CableLabs®

NETWORK TECHNOLOGIES

ACTIVE QUEUE MANAGEMENT IN DOCSIS 3.X CABLE MODEMS

Prepared by:

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

CableLabs R&D Lead:
Dan Rice
Vice President, Network Technologies
d.rice@cablelabs.com

May 2014

DISCLAIMER

This document is published by Cable Television Laboratories, Inc. ("CableLabs®") to provide information to the cable industry. CableLabs reserves the right to revise this document for any reason including, but not limited to, changes in laws, regulations, or standards promulgated by various agencies; technological advances; or changes in equipment design, manufacturing techniques or operating procedures described or referred to herein. This document is prepared by CableLabs on behalf of its cable operator members to facilitate the rendering, protection, and quality control of communications services provided to subscribers.

CableLabs makes no representation or warranty, express or implied, with respect to the completeness, accuracy or utility of the document or any information or opinion contained in this document. Any use or reliance on the information or opinion is at the risk of the user, and CableLabs shall not be liable for any damage or injury incurred by any person arising out of the completeness, accuracy or utility of any information or opinion contained in this document.

This document is not to be construed to suggest that any manufacturer modify or change any of its products or procedures, nor does this document represent a commitment by CableLabs or any member to purchase any product whether or not it meets the described characteristics. Nothing contained herein shall be construed to confer any license or right to any intellectual property, whether or not the use of any information herein necessarily utilizes such intellectual property.

This document is not to be construed as an endorsement of any product or company or as the adoption or promulgation of any guidelines, standards, or recommendations.

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 assistance on tuning it for the DOCSIS MAC; and to the members of the DOCSIS 3.1 AQM Working Group:

Alon Bernstein (Cisco)
Jim Chen (Huawei)
Jason Combs (Comcast)
Margo Dolas (Broadcom)
Hesham ElBakoury (Huawei)
Kirk Erichsen (Time Warner Cable)
Bill Hanks (ARRIS)
Charaf Hanna (STMicroelectronics)
Barak Hermesh (Intel)
Victor Hou (Broadcom)
Jeff Howe (ARRIS)
Lee Johnson (STMicroelectronics)
Amos Klimker (Intel)
Ramdane Krikeb (Videotron)
James Martin (Clemson University)
Satish Mudugere (Intel)
Rong Pan (Cisco)
David Pullen (Broadcom)
Pawel Sowinski (Cisco)
Rick Vetter (Comcast)

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Active Queue Management (AQM) provides a solution to the problem of providing good application layer Quality of Experience when multiple applications share a network connection. The need for AQM arises due to the presence of packet buffering in network elements and due to the mechanics of the Transmission Control Protocol (TCP) congestion avoidance algorithm.

Every network element supports buffering of some amount of packets that are destined to be forwarded on the next link. This buffering is important to ensure good utilization of the network link, especially in cases where the incoming traffic rate exceeds the outgoing link rate. In these bottleneck situations, the buffer serves to absorb high-rate traffic bursts so that they can then be played out on the slower outgoing link. Without buffering, most of the packets in the high-rate burst would simply be dropped.

The Transmission Control Protocol (TCP) is by far the most widely used protocol for reliable delivery of content on the Internet. It is used for the vast majority of Internet video streaming (e.g. Netflix, YouTube, HBOGo, Hulu, Amazon Prime Video, etc.), and all web browsing, email, and file transfers. When sending content, the TCP utilizes a "congestion avoidance" algorithm in order to automatically adjust its sending rate to approximately its fair share of the available capacity in the network path to the receiver. In essence, this congestion avoidance algorithm works by sending data at an ever-increasing rate until a packet loss is experienced. Once a packet loss is detected, the algorithm cuts its rate in half immediately, and then resumes again the process of increasing its rate. In many flavors of TCP, this process is referred to as Additive Increase, Multiplicative Decrease (AIMD).

The "Additive Increase" part of the process results in the sending rate increasing until the bottleneck link can no longer support it. At that point, a standing queue begins to form at the bottleneck link. In the absence of AQM, that standing queue grows until the buffer is full, at which time a packet drop occurs, triggering the "Multiplicative Decrease" and the process repeats. The result is that TCP sessions effectively seek to keep network packet buffers full, which results in poor performance for interactive applications, which are generally sensitive to latency.

The term "Bufferbloat" has been coined to refer to the practice (sometimes inadvertent) of sizing network buffers to be significantly greater than needed to ensure good link utilization, and the resulting significant degradation of interactive applications in the presence of concurrent TCP traffic.

As awareness of the topic of "Bufferbloat" has risen, so too has interest in methods to resolve it. AQM is currently the most promising approach because significant network-wide benefits can be derived by implementing it in a relatively small number of bottleneck network elements (e.g. broadband modems). Current AQM approaches seek to detect the "standing queue" created by TCP, and once detected, send TCP a congestion signal (by dropping a packet). The modern algorithms do this without the need to be tuned for the network conditions.

Based on the simulated performance and the implementation considerations, a customized version of the PIE algorithm, called DOCSIS-PIE is now specified in the DOCSIS 3.1 and DOCSIS 3.0 specifications. Implementation of DOCSIS-PIE is mandatory for implementation in DOCSIS 3.1 cable modems, and recommended for implementation in DOCSIS 3.0 cable modems. In addition to the mandatory/recommended algorithm, DOCSIS 3.1 & 3.0 CM vendors are free to support additional AQM algorithms of their choosing. However, even in that case, DOCSIS-PIE is the default and other algorithms require explicit selection by the operator.

1 Introduction

From June 2013 through January 2014, CableLabs worked with the developers of DOCSIS equipment to define an Active Queue Management algorithm that would be mandatory for implementation of a cable modem compliant with the DOCSIS 3.1 specification. This DOCSIS 3.1 AQM Working Group evaluated several existing candidate algorithms (extending one of these to improve performance) and two new algorithms developed by CableLabs.

In previous work ([White], [White2]), we examined the performance of several AQM algorithms in the context of a DOCSIS 3.0 cable modem. In this white paper, we extend the set of algorithms that we examine, and extend the examination to DOCSIS 3.1 data rates and expected application traffic patterns.

While our initial look at the PIE algorithm in [White2] showed performance that was not quite on par with the competitor CoDel algorithm, we worked closely with Cisco and other stakeholders to improve upon the PIE algorithm to the point that it provides equivalent performance to CoDel.

