ACCESS NETWORK TECHNOLOGIES

ACTIVE QUEUE MANAGEMENT ALGORITHMS FOR DOCSIS 3.0

A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks

Prepared by:

Greg White
Principal Architect, Access Network Technologies
g.white@cablelabs.com

CableLabs R&D Lead:

Dan Rice
Vice President, Access Network Technologies
d.rice@cablelabs.com

April 2013

DISCLAIMER

ACKNOWLEDGMENTS

The author wishes to thank: Kathleen Nichols for her contribution of the CoDel and SFQ-CoDel implementations and for her development of a number of the traffic models; Joey Padden, Takashi Hayakawa, and Dave Täht for their significant contributions to the development of the simulation platform and testing methodology; Rong Pan, Preethi Natarajan, Mythili Prabhu and Chiara Piglione for providing the PIE implementation, and for their work on tuning it for the DOCSIS MAC.

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

This paper describes the results of a simulation study of three active queue management algorithms applied to the upstream transmission buffer in a DOCSIS 3.0 cable modem. This paper is a follow-on to an earlier study which examined the "Controlled Delay" (CoDel) active queue management algorithm in a simulated DOCSIS 3.0 cable modem. This expanded study looks at CoDel in more depth, and compares it to two other promising active queue management algorithms, Stochastic Flow Queue - CoDel (SFQ-CoDel) and Proportional Integral Enhanced (PIE). These three queue management algorithms are compared to existing (tail drop) buffering implementations that exist in current cable modems across a range of latency-sensitive applications.

It is demonstrated that current cable modem implementations result in severe degradation of user experience for latency-sensitive applications in situations where the user is simultaneously uploading a file via TCP. The goal of the active queue managers in this study is to prevent the degradation of latency-sensitive applications, while not impacting the TCP upload performance.

The "Stochastic Flow Queue - Controlled Delay" active queue manager displays extremely good performance in most traffic scenarios, enabling up to 200x reduction in latency for gaming traffic, 10x reduction in web page load time, and pristine VoIP quality, all while minimally impacting TCP upload performance.

The "Proportional Integral Enhanced" active queue manager similarly provided very good performance, and is optimized for efficient implementation in existing cable modems.

1 INTRODUCTION

1.1 WHY IS LATENCY IMPORTANT?

Packet forwarding latency can have a large impact on the user experience for a variety of network applications. The applications most commonly considered as latency-sensitive are real-time interactive applications such as VoIP, video conferencing and networked "twitch" games such as first-person shooter titles. However, other applications are sensitive as well; for example, web browsing is surprisingly sensitive to latencies on the order of hundreds of milliseconds.

There are established models for the degradation in user experience for VoIP caused by latency. In the model we use for estimating VoIP quality, every additional 20 ms in latency causes a decrease in the VoIP Mean Opinion Score (MOS) of approximately 0.005 MOS points up to a latency of 177 ms. Beyond the threshold of 177 ms, each additional 20 ms of latency reduces VoIP quality MOS score by approximately 0.13 MOS points.

While online games don't have similar well-vetted models for the impact that network parameters have on user experience, a number of researchers have studied the topic, and some data exists to indicate that access network latencies should be kept below 20 ms in order to provide a good user experience.

Loading a web page involves an initial HTTP GET method to request the download of an HTML file, which then triggers the download of dozens or sometimes hundreds of resources that are then used to render the page. While many servers may be involved in providing the page contents, generally speaking, the majority of the resources are served from a small number (4 or 5) of servers. Web browsers will typically fetch the resources from each server by opening up multiple (typically 6) TCP connections to the server, and requesting a single resource via each connection. Once each individual resource is received, the browser will close the TCP connection and open a new one to request the next resource, thus keeping the same number of connections open at a time. The result of this hybrid parallel-serial download is that the page load time is in some cases driven by the serial aspect, i.e., the number of sequential downloads (one completing before the next can start), of which there may be a dozen or more. Round-Trip latency can impact the page load time due to the fact that completion of each resource download is delayed by any additional round-trip time in the network. Thus, when RTT increases, page load time can increase by 10x-20x that amount.

Related to their SPDY protocol, the developers at Google presented a "Google Tech Talk" [Peon]. That talk was intended to provide motivation for the development of SPDY as HTTP/2.0, a replacement for HTTP 1.1, and they illustrate the sensitivity that page-load time has relative to the round-trip time.

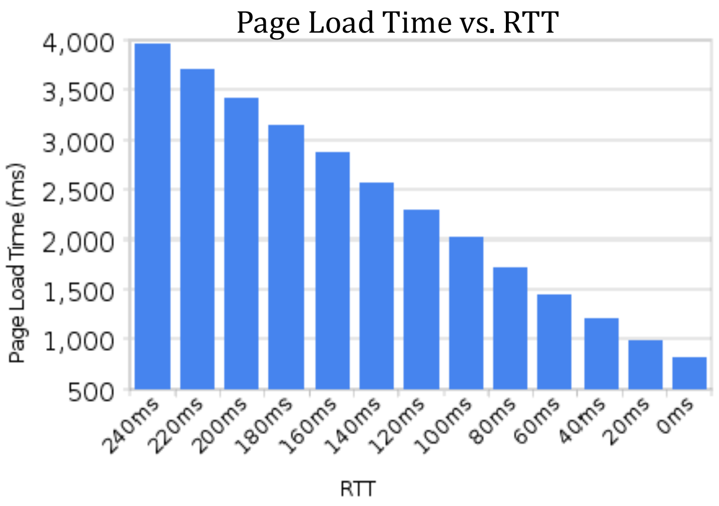

Figure 1 - Page Load Time vs. Round-Trip Time

Figure 1 shows that as the round-trip time between the Web browser and the servers decreases, the page-load time decreases linearly. For the Web page that they used in generating that plot, it shows a 14x multiplier. For example, a 200 millisecond increase in round-trip time results in a 2.8 seconds increase in page load time, and it doesn't matter whether that increase in round-trip time comes on the upstream leg of the connection or the downstream leg.

So it is clear that there is a benefit to keeping network latency low if we are interested in ensuring good user experience for a range of applications.

1.2 Megabits Myth?

Contrast the above with the sensitivity of page load time versus the link bandwidth and you can see that, at the rates that cable modem customers are getting today, we are really in the space of diminishing returns. Anything beyond about 6 Mbps returns almost imperceptible improvements in page load time.

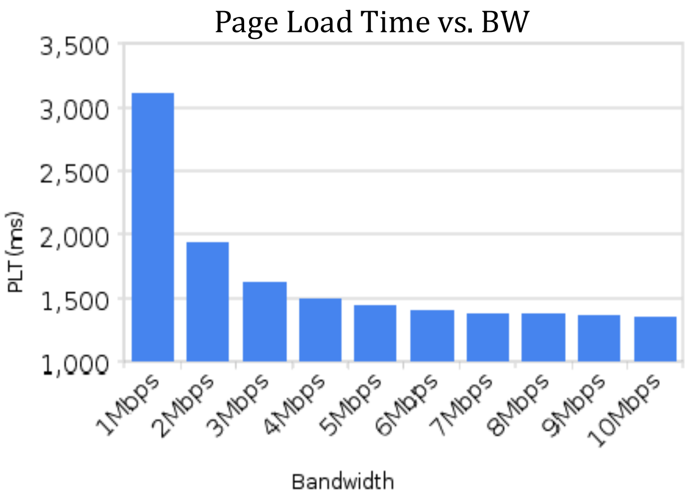

Figure 2 - Page Load Time vs. Bandwidth

Similarly, many of the other network applications operate at data rates well below what is commonly provisioned for cable modem service.

There is a lot of focus on bandwidth: it's the top-line number that has been used to market high-speed data service. For the foreseeable future that will probably be the case, but when it comes down to the user experience for the actual applications that broadband customers are using, improvements in latency are more important at this point than improvements in bandwidth.

Some aspects of latency are hard to fix. There is the propagation delay from the user to the server, or for a VoIP session, between two users. There's not much you can do about the speed of light, but routing paths can be made as short as possible, and CDNs can reduce the physical distance and number of hops for some content.

On the other hand, there is a significant issue which has gained a lot of press in technical circles in the past few years that is pointing to the fact that a lot of network elements have more buffering memory in them than is really good for application performance. The term "buffer bloat" has been coined to describe this.

1.3 "Bufferbloat"

Every piece of network equipment has to have some amount of buffering in order to handle bursts of packets on an ingress link, and then to play them out on an egress link. It's particularly important in cases where there is a mismatch between the rate into the device and the rate out. For example, imagine a switch with a GigE ingress link and a 100 Mbps egress link. Even if the average ingress rate is 100 Mbps, the ingress link will often provide that traffic in bursts of packets at 1 Gbps. Buffering is pretty important to make sure that the switch can accept those bursts and play them out on the 100 Mbps link. From the perspective of egress link utilization, more buffering is better, since it reduces the chance that the egress link will go idle. For bulk TCP traffic (file transfers), user experience is driven by how quickly the file transfer can complete, which is directly related to how effectively the protocol can utilize the network links, again supporting the view that more buffering is better.

But the downside to large buffers is that they result in excessive latency. While this isn't an issue for the bulk file transfers, it is clearly an issue for other traffic, and the issue is exacerbated by the TCP itself. The majority of TCP implementations use loss-based congestion control, which means the TCP ramps up its congestion window (effectively ramps up its sending rate) until it sees packet loss, then it cuts its congestion window in half, and then starts ramping back up again until it sees the next packet loss, and that saw-tooth continues. In a lot of networks, especially wired networks, packet loss doesn't come from noise on the wire. It comes from buffers being full, and when a packet arrives at a full buffer it has to be discarded. This is how the TCP automatically adjusts its transmission rate to match the available capacity of the bottleneck link.

The result of this saw-tooth behavior being driven by buffer exhaustion is that the buffer at the head of the bottleneck link is going to saw-tooth between partially full and totally full. Depending on the particular flavor of TCP congestion control (Reno, New Reno, CUBIC, etc.) the portion of time spent in the full (or nearly full) state will vary, and if there are multiple TCP sessions sharing that bottleneck link, then the average buffer occupancy will increase. Furthermore, if the buffer is oversized, its average occupancy will be higher as well.

In DOCSIS networks, the cable modem is generally at the head of the bottleneck link for upstream traffic. Historically, and still typically today, CMs have had a much bigger buffer than is needed to keep TCP working smoothly.

The two of those factors together – the modem being at the head of the bottleneck link and having an oversized buffer – plus the fact that TCP is going to try to keep that buffer full, results in high upstream latency through the modem whenever there is an upstream TCP session. The terms that have been coined to describe this phenomenon are "bufferbloat" or "latency under load".

The result of bufferbloat is that applications other than upstream TCP suffer. Even though the other applications might be low bandwidth, and TCP will back off to accommodate them on the link, their packets arrive to a full or nearly full buffer that may take hundreds of milliseconds or even seconds to play out. This can make web browsing perform poorly and make VoIP, video chat, or online games unusable. In addition, this could potentially affect downstream TCP performance as well, since the upstream ACKs would experience similar latencies. However, this effect was identified some time ago, and as a result all cable modems have for years supported some kind of ACK prioritization scheme that allows upstream TCP ACKs to bypass the large queue.

One reason this situation has persisted in DOCSIS cable modems is that since DOCSIS 1.1, modems have supported multiple service flows. The presumption on the part of modem developers has been that if operators are concerned about latency for certain traffic flows, they can create a separate service flow to carry that traffic. Unfortunately this isn't a feasible solution in the vast majority of cases.

1.4 Measuring Bufferbloat

There have been a number of efforts in recent years to characterize buffer bloat. Figure 3 comes from a paper [Dischinger] that really kicked off a lot of the interest in solving this problem. It shows the amount of buffering delay or queuing delay in DSL and cable networks, circa 2007. It shows delays on the order of seconds on the upstream for cable modems. Consider again the 14x multiplier effect described earlier. This amount of buffering would result in page load delays on the order of 30 seconds to a minute.

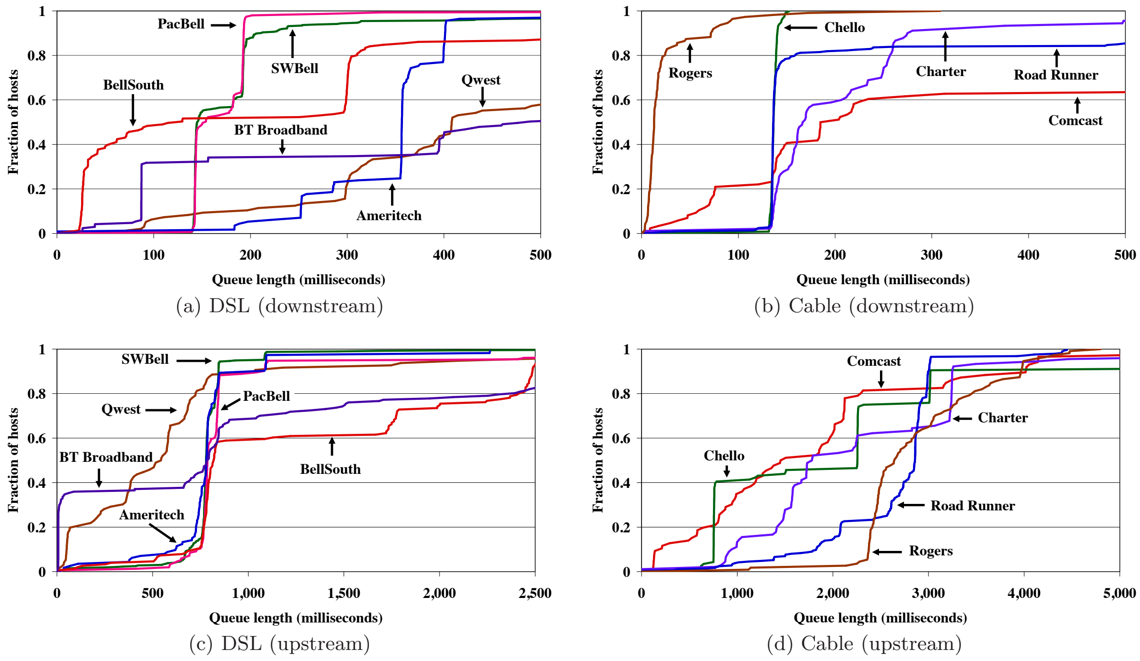

Figure 3 - Buffering Delay in Residential Broadband Networks

Figure 3 does show that this problem isn't limited to cable modems. CMTSs appear to also have a significant amount of buffering, and DSL systems suffer as well.

Another data point comes from the SamKnows testing that has been going on in the US for the past couple of years. SamKnows conducts tests (and the FCC produces reports) on latency under load as well. Figure 4 is a graphic from a paper [Sundaresan], which analyzed the data that was published in the 2011 version of the Measuring Broadband America report from the FCC.

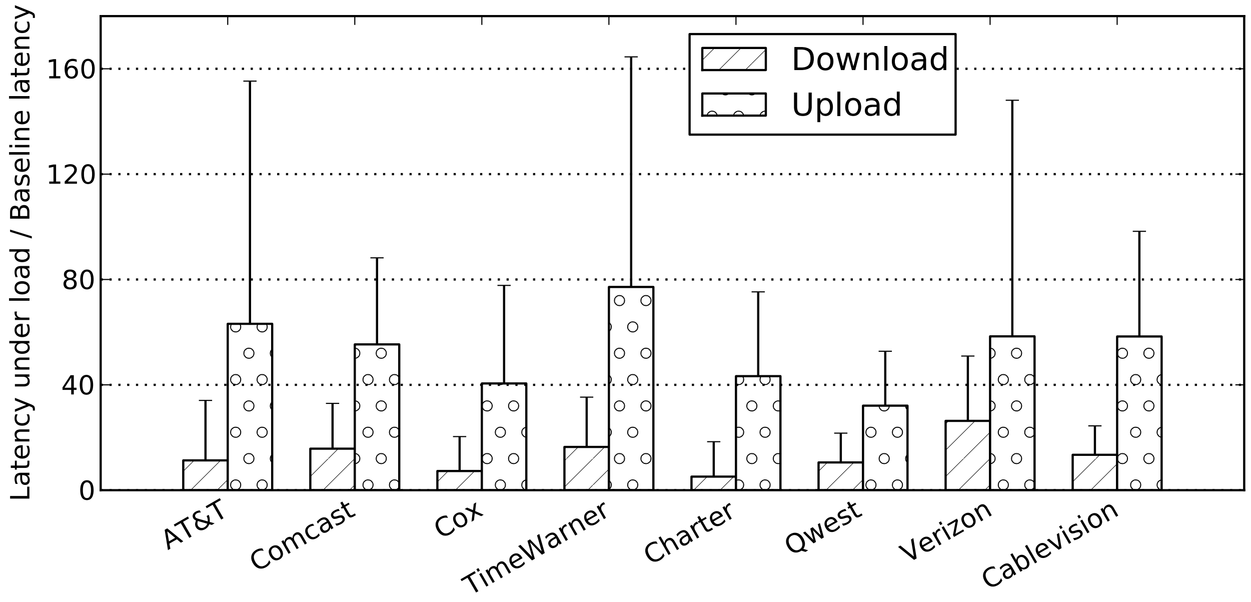

Figure 4 - FCC/SamKnows Data on Latency Under Load

This shows the ratio of "latency under load" to baseline latency for both upstream and downstream. Each bar shows the mean value of that ratio and the top of the whisker is the highest ratio that they saw in their testing.

