3
4
5
6
7
8
,93.26614247
,20.94763227


1
2
3
4
5
6
7
8

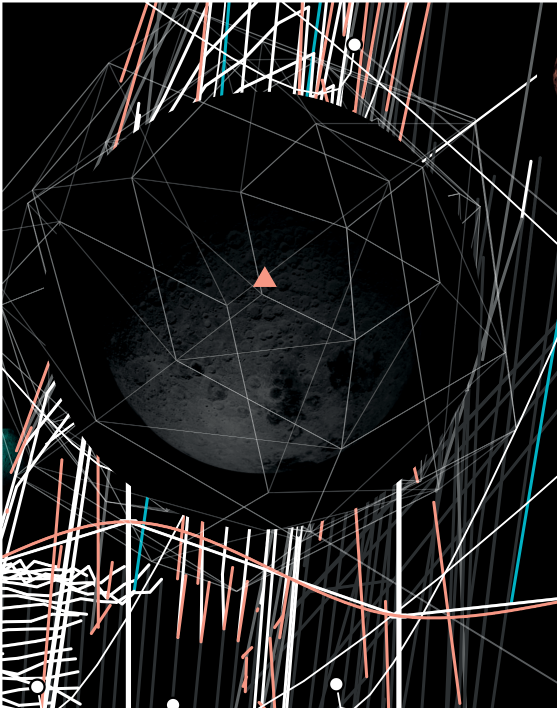


Figure 1.TCP connection startup.

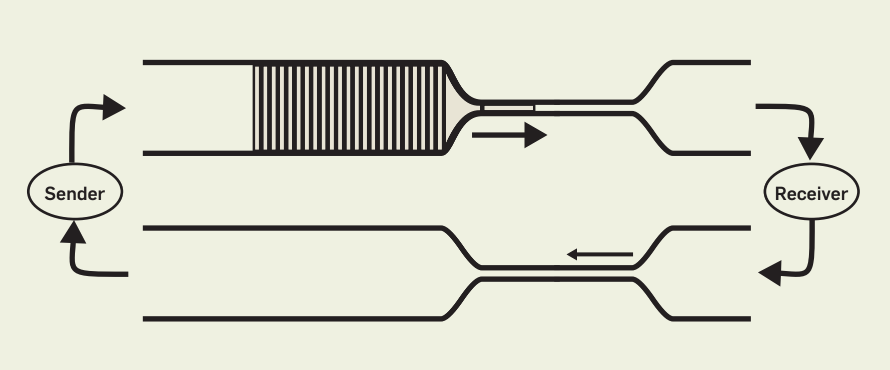
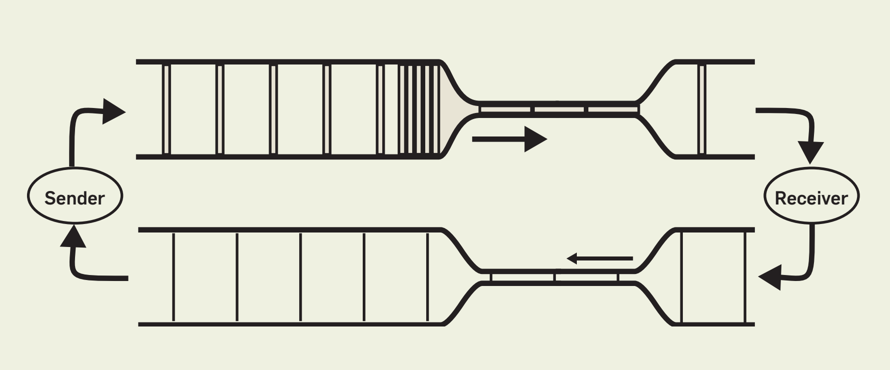


Figure 2. TCP connection after one RTT.



These measurement studies and the bufferbloat project[2] document the harmful effects of large, unmanaged buffers at the network edge.

Correct buffer sizing is not an easy problem. Undersizing—making buffers smaller than the traditional BDP (bandwidth-delay product)—is fraught with problems,[8] nicely illustrated by Guillaume Vu-Brugier et al.[20] Today's links vary in bandwidth, and individual connections vary in round-trip times (the "delay" used in BDP). This makes it impossible to properly pick a static size for most edge links.[13,14] A simple, robust algorithm that can manage buffer delay regardless of buffer size and link bandwidth without a negative impact on utilization can make oversized buffers irrelevant.

Understanding Queues

Developing effective active queue management has been hampered by misconceptions about the cause and meaning of queues. Network buffers exist to absorb the packet bursts that occur naturally in statistically multiplexed networks. Queues occur in the buffers as a result of short-term mismatches in traffic arrival and departure rates that arise from upstream resource contention, transport conversation startup transients, and/or changes in the number of conversations sharing a link. Unfortunately, other network behavior can cause buffers to fill, with effects that are not nearly as benign. With the wrong conceptual model for queues, AQMs have limited operational range, require a lot of configuration tweaking, and frequently impair rather than improve performance.

Figure 1 shows a TCP connection shortly after startup (see "Congestion Avoidance and Control"[8] for more discussion). The sender launches its window of 25 packets back-to-back, and they flow through the network until they hit a bottleneck (bandwidth reduction). There, as each packet is squeezed down in bandwidth, it must stretch out in time since its size stays constant. The vertical direction is bandwidth (bits/sec), and the horizontal direction is time (sec), so the area of each rectangle is the packet size (bits/sec × sec = bits). For example, if the packets were 1ms long and the bandwidth reduction was 100Mbps to 10Mbps, then the second packet would arrive 1ms after the first but would have to wait for an additional 9ms, since it takes 10ms for the first packet to depart. The third packet would have to wait for an additional 18ms for both one and two to depart, and so on. This bottleneck-induced waiting is what creates the queues that form in the packet buffers at the link.