This study, and the previous one ([White2]), investigated a multiple queue variant of CoDel, referred to as stochastic flow queue CoDel (SFQ-CoDel). This study extended the investigation to also include a CableLabs-designed SFQ-PIE. These SFQ algorithms provided the best performance compared to the other algorithms, but our conclusion was that the performance delta was insufficient to justify the large delta in implementation complexity. Also, SFQ brought with it too many unknowns, such as number of queues, best hashing function, issues with VPNs and tunneled traffic, etc.

The PIE algorithm and CoDel-DT (a variant of CoDel developed by CableLabs) were the most attractive candidates due to implementation complexity and alignment with the DOCSIS 3.0/3.1 MAC layer.

PIE has a distinct advantage over the other algorithms (including CoDel-DT) in that the most important parts of the algorithm lend themselves to be implemented in software in D3.1 cable modems. This has a couple of advantages. One advantage is that it reduces the development risk for each DOCSIS 3.1 CM silicon vendor, since the algorithm doesn't need to be extensively tested prior to taping out the SOC. Another is that it reduces risk for the DOCSIS 3.1 platform in that it allows the algorithm to be modified in the future, in devices that are in the field. An additional benefit of PIE (a benefit shared with CoDel-DT) is that the algorithm has the potential to be implemented in existing DOCSIS 3.0 cable modems.

This white paper presents the results of the AQM selection process and specification definition for DOCSIS 3.1 technology.

2 SIMULATION CONDITIONS

The simulation and analysis methodology utilized for evaluating AQM algorithms for DOCSIS 3.1 is an evolution of the approach presented in [White2]. In this section, we build on the methodology description provided in [White2] rather than presenting it again.

As our goal is to select an AQM algorithm for managing the upstream queue in the cable modem, the simulation conditions are focused on scenarios that will the best illustrate the difference between the performance of the different algorithm choices in that context. Further, our focus is on scenarios that we anticipate will be particularly relevant for DOCSIS 3.1 network deployments in the 2018 timeframe.

2.1 SERVICE MODEL

The configured data rates for the service are extrapolated for 2018 as follows.

Upstream

- Maximum Sustained Traffic Rate: 200 Mbps
- Maximum Traffic Burst: 30 MB
- Peak Traffic Rate (burst rate): 250 Mbps

Downstream

- Maximum Sustained Traffic Rate: 1000 Mbps
- Maximum Traffic Burst: 330 MB
- Peak Traffic Rate (burst rate): 1500 Mbps

2.2 TRAFFIC MODELS

2.2.1 MODELS USED FOR VOIP, GAMING, WEB PERFORMANCE METRICS

For traffic conditions, we extrapolate from the traffic scenarios used in [White2] in order to model possible future traffic loads. In addition, we reduced the number of scenarios by including FTPs with short and long RTT in each FTP scenario rather than maintaining individual scenarios with short and with long RTT.

For our 2018 model of Internet traffic we're using the following 9 traffic loads:

Table 1 - Test Conditions

N	F1	F2	Fs	W	VG	C	T
1	0	0	-	1	1	0	0
2	1	1	infinite	1	1	0	0
3	3	3	50MB	1	1	0	0
4	3	3	50MB	1	1	6	0

5	1	1	19MB*	1	1	0	0
6	3	3	50MB	4	4	0	0
7	3	3	50MB	4	4	6	0
8	5	5	2.5MB	4	4	6	0
9	3	3	50MB	4	1	0	100

where:

N: traffic load index

F1: number of simultaneous FTP uploads with 20ms RTT

F2: number of simultaneous FTP uploads with 100ms RTT

Fs: FTP file size

W: number of simultaneous web users

VG: number of simultaneous VoIP/gaming sessions

C: CBR data rate (Mbps)

T: number of torrent (LEDBAT) connections

*Filesize and repetition pattern chosen to exercise DOCSIS "powerboost" feature

The web user model is not fundamentally changed from what was reported in [White2]. In it the client fetches a single file (representing the html file) and then upon completion of this file transfer proceeds to download 100 resources (of log-normally distributed size) that are spread evenly across 4 servers. The client maintains 6 active TCP connections to each server until all 25 resources have been requested from that server. What has changed in our model are the sizes of the web resources. Extrapolating from http://httparchive.org/trends.php, we've selected a total page size (sum of all 101 resources) of 7 MB. Our web user downloads the web page, waits 5 seconds and then repeats. The metric of interest here is page load time, calculated from the initiation of the TCP connection to download the initial file, to the completion of the TCP session that downloads the final resource.

As in [White2] we use a single traffic type to model both VoIP and online gaming. This traffic type consists of UDP packets of 218 bytes at 50 packets per second. We monitor packet loss and per-packet latency, and we estimate a VoIP MOS score using the methodology described in [White].

In our updated model, we rate-limit the aggregate upstream torrent traffic to 50% of the upstream Maximum Sustained Traffic Rate as an approximation for what typical client behavior would be.

2.2.2 Model Used for TCP Performance Metrics

To assess performance of TCP applications we use a model somewhat inspired by Ookla Speedtest.net. The goal of this work is not to optimize the performance specifically for the speedtest.net result, however, this scenario is nonetheless an interesting one both because it represents a commonly utilized methodology for assessing TCP performance, and because it is a common scenario in which the user's experience is directly driven by the average TCP upload throughput. In many other upload cases (email, cloud storage and cloud backup, etc.) uploads happen more-or-less in the background, with the user's interaction (when there is direct interaction) ending with the initiation of the upload task (e.g. clicking "send" on the email message) rather than on its completion.

In the Ookla test, upstream TCP throughput is measured via the use of two simultaneous upstream TCP sessions that transfer data for a total of approximately 10 seconds. The closest server is chosen by default, but the user can select to run a test to any server in the world. We simulate two TCP sessions, but don't terminate the test at 10 seconds, instead choosing to allow it to continue for up to 100 seconds. We simulate four values for RTT (20ms, 50ms, 100ms and 200ms). The data point from our simulation set that most closely represents typical Ookla throughput results is the average throughput for a 10 second transfer using an RTT of 20ms, but the other results provide interesting insight into other file transfer conditions.

2.3 RF Congestion Models

We utilize 4 different levels of RF congestion to examine the ability of the AQM to respond to changes in available link capacity.