So again, it's not a problem that's specific to cable, AT&T, Qwest, Verizon are showing up here as well. And again, it seems to be a bigger problem on the upload side, where we see ratios of 40 to 80 times as much as latency under loaded conditions than you see in the nominal or baseline condition when there's effectively no TCP running to clog up the upstream buffer in the modem.

Additionally, the ICSI Netalyzer test can measure upstream and downstream buffering. Below are a few data points collected in March/April 2013 using Netalyzer by Matt Tooley of NCTA. In Table 1, speeds are reported in kbps, and latencies are in ms. The Netalyzer tool does not measure speeds in excess of 20 Mbps, so a number of the data points are shown as >20000 kbps.

Table 1 - Snapshot of Netalyzer Tests of Speed and Buffering Latency

ISP	Technology	Down		Up	
		Speed	Latency	Speed	Latency
Comcast	DOCSIS	>20000	710	4100	480
Comcast	DOCSIS	>20000	720	6000	240
Cox	DOCSIS	>20000	710	4000	480
Rogers	DOCSIS	>20000	89	1900	270
ATT	U-Verse	5600	83	1000	1400
ATT	U-Verse	5600	59	1000	1400
ATT	DSL	1200	3800	330	6700
Verizon	FIOS	8000	270	7000	89
Verizon	FIOS	12000	240	9000	200
Comcast	EPON	>20000	56	>20000	99
ATT	3G	2000	850	1000	1700
Clearwire	WiMAX	8000	1700	340	1700

This recent data seems to indicate that upstream latencies in DOCSIS may be coming down relative to what they were in 2007. One probable explanation is that the amount of buffering (measured in packets or bytes) hasn't changed, but as upstream data rates have risen over the intervening 6 years, the latency caused by that buffer has dropped. Still, there is perhaps 5x or 10x as much upstream buffering as is needed at these speeds. Also notable is that the downstream buffering latency in DOCSIS is actually higher than upstream for all except Rogers.

All of these reports point to what happens if you set up a specific test case that is designed to uncover this issue. The next question is, how often does it really happen in practice for residential cable customers? How often are they seeing the effect of bufferbloat? A researcher from ICSI did a study to try to identify that. It's not a direct measure, but rather an estimate of how much buffering latency is seen in the real world by users doing real world activities. The paper [Allman] goes into a lot of detail on the methodology they used, but the main result is provided in Figure 5 below.

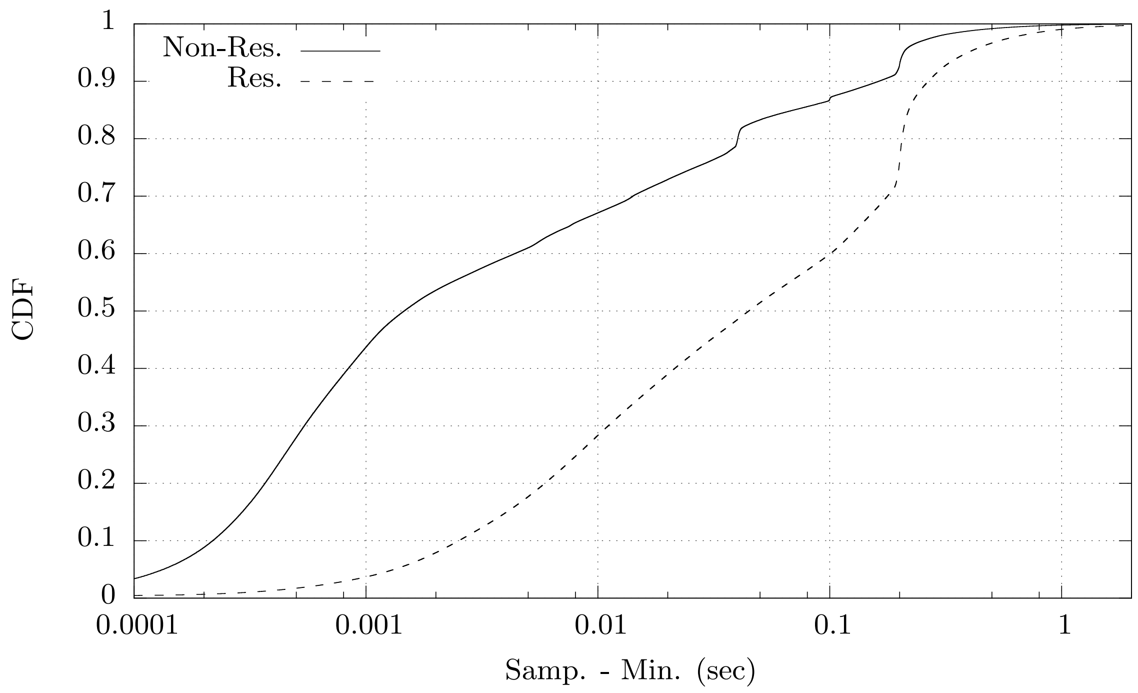

Figure 5 - Estimated Cumulative Probability of Buffering Latency

This plot separates residential customers "Res." from non-residential customers "Non-Res.", but it contains a random sampling of hosts on the Internet, so each curve contains a mix of different network technologies.

But this is a conservative, and somewhat flawed estimate. It's effectively a lower bound on what the buffering latency was, packet by packet. However, the measurements were collected only using packets from long duration downstream TCP sessions and their upstream ACKs. Due to the TCP ACK prioritization schemes mentioned previously, these measurements are invalid as a predictor for the impact that buffer bloat has on upstream traffic (other than TCP ACKs).

Flawed as it is, they show a median value for residential customers of about 50 milliseconds of buffering latency, and a 95th percentile of about 250 milliseconds of latency. The author surprisingly concludes that this isn't too bad. But, these values are sufficient to significantly degrade many interactive applications, and would result in multi-second delays in page load time.

To get around the upstream ACK flaw in the ICSI methodology and to isolate the test to a cable modem user, we performed a weeklong network capture on the link between a single residential cable modem and the customer's home router. Other than that, we followed the ICSI methodology.

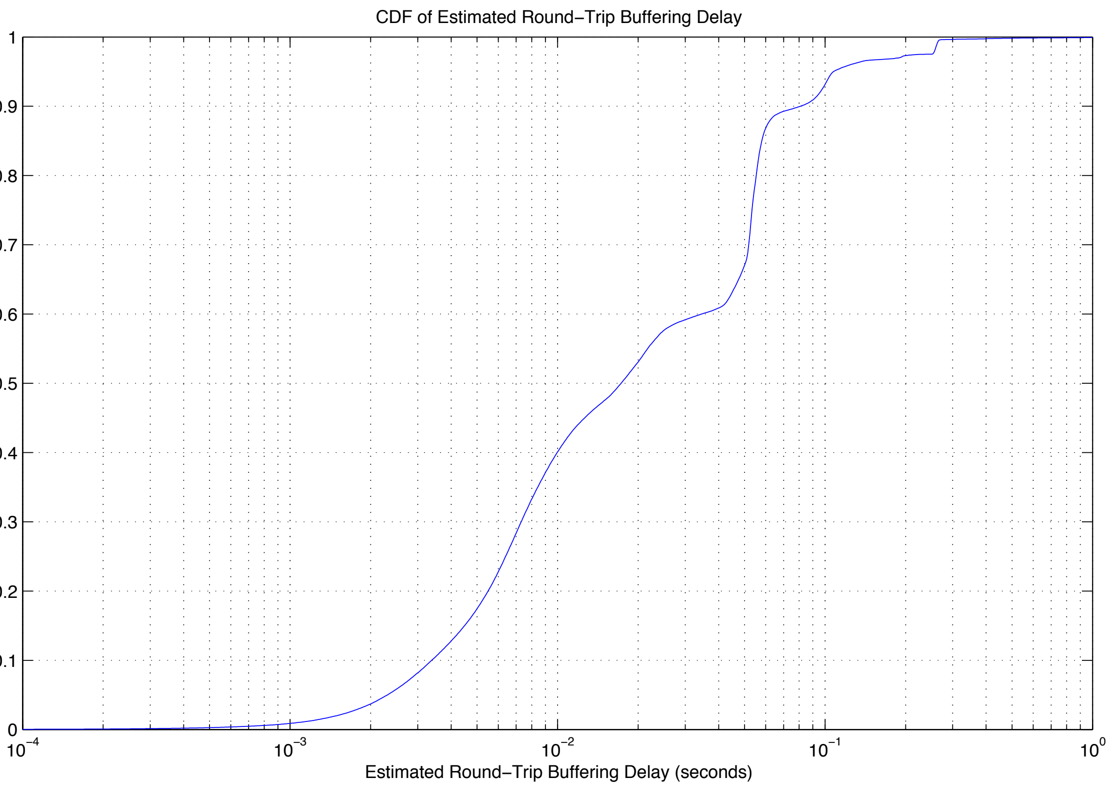

Figure 6 - CDF of Estimated Round-trip Buffering Delay

The plot shows that this customer experienced buffering latency greater than 50 ms about 25% of the time, and greater than 100 ms about 6% of the time.

1.5 Solutions?

There are several potential solutions to this problem.

One solution is to fix TCP, meaning move away from the loss based congestion control algorithm and instead use delay rather than loss. This has been a topic of research for some time, and is in fact implemented in a couple of important cases. One is the LEDBAT congestion avoidance that is used in the uTP BitTorrent protocol. The other is the Compound TCP implementation that is supported in Windows (not enabled by default). However, in many situations, when a loss-based congestion control algorithm and a delay-based algorithm share a bottleneck link, the delay-based algorithm backs off before the loss-based algorithm. In the case of BitTorrent/uTP/LEDBAT that is desirable, but for general-purpose TCP traffic it isn't. There are a lot of TCP endpoints out there that may not be updated any time soon, or may never be updated, so from a practical perspective there is a disincentive for any individual device to switch to delay-based congestion control.

Another solution is to tune the buffers. For cable modems, that means setting the upstream buffers to be a more appropriate size that minimizes the impact that a full buffer will have on interactive applications, but doesn't make it so small that it harms TCP performance. This solution is available in the DOCSIS 3.0 Buffer Control feature.

A third solution is to deploy quality of service configurations that segregate latency-sensitive traffic from bulk TCP data. This solution is available in all DOCSIS modems since DOCSIS 1.1. This solution presents several challenges. One is that it involves developing classifiers that can discriminate between latency-sensitive and non-latency-sensitive traffic. Doing so in a manner that is accurate, reliable, and that would be perceived as "network neutral" is a significant challenge. A second challenge is that rate shaping as defined in DOCSIS does not provide aggregate service limits. In other words, if the operator were to configure two service flows for each user, they would not be able to limit the total traffic rate for the combination of the two service flows. Instead they would have to separately limit each service flow. This

A fourth solution is active queue management. Active queue managers make packet drop decisions based on criteria other than "is the buffer full or not". Their goal is to send TCP the signal to slow down before the buffer is completely full. Thus, the device can support enough buffering so that it can absorb bursts of packets on the ingress, but doesn't let a standing queue build up. This has been an area of research for many years, but in the past year several new algorithms have been proposed that look very promising.

2 Active Queue Management Algorithms

We have studied the performance of three new active queue management algorithms; the first is called control delay, or CoDel; the second is a variant of CoDel, called Stochastic Flow Queue CoDel, and the third is called Proportional Integral Enhanced, or PIE.

2.1 CoDel

CoDel was first published in May of 2012 by Kathy Nichols and Van Jacobson [Nichols]. CoDel is described as a "knobless" active queue management algorithm, in that there are no parameters that need to be tuned based on the network conditions. There are, in fact, a couple of parameters built into CoDel that appear to be candidates for tuning, but the authors claim that the default settings work very well for a large class of conditions. In our experiments, we needed to modify the values in order for them to work well with the DOCSIS 3.0 MAC.

CoDel looks at the actual queuing latency of packets in the device. To do that, it timestamps each packet on ingress. Then when it de-queues the packet on the egress side, it compares the current time to the timestamp in order to calculate the queuing latency (or "sojourn time") for the packet. It then does some fairly simple decision operations based on the sojourn time. First, it compares the sojourn time to a "target" (5 ms in the original definition of CoDel). If the sojourn time is below target, then the packet is simply forwarded. If the sojourn time crosses above the target, the algorithm sets a next-drop-time equal to one "interval" in the future (100 ms in the original definition of CoDel), and the packet is forwarded. As more packets are dequeued, CoDel continues to track sojourn time, and if it ever falls below target, the next-drop-time is cleared.

However, if the next-drop-time arrives (meaning the sojourn time has been above target for one interval), CoDel drops the next packet from the head of the queue and schedules a new next-drop-time equal to interval/sqrt(2). If again the next-drop-time is reached, the third next-drop-time is scheduled at interval/sqrt(3), and the process continues with the drops becoming more closely spaced following the interval/sqrt(N) function (which equates to a linear increase in drop rate over time).

The result is that it can allow a queue to build up, and the latency through the device to be reasonably high, for short periods of time as long as at least every interval the queue is drained below the target latency. As such, the algorithm accommodates bursty traffic, without triggering packet loss. However, if a standing queue builds up (lasting for an interval or more), that's when CoDel enters the dropping state, and starts trying to send the signal to TCP. CoDel starts with a pretty low drop rate, basically one packet drop every interval, and ramps up the drop rate linearly as long as the sojourn time remains above the target value. Then, as soon as it drops below target, CoDel stops dropping packets. Using head-drop rather than tail-drop additionally results in the loss signal reaching the TCP sender sooner, which helps get the queue-depth under control quickly.

The diagram in Figure 7 illustrates the behavior of the CoDel algorithm conceptually. The actual implementation more closely follows the description above, but this illustration is intended to describe the behavior. In the figure, "local min" refers to the minimum sojourn time seen in the current interval.

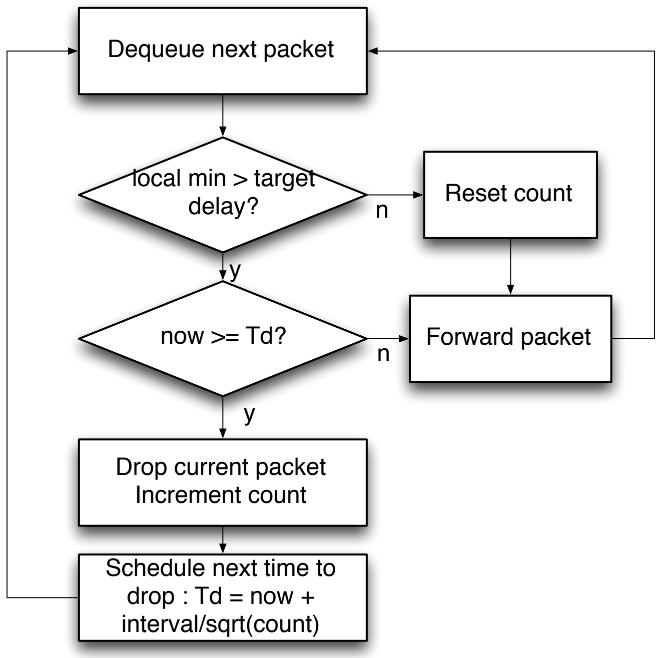
Dequeue next packet
local min > target delay?
n
Reset count
y
now >= Td?
n
Forward packet
y
Drop current packet Increment count
Schedule next time to drop : Td = now + interval/sqrt(count)

Figure 7 - Simplified CoDel Algorithm Behavior

2.1.1 CoDel in DOCSIS 3.0

To implement CoDel in the DOCSIS 3.0 modem simulator, we measured sojourn time as the total time spent in queue from ingress, until a grant arrives and the packet is able to be transmitted (or begin transmission in the case of a fragmented packet). As a result we need to consider the fact that packets will experience a MAC latency of at least one request-grant loop, even if the queue itself is small. Since CoDel uses head-drop, there actually is the possibility that a packet could experience a MAC latency of less than one request-grant loop, but this will only be as a direct result of a prior packet being dropped, making way for a newer packet to be forwarded in its place. Since we measure sojourn time such that it includes this MAC latency, we simply increased the value for target (to 10 ms in our experiments) such that a packet arriving to an empty queue would be assured (absent RF congestion) that it would have a sojourn time less than the target value.