Figure 2 shows the connection one RTT (round-trip time) later. The bottleneck spaces out the packets, and they retain that spacing after leaving. The receiver just turns a data packet into an ack (acknowledgment packet), so the ack stream retains the spacing on the return path. The sender turns each ack into a data packet, so, after just one RTT, the packet-arrival rate at the bottleneck exactly equals the departure rate, and the queue won't grow. This is clearly visible in Figure 3, which shows the bottleneck queue size vs. time. There is an initial peak corresponding to the initial window's worth of back-to-back packets, but it dissipates after one RTT, leaving just a ±1 packet variation resulting from small phase differences in the arrival and departure processes.

Note, however, that the steady-state queue in Figure 3 is not zero. Figures 1 and 2 are sized so that the distance in time between the sender and receiver is 10 bottleneck packet times (for example, 100ms if the bottleneck packet spacing is 10ms); thus, it takes 20 packets in flight to "fill the pipe" and use 100% of the bottleneck capacity. The window in this example is 25 packets—five more than will fit in the pipe (the BDP, measured in packets). These create a *standing queue* that cannot dissipate. This is clearly visible in Figure 2: as the 26th packet is about to arrive at the bottleneck, five packets are still in its queue; but, from this time on, whenever one packet leaves, another will arrive. Thus, there will always be five packets in the queue (±1), even though the sender's rate is constant. The standing queue has nothing to do with the sender's rate but rather with how much the sender's window exceeds the pipe size.

This standing queue, resulting from a mismatch between the window and pipe size, is the essence of bufferbloat. It creates large delays but no improvement in throughput. It is not a phenomenon treated by queuing or traffic theory, which, unfortunately, results in it being almost universally misclassified as *congestion* (a completely different and much rarer pathology). These theories usually assume Poisson arrival processes, which are, by definition, uncorrelated. The arrivals of a closed-loop, reliable transport process such as TCP are completely correlated, resulting in an arrival and departure rate equality that theorists have dismissed as unnatural and wildly improbable. Since normal cures for congestion such as usage limits or usage-based billing have no effect on bufferbloat but annoy customers and discourage network use, addressing the real problem would be prudent.

The bufferbloat problem, making the window match the pipe size, is hard to address. Window sizes are chosen by senders while queues manifest at bottleneck gateways. It is difficult for senders to compute a window size, since both terms—bottleneck bandwidth and RTT—change constantly as connections come and go, paths change because of rerouting, layers 1 and 2 protocols adapt bandwidth for changing physical conditions, and so on. Since queues can be directly measured at the bottleneck, the most promising approach is to detect the problem there, then signal senders to reduce their windows (via TCP congestion-control mechanisms).

Reliable detection is difficult, however. The big queue at the start of Figure 3 is necessary for connections to get started, so queue magnitude contains no information about excess queue. The fact that the problem lies in the flat part of Figure 3 has caused some researchers to look at the time the buffer is occupied as an indicator.

Figure 4 shows the queue vs. time for a TCP receiver that sends one ack per window rather than one per packet. This is a legal, though brittle, operating mode used by some commercial operating systems with very high per-packet overhead to reduce the number of acks generated so as to perform credibly in benchmark tests. This kind of queue length variation is also seen in Web traffic—a superposition of mostly small transfers that are essentially all startup transient. Since this ack policy means a full window of packets is always delivered as a back-to-back burst, the initial turn-on transient repeats every RTT and the buffer is always occupied. The window exactly fills the pipe, however, and there is no excess queue, so any attempt to reduce this queue will result in poor utilization of the bottleneck. Thus, occupancy time contains no information about excess queue.

Figure 4 and the leading part of Figure 3 show queues doing their job—acting as shock absorbers to convert bursty arrivals into smooth, steady departures. This is *good queue*. The tail end of Figure 3 shows a queue doing nothing but creating excess delay. This is *bad queue*. The core of the bufferbloat-detection problem is separating good from bad. Figure 3 hints at the successful strategy: good queue is occupancy that goes away in about one RTT; bad queue persists for several RTTs. An easy, robust way to separate the two is to take the minimum of the queue length over a sliding time window that is longer than the nominal RTT.

Controlled Delay Management

In early 1998, we set out to understand why the original RED was difficult to configure and came up with a new RED algorithm (described in a talk and an unpublished paper[10,12]) that needed only one parameter: the queue's output bandwidth or average departure rate. Despite improvements in performance, issues remained—almost anything worked for long-lived TCP transfers and almost nothing worked for bursty traffic. In the decade since, many researchers have made strides in AQM, but no one has produced an AQM that has the following characteristics:

- It is parameterless—it has no knobs for operators, users, or implementers to adjust.
- It treats good queue and bad queue

Figure 3. Queue size vs. time.