- No Congestion: Channel capacity exceeds the Peak Traffic Rate of 250 Mbps.
- Light RF Congestion: Channel capacity varies among 165, 200, 225, 250 Mbps.
- Moderate RF Congestion: Channel capacity varies among 185, 190, 200, 225 Mbps.
- Heavy RF Congestion: Channel capacity varies among 100, 120, 180, 200 Mbps.

For each congestion case, the channel capacity remains at each value for 10 seconds, and changes in a repeating pattern that exercises all 12 possible rate transitions as discussed in [White2].

2.4 Downstream Queueing

For purposes of this study, downstream traffic sees a drop-tail queue at the CMTS with 250 KB of buffering. The CMTS requirements that were the outcome of this project include mandatory support for AQM for DOCSIS 3.1 CMTS and recommended support of AQM for DOCSIS 3.0 CMTS, but no specific algorithm is required.

3 Implementation Concerns for AQM in the Cable Modem

Cable modems are high-volume, low-margin commodity devices that use special purpose silicon designs to implement the DOCSIS MAC and PHY layers. The silicon designs rely on custom hardware engines in order to provide high packet forwarding performance at low cost, and due to the evolution of the DOCSIS protocol versions (going back to DOCSIS 1.0 in the mid-1990s), the requirements for backward compatibility, and the long history that some of the chip companies have in developing them, the designs are both sophisticated and highly optimized. As a result, there are some DOCSIS-specific implementation considerations that need to be factored into the selection of an AQM algorithm, in addition to the typical complexity vs. performance considerations.

Two of these implementation considerations that have a strong bearing on the selection arise from the Service Flow requirements and the timing requirements for MAP processing in channel-bonded upstream transmission.

3.1 Service Flow Requirements

The DOCSIS Media Access Control (sub-)layer provides tools for configuring differentiated Quality of Service for different applications by the use of Packet Classifiers and Service Flows.

Each cable modem can be configured with multiple Packet Classifiers and Service Flows for managing upstream traffic. The maximum number of such entities that a cable modem supports is an implementation decision for the chip designer, but modems typically support 16 or 32 Service Flows and at least that many Packet Classifiers.

Packet Classifiers can match packets based upon several fields in the packet/frame headers including the Ethernet header, IP header, and TCP/UDP header. Matched packets are then queued in the associated Service Flow queue.

Each Service Flow has an associated Quality of Service (QoS) parameter set that defines the treatment of the packets that traverse the Service Flow for transmission on the coax media. These parameters include (for example) Minimum Reserved Traffic Rate, Maximum Sustained Traffic Rate, Peak Traffic Rate, Maximum Traffic Burst, and Traffic Priority. Each upstream Service Flow corresponds to a queue in the cable modem, and each downstream Service Flow corresponds to a queue in the CMTS. Each Service Flow queue in the cable modem has a hardware engine that manages media access more-or-less independently from the other Service Flows present on the device, and AQM functionality similarly needs to be implemented such that it operates on each Service Flow queue independently.

In this context, considering flow queuing algorithms such as SFQ-CoDel or SFQ-PIE which call for 1024 queues could be daunting considering that this needs to be replicated 16 or 32 times, with each of these 16384 or 32768 queues integrating into the hardware engines that manage the upstream media access. In our experimentation we utilized 32 queues in SFQ-CoDel and SFQ-PIE yet even that number of hardware queues (512 or 1024) has an impact on silicon complexity and die size and so needs to be considered carefully.

3.2 MAP Processing Requirements

The upstream media access function in the DOCSIS specifications operates using a request-grant mechanism. The CMTS schedules the individual transmission bursts of all of the cable modems sharing the link, and it communicates this schedule via a broadcast bandwidth allocation map message (called a MAP for short). Each MAP message describes the upstream transmission opportunities (grants) in a finite span of time (typically 2-4 ms in duration) and is sent shortly before the interval to which it applies.

When a particular Service Flow has data that is eligible to be sent (i.e. it has cleared rate shaping) it scans the MAP messages for an upstream "contention request" transmission opportunity and when one comes available, it sends a short request message identifying itself and how much data it has to send. It then waits for a MAP message granting it a transmission opportunity in which to send its data.

Once the MAP containing a grant for this Service Flow arrives, the modem has on the order of 650 µs to prepare the burst for transmission. Depending on channel congestion and CMTS scheduler policies, the Service Flow may not be granted a transmission opportunity that fully satisfies its request. In the case of multiple transmit channel operation ("channel bonding") the modem may need to prepare to transmit multiple (4 or 6) bursts simultaneously on different channels using data that is drawn serially from the Service Flow queue. Since the modem is unaware of how much it will be allowed to transmit and on which channels (each of which may have different modulation and forward error correction parameters) until it receives the MAP message, it cannot prepare bursts prior to the MAP arrival. As a result, data needs to be dequeued from the Service Flow into multiple parallel hardware buffers at a rate that far exceeds the line rate of the actual transmissions.

Thus, AQM algorithms that are designed to operate at the head of the queue (during the dequeue operation) must similarly be scaled to function at a data forwarding rate that far exceeds the rate required for algorithms that are designed to operate at the tail of the queue (during the enqueue operation). This strongly favors tail-drop based algorithms. Further, it implies that flow queuing type algorithms, such as SFQ-CoDel and SFQ-PIE, likely need to perform their deficit round robin packet scheduling operation ([Shreedhar]) prior to MAP arrival so that the high-speed dequeue function can operate on a single transmit queue. This reduces the benefit that would otherwise be achieved by the flow queuing approach.

4 AQM Algorithms Under Study

4.1 Drop Tail

A simple FIFO queue is the base case. Here we use two scenarios, "Bufferbloat" and "Buffer Control".

4.1.1 Bufferbloat

For a DOCSIS 3.1 CM, we model "Bufferbloat" as 250ms of buffering at the Maximum Sustained Traffic Rate. In terms of buffering time, this is considerably less than CMs historically have supported. However, as the negative impacts of overbuffering have been brought to light, and considering that DOCSIS 3.1 modems support data rates an order of magnitude larger than DOCSIS 3.0 modems, chip vendors have indicated that 250ms might be a reasonable upper limit for DOCSIS 3.1 modem implementations.