2.2 SFQ-CoDel

SFQ-CoDel is a combination of two different queue management concepts. One component, CoDel, would really be referred to as "active queue management", and the other component is stochastic flow queuing. These two concepts were first brought together by Eric Dumazet in an implementation in the Linux kernel in July 2012, and later ported into the OpenWRT router code base by Dave Täht, where in both cases it is referred to as "fq-codel" (flow-queue CoDel). The ns2 implementation by Kathleen Nichols bears the name SFQ-CoDel[1].

Figure 8, below, is an illustration of the SFQ concept that shows packets arriving into the network device on the left, and then exiting on the right.

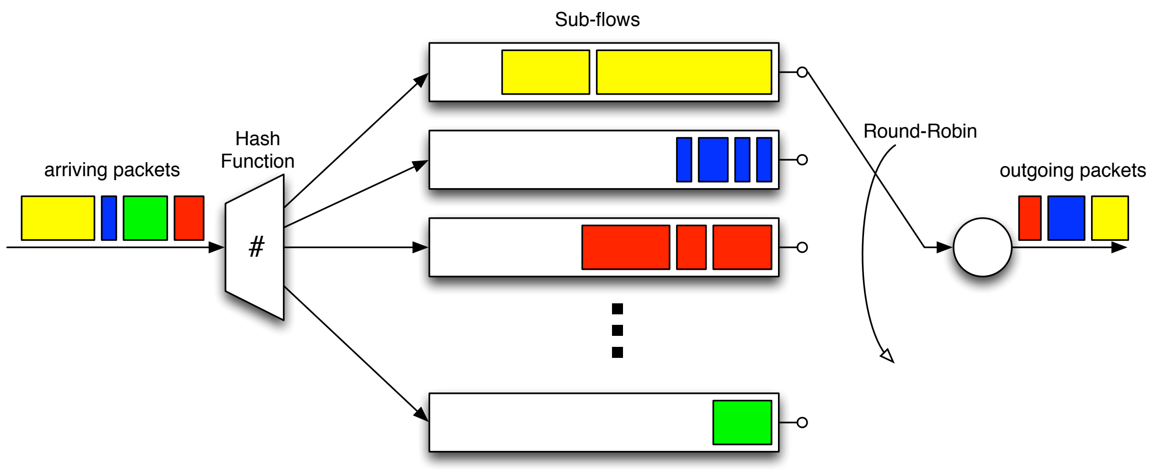

Figure 8 - Stochastic Flow Queuing

The diagram in Figure 8 bears some similarities to the DOCSIS quality of service functionality. There is a set of different "flows" where incoming packets get queued, and a function that looks at each incoming packet and maps it into one of the different flows. The flows then get multiplexed together on the wire on the upstream. But, that's where the similarity ends. In the context of a DOCSIS modem, this SFQ functionality would all be happening within a single DOCSIS Service Flow. The separation of traffic is not done via a configured classification function. Rather, it's done via a hash function, which utilizes 3 or 5 elements in the packet header: source IP, destination IP, protocol type, and then source port and destination port for UDP and TCP packets. Effectively, all the packets for a given connection between a particular source and destination will get hashed into the same sub-flow, and traffic for other connections will get hashed into other sub-flows.

However, there is a fixed number of sub-flows, so there is the potential for hash collisions, where two different flows get mapped to the same sub-flow. But if the number of sub-flows is greater than the number of simultaneous connections going through the device, there is a good chance that the different connections will be separated onto different sub-flows. From this perspective, having more sub-flow queues provides an advantage in that it gives a lower probability of hash-collisions. Since the sub-flows share a single pool of buffering memory, the only memory overhead that is dependent on the number of sub-flows is a small amount of per-sub-flow state.

Servicing these sub-flow queues on the egress side is done via a round-robin mechanism (either packet-based or byte-based) that aims to give each "active" sub-flow an equal share of the link bandwidth (either from a packet-per-second or bit-per-second perspective).

The result of this arrangement is that, when there are multiple concurrent connections, the bulk TCP connections get hashed into queues that are separate from the latency-sensitive connections. So, even if

[1] In initial implementations of SFQ-CoDel there were some differences between it and fq-codel. The most significant differences were the following: 1) The fq-codel implementation utilized a byte-based "deficit round-robin" approach, while SFQ-CoDel used a packet-based round-robin, 2) fq-codel included a capability to discard packets from the "largest" sub-flow upon exhaustion of queue memory. CableLabs has worked with Kathleen Nichols to extend SFQ-CoDel so that it now supports (2), and can be configured to operate either with byte-based or packet-based round robin.

each TCP connection creates a buffer backlog, its impact on the latency-sensitive connections is almost nil. So, a periodic or occasional upstream packet for a VoIP call or a gaming session, or an HTTP GET from a web page download gets hashed into a different sub-flow from the TCP connection. It goes effectively to the head of its queue, and then when the round-robin comes around, it gets serviced pretty quickly.

SFQ-CoDel runs the CoDel active queue manager on top of the SFQ structure. The result is that the bulk TCP sessions then don't maintain a standing queue. This minimizes the impact of hash-collisions.

2.3 PIE

The third algorithm we've studied is called PIE, proportional integral enhanced [Pan]. This is being developed by Cisco and was first reported on in the October 2012 IETF meeting. It's based on the theory of linear feedback control.

The concept is based on some earlier work [Misra] that modeled TCP congestion control behavior as a linear system. It showed that in certain, fairly specific, situations, long-term TCP sessions can respond to packet loss probability in a way that is similar to a linear-dynamic system.

Under that assumption, PIE attempts to control the queue depth via linear-feedback control, it monitors the queue depth, and adjusts drop-probability via linear control-theory mechanisms.

However, in general, most real world traffic isn't a set of long-term bulk TCP sessions. It's made up of different sessions coming and going, some only sending a few packets, some sending thousands. Some being rate controlled via the TCP congestion control algorithm, some being limited by the application. For the fraction of the sessions that are in TCP congestion control, they are all in different states (slow-start, congestion avoidance, etc.) and there are different TCP algorithms as well (NewReno, cubic, etc.).

The PIE approach addresses this by establishing three operating points for its linear controller, and switching between them in an attempt to select the mode that results in the best behavior.

The core of the algorithm is that is develops a prediction of the queuing latency for the packet that is currently at the tail of the queue, based on the current queue depth and an estimate of the egress data rate. It calculates a latency "error" - the difference between the estimated latency and a reference value, e.g. 5 or 10 milliseconds. It then sets the drop probability as a weighted sum of the latency error and the running sum of the latency error over time. So, if the latency is significantly greater than (e.g.) 5 ms, and/or has been greater than 5 ms for some time, then PIE will set a fairly high drop-probability, and if it's below the reference value and/or has been below for some time, then it will set a low (or zero) drop-probability. Additionally, rather than performing head-drop, PIE performs tail-drop.

The PIE algorithm may have some advantages from an implementation perspective. First, it does not require timestamping of ingress packets, and second, tail-drop is generally considered simpler than head-drop. Additionally, the computations involved in estimating the egress link rate, and in updating the drop probability are performed periodically, e.g., every 30 ms, so that CPU demands can be moderated (at the expense of performance) if need be.

3 Simulation Model

3.1 DOCSIS Model Updates

The ns-2 simulation model of a DOCSIS 3.0 cable modem and CMTS used in this study is an updated version of the model described in [White]. The changes are as follows:

- The rate shaping queue (RsQ) and transmit queue (TxQ) have been combined into a single queue that implements both rate shaping (per the DOCSIS 3.0 specification) and the Request-Grant DOCSIS 3.0 MAC. While the previous, dual-queue, approach is a potential implementation for DOCSIS modems, we believe that the single-queue is more realistic, and when AQM is used, the single-queue approach provides significantly better performance than dual-queue.
- The DropTail implementations of the cable modem queue now limit the queue depth in bytes rather than in packets. This is a more accurate model of the way that the DOCSIS 3.0 Buffer Control feature is defined.
- The CMTS provides the CM a grant that is located randomly in the MAP interval (via a uniform distribution), rather than always occurring at the beginning of the MAP interval. We feel that this is a more realistic model for grant distribution.

3.2 Queue Manager Configurations

We utilize the following configurations for the queue managers under study:

- BufferBloat - DropTail queue with a buffer size set to 625000 bytes (equivalent to 1000 ms at the Max Sustained Rate of 5 Mbps, and 250 ms at the Peak Rate of 20 Mbps). This is intended to model current modems as deployed in the field today. As noted in section 2.4, there is some evidence that buffering latencies for current DOCSIS 3.0 modems may be on the order of 240-480 ms rather than 1000.
- Buffer Control - DropTail queue with a buffer size set to 31250 bytes (equivalent to 50 ms at the Max Sustained Rate, and 12.5 ms at the Peak Rate). This is intended to model a current DOCSIS 3.0 modem with the buffer control feature enabled, and configured to a target buffering latency of 50 ms.
- CoDel - "target" = 10 ms, "interval" = 150 ms, buffer size 625000 bytes. In this case, the buffer size was kept the same as the BufferBloat case in order to examine the ability of CoDel to manage the large buffer of an existing D3.0 modem.
- SFQ-CoDel - 32 sub-flows, byte-based deficit round-robin with 300 byte quantum, buffer size 625000 bytes, "target" = 10 ms, "interval" = 150 ms.
- PIE - delay reference = 5 ms, buffer size 300000 bytes, queue-depth measured in bytes, a=0.25, b=1.25, update interval 15 ms.

3.3 Congestion Scenarios

As a result of combining the RsQ and TxQ, it is now meaningful to compare the relative performance of the buffer management schemes in the presence of upstream RF congestion. So this study includes that variable. In the previous implementation, RF congestion would only impact the egress rate of the TxQ, which was always DropTail. Since the buffer management algorithms under study operated only on the

RsQ, the result was that there was very little difference in performance between the studied algorithms during simulations of congested upstreams.

We chose three different RF congestion scenarios.

1. No RF congestion from other DOCSIS modems. In this case, the test modem can achieve its 5 Mbps upstream rate limit and get the full 20 Mbps power boost.
2. Light RF congestion, where the channel capacity varies across 2.5 Mbps, 5 Mbps, 12.5 Mbps, 20 Mbps (average capacity 10 Mbps).
3. Moderate RF congestion, where the channel capacity varies across 1.7 Mbps, 3.3 Mbps, 5 Mbps, 12.5 Mbps (average capacity 5.625 Mbps).

In the light-congestion and moderate-congestion cases, the amount of available capacity in the channel varies over time. In both cases, the channel cycles through a set of four different available capacity numbers in a pattern that repeats every 120s as is shown in Figure 9 and Figure 10 below. This particular pattern exhaustively covers all 12 possible data rate transitions.

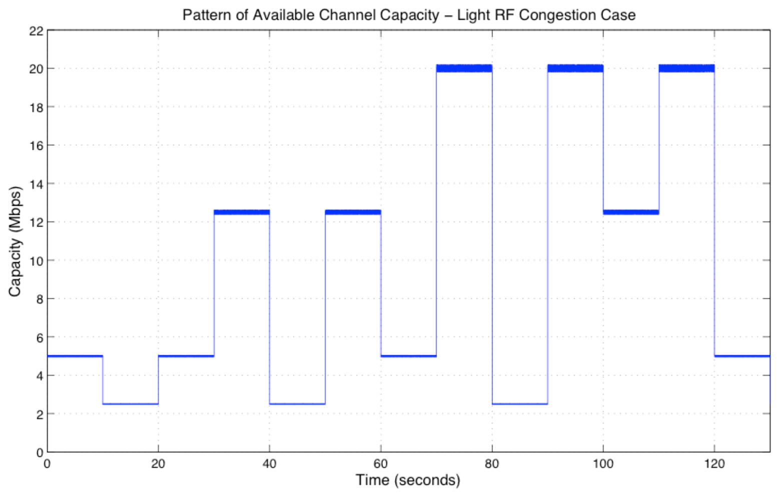

Figure 9 - Light RF Congestion

In the light-congestion case, the average capacity was greater than the user's provisioned data rate of 5 Mbps. Only occasionally, were they limited to a rate below their provisioned rate. Free capacity on the upstream channel varies across 2.5 Mbps, 5 Mbps, 12.5 Mbps, 20 Mbps (average capacity 10 Mbps)

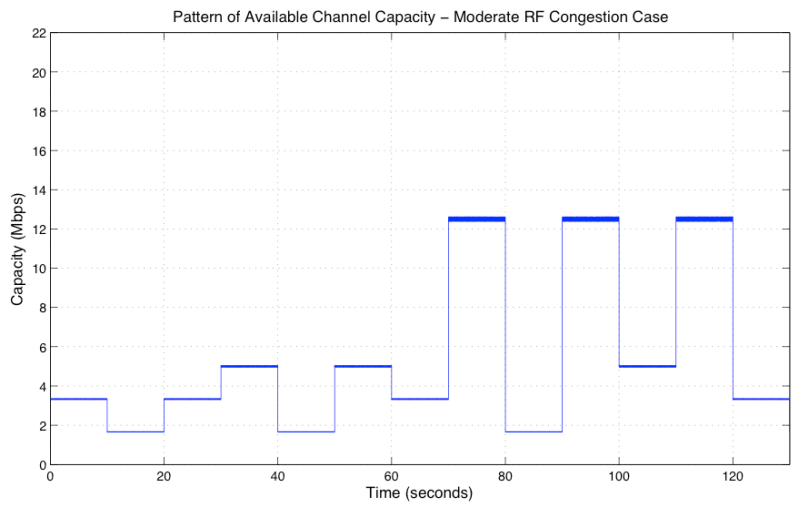

Figure 10 - Moderate RF Congestion

In the moderate-congestion case, the average capacity is just slightly above that provisioned rate, and for half the time was below that 5 Mbps rate. Free capacity on the upstream channel varies across 1.7 Mbps, 3.3 Mbps, 5 Mbps, 12.5 Mbps (average capacity 5.625 Mbps).