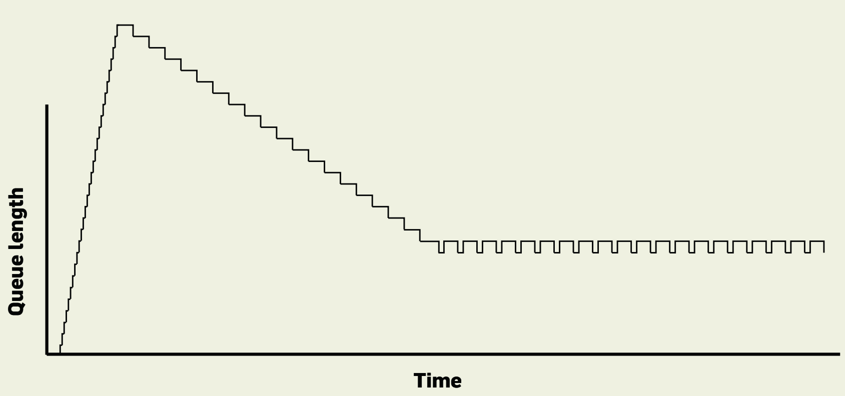
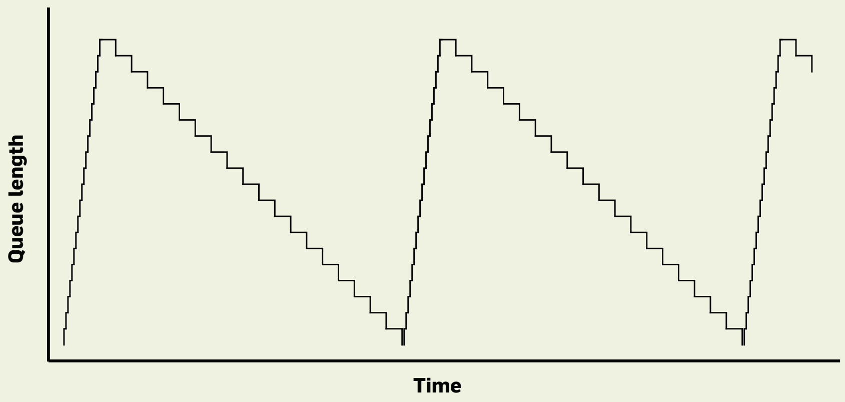


Figure 4. Queue size vs. time for ack-per-window receiver and 20-packet window.



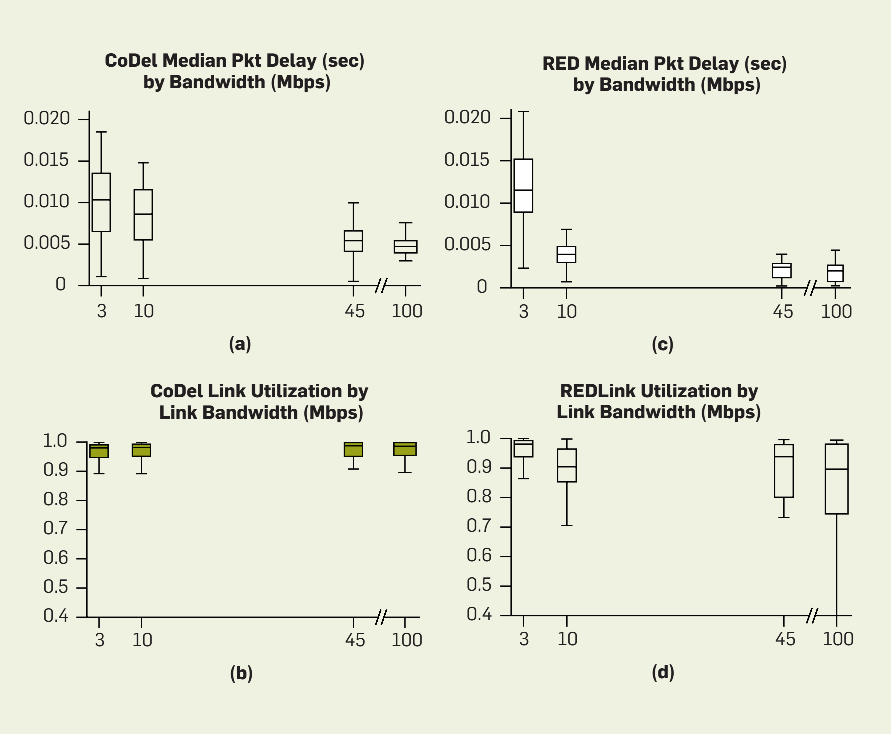
differently—that is, it keeps the delays low while permitting bursts of traffic.

- It controls delay, while insensitive to round-trip delays, link rates, and traffic loads.
- It adapts to dynamically changing link rates with no negative impact on utilization.
- It is simple and efficient—it can easily span the spectrum from low-end, Linux-based access points and home routers up to high-end commercial router silicon.

One Code Module, No Knobs, Any Link Rate

CoDel (Controlled Delay Management) has three major innovations that distinguish it from prior AQMs. First, CoDel's algorithm is not based on queue size, queue-size averages, queue-size thresholds, rate measurements, link utilization, drop rate or queue occupancy time. Starting from Van Jacobson's 2006 insight,[11] we used the local minimum queue as a more accurate and robust measure of standing queue. Then we observed that it is sufficient to keep a single-state variable of how long the minimum has been above or below the target value for standing queue delay rather than keeping a window of values to compute the minimum. Finally, rather than measuring queue size in bytes or packets, we used the packet-sojourn time through the queue. Use of the actual delay experienced by each packet is independent of link rate, gives superior performance to use of buffer size, and is directly related to the user-visible performance.