4.1.2 Buffer Control

To simulate the use of the DOCSIS 3.0 Buffer Control feature (a feature carried forward to DOCSIS 3.1 specifications as well), where the operator can configure the buffer size for the service flow, we model the recommended 50ms of buffering at the Maximum Sustained Traffic Rate.

4.2 CoDel

The CoDel algorithm is described in [Nichols] and also summarized in [White]. In our experiments, we have increased the target sojourn time to 20ms from the default of 5ms (to account for the intrinsic Request-Grant delay of the DOCSIS MAC, and to provide a bit better performance for single TCP sessions), and we've set interval to 150ms based on recommendations from Kathie Nichols. The CoDel algorithm requires timestamps be added to packets at ingress, and then at dequeue, the sojourn time is calculated on each packet and drop decisions are made. As discussed in section 3.2, this head-of-queue processing is problematic in DOCSIS cable modems.

4.3 CoDel-DT

CoDel-DT is an algorithm of our own design. It replaces the problematic packet timestamping and head-drop functions of CoDel with latency prediction and tail-drop. The result is an algorithm that may be more amenable to implementation in DOCSIS cable modems.

In many contexts, it is claimed that the head-drop aspect of CoDel provides a benefit in that congestion signals (drops or possibly ECN marks) aren't waiting in queue (in contrast to tail-dropping AQMs), and instead are transmitted immediately. However, this benefit is counterbalanced by the fact that CoDel makes its drop decisions based on the latency already experienced by a packet in queue (the calculated sojourn time). In CoDel-DT we replace the per-packet sojourn time calculation at dequeue with a per-packet sojourn time prediction at enqueue, and then use the identical drop decision logic as CoDel, but applied at enqueue. In the case where the latency prediction function is omniscient, CoDel and CoDel-DT would thus produce identical results.

Our latency prediction function is identical to the one used in the PIE algorithm, it utilizes an egress rate estimator and the current queue depth to predict latency.

We configure CoDel-DT with a latency target of 20ms and an interval of 150ms.

4.4 SFQ-CoDEL

Motivated by the discussion in Section 3.1, we simulate SFQ-CoDel [Hoeiland] with 32 queues rather than the typical 1024. We refer the reader to [White2] for a discussion of SFQ-CoDel. In brief, it involves a number of CoDel managed queues that are given access to the channel via a deficit-round-robin scheduler, with packets being mapped into queues based on a hash of certain packet header fields (such as IP addresses, protocol, and port numbers). We use an interval of 200ms and a target of 50ms with SFQ-CoDel. We're using increased interval and target values here in order to allow individual TCP sessions to achieve better throughput. In the majority of cases, the flow queueing component of SFQ-CoDel succeeds in isolating the flows, so the impact of TCP buffering isn't felt by the latency sensitive traffic. Note that this brings some risk. The risk arises from the fact that there is a small but non-zero probability of a latency sensitive flow getting hashed into the same queue as a concurrent TCP flow. When this occurs, the latency sensitive flow will suffer as a result of the target value of 50ms. In our view, this is a worthwhile tradeoff.

4.5 PIE

For simulations of the PIE algorithm [Pan] we utilized a delay reference of 15ms. The value here is less than that used for CoDel to compensate for the fact that the PIE latency prediction does not include the Request-Grant latency. In order to have a valid comparison, the intent was to obtain similar latency performance between both algorithms.

4.6 SFQ-PIE

We developed an SFQ-PIE AQM in ns2 for comparison with SFQ-CoDel. This is a bit more complex than it might appear at first. The PIE algorithm calculates a drop probability based on a prediction of queuing latency (and the history of that prediction) that is calculated as the buffer depth divided by the estimated egress rate of the interface. For SFQ-PIE, we'd like to have an independent value of drop probability for each queue, so that queues that are backing up (e.g. with bulk TCP traffic) are given the appropriate congestion signals, while queues that are in a good state (empty or nearly empty) are protected from packet loss.

A first approach might be to simply use the PIE algorithm to calculate the drop probability on each queue independently. This is unfortunately not a realistic approach, since the egress rate of each queue is much more highly variable than the egress rate of the outbound link. Even in the case where the link egress rate is stable at a value R, the egress rate of a queue will be approximately R/N where N is the number of queues occupied with traffic, an unpredictable and unstable value. As a result it is very difficult to obtain an accurate prediction of the queuing latency for an individual SFQ queue.

The approach we chose was to calculate an overall drop probability that is unaware of the SFQ structure, i.e. it is identical to the drop probability that would be calculated by the PIE algorithm (using total bytes in queue divided by estimated egress rate as a predictor of queuing latency). Then, at enqueue time, the drop probability applied to a packet destined for queue X is scaled based on the ratio of the queue depth of queue X and the queue depth of the current largest queue. This approach is reasonably simple, and has a couple of nice properties. One, if a packet is arriving to an empty queue, it is given immunity from packet drops altogether, regardless of the state of the other queues. Two, in the situation where only a single queue is in use, the algorithm behaves exactly like the single-queue PIE algorithm.

In cases where traffic is present across multiple queues and sessions come and go, the overall drop probability calculated by this approach is actually based on an underestimate of the queuing latency for the largest queue. In essence, the latency predictor is accurate in the case where all future traffic is destined for the largest queue. If traffic arrives for other queues, it will be forwarded ahead of the last packet in the current largest queue. To compensate for this, we utilize a delay reference 45ms (rather than 50ms as was used in SFQ-CoDel).

Similar to our testing of SFQ-CoDel we utilize 32 queues for SFQ-PIE.

4.7 Relative Complexity of the AQM Algorithms Under Study

The various options for AQM present different levels of implementation complexity, and the performance of each algorithm needs to be considered in light of its relative complexity.

Based on analysis performed by the DOCSIS chip vendors, the algorithms can be ranked in order of decreasing complexity as follows:

- SFQ-CoDel (most complex due to SFQ structure, time-stamping, head-of-queue operations)
- SFQ-PIE (relatively high complexity due to SFQ structure)
- CoDel (time-stamping & head-of-queue operations)
- CoDel-DT
- PIE

This analysis takes into account the specifics of the DOCSIS MAC layer as discussed in section 3, as well as detailed knowledge of existing silicon designs, and so may not be reflective of implementations in other contexts.