3.4 DOCSIS Service Configuration

As in [White], the cable modem is configured with a single best effort upstream service flow with the following parameters:

- Maximum Sustained Traffic Rate: 5 Mbps
- Maximum Traffic Burst: 10 MB
- Peak Traffic Rate: 20 Mbps

And a single downstream service flow with the following parameters:

- Maximum Sustained Traffic Rate: 20 Mbps
- Maximum Traffic Burst: 10 MB
- Peak Traffic Rate: 50 Mbps

3.5 Topology Updates

As a result of the changes in the cable modem model, the topology was updated as shown in Figure 13:

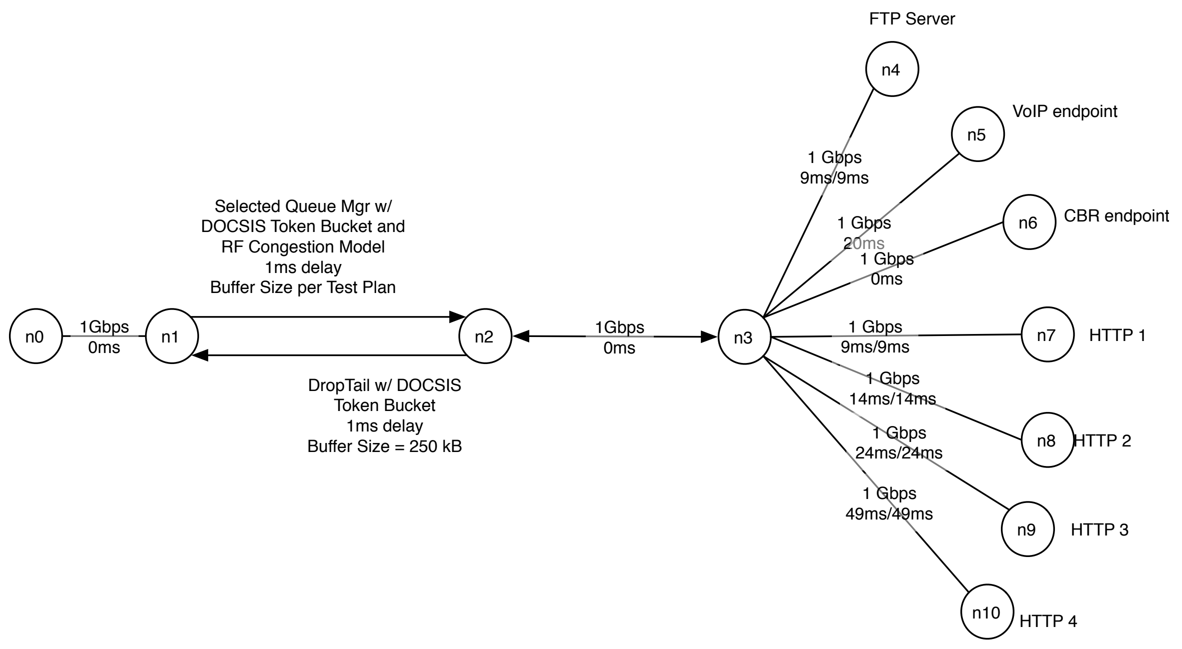

Figure 11 - Simulator Topology

In this topology, n0 represents the client endpoint (or router), n1 represents the CM, and n2 represents the CMTS. The remaining nodes represent servers and other network elements.

3.6 Traffic Model Updates

Updates have been made to the traffic models relative to what was described in [White]. In particular: there have been two significant changes to the web browsing model, we've added a model of BitTorrent, and we utilize the VoIP traffic both as a model of a VoIP session and as a model for an online "twitch" game.

The changes to the web model include the use of a log-normally distributed set of object sizes as was described in Appendix A of [White]. This provides a more realistic model of a web page download. Additionally, the web model was updated to allow testing using multiple simultaneous clients. Notably, our web model does not include the DNS lookups which would be present in many real-world page loads. While the number of DNS lookup packets is extremely small relative to the total number of packets involved in loading a page, they are very sensitive to packet loss. The retry timeout for DNS clients is commonly 5 seconds, so a single DNS packet loss would increase the page load time by approximately that amount.

The BitTorrent model uses the linux tcp-ledbat implementation from [Rossi] and described in [LEDBAT]. For our model of a BitTorrent uploader, we chose to configure 32 simultaneous upload connections and set the size of each upload "piece" at 256,000 bytes.

The VoIP traffic is unchanged from [White] (G.711 voip model, 87.2 kbps, 218 byte UDP packets at 50 pps). In addition to estimating the VoIP MOS as described in [White], we also report latency and loss statistics so that this traffic can be viewed as representing an online twitch game, and an understanding of the impact to gaming applications can be reached.

3.7 Traffic Scenarios

We tested 17 different user traffic scenarios that comprise different mixes of the four applications:

- VoIP/Gaming
- Web browsing
- File upload (either FTP or BitTorrent)
- Constant bit rate UDP traffic

The 17 scenarios lean toward more active use of the upstream connection than would be considered "average", and this is by design. Cable modems in residential service may have fairly low average upstream utilization, but it is generally marked by sporadic periods of activity interspersed among idle periods. It is during the periods of activity where buffer management has an impact on performance. It is also during these periods that the user may be most likely to be aware of any impact on user experience.

The 17 scenarios are grouped into three groups: light, moderate and heavy traffic. The light traffic group consists of seven traffic scenarios, the moderate traffic group contains 4 scenarios, and the heavy traffic group contains 6 scenarios. These scenarios are detailed in Table 2 below.

In Table 2, "VoIPs" indicates the number of simultaneous 87.2kbps UDP streams (representing both a VoIP application and an online game), "Webs" indicates the number of simultaneous web users (repeated downloads of a 700 kB page as described in Appendix A of [White]), "CBR" indicates presence or absence of a single 1 Mbps UDP flow, "FTPs" indicates the number of simultaneous bulk TCP uploads, and are detailed in the table.

Table 2 - Traffic Scenarios

Test Case	VoIPs	Webs	CBR traffic (Mbps)	File Upload Traffic				Traffic/Load "Group"
				FTPs	RTT (ms)	FTP detail	TCP	
x01	1	1	0	0	n/a	n/a	n/a	L
x02	1	1	0	1	20	continuous file tx	linux-cubic	L
x03	1	1	0	1	100	continuous file tx	linux-cubic	L
x04	1	1	0	5	20	repeating 5MB file tx	linux-cubic	M
x05	1	1	0	5	100	repeating 5MB file tx	linux-cubic	M
x06	1	1	1	5	20	repeating 5MB file tx	linux-cubic	M
x07	1	1	1	5	100	repeating 5MB file tx	linux-cubic	M
x08	1	1	0	1	20	bursty 18.75MB file tx	tcp-reno	L
x09	1	1	0	1	100	bursty 18.75MB file tx	tcp-reno	L
x10	4	4	0	1	20	continuous file tx	linux-cubic	L
x11	4	4	0	1	100	continuous file tx	linux-cubic	L
x12	4	4	0	5	20	repeating 5MB file tx	linux-cubic	H
x13	4	4	0	5	100	repeating 5MB file tx	linux-cubic	H
x14	4	4	0	10	20	repeating 250KB file tx	linux-cubic	H
x15	4	4	0	10	100	repeating 250KB file tx	linux-cubic	H
x16	4	1	0	32	20	repeating 250KB file tx	ledbat	H
x17	4	1	0	32	100	repeating 250KB file tx	ledbat	H

Note that the traffic grouping (L,M,H) is based on the number of simultaneous file transfer sessions and the number of simultaneous web users. Cases where there is 0 or 1 file transfer form the "light" traffic group. Cases where there 5 are file transfers and only a single web user form the "moderate" traffic group. The remaining cases (5 file transfers and multiple web users; or >5 file transfers) form the "heavy" traffic group. As traffic increases from light to heavy, it is expected that the VoIP/Gaming traffic will experience decreasing user experience. The same may not be precisely true for the web page load time, since in some of the "heavy" traffic cases the web traffic constitutes a significant portion of that traffic load.

3.8 Application Metrics

As noted above, different applications are impacted differently by network transport behaviors. As a result, we look at different metrics to gauge the user experience for each application.

For VoIP applications, we utilize the same Mean Opinion Score (MOS) estimator as described in [White], for gaming applications we track statistics on packet latency and loss, for web browsing we track page load time statistics, and for file uploads we track long term average good-put.

Additionally, to better understand short timescale TCP performance, we initiate single file transfers, and monitor throughput and buffering latency over time (averaging on 100ms intervals).

4 Simulation Results

The simulation results presented here are the result of simulating 2 hours of user activity for each data point. In all of the sections below aside from section 5.4, the simulated scenarios are those described in Section 4.7. Section 5.4 instead uses a single isolated TCP session.

4.1 Gaming Traffic

4.1.1 Gaming Packet Latency

Figure 12 shows a detailed view of the impact that the various queue management techniques can have on gaming traffic packet latency across the various test conditions. This figure breaks out the test conditions as a 3x3 grid of plots, where the rows represent the different RF congestion levels, and the columns represent the different traffic load groups.

Figure 13 provides a "one-level-up" summary of the results, where each plot represents a row-wise or column-wise combination of three of the plots from Figure 12. The first column of plots summarizes the three different RF congestion levels (weighting each traffic load group equally), whereas the second column summarizes the three different traffic load groups (weighting each RF traffic level equally).

Finally, Figure 14 provides an overall summary of the results (weighting each of the nine conditions from Figure 12 equally).

All of these plots show the buffering latency for the VoIP/Gaming traffic, including any MAC access time. The latency values used for generating these plots are measured using a process that rounds each measurement up to the nearest integer millisecond.

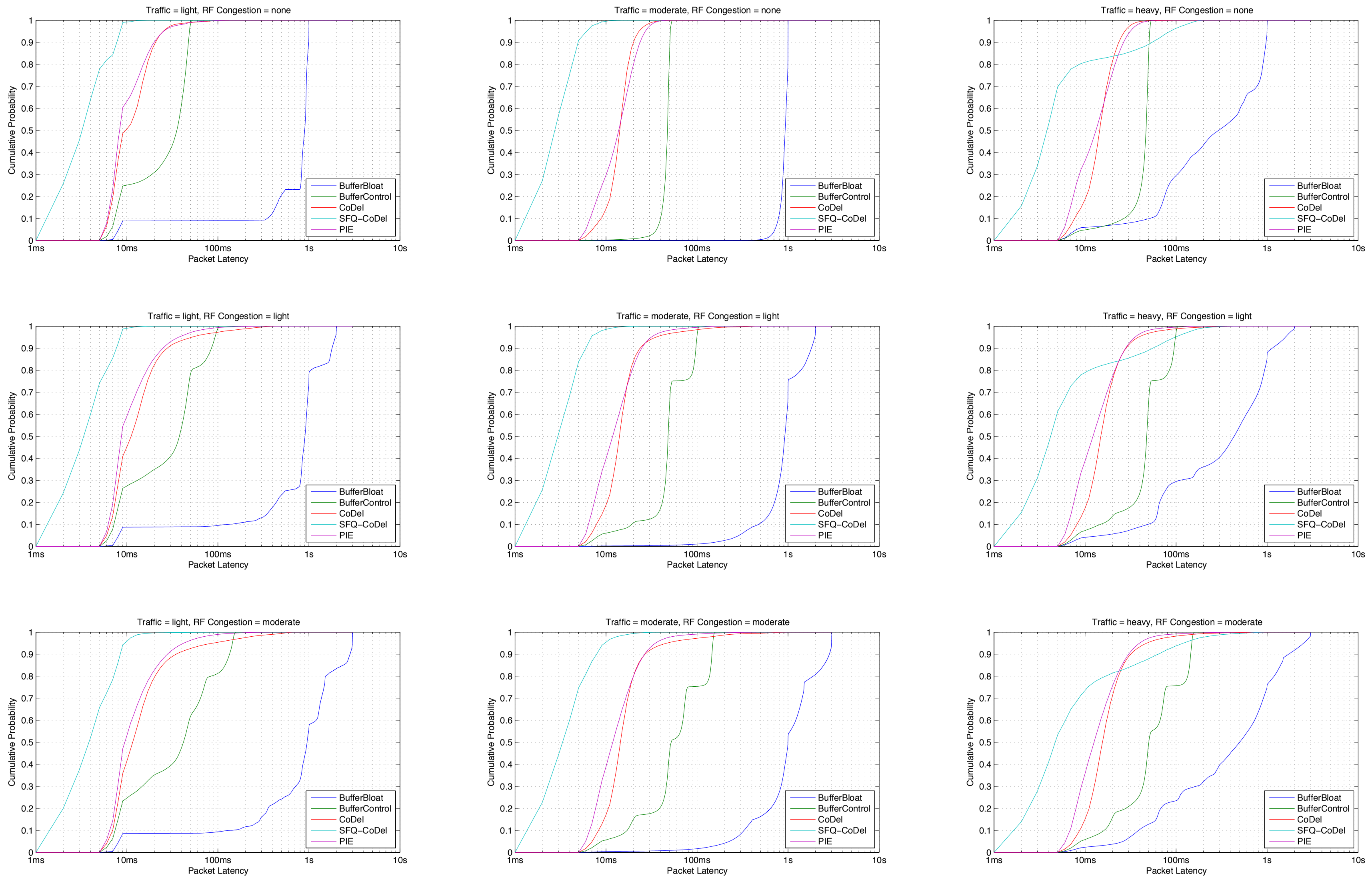

Figure 12 - Gaming Traffic Latency Detail

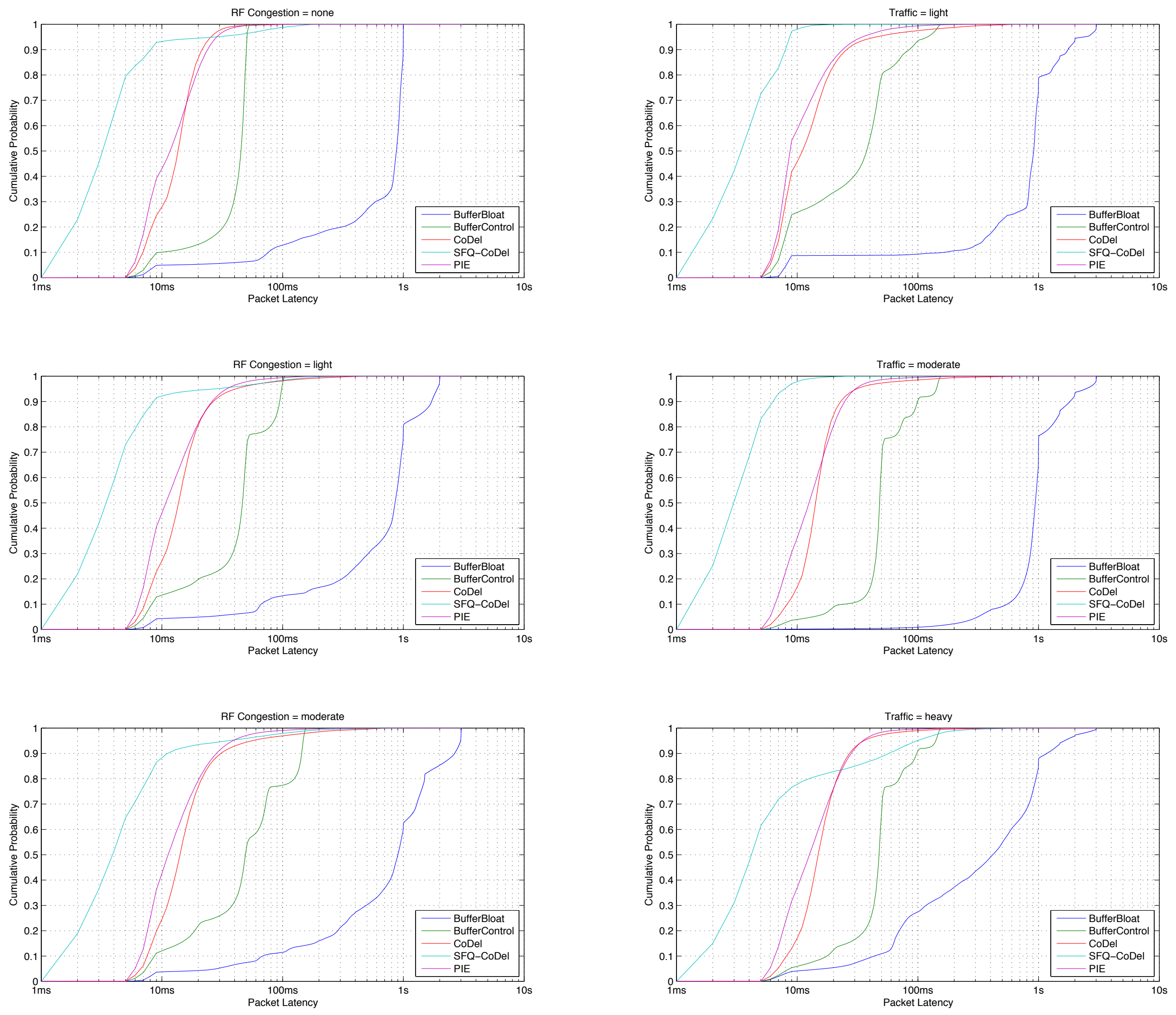

Figure 13 - Gaming Traffic Latency vs. RF Congestion and vs. Traffic Load

In this view, we see a few notable characteristics.