Figure 5. CoDel and RED performance variation with link bandwidth.

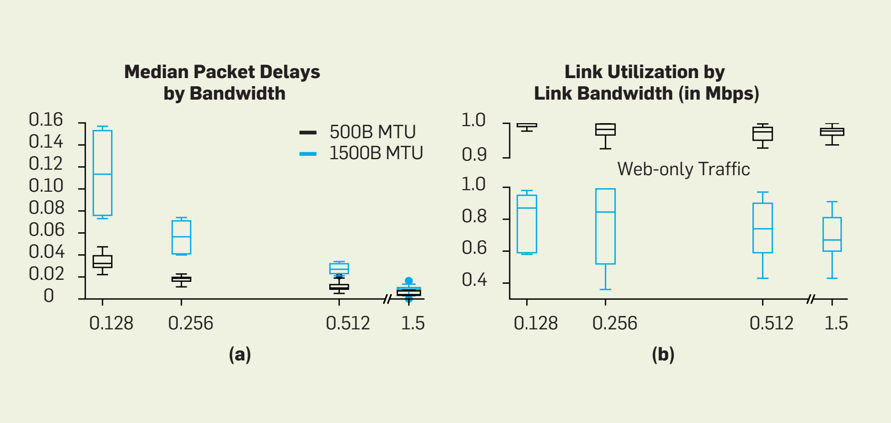

CoDel Median Pkt Delay (sec) by Bandwidth (Mbps)

RED Median Pkt Delay (sec) by Bandwidth (Mbps)

CoDel Link Utilization by Link Bandwidth (Mbps)

REDLink Utilization by Link Bandwidth (Mbps)

Figure 6. CoDel performance variation with link bandwidth for low bandwidths.


Median Packet Delays by Bandwidth

Link Utilization by Link Bandwidth (in Mbps)

Using the minimum value has some important implications. The minimum packet sojourn can be decreased only when a packet is dequeued, which means all the work of CoDel can take place when packets are dequeued for transmission and that no locks are needed in the implementation. The minimum is the only statistic with this property. The only addition to packet arrival is that a timestamp of packet arrival time is created. If the buffer is full when a packet arrives, then the packet can be dropped as usual.

CoDel assumes a standing queue of `target` is acceptable and that it is unacceptable to drop packets when there are fewer than one MTU's (maximum transmission unit's) worth of bytes in the buffer. CoDel identifies the persistent delay by tracking the (local) minimum queue delay packets experience. To ensure the minimum value does not become stale, it has to have been experienced within the most recent `interval`. When the queue delay has exceeded `target` for at least `interval`, a packet is dropped and a control law sets the next drop time. The next drop time is decreased in inverse proportion to the square root of the number of drops since the dropping state was entered, using the well-known relationship of drop rate to throughput to get a linear change in throughput.[12,16] When the queue delay goes below `target`, the controller stops dropping. No drops are carried out if the buffer contains fewer than an MTU's worth of bytes. Additional logic prevents reentering the dropping state too soon after exiting it and resumes the dropping state at a recent control level, if one exists.

`Target` and `interval` are constants with straightforward interpretations: acceptable standing queue delay and a time on the order of a worst-case RTT of connections through the bottleneck. We experimented to determine values for `target` and `interval` that give a consistently high utilization with a controlled delay across a range of bandwidths, RTTs, and traffic loads. Below a `target` of 5ms, utilization suffers for some conditions and traffic loads; above 5ms there is very little or no improvement in utilization. `Interval` is loosely related to RTT since it is chosen to give endpoints time to react without being so long that response times suffer. A setting of 100ms works well across a range of RTTs from 10ms to 1 second (excellent performance is achieved in the range from 10ms to 300ms). (Pseudocode for CoDel is included in the appendix, available at http://queue.acm.org/appendices/codel.html.)

CoDel's efficient implementation and lack of configuration are unique features that make it suitable for managing modern packet buffers. The three innovations—using minimum rather than average as the queue measure, simplified single-state variable tracking of minimum, and use of queue-sojourn time—lead directly to these unique features.

Simulation Experiments

No network simulation provides a high-fidelity representation of reality; the real test of CoDel will come through deployment on networks. Though CoDel's functionality and an evaluation of its merits are our main concerns, some AQM comparisons can serve as a sanity check. We did several thousand simulation runs, varying bottleneck bandwidth from 64kbps to 100mbps, end-to-end delay from 5ms to 1sec., simultaneous bulk data transfers from 0–50, PackMime Web traffic intensity from 0–80, with and without CBR traffic, with both Reno and CUBIC TCP congestion control, and all with CoDel using the same, constant configuration values. As the next section will show, CoDel performed very well. Most importantly, when link rates vary dynamically—even as much as two orders of magnitude—CoDel adapts. Results are compelling enough to move on to the next step of extensive real-world testing in Linux-based routers.

Simulator Configuration. For expediency, we used the freely available ns-2 simulator.[18] Our file-transfer application used the Linux TCP suite, primarily with CUBIC congestion control. Results (available online) were similar with New Reno but with slightly lower utilizations, as expected. To emulate Web-browsing loads, we used PackMime,[21] which runs with the ns-2 Full-TCP protocol and uses the New Reno congestion-control algorithm. A constant bit-rate application used UDP (User Datagram Protocol) and was configured at 64Kbps in 100-byte packets. All TCPs were configured with modern parameters, including use of SACK. In most cases, CoDel's managed buffer was set very large (~8xBDP packets) to verify that the actual size of the buffer does not matter. For comparison, we also ran many of our test scenarios with the ns-2 RED AQM module substituted for CoDel. We used the most recent settings and code for ns-2 RED, which reads the initial link bandwidth and delay to adjust its settings, rather than the original RED of Floyd and Jacobson.[5] We also attempted to run the scenarios with BLUE[22] but its default settings performed badly and limited experimentation did not result in settings that worked well so we did not pursue it.