Between CoDel-DT and PIE, there is a small complexity difference due to the fact that the CoDel control law requires more calculations that need to be scaled to occur on a per-packet basis (including the inverse square-root function[1]). In the majority of CM implementations, per-packet processing is implemented in hardware, so this results in slightly higher hardware complexity for CoDel-DT. In addition, with PIE, the majority of "important" calculations are done periodically during an update interval (e.g. every 15 or 16 ms) and thus can be performed by a CPU core. This is an advantage in that the PIE control law would be available for future update and modification via a firmware update, whereas this would likely not be possible with CoDel-DT. Since none of these algorithms have been extensively used in the wild, and since the AQM space seems ripe for future evolution, this is a valuable attribute.

[1] We investigated multiple approaches to simplify the inverse square root function, via table lookup or approximations using Newton's method (as described in [Pollere]), or hybrids of the two.

5 Comparative Results of AQM Algorithms

5.1 VoIP/Gaming Traffic Performance

In [White2] we summarized some work by [Bredel] & Fidler on the impact that network impairments have on the user experience for gaming traffic. That work indicated that latency was the impairment that caused the most impact to game performance.

For VoIP traffic, we again utilize the Mean Opinion Score (MOS) estimator that we utilized in [White].

Figure 1 shows aggregated packet latency results for simulations of the AQM algorithms in the various network and traffic load scenarios. In it we can clearly see the benefit provided by the SFQ-* algorithms, where the 90th percentile latency is approximately 13ms. However, the single-queue AQMs (CoDel, CoDel-DT, PIE) provide good results as well, with 90th percentile latencies between 20ms and 26ms, as compared to Buffer Control where the 90th percentile is above 40ms and Buffer Bloat where it is around 170ms.

Between the single-queue AQMs, CoDel and PIE show very similar performance, and CoDel-DT shows slightly better performance. This is likely due to some offset in the latency prediction algorithm that is biasing the results towards lower queuing delay. It is expected that an adjustment in the latency target could compensate for this offset and provide performance more equivalent to CoDel and PIE.

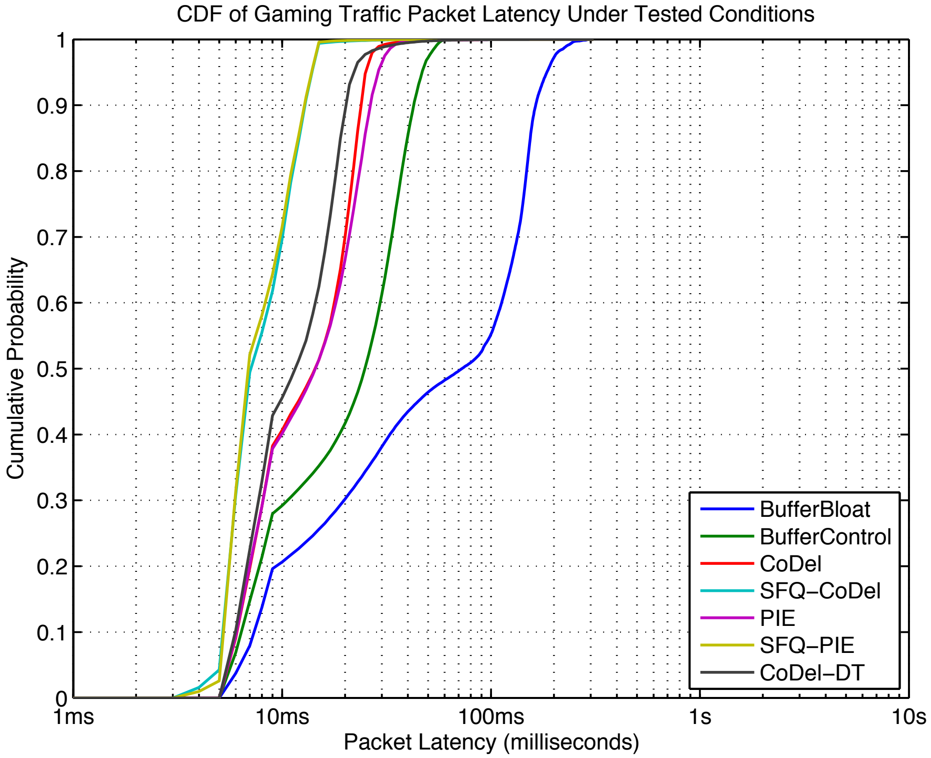

Figure 1 - Packet Latency Statistics for VoIP/Gaming Traffic

Figure 2 shows the statistics of packet loss for gaming traffic. The SFQ-* algorithms don't appear in the plot since they did not result in VoIP/gaming packet loss in any of our tested conditions. Each of the other algorithms also returned zero packet loss in some test conditions, and so, due to the logarithmic scale, the CDF lines do not extend all of the way to the bottom of the graph. Among the single-queue AQMs, PIE provides the lowest packet loss rates, followed closely by CoDel. Again we see different performance from CoDel-DT, in this case we see higher packet loss. This is consistent with the view expressed above that CoDel-DT, as configured, is in effect targeting a lower queuing latency. It achieves this via an increase in packet drops.

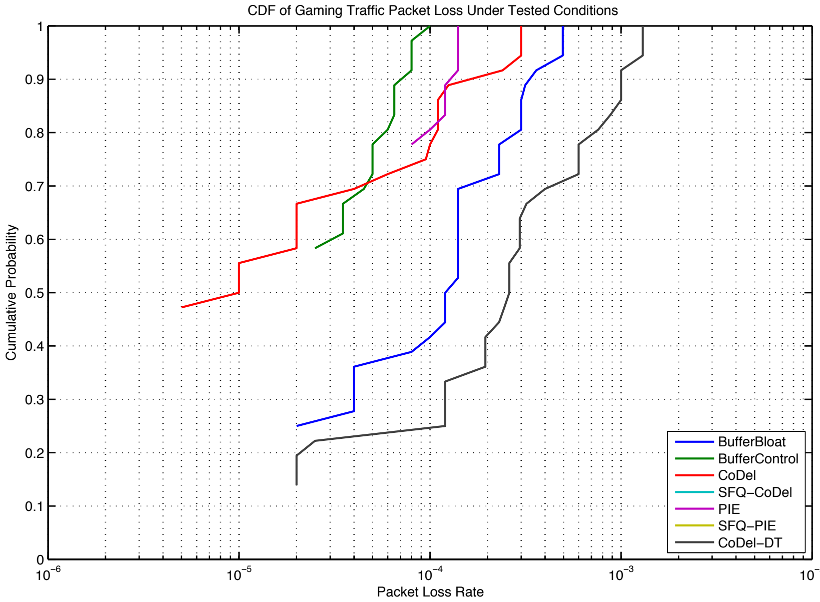

Figure 2 - Packet Loss Statistics for VoIP/Gaming Traffic

In terms of VoIP user experience, recall that MOS is a 5 point scale with the values representing:

MOS	QUALITY
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

The MOS estimator used for this analysis assumes a G.711 codec with a maximum MOS of approximately 4.4, and takes into account the latency, jitter and packet loss experienced by the stream.