One notable characteristic is that the two DropTail queue managers (BufferBloat and BufferControl) experience a worst-case latency that is equivalent to their configured target buffer depth when there is no RF congestion. However, when RF congestion is present, the buffering latency increases beyond the targeted limit. This is due to the fact that the effective data rate that the modem is able to achieve during congestion is less than the provisioned service rate, and that the buffer size is set in terms of bytes (and doesn't adjust based on congestion). In the light congestion case we see a worst-case latency that is double the target value, and in the moderate congestion case we see triple the target value, exactly as would be predicted based on the RF congestion configuration. On the other hand, the three active queue management algorithms are much less affected by the RF congestion scenarios. This is a result of the fact that they are each utilize target latency in a way that does not make any a priori assumptions about the egress data rate. In the case of CoDel and SFQ-CoDel, the latency is measured directly. In the case of PIE, latency is predicted based on recent history of egress data rate. In these AQM cases, there will be episodes of increased latency that result immediately after the egress rate drops, but the AQM then recovers and returns the packet latency to the desired amount.

Another notable characteristic is that the DropTail queue managers show buffering latency that is heavily weighted toward the maximum value, which corroborates the theory that TCP is keeping the buffer full. However, in the heavy traffic scenario, the BufferBloat curve doesn't exhibit the same behavior. This is a result of the fact that in these scenarios, much of the bulk file-transfer traffic uses LEDBAT, which will back off when it senses buffering delays on the order of 100ms.

We also note that all of the "single queue" approaches (i.e., everything except SFQ-CoDel) show a minimum latency of 5-6 ms (shown as 6 ms due to rounding-up), whereas SFQ-CoDel can provide latencies as low as 1-2 ms (shown as 2 ms). The 5-6 ms best case for the single queue approach results from the DOCSIS 3.0 Request-Grant process, where a packet arriving to an empty queue will trigger the CM to send a Request message during the first MAP interval, and then transmit during a grant occurring in the third MAP interval. In the SFQ-CoDel case, our implementation does not dequeue packets, or even iterate the round-robin algorithm, until the grant arrives. As a result, approximately 80% of the time, the VoIP/Gaming packet sees less than 5ms of buffering latency! This phenomenon is caused by a request being triggered by a packet for another flow (bulk file transfer or web traffic), and then the VoIP/Gaming traffic being selected to utilize the resulting grant (thereby delaying the bulk file transfer or web traffic slightly). This was an unexpected benefit of the SFQ approach.

Finally, we note that while SFQ-CoDel outperforms all other approaches in the majority of cases, the heavy traffic scenarios have a crossover point where the SFQ-CoDel approach actually performs worse than CoDel (or PIE). This appears to be the result of competition with a large number of other concurrent sessions. In these scenarios, there are anywhere from 60 to 110 simultaneous sessions operating, of which 4 are VoIP/Gaming traffic. In the FTP cases the majority of the sessions are web upstream traffic (GETs and ACKs) which are small packets, but in the BitTorrent cases over half of the sessions are large packet flows. The most significant degradation in performance occurred in the BitTorrent cases. Given that our SFQ implementation is configured for 32 sub-flows, it is unlikely that all of the VoIP/Gaming traffic flows get hashed into their own, dedicated SFQ sub-flows, and some of them undoubtedly get hashed into a sub-flow with upstream bulk traffic. It is in these cases that the SFQ round robin seems to work against us. Unfortunately, in our testing, increasing the number of SFQ sub-flows did not improve performance. This is due to the fact that each upstream VoIP/Gaming flow was 87.2 kbps, or about 1/57 of the full upstream rate of 5 Mbps. Since the SFQ round robin seeks to give each active sub-flow an equal share of the link bandwidth, more than 57 active sub-flows would result in the VoIP/Gaming flows being starved for bandwidth. We tested with different power-of-two counts for sub-flows (e.g. 8, 16, 32, 64, etc.) since this restriction provides some implementation benefits in the hash calculation, and found that 32 sub-flows provided the best performance. Notably, 32 is the highest power-of-two that is less than 57, but we have not explored non-power-of-two sub-flow counts to determine whether better performance would be achieved with sub-flow counts between 32 and 57.

Further work may uncover a solution for some of these cases. For example, there are some reports that BitTorrent clients mark their upstream traffic with the CS1 DiffServ code point (denoting "background" or "scavenger" traffic). If this is widespread, it could be utilized to map all such traffic into a single dedicated sub-flow (perhaps one that is treated as lower priority).

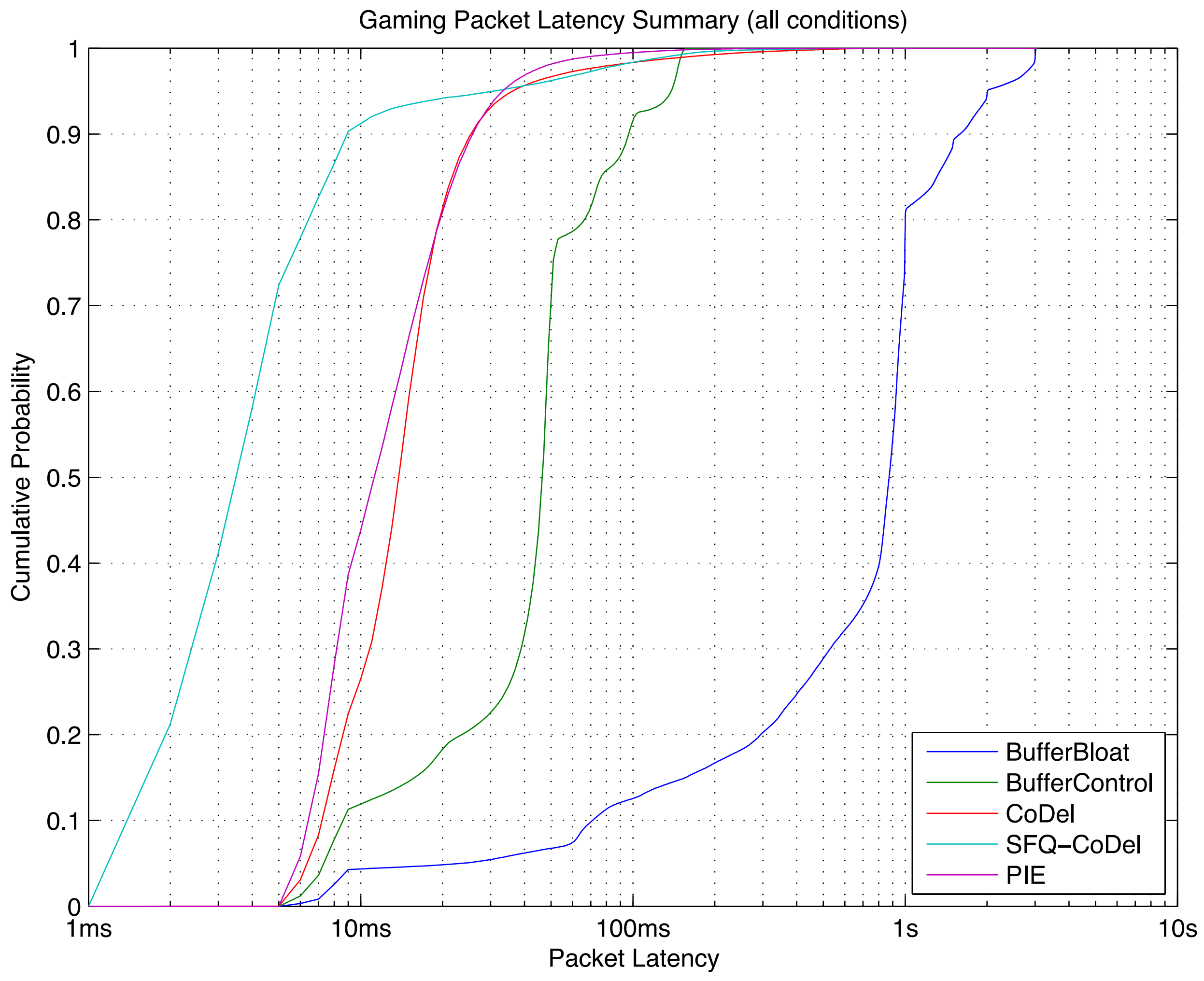

Figure 14 - Gaming Traffic Latency Summary

4.1.2 Gaming Packet Loss

The other side of the user experience metric for gaming traffic that we looked at is packet loss. The next 3 figures show the packet-loss performance of the different queue management algorithms across the different scenarios. The PIE algorithm and the simple DropTail queues show fairly low loss across the test conditions. The CoDel algorithm returns moderate loss levels across the different scenarios. Finally, the sfqCoDel algorithm resulted in almost zero packet loss (rates less than 0.01%) in the light and moderate traffic scenarios, but had significant packet loss in some of the heavy traffic cases.

In the cases with a total of 110 simultaneous flows: 4 VoIP/Gaming flows, 4 web clients (each opening 24 connections), and 10 FTPs, sfqCoDel resulted in approximately 5% packet loss. In the BitTorrent cases however, sfqCoDel resulted in packet loss rates between 9% and 34%. This is a result of the same phenomenon that impacted the latency performance.

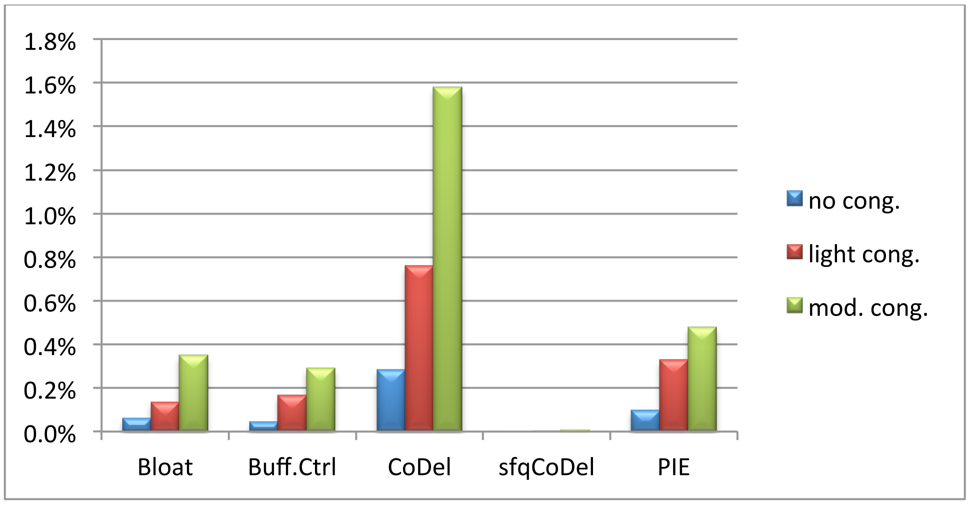

Figure 15 - Gaming Packet Loss - Light Traffic Scenarios

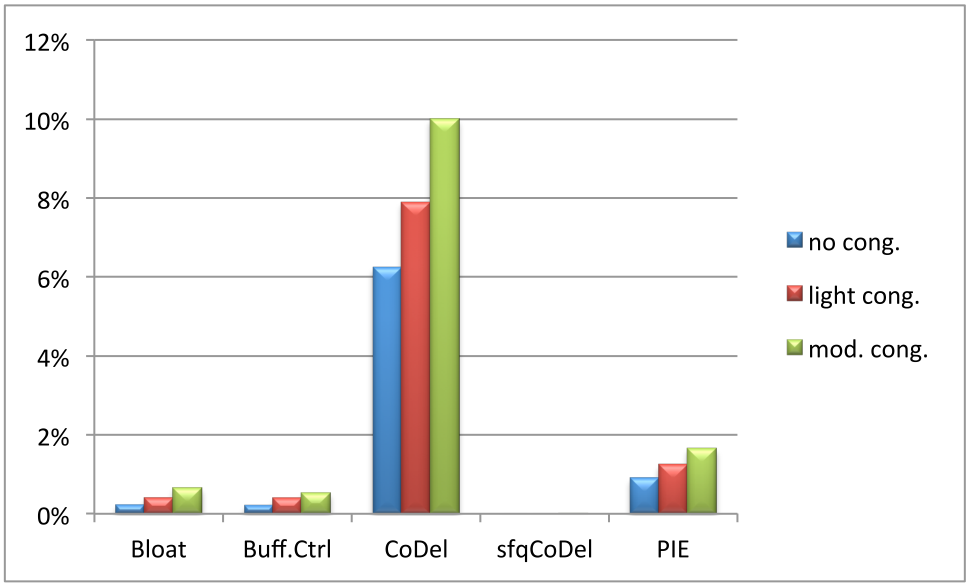

Figure 16 - Gaming Packet Loss - Moderate Traffic Scenarios

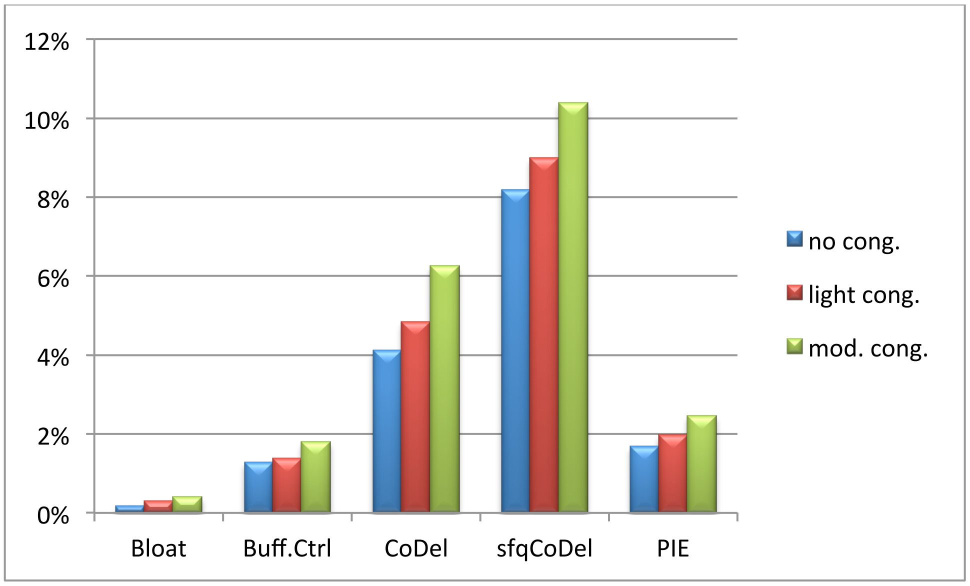

Figure 17 - Gaming Packet Loss - Heavy Traffic Scenarios

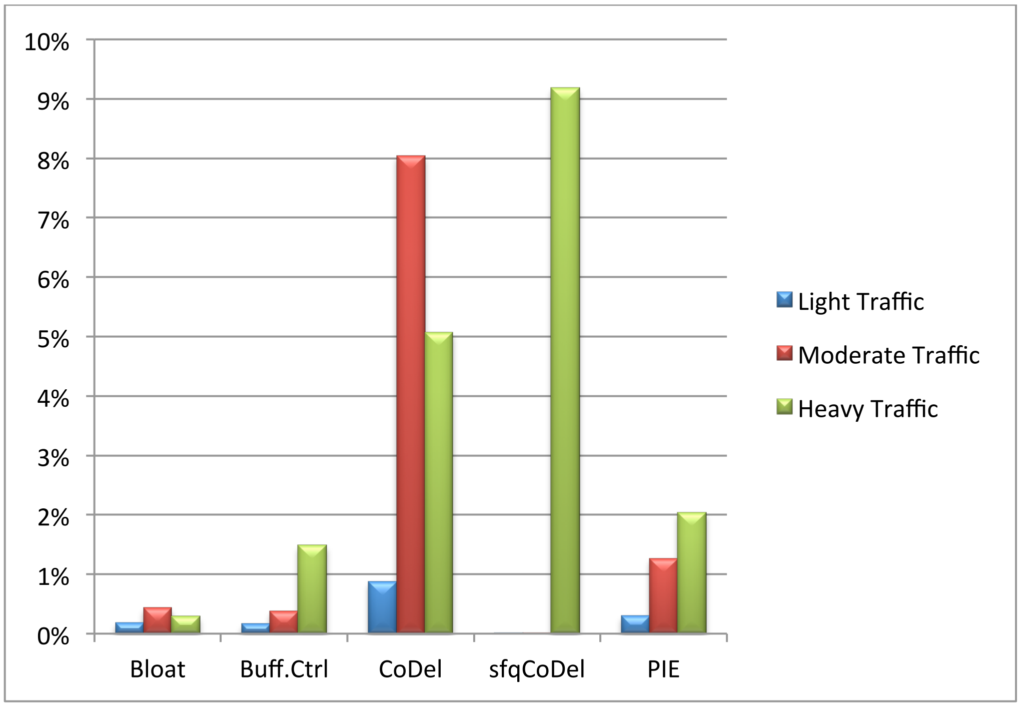

Figure 18 - Gaming Packet Loss - All Scenarios

4.1.3 Sensitivity of Gaming Applications to Packet Latency and Loss

In assessing the potential impact to user experience for gaming applications, we have looked at two metrics: latency and loss, with the result being that some queue managers did well against one metric, and not as well against the other. This is summarized qualitatively in Table 3 below.