Metrics. The metrics of interest for AQMs—queue delay and size, link utilization, and "fairness" of drops between flows—can be quite sensitive to the types of traffic mixes and link bandwidths,[10,12,20] so we tested a range of traffic loads and conditions. The per-packet queue delay for successfully transmitted (not dropped) packets is particularly useful. This can be viewed against simulation time for a single run or the statistics over the run (for example, median and 95th percentile) can be used for comparisons and trends. Monitoring link utilization ensures queue management does not have a negative impact on throughput. Though not our primary concern, we used the Jain fairness index to see if drop share per source was somewhat in proportion to the number of packets transmitted, computed for n samples as:

$$\frac{(\Sigma_i x_i)^2}{n \cdot (\Sigma_i x_i^2)}$$

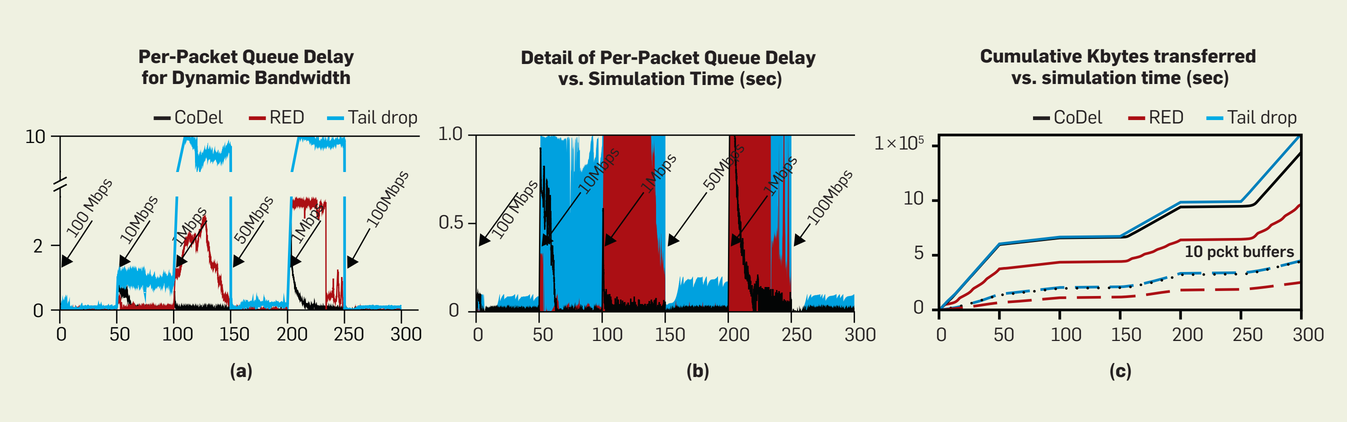
Performance Across a Range of Static Link Rates

CoDel is the same code with the same settings, regardless of egress link rate. To see how well it works, we collected the median packet delay and link utilization values from a number of traffic loads (FTPs with and without added Web-browsing and constant-bit-rate applications) and RTTs from 10ms–500ms, sorted them by link bandwidths, and box plotted the results. These are displayed separately for larger bandwidths (3Mbps, 10Mbps, 45Mbps, and 100Mbps in Figure 5) and smaller bandwidths (128Kbps, 256Kbps, 512Kbps, and 1.5Mbps in Figure 6). The results for RED are also shown for the larger bandwidths, but median delays for the smaller set were excessive (100ms–200ms).

CoDel drops only when its minimum delay statistic exceeds 5ms and the buffer contains at least a link MTU's worth of bytes. For links with MTUs transmitting in less than 5ms (the larger bandwidth group) and traffic loads consisting of all long-lived FTPs, the median delay should approach this target. CoDel is designed to permit short-term bursts; thus, median values for bursty traffic patterns should be higher than the 5ms target. For larger bandwidths, delay resulting from bursts of packets is small compared with the target (for example, a 1,500-byte packet is transmitted in 0.1ms at 100 Mbps), so load variations have less effect on the median. At smaller bandwidths the delay caused by an additional packet is significant compared with the target (or example, 4ms for a 3Mbps link), and median delays will be noticeably larger. This is exactly the desired behavior: the longer delays are not excessive and permit link utilizations to be maintained at high levels. Figure 5 shows CoDel delays are as expected and the link utilizations are good. RED's delays and utilizations are similar for 3Mbps links, but delays and utilizations are smaller for 10Mbps and above. The low utilizations show that RED is likely overcontrolling.

The lower bandwidth values depicted in Figure 6 use a smaller range of RTTs (30ms–100ms). Such link rates can be expected at the consumer Internet-access edge (handheld, home) or on degraded Wi-Fi links. Static low-bandwidth links usually use a smaller link MTU, so we collected data where the link MTU was set to 500 bytes, as well as the 1,500-byte MTU used for all other runs. As expected, the larger MTU

Figure 7. Wireless example.