The MOS estimator shows that Buffer Control and all of the AQMs provide excellent performance, whereas the Buffer Bloat case provides unacceptable performance in the majority of cases. It can be seen in Figure 3 that only the Buffer Bloat case shows degradation of performance, with all others showing almost no degradation across the full range of test conditions. Figure 4 provides a closer look at the very minimal degradation experienced with Buffer Control or any of the AQM algorithms.

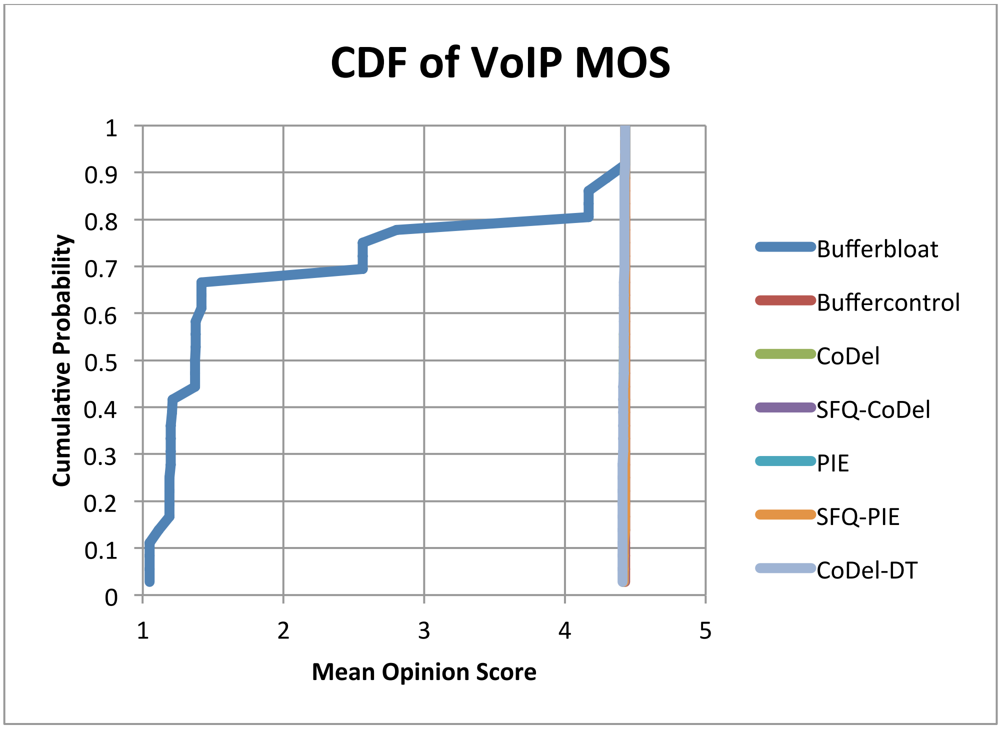

Figure 3 - VoIP MOS Statistics

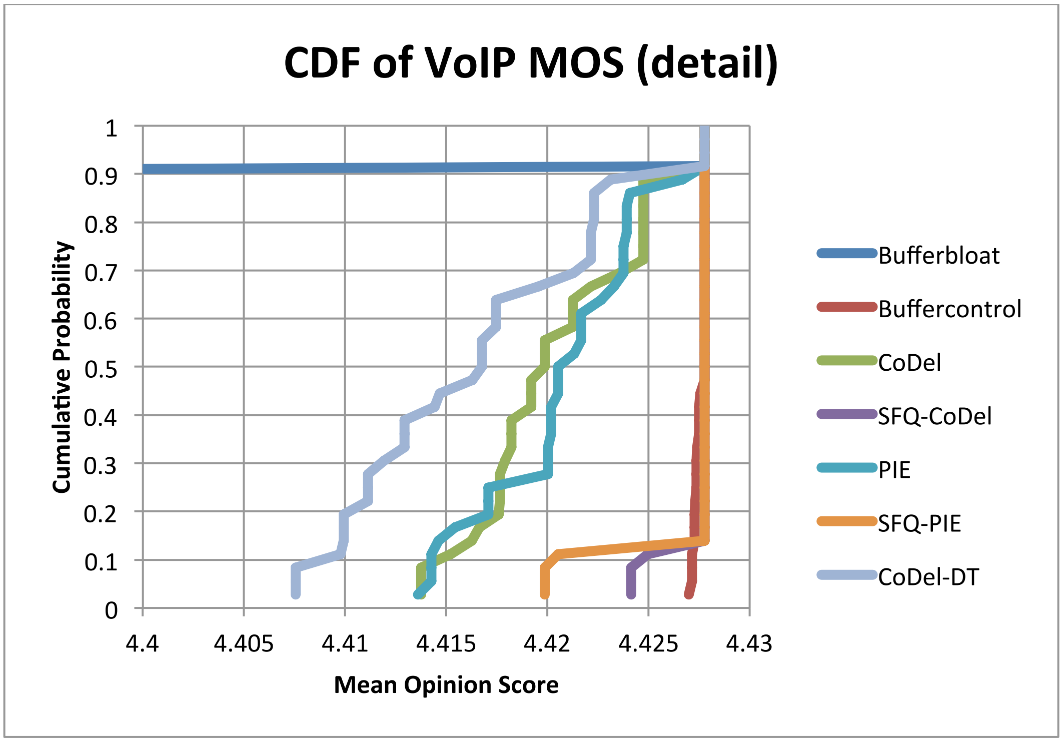

Figure 4 - VoIP MOS Statistics (detail)

5.2 Web Performance

Figure 5 shows the statistics of web page load time. In this case, all of the AQM algorithms provided very good performance, with 90th percentile values between 2.7 and 3 seconds. Buffer Control shows somewhat poorer performance (90%-ile = 3.5 seconds), and Buffer Bloat shows the worst performance (90%-ile = 5.5seconds).