Table 3 - Qualitative Summary of Gaming Performance

QUEUE MANAGER	LATENCY PERFORMANCE	LOSS PERFORMANCE
BufferBloat	Extremely poor	Very good
Buffer Control	Poor	Very good
CoDel	Good	Good
SFQ-CoDel	Very good in most conditions Good in BitTorrent cases	Very good in most conditions Poor in BitTorrent cases
PIE	Good	Very Good

This begs the question, what is the relative sensitivity of online games to packet loss versus latency? Latency appears to have a bigger effect than packet loss, but it almost certainly depends on the game.

Two game types that are likely to be very sensitive to network impairment are first-person-shooter (FPS) games and cloud-games. In FPS games, the game environment is typically hosted on a server, and each client receives periodic game-state updates from the server, renders the player's immediate surroundings, and sends player-state updates back to the server. While the player's surroundings and motion within those surroundings is handled locally on the player's console, and so would not be directly impacted by network conditions, the interactions with other players are what determine the outcome of the game. Since performance in these games can be impacted by a player's reaction-time, it stands to reason that delays or loss in delivering game or player state could negatively impact game play. Cloud-games, in contrast, offload all of the rendering functionality to the cloud server, with the server delivering an audio/video stream to the client, and the client returning controller inputs. As a result, all user interactions with the game are subjected to the effect of network impairments.

There have been a number of studies of the impact that network impairments have on networked first-person-shooter (FPS) games. In 2004, [Beigbeder], et al., performed a study of the relative impact of packet loss and latency for one FPS game, "Unreal Tournament 2003".

> *Through numerous user studies, we find that packet loss has no measurable affect on player performance in any user interaction category. Moreover, users rarely even notice packet losses even as high as 5% during a typical network game. Latency has no measurable effect on simple, straight- line or more complex movements. Shooting, however, is greatly affected by latency with even modest (75-100 ms) amounts of latency, decreasing accuracy and number of kills by up to 50% over a common Internet latency ranges. While combinations of movement and shooting somewhat mask the effects of latency on user performance, even unrestricted games show trends which indicate that latency degrades user performance. This is reflected in the subjective comments collected during our user studies in which loss rates went unnoticed, but latencies as low as 100 ms were noticeable and latencies around 200 ms were annoying. [Beigbeder]*

Around the same time, [Bussiere] studied two FPS games: "Halo 2" and "Quake 3", concluding:

> *While for both games the players perceived quality is not much affected by loss, it is very surprising that Halo 2 players experience no problems at high latencies whereas Quake 3 players do.*

And, in a somewhat earlier paper, [Zander] concluded that for "Quake 3":

> *Delay has a larger impact on the player's perceived quality and performance than loss (for values typical occurring in the Internet).*

Finally in 2010, [Bredel] & Fidler, also studied "Quake 3". Rather than utilizing human subjects and collecting subjective perceptions of QoE, they developed a methodology in which two identical "bots" would play repeated 1-on-1 matches under various network impairment conditions. They concluded that while both delay and loss had an impact on game performance, delay has a linear impact on scoring probabilities and rates, and is independent of the direction in which the delay occurs, whereas loss has less of an impact, but it is important which direction the loss occurred.

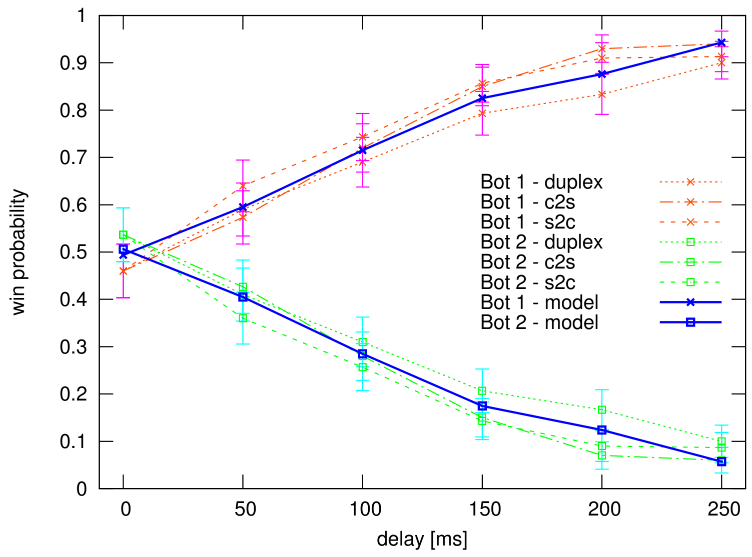

Figure 19 - Impact of Latency on Win Probability for Quake 3

Figure 19 from [Bredel] shows the relationship between delay and win probability for two identical bots playing against one another in a Quake 3 two-player game. In the experiment, Bot 2 was subjected to additional delay (either in one direction only or symmetrically) relative to the very low baseline delay experienced by Bot 1. It is clear to see the degradation in game performance for Bot 2 as delay increases.

In contrast, the impact of loss is dependent on the direction in which loss occurs, with no measurable impact resulting from loss rates of up to 20% in the server-to-client direction, and up to 50% in the client-to-server direction.

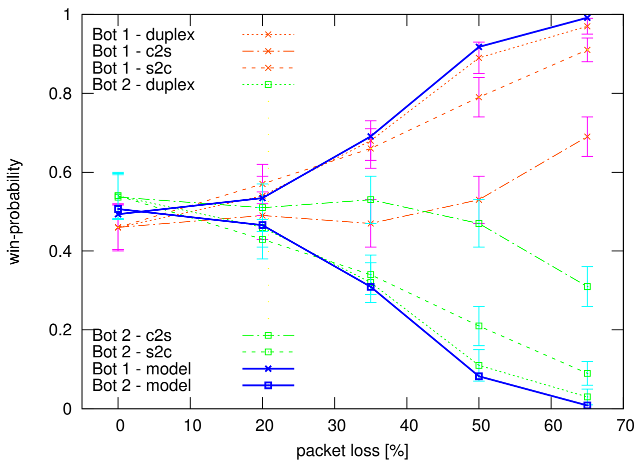

Figure 20 - Impact of Packet Loss on Win Probability for Quake 3

Additionally, while we did not explicitly examine jitter in our simulations, [Bredel] concludes that jitter has less influence on scoring probabilities and rates for Quake 3. It is also notable that there does not appear to be a trend in the data such that increasing jitter decreases game performance. Instead it appears that the presence of any amount of jitter causes a more-or-less fixed impairment.

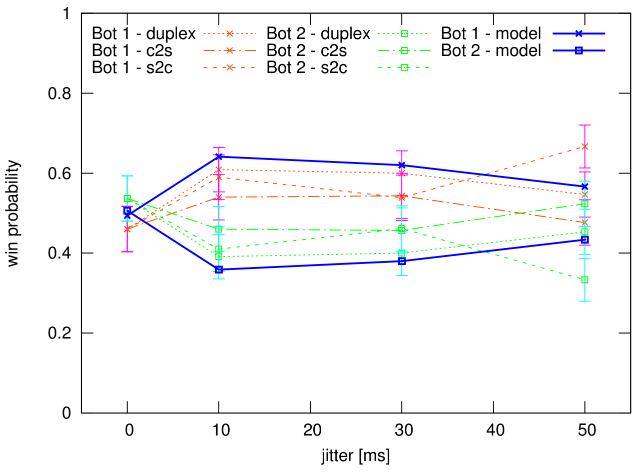

Figure 21 - Impact of Jitter on Win Probability for Quake 3

While these studies appear to have consistent conclusions for networked FPS games, they have only studied a few selected game titles, so it is possible that other games are more sensitive to packet loss (or to jitter).

Nonetheless, if we use the more conservative 20% loss threshold described by [Bredel], we find that BufferBloat, Buffer Control, and CoDel achieve loss rates less than the threshold in all tested conditions, while sfqCoDel and PIE only exceed the threshold for the most severe BitTorrent traffic scenario (regardless of RF congestion). The 50% loss threshold was not exceeded in any of our test cases by any of the queue managers.

From the latency perspective, there is not a threshold below which user experience is unaffected. This, coupled with the fact that modem buffering latency is only one component of the overall round-trip time, points to keeping the upstream latency at a minimum. The active queue managers (CoDel, SFQ-CoDel or PIE) would be the most attractive approaches for ensuring satisfactory user experience for networked FPS games.

Cloud games are a more recent phenomenon and so have been studied a bit less. [Jarschel], et al., performed a subjective study of user-perceived quality of experience for three different types of single-player cloud-games (a soccer game, a 3rd person role-playing game, and a first person driving game). They found that downstream packet loss had the greatest impact on user experience with packet loss rates as low as 0.3% causing a significant degradation in user experience. The fact that downstream packet loss plays a bigger role in determining user experience in cloud-games is understandable, since downstream loss will result in distracting visual artifacts that would not be present in console-based networked games.

This study did not examine upstream and downstream packet loss independently with as much detail as was the case with Bredel & Fidler's study of Quake 3, so it is difficult to draw specific conclusions about the sensitivity to upstream loss. However, the one test in which upstream packet loss is present and downstream packet loss is absent was performed with 1% packet loss and 120 ms RTT, and returned an average MOS of 3.63, as compared to an average MOS (interpolating between test points) of 3.88 for 0% packet loss and 120 ms RTT. So, the addition of 1% upstream packet loss resulted in an estimated 0.25 decrease in MOS, which would be a noticeable impact.

It is similarly difficult to infer sufficient details on the impact that latency has on cloud-game MOS, but the data provided seems to show a fairly linear decline in MOS as RTT increases (similar to what was seen in [Bredel] for Quake 3). This result, and some similar studies, led Choy, et al. [Choy], to suggest that 80 ms should be the round-trip budget for the network between the client and the server. Based on this, we suggest that upstream buffering latency be kept below 20 ms for satisfactory cloud-game performance.

The only queue manager that can achieve both of these stringent latency and loss targets is SFQ-CoDel, and even then only in moderate to light traffic loads.

4.2 WEB PAGE LOAD TIME

Figure 22 shows a detailed view of the impact that the various queue management techniques can have on web page load time across the various test conditions. This figure breaks out the test conditions as a 3x3 grid of plots, where the rows represent the different RF congestion levels, and the columns represent the different traffic load groups.

Figure 23 provides a "one-level-up" summary of the results, where each plot represents a row-wise or column-wise combination of three of the plots from Figure 22. The first column of plots summarizes the three different RF congestion levels (weighting each traffic load group equally), whereas the second column summarizes the three different traffic load groups (weighting each RF traffic level equally).

Finally, Figure 24 provides an overall summary of the results (weighting each of the nine conditions from Figure 22 equally).

In each of these plots the plot legend indicates the number of page loads that are included in the CDF. Note that the moderate traffic group consisted of test scenarios where there was always a single web client, whereas some of the light traffic group scenarios and most of the heavy traffic group scenarios involved 4 simultaneous web clients. As a result the total number of page loads is lower for the moderate traffic group.