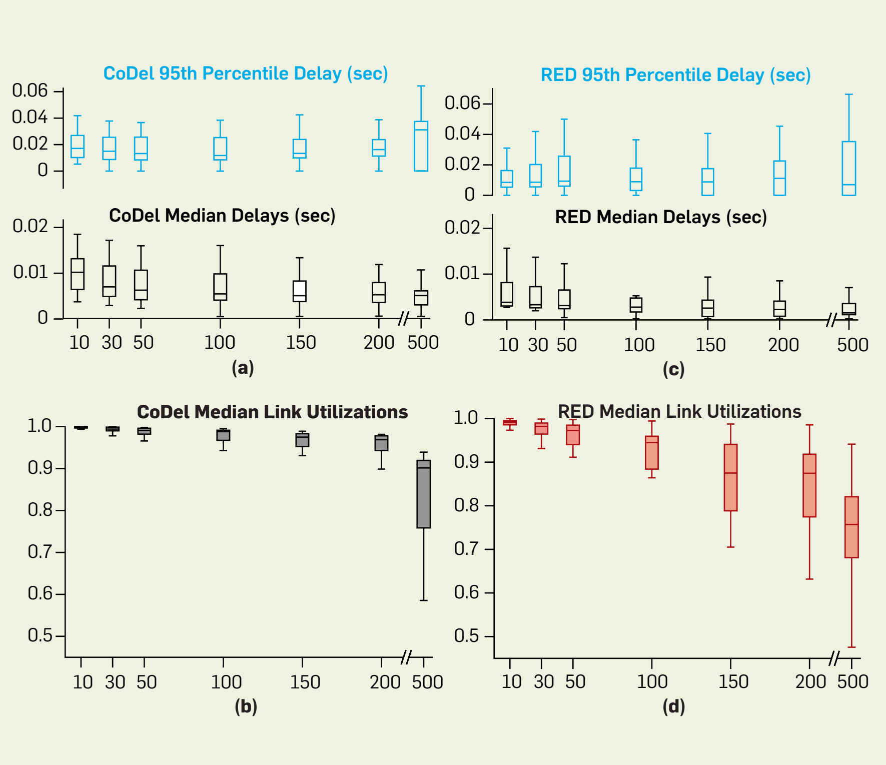
increases delays, but less significantly as bandwidth increases. Utilizations for low-bandwidth links are generally good since they are easy to fill with mixes of FTPs. A few runs were done at each bandwidth with only the PackMime Web browsing, which makes it difficult to get high utilizations; we had to accept a drop rate of more than 10% to get over 60% utilization.

Performance on Dynamic Links

Some dynamic link testing is possible in simulation. To roughly emulate a (nominal) 100Mbps Wi-Fi link subject to degradation, we used a load of four FTPs and five Web connections per second and changed link rates at 50-second intervals (over the 300 simulated seconds), first dropping to 10Mbps, then to 1Mbps, then jumping to 50Mbps, dropping to 1Mbps, and finally jumping back to 100Mbps. Buffer capacity is a single BDP (830 packets) for the nominal rate. This scenario was repeated for CoDel, Tail Drop, and RED.

Figure 7 shows the per-packet queue delay and the cumulative number of kilobytes transferred as simulation time progresses. As expected, Tail Drop generally keeps its buffer full, delaying packets by the amount of time it takes to transmit a buffer full of packets at the current link rate. That delay can be as much as 10 seconds. RED keeps the queue delay smaller than Tail Drop but does not respond to changes as quickly as CoDel. CoDel permits an initial spike as the FTPs get started and a dropping rate is learned for the current conditions. The delays spike when the link rates drop (at 50, 100, and 250 seconds), as the queue size is now too long for the new rate. CoDel computes a new control point within 100 ms, the maximum interval a local minimum is valid. It is an open question whether anything should be done to speed this; preliminary studies show it is better not to attempt to "clear out" the backlog. If rate changes of an order of magnitude or more several times a minute are common, then the issue may require further study.

Figure 8. CoDel and RED performance by RTT.



Comparing the kilobytes transferred with the queue delay, it is easy to see that long delays are not required for high utilization. CoDel transfers almost the same total kilobytes as Tail Drop, with the differences coming at the rate jumps (150 and 250 seconds), where CoDel's FTPs have to ramp up to fill the new pipe size while Tail Drop's queues send their backlog. Undersized buffers are not the answer to bufferbloat. Figure 7 shows the same scenario using 10 packet buffers, a size suitable for the 1Mbps rate. Throughput is about 75% *less* for all three schemes (Tail Drop and CoDel are identical). Tail Drop tends to keep its 10-packet buffer full, which would result in a worst-case delay of 120ms but with 25% of the throughput that can be achieved with a large CoDel-managed buffer—which gets a median delay of 2.7ms, 75th percentile delay of 5ms, and is less than 90ms 95% of the total simulation time.

Dropping the Right Packets

Although most network analysis today assumes connections with an unloaded 100-ms RTT, in practice RTTs vary. Unless a path includes a satellite link, RTTs in the one-second range are usually caused by bufferbloat, not the intrinsic path characteristics. At the consumer edge, few connections will have less than a 30ms RTT. Since CoDel's interval is weakly related to RTT, we tested the effectiveness of a 100-ms setting over a wide range of likely RTTs and report on the range from 10ms–500ms.