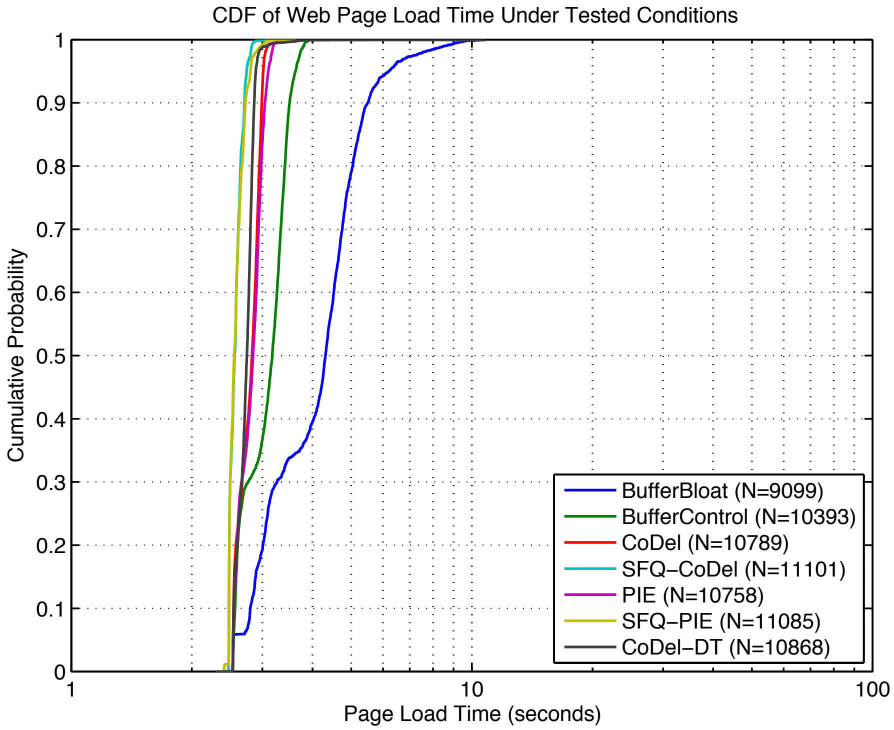

Figure 5 - Web Page Load Time Statistics

5.3 TCP Throughput Performance

In terms of TCP throughput performance, all of the queue management approaches are capable of providing essentially equivalent long-term average performance, especially when the session has a short RTT. Where we see more differences is in the short-term, and at longer RTTs. Figure 6 shows the averaged TCP performance over moderately short time scales for the four values of RTT we simulated. In many locales, and in particular in the United States and other developed regions, RTTs in the 10-50ms range are likely the most common for upload RTT (due in part to the proliferation of geographically distributed data centers).

As mentioned in Section 2.2.2 the data point that most closely represents the result one would expect from using speedtest.net is the average throughput at 10 seconds in the 20ms RTT case. At that data point, we see all of the AQM algorithms performing nearly identically.

Comparing CoDel-DT and PIE at 20ms RTT, we see better performance from PIE for file transfers lasting between 2 seconds and 7 seconds, but identical performance between the two outside of that range. At the 50ms RTT case, we see almost identical performance between CoDel-DT and PIE from 0 to 7 seconds, better performance from PIE between 7 seconds and 18 seconds, followed by better performance from CoDel-DT from 18 seconds until the end of the test at 25 seconds. At 100ms RTT, PIE performs better for the first 20 seconds, but for the remainder of the test CoDel-DT performs as well or better than PIE. At 200ms RTT (a fairly rare case) we see a significant difference between CoDel-DT and PIE, with PIE showing inferior performance. It is important to note here that this situation (200 Mbps link with 200ms RTT) has a very high bandwidth-delay product (BDP) of 5 MB, equivalent to over 3300 packets. When in congestion avoidance, any Reno-based TCP (such as what is present in Windows and Mac OS X) will take a very long time to recover from a congestion window decrease. For example, in a case where 20ms of buffering results in a packet drop, the congestion window will drop from ~3600 packets to ~1800 packets, and then will take 360 seconds to recover. A small number of unlucky packet drops can thus have a significant effect on throughput in this case. More advanced TCPs such as cubic should recover much more quickly in this situation.