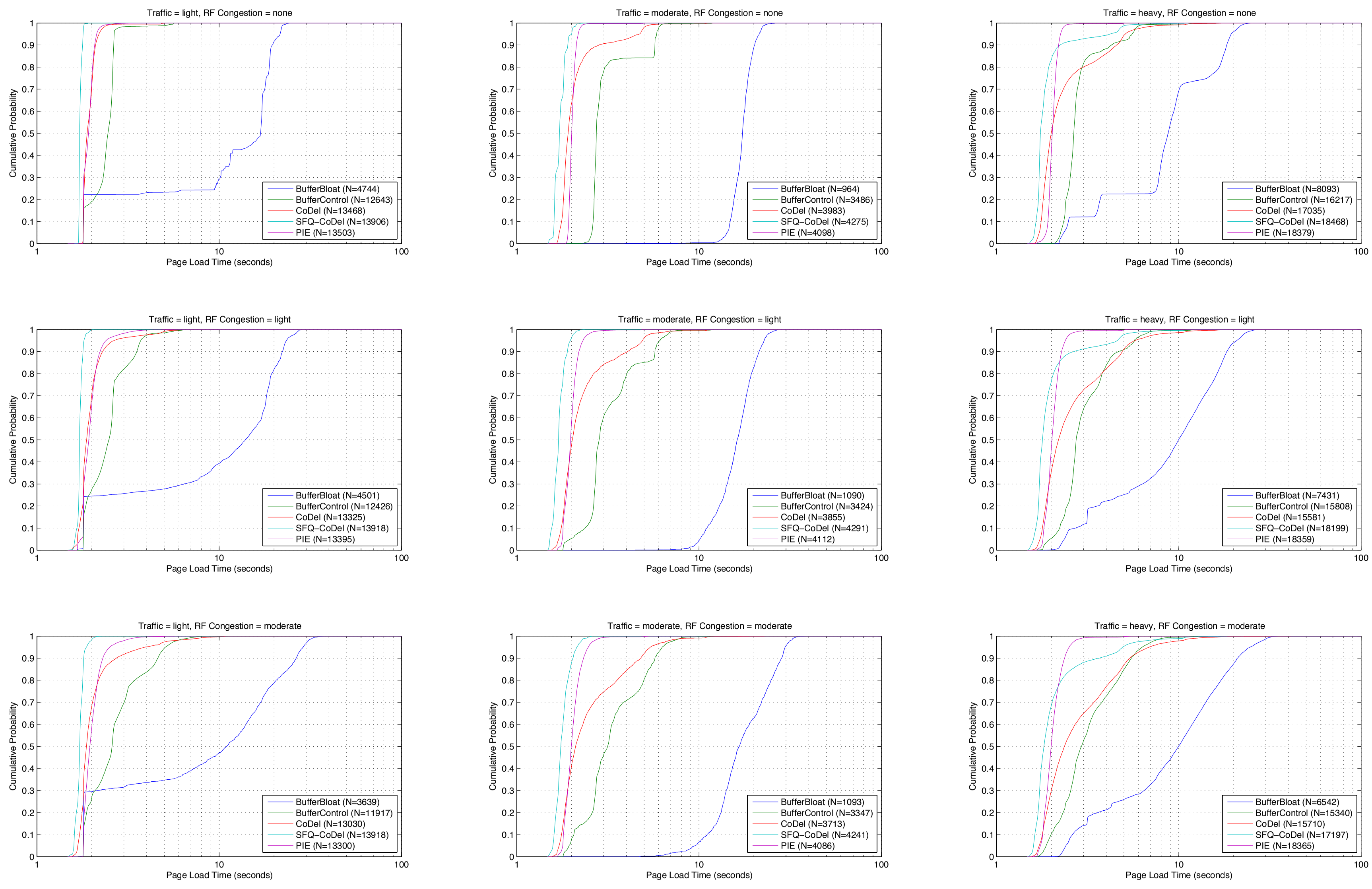

Figure 22 - Web Page Load Performance Detail

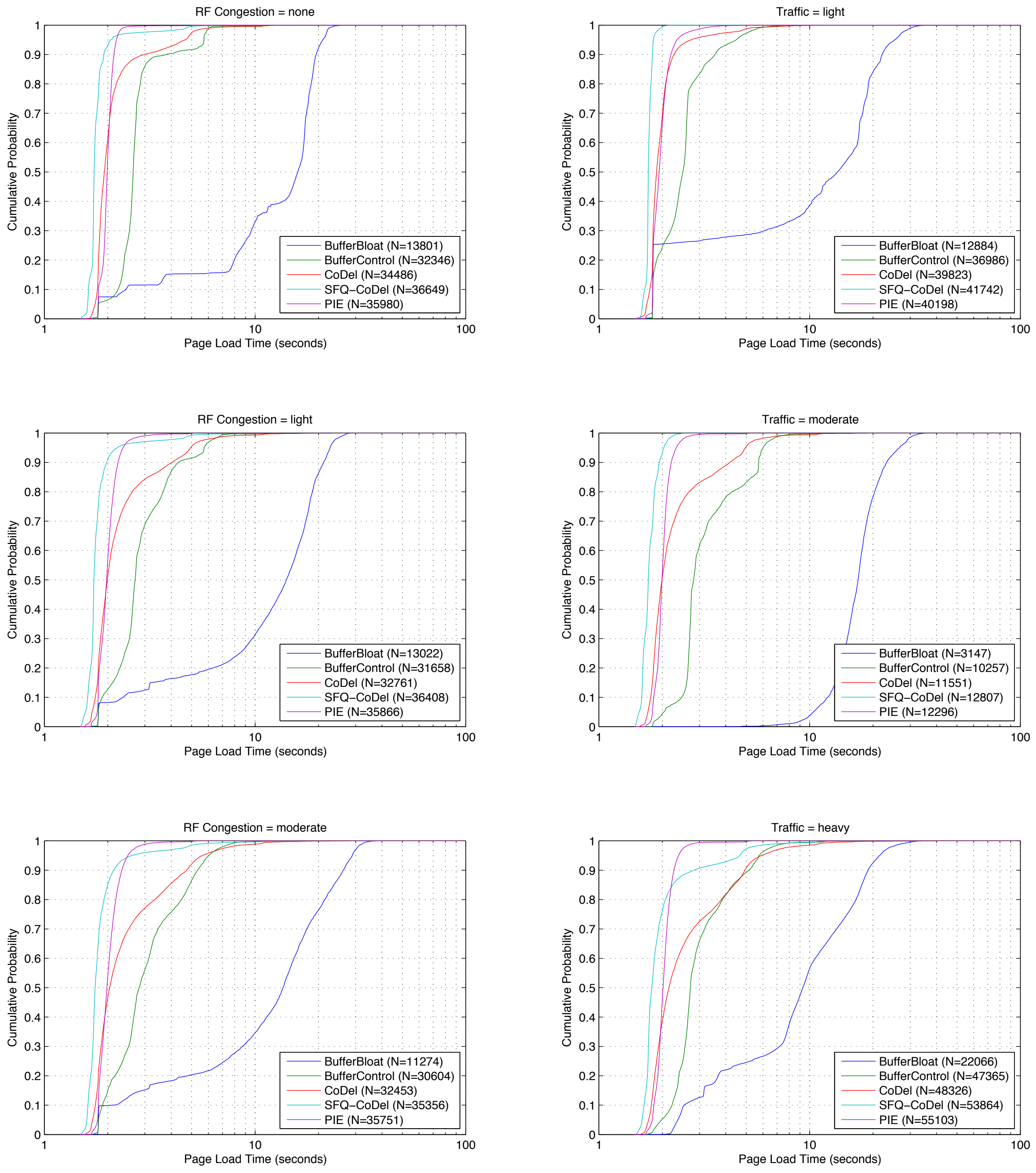

Figure 23 - Web Page Load Performance vs. RF Congestion and vs. Traffic Load

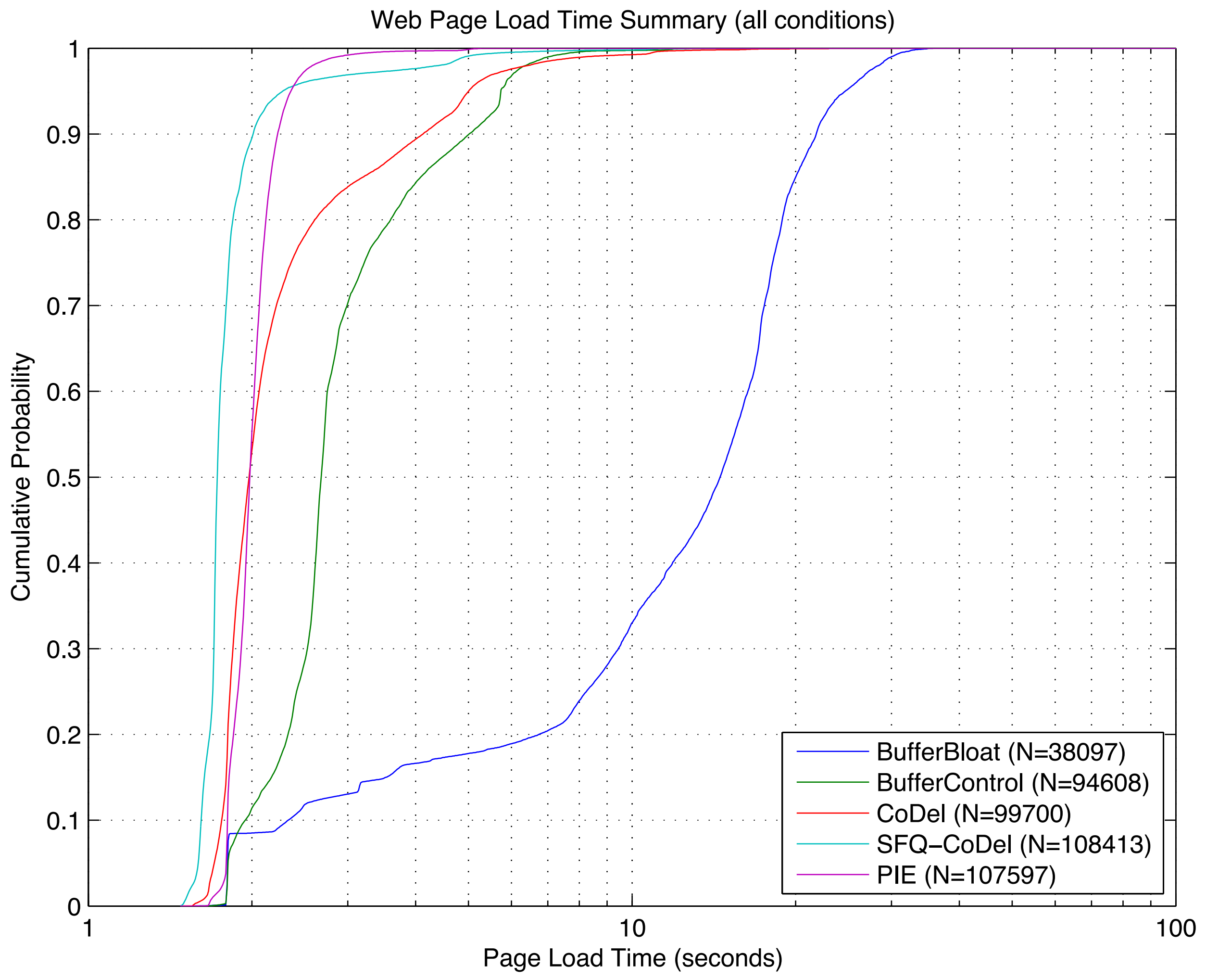

Figure 24 - Web Page Load Performance - Summary

These results show a significant benefit can be achieved by using Buffer Control or any of the active queue management techniques, and extremely good performance can be expected by using the SFQ-CoDel or PIE queue managers.

As noted in Section 3.6, our web model does not include DNS lookups. The most interesting impact that the DNS process would have on these page load time results would come from the potential for packet loss. An estimate of the impact that would be caused by DNS packet loss can be obtained by examining the gaming traffic packet loss statistics. Since CoDel in moderate and heavy traffic cases, and SFQ-CoDel in heavy traffic cases showed the highest levels of gaming packet loss, we would conclude that those scenarios would see the most negative impact as a result of DNS packet loss.

4.3 VoIP Audio Quality

Using the methodology described in [White], we estimate voice quality for a G.711 VoIP application in the simulated conditions. The figures below illustrate the Mean Opinion Score results. Based on the choice of codec, the ideal MOS score that can be achieved is a value of 4.4. It can be seen that the BufferBloat case provides very poor VoIP quality across all tested conditions. Buffer Control succeeds in providing nearly ideal MOS scores in test conditions with no RF congestion, but performance degrades quickly when RF congestion appears. CoDel provides reasonably good VoIP performance across the entire range of conditions. SFQ-CoDel provides ideal VoIP performance in all of the light and moderate traffic cases, but performance degrades for the heavy traffic cases, for the reasons noted above. Finally, the PIE algorithm provided near-perfect MOS scores across all of the tested conditions.

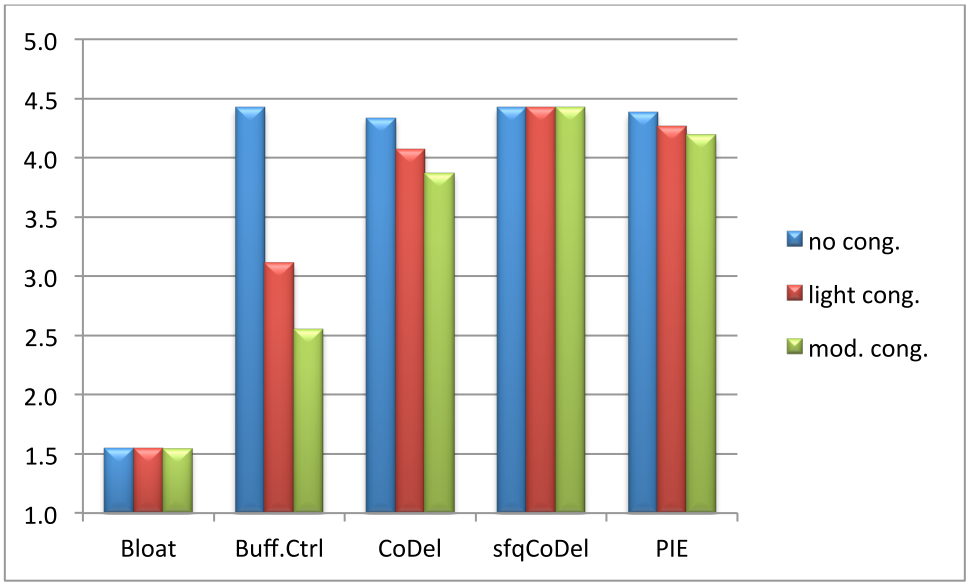

Figure 25 - VoIP Audio Quality - Light Traffic Scenarios

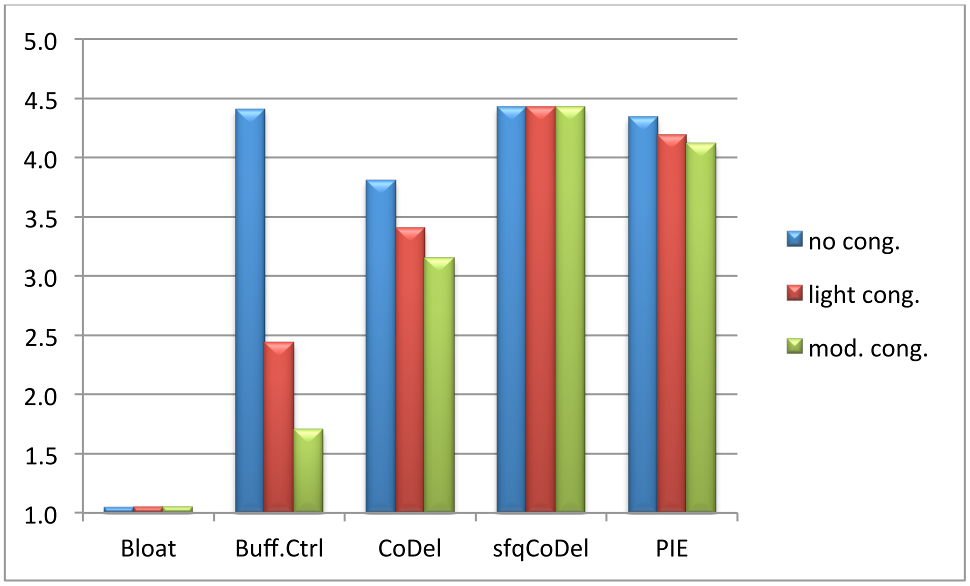

Figure 26 - VoIP Audio Quality - Moderate Traffic Scenarios

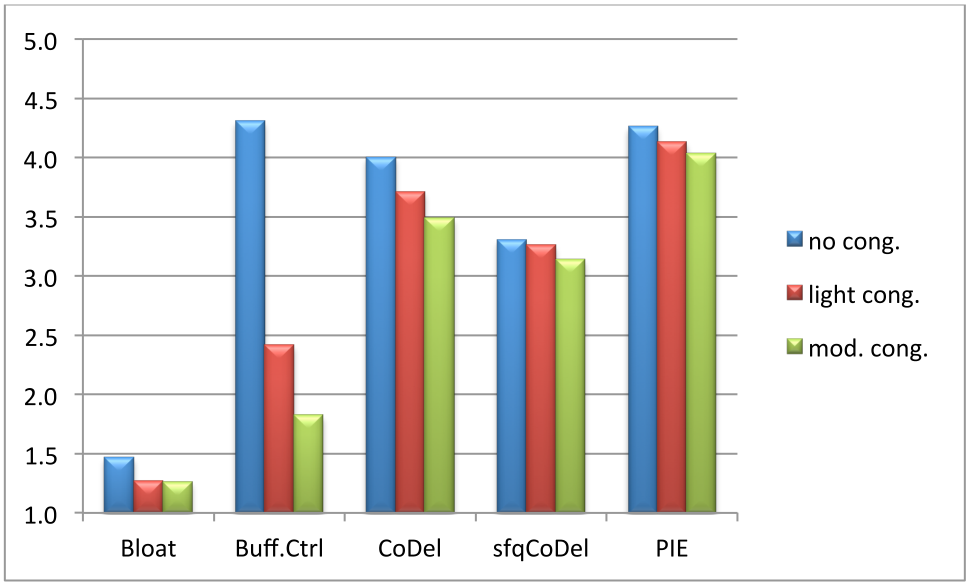

Figure 27 - VoIP Audio Quality - Heavy Traffic Scenarios

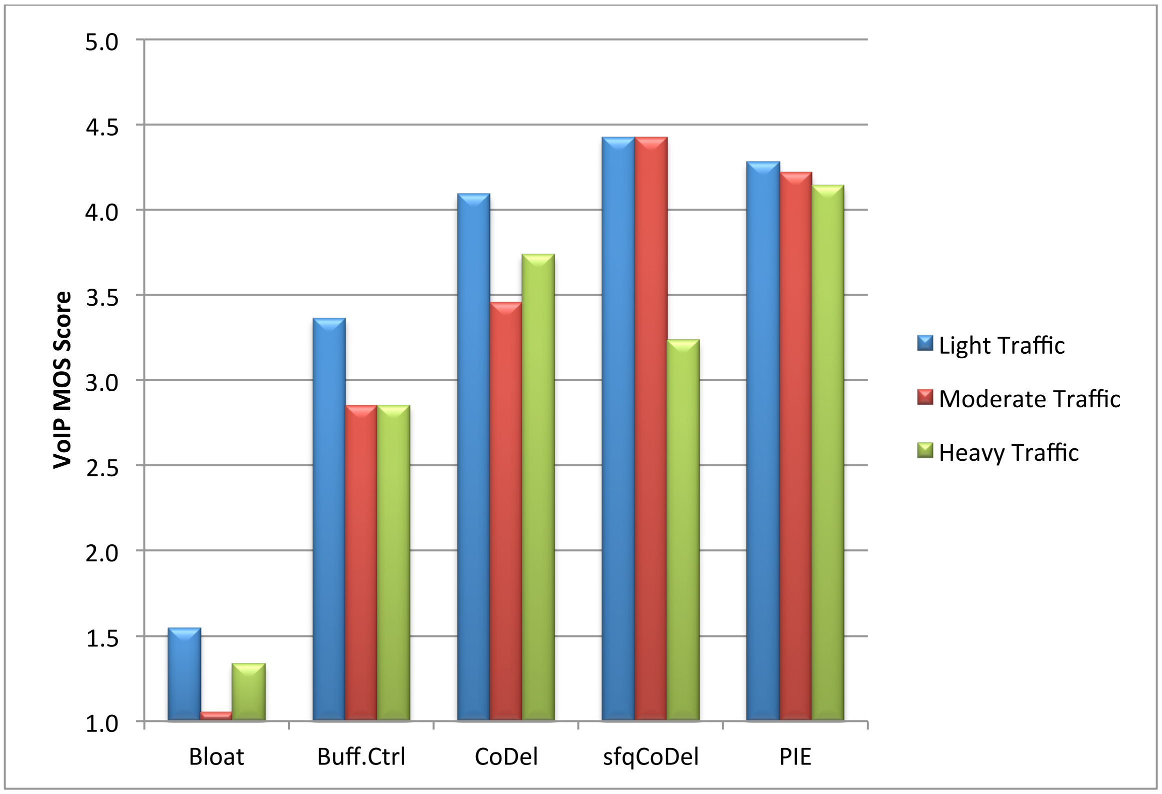

Figure 28 - VoIP Audio Quality - All Scenarios

There are many possible VoIP implementations, and some may show better performance in certain test conditions than the approach we chose. In particular, some over-the-top VoIP services use deep and variable dejitter buffers along with sophisticated playback rate modulation techniques in order to overcome temporary events of high latency. We have not attempted to implement such techniques, nor are we aware of analytical models that could be used to predict MOS scores for them.