Figure 8 shows results for a variety of traffic loads, sorted by RTT. Both CoDel and RED keep the median delay low, but CoDel has higher link utilizations and better drop-share fairness, showing that CoDel's design is more effective at dropping the *right* packets. CoDel's utilization is very close to that of Tail Drop (except for an RTT of 500ms) but with much less delay. CoDel's performance metrics are not significantly different between 30ms and 200ms RTTs. Utilizations are slightly less and have a larger range as the RTT increases, because some traffic loads have difficulty keeping the larger pipe full. The 500ms RTT utilization shows more variation, the low end corresponding to single FTPs, which have difficulty keeping a very large pipe full.

Figure 9 compares the Jain fairness index for the source-drop shares of CoDel and RED for these runs. CoDel consistently outperforms RED for this metric. This seems to be, in part, because the changes to the original RED make drops less randomly distributed while CoDel gets randomness from the independence of drop intervals and packet arrivals.

Consumer Edge

This scenario roughly emulates a consumer edge for two (symmetric) bandwidths: 512KB and 1.5MB. The load includes a two-way 64Kbps CBR (VoIP-like), an infinite FTP as a download, Web browsing at a rate of two connections per second, and uploads of small FTPs—1MB with short idle periods (5 to 15 seconds, uniformly distributed) between. The table lists results, where "C" is CoDel and "T" is Tail Drop and each link direction is shown separately. Although CoDel never drops packets at a higher rate than Tail Drop, it keeps a much smaller queue and transfers similar amounts of data, offering encouragement for taming bufferbloat.

The experiments presented here mainly consist of "forward" traffic where all the data traffic is going in the analyzed direction. Reverse traffic has well-known issues of ack compression or data pendulum, which tend to push the delay up and the utilization down. There are known mitigations to improve the mixing of acks and data packets that will be performed in the home-router implementation. Even the unmitigated simulation experiments showed acceptable performance.

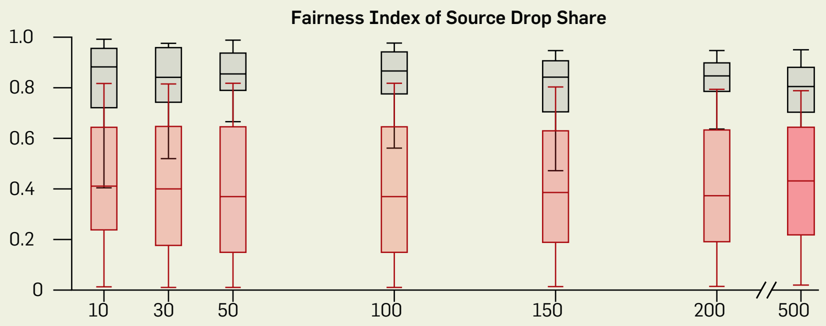
AQM is not a substitute for differentiated queuing to provide priority for packets that need low latency and jitter. We have had a lot to say in the past about solutions for that type of traffic; AQM should be employed on the packet queues handling common Internet traffic and a Delay Bound per-hop behavior used for latency sensitive traffic.

CoDel lends itself to efficient implementation in Linux-based routers, crucial to deployment at the edge. Other AQM implementations require locks on the queue that are not necessary for CoDel. There are also a small number of state variables. We believe CoDel's algorithm can be efficiently implemented in silicon.

Manage the Right Queue

At this point, a savvy user could be tempted to deploy CoDel through

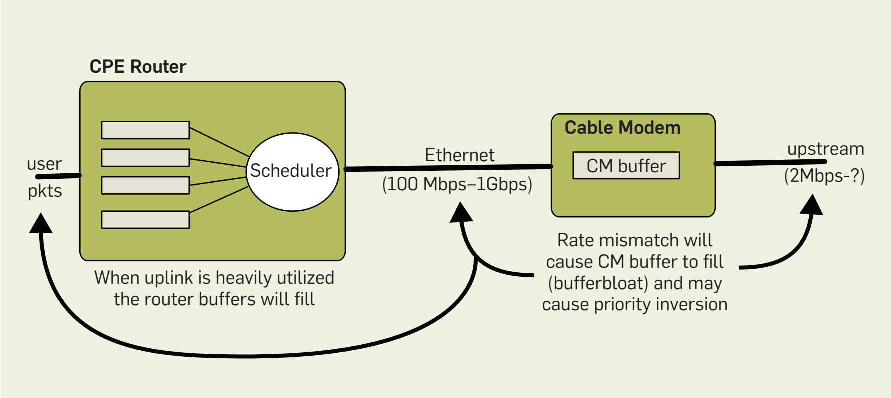
Figure 9. Jain fairness for drop share, CoDel (shown in black) and RED (shown in red).



Consumer edge example.

	512Kbps Links						1.5Mbps Links					
	download			upload			download			upload		
metric	C	T	C/T	C	T	C/T	C	T	C/T	C	T	C/T
pkt drop %	8	8	100%	1.5	5.8	25%	3.5	4.7	75%	1.4	2.5	56%
median delay (ms)	18	73	25%	9	37	25%	8	49	17%	0	0	100%
total Mbytes	17	18	95%	13	14	92%	37	40	92%	22	21	103%
fairness of drops	0.87	0.9	98%	0.9	0.9	104%	0.8	1	84%	0.6	0.7	85%

Figure 10. Rate mismatch in a cable modem.



a CeroWrt-enabled edge router to make bufferbloat disappear. Unfortunately, large buffers are not always located where they can be managed but can be ubiquitous and hidden.[6,7] Examples include consumer-edge routers connected to cable modems and wireless access points with ring buffers. Many users access the Internet through a cable modem with varying upstream link speeds: 2 Mbps is a typical value. The home network or home computer connects to the cable modem with an Ethernet cable in the range of 100Mbps–1Gbps (Figure 10). The modem's buffers are at the fast-to-slow transition, and that is where queues will build up: inside a sealed device outside of user control. Any differentiated services (DiffServ) queuing and buffer management at the router can be defeated by a full queue in the cable modem. Three approaches have been suggested: limit the Ethernet link to the upstream rate; put buffer management and DiffServ queues in the cable modem with a configuration interface for the DiffServ queues; and implement Ethernet flow control between the modem and the upstream router.

Option 1 has the advantage that it can be implemented by a user without cable-modem changes and the disadvantage that it must rate limit to the expected rate of the upstream. If the rate drops, then a queue will still build up in the cable modem, and any additional short-term bandwidth cannot be utilized. Option 2 puts buffer management right at the bottleneck link but requires the vendor to make (possibly significant) changes to the modem architecture and permit configuration. Option 3 also requires vendors to make changes but uses Ethernet flow control to permit only the number of packets in the modem buffer needed for good transmission utilization while pushing the queue into the router where it can be managed and where new algorithms can be more readily deployed. Options 2 or 3 are preferable but require a cable-modem vendor and/or a cable data network service provider to make this part of the modem requirements.

A modern AQM is just one piece of the solution to bufferbloat. Concatenated queues are common in packet communications with the bottleneck queue often invisible to users and many network engineers. A full solution has to include raising awareness so that the relevant vendors are both empowered and given incentive to market devices with buffer management.

Next Steps

The open source project CeroWrt[3] is using OpenWrt to explore solutions to bufferbloat. A CoDel implementation is in the works, after which real-world data can be studied. We plan to make our ns-2 simulation code available, as well as some further results.[19]

Related articles on queue.acm.org

Bufferbloat: Dark Buffers in the Internet
Jim Gettys, Kathleen Nichols
http://queue.acm.org/detail.cfm?id=2071893

Revisiting Network I/O APIs: The netmap Framework
Luigi Rizzo
http://queue.acm.org/detail.cfm?id=2103536

The Robustness Principle Reconsidered
Eric Allman
http://queue.acm.org/detail.cfm?id=1999945

References

1. Braden, R. et al. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309 (1998).
2. Bufferbloat Project; http://www.bufferbloat.net.
3. CeroWrt Project; http://www.bufferbloat.net/projects/cerowrt.
4. Dischinger, M. et. al. Characterizing residential broadband networks. In *Proceedings of the Internet Measurement Conference*, San Diego, CA, 2007.
5. Floyd, S. and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, (1993).
6. Gettys, J. Bufferbloat: Dark buffers in the Internet. Backspace Column. *IEEE Internet Computing 15*, 3 (2011), 95-96.
7. Gettys, J. and Nichols, K. 2011. Bufferbloat: Dark buffers in the Internet. *Commun. ACM 9*, 11 (Sept. 2011) 57–65.
8. Jacobson, V. Congestion avoidance and control. *Proceedings of SIGCOMM '88*, (Stanford, CA, 1998).
9. Jacobson, V. Reported in Minutes of the Performance Working Group. *Proceedings of the Cocoa Beach Internet Engineering Task Force.* Corporation for National Research Initiatives, Reston, VA, 1989.
10. Jacobson, V. Notes on using RED for queue management and congestion avoidance. Talk presented at North American Network Operators' Group (1998); ftp://ftp.ee.lbl.gov/talks/vj-nanog-red.pdf.
11. Jacobson, V. A rant on queues. A talk presented at MIT Lincoln Labs, Lexington, MA, 2006; http://www.pollere.net/Pdfdocs/QrantJul06.pdf.
12. Jacobson, V., Nichols, K. and Poduri, K. RED in a different light, 1999; http://www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf.
13. Kreibich, C. et. al. Netalyzr: Illuminating the edge network. In *Proceedings of the Internet Measurement Conference*, (Melbourne, Australia, 2010).
14. Li, T. and Leith, D. Adaptive buffer sizing for TCP flows in 802.11e WLANs. In *Proceedings of the 2008 Communications and Networking in China.*
15. Mankin, A. Random drop congestion control. In *Proceedings of SIGCOMM '90.*
16. Mathis, M., Semke, J. and Mahdavi, J. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review 27*, 3 (1997).
17. Nagle, J. Congestion control in IP/TCP internetworks. RFC 896 (1984); http://www.ietf.org/rfc/rfc896.txt,.
18. Network Simulator - ns-2; http://nsnam.isi.edu/nsnam/index.php/User_Information.
19. http://www.pollere.net/CoDel.html.
20. Vu-Brugier, G., et. al. 2007. A critique of recently proposed buffer-sizing strategies. *ACM SIGCOMM Computer Communication Review 37*(1).
21. Weigle, M. C. 2002. Web traffic generation in ns-2 with PackMime-HTTP; http://www.cs.odu.edu/~mweigle/research/packmime.
22. Feng, W. et. al. 2002. The BLUE Active Queue Management Algorithm. *IEEE/ACM Transactions on Networking 10*, 4 (2002), 513–528.

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.

Van Jacobson is a Research Fellow at PARC where he leads its content-centric networking research program. He has worked at Lawrence Berkeley National Laboratory, Cisco Systems, and was a cofounder of Packet Design.