4.4 Bulk TCP Upload Performance - Short Time Scale

The last set of application metrics is TCP performance. The figures below show the initial 30 seconds of a single TCP-Reno upload with no other competing traffic. For all queue managers, the modem starts off with a full token bucket, so we see the power boost rate of 20 Mbps initially, followed by the 5 Mbps Max Sustained Rate. The bufferbloat case shows the ideal performance, where TCP ramps up very quickly to a steady 20 Mbps rate, and then once the token bucket is exhausted, drops down to a steady 5 Mbps. This is one explanation for why the problem of buffer bloat exists today. The advantage of having an oversized buffer is that TCP performance is maximized, i.e., the presence of a standing queue at the bottleneck link ensures that the link never goes idle.

The buffer control plot shows the downside to simply limiting the buffer size. While the buffer was adequately sized for TCP to ramp up fairly quickly to 5 Mbps, it was undersized for reaching 20 Mbps. As a result, the rate slowly increases and only reaches 20 Mbps after 15 seconds. In situations in which the power boost configuration is less than 4x the Max Sustained Rate, this effect will not be as pronounced.

CoDel and SFQ-CoDel show similar performance to one another. They both have a sufficient buffer to allow TCP to ramp up quickly to 20 Mbps; however, CoDel/SFQ-CoDel induce some packet drops as they try to react to the rapidly increasing congestion window. The result is that the data rate temporarily drops to 12 Mbps before beginning to ramp back up to 20 Mbps. Additionally, after the initial power boost period ends, we do see quite a bit of short-term variability in data rate as they try to maintain a minimal standing queue. Note that the active queue managers are helped by the fact that the token bucket rate limiter allows the modem to earn credit during periods where its transmit rate is less than 5 Mbps, and then spend that credit later to achieve a rate that is greater than 5 Mbps (particularly in the case where a large token bucket is configured). So, if the AQM happens to send too strong a signal to TCP, forcing it to momentarily back-off too severely, that doesn't have a big effect on the average throughput over time.

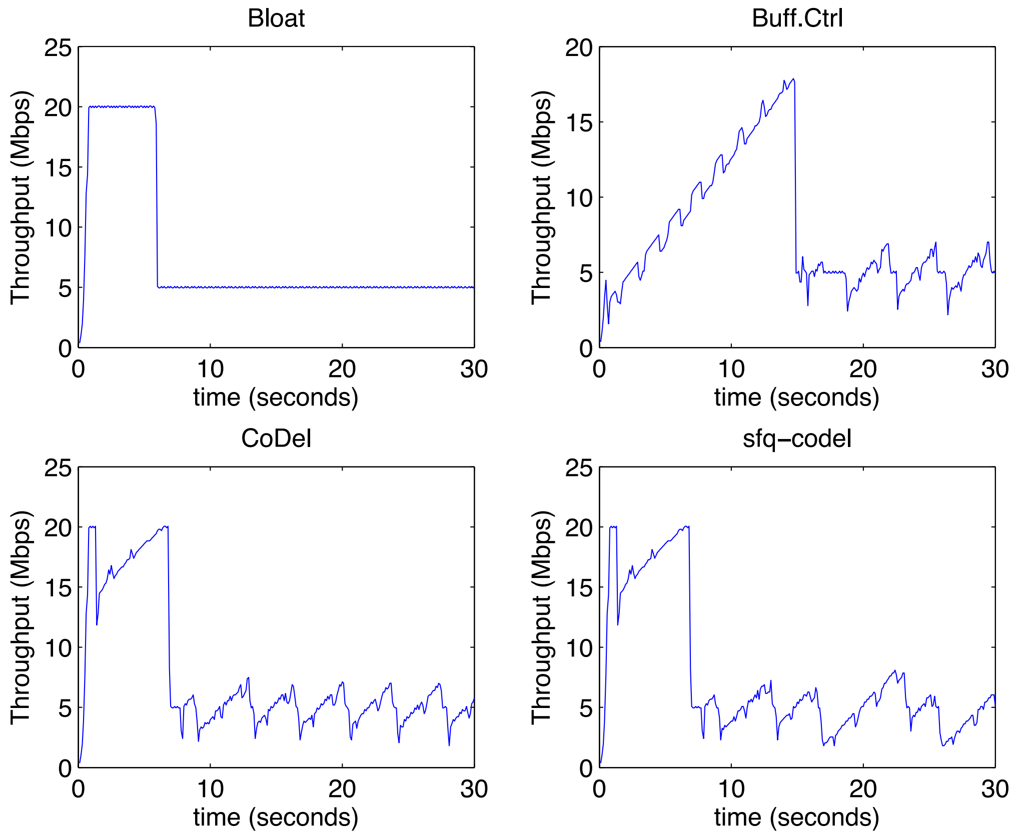

Figure 29 - Short Term TCP Performance

Results with PIE don't show as good performance as compared to the other AQMs. While the data rate does ramp up quickly to 20 Mbps, PIE induces significant packet drops that cause the data rate to plummet to as low as 1 Mbps before beginning the Reno linear ramp back up. Unfortunately, it appears that PIE induces further packet drops when the rate reaches about 12 Mbps, which results in the data rate never recovering to the full 20 Mbps power boost rate. The data rate then continues to oscillate a bit more heavily than was the case with CoDel or SFQ-CoDel.

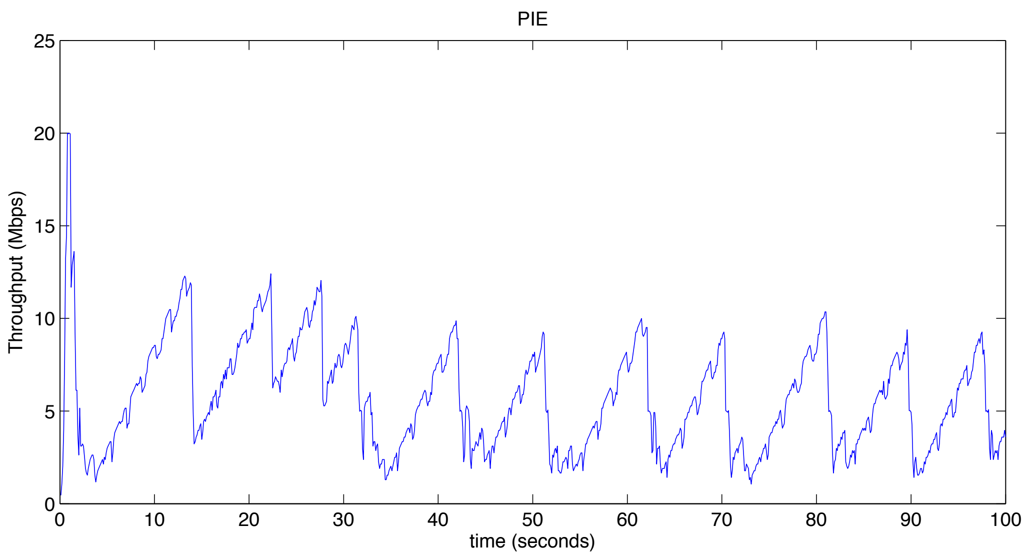

Figure 30 - Short Term TCP Performance using PIE

4.5 Bulk TCP Upload Performance - Long Time Scale

Over a long time scale, we see similar performance for all of the approaches. There is a slight benefit to using the large buffer (the bufferbloat case), since it is able to keep the link busy consistently. As was seen in the short term TCP performance graphs in the previous section, the other approaches are not able to consistently keep the link at maximum capacity. However, the other approaches are benefitted by the token bucket configuration. If they ever allow the link to go underutilized, the DOCSIS token bucket rate shaper earns tokens that allow the modem to exceed the Maximum Sustained Rate for an equivalent amount of data. In other words, the token bucket enforces a long-term average rate limit, and allows for variability in data rate to achieve it.

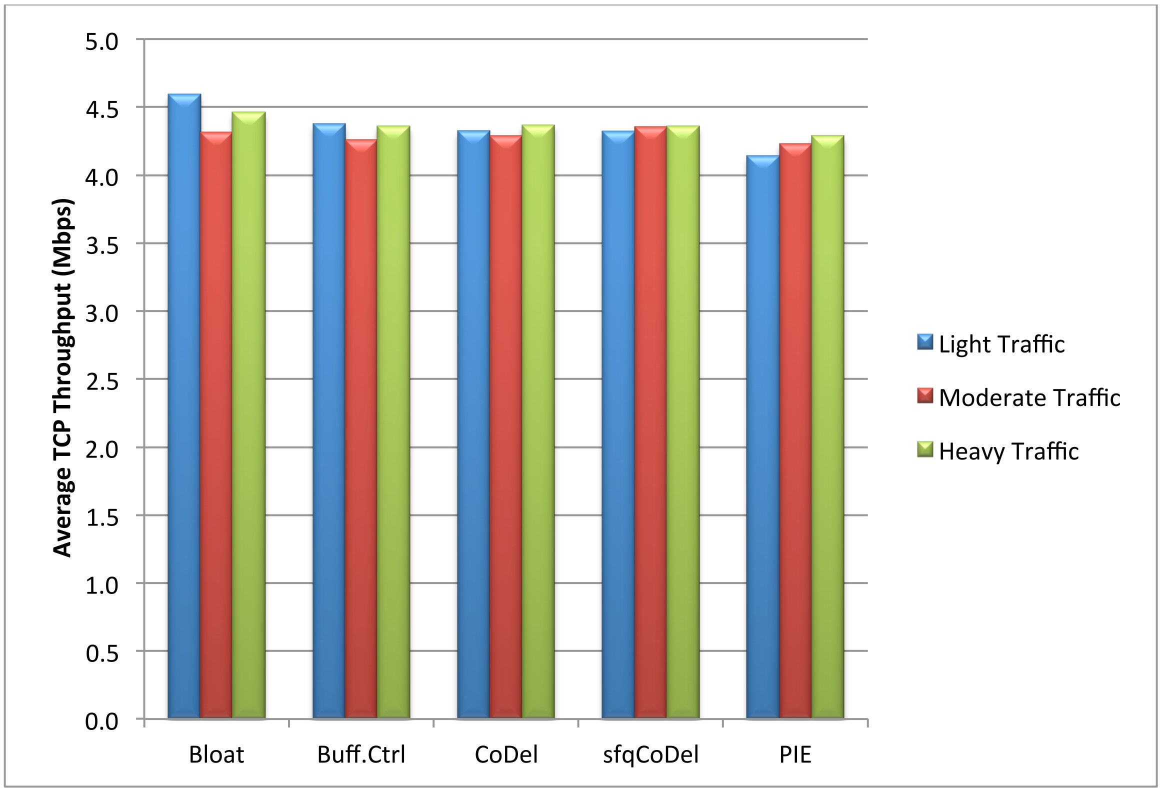

Figure 31 - Long Term TCP Performance

5 Conclusion

In summary, the buffer control feature is available today on many DOCSIS 3.0 modems, but is not enabled by default. When configured appropriately, buffer control may provide a 10 to 20x reduction in gaming latency, three to six times reduction Web page load time, improved VoIP quality, and only a slight degradation in TCP performance (mainly on short time scales).

In looking at the three active queue management approaches that we studied, CoDel provides some attractive benefits relative to Buffer Control. The main benefits being improvements in gaming latency, page load time and short term TCP performance, as well as improvements in VoIP performance in certain conditions.

SFQ-Codel shows extremely good performance in the majority of the tested scenarios, although the issue with BitTorrent traffic deserves some further research to see if it can be addressed.

The PIE algorithm outperforms buffer control, and in most cases outperforms CoDel. The performance of PIE for TCP traffic warrants some additional investigation to see if further tuning of PIE can resolve it, or if performance differs considerably with other TCPs. Another concern is that PIE may require more elaborate tuning based on the network technology and conditions. One potential improvement in PIE that has been discussed by Cisco is to marry it with the SFQ concept. While this may give even further improvements in some scenarios, it would introduce the same issue that SFQ-CoDel has with BitTorrent.

5.1 Next Steps

One next step is for cable operators to plan to enable the use of buffer control in their deployed modems.

Another is for cable modem vendors to examine the implementation feasibility of the various AQM approaches described here. That these algorithms are more complex than drop tail is certain. But the improvement in performance likely outweighs the implementation complexity.

In addition, we plan to extend the investigation of AQM techniques to DOCSIS 3.1 speeds, and to the DOCSIS 3.1 MAC as that starts to take shape. The goal would be to provide some recommendations for vendors, or potentially some requirements in DOCSIS 3.1 to implement some the advanced flow queuing concepts. Further, we'd like to understand the benefit that SFQ on its own provides (without CoDel) so that we can share those results with the community as well.

Finally, this work has only looked at the upstream side, since the view has been that upstream is the source of the biggest problem. Downstream has been viewed to be not quite as bad, but some of the more recent data from Netalyzer testing seems to indicate that it is a source for bufferbloat congestion as well.

APPENDIX A REFERENCES

[Allman]	M. Allman, "Comments on Bufferbloat", ACM SIGCOMM Computer Communication Review, 43(1), January 2013, http://www.icir.org/mallman/papers/bufferbloat-ccr13.pdf
[Beigbeder]	T, Beigbeder, R, Coughlan, C. Lusher, J. Plunke, E. Agu, M. Claypool, “The Effects of Loss and Latency on User Performance in Unreal Tournament 2003”, Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames-04), August, 2004.
[Bredel]	M. Bredel and M. Fidler, "A Measurement Study regarding Quality of Service and its Impact on Multiplayer Online Games", 2010 9th Annual Workshop on Network and Systems Support for Games (NetGames), 16-17 Nov. 2010.
[Bussiere]	J. Bussiere, S. Zander, "Empirical Measurements of Player QoS Sensitivity for the Xbox Game Halo2", CAIA Technical Report 050527A, May 2005
[Choy]	S. Choy, B. Wong, G. Simon, C. Rosenberg, "The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency", Network and System Support for Games (NetGames-2012), November, 2012.
[Dischinger]	Dischinger, M., et al., “Characterizing Residential Broadband Networks”, Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, Oct. 24-26, 2007, San Diego, CA, USA http://broadband.mpi-sws.org/residential/.
[LEDBAT]	Dario Rossi, Claudio Testa, Silvio Valenti, Luca Muscariello, LEDBAT: the new BitTorrent congestion control protocol, International Conference on Computer Communications and Networks (ICCCN 2010), Zurich, Switzerland, August, 2010.
[Jarschel]	M. Jarschel, D. Schlosser, S. Scheuring, T. Hossfeld, "An Evaluation of QoE in Cloud Gaming Based on Subjective Tests", 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, June-July 2011.
[Misra]	V. Misra, W.B. Gong, and D. Towsley, “Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red,” in *Proceedings OF ACM SIGCOMM*, 2000, pp. 151–160.
[Nichols]	K. Nichols, V. Jacobsen, "Controlled Delay Active Queue Management", draft-nichols-tsvwg-codel-01.txt
[Pan]	R. Pan, et al., "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-pan-tsvwg-pie-00.txt
[Peon]	R. Peon & W. Chan, SPDYEssentials, Google Tech Talk, 12/8/11
[Rossi]	D. Rossi, software::LEDBAT, http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.LEDBAT#howtorun

[Sundaresan]	S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, A. Pescapè, "Broadband Internet Performance: A View From the Gateway", SIGCOMM'11, http://sites.noise.gatech.edu/~srikanth/docs/gateway-sigcomm2011.pdf
[White]	G. White, J. Padden, "Preliminary Study of CoDel AQM in a DOCSIS Network", http://www.cablelabs.com/downloads/pubs/PreliminaryStudyOfCoDelAQM_DOCSISNetwork.pdf
[Zander]	S. Zander, G. Armitage "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players", Australian Telecommunications Networks & Applications Conference 2004 (ATNAC2004), Sydney, Australia December 8-10 2004.