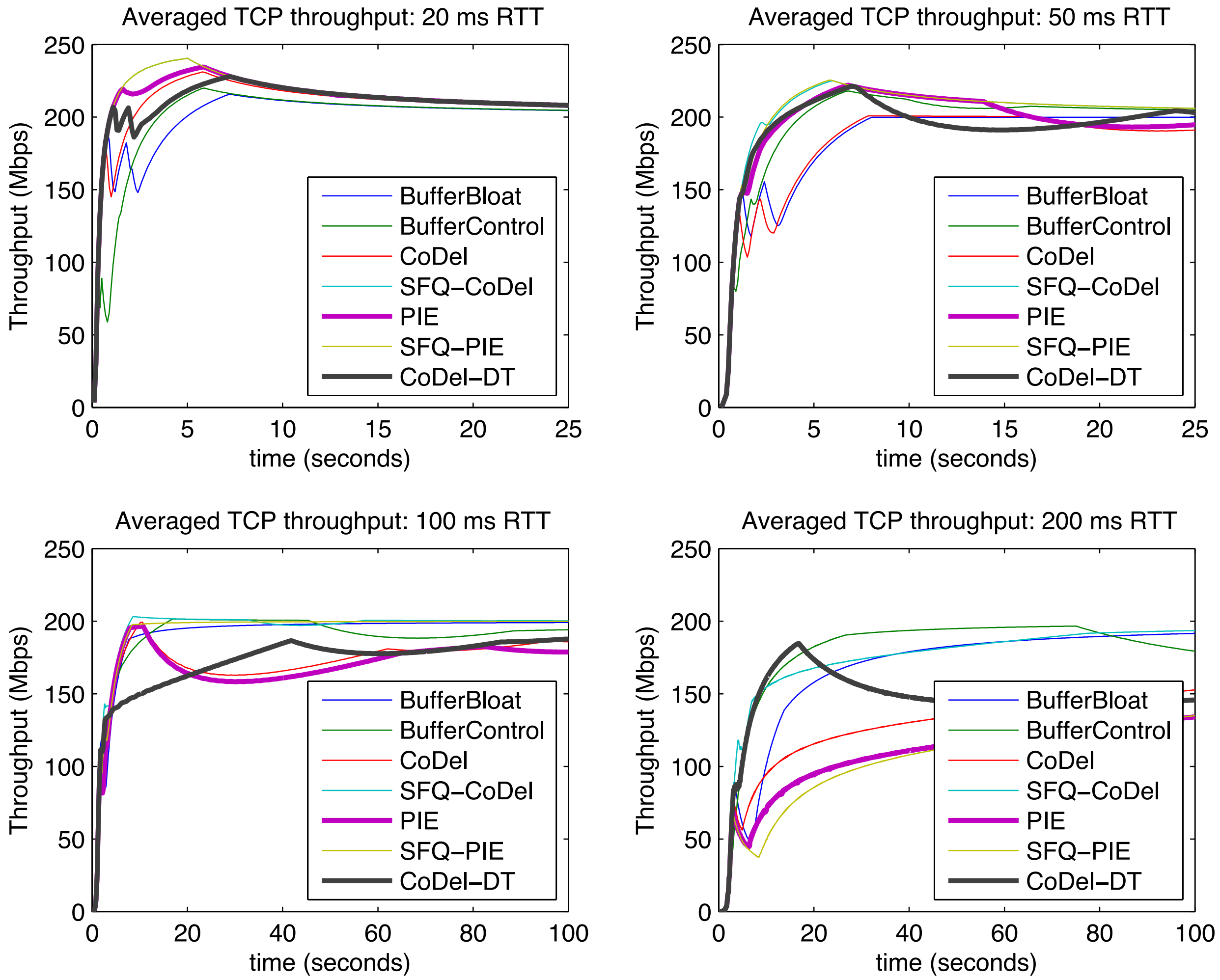

Figure 6 - TCP Performance

6 Algorithm Selection

In summary, the performance testing showed good performance across all of the AQMs. We saw some benefit with the SFQ-based AQMs relative to the single-queue approaches, but not enough in our view to warrant the significant increase in implementation complexity. Between the single-queue AQMs, we see marginal difference in performance between the three approaches (CoDel, PIE, CoDel-DT), with one algorithm slightly out-performing another in certain test conditions, but none of them consistently out-performing or under-performing the other two. As a result, we based our selection of a single-queue AQM approach on the implementation aspects (complexity and feasibility of making implementation changes in the field). Based on these aspects, PIE provides the best attributes.

In preparing the simulation results and the detailed requirements for AQM implementation in DOCSIS 3.1 cable modems, we worked very closely with the CM silicon implementers as well as with Rong Pan of Cisco Systems, the developer of the PIE algorithm. As part of that process, a number of improvements to the original PIE algorithm were made in order to improve performance in the context of a DOCSIS 3.1 cable modem. These changes are documented in [White3]. Some of these changes were made prior to algorithm selection, so their impact is represented in the plots in Section 5. Others, including the latency prediction algorithm were made after algorithm selection.

7 DOCSIS-PIE Latency Prediction algorithm

The PIE algorithm utilizes a departure rate estimator to track fluctuations in the egress rate for the queue. It then utilizes this egress rate estimate along with queue depth to predict queuing latency for use in the drop probability calculation. This departure rate estimator may be well suited to many link technologies, but is not ideal for DOCSIS upstream links for a number of reasons.

First, the bursty nature of the upstream transmissions, in which the queue drains at line rate (up to ~100 Mbps for DOCSIS 3.0 and ~1 Gbps for DOCSIS 3.1) and then is blocked until the next transmit opportunity, results in the potential for inaccuracy in measurement, given that the PIE departure rate estimator starts each measurement during a transmission burst and ends each measurement during a (possibly different) transmission burst. For example, in the case where the start and end of measurement occur within the same burst, the PIE estimator will calculate the egress rate to be equal to the line rate, rather than the average rate available to the modem.

Second, the latency introduced by the DOCSIS request-grant mechanism can result in some further inaccuracy. In typical conditions, the request-grant mechanism can add between ~4 ms and ~8 ms of latency to the forwarding of upstream traffic. Within that range, the amount of additional latency that affects any individual data burst is effectively random, being influenced by the arrival time of the burst relative to the next request transmit opportunity, among other factors.

Third, in a significant majority of cases, the departure rate, while variable, is controlled by the modem itself via the pair of token bucket rate shaping equations described in Appendix C. Together, these two equations enforce a maximum sustained traffic rate, a peak traffic rate, and a maximum traffic burst size for the modem's requested bandwidth. The implication of this is that the modem, in a significant majority of cases, will know precisely what the departure rate will be, and can predict exactly when transitions between peak rate and maximum sustained traffic rate will occur. This can be compared to the PIE estimator, which would be simply reacting to (and smoothing its estimate of) those rate transitions after the fact.

Finally, since the modem is already implementing the dual token bucket traffic shaper, it contains enough internal state to calculate predicted queuing delay with a minimum of computations. Furthermore, these computations only need to be run every drop probability update interval, as opposed to the PIE estimator, which runs a similar number of computations on each packet dequeue event.

For these reasons, the DOCSIS-PIE algorithm utilizes the configuration and state of the dual token bucket traffic shaper to translate queue depth into predicted queuing delay, rather than implementing the departure rate estimator defined in PIE.

8 DOCSIS-PIE Packet Drop De-Randomization

The PIE algorithm generates and applies a drop probability on incoming packets in order to manage queue depth. The drop decisions are made using an independent and identically distributed (iid) Bernoulli random variable with p equal to the calculated drop probability. While this results in a long-term drop probability equal to the calculated value, the possibility for localized excursions in drop probability (over a small number of packets) from the calculated value is fairly high. In fact, by examining the statistics of the run-length (defined here as the number of packets in a sequence, for which all but the last packet are forwarded) it can be seen that the most likely run-length is 1 (i.e. back-to-back packet drops) regardless of the value of p. Figure 7 shows the probability mass function (PMF) of the run-length for p=0.01. For this value of p, the mean run-length is 100 packets (1/p), but it is clear that there is a high probability that either shorter or much longer run-lengths will be experienced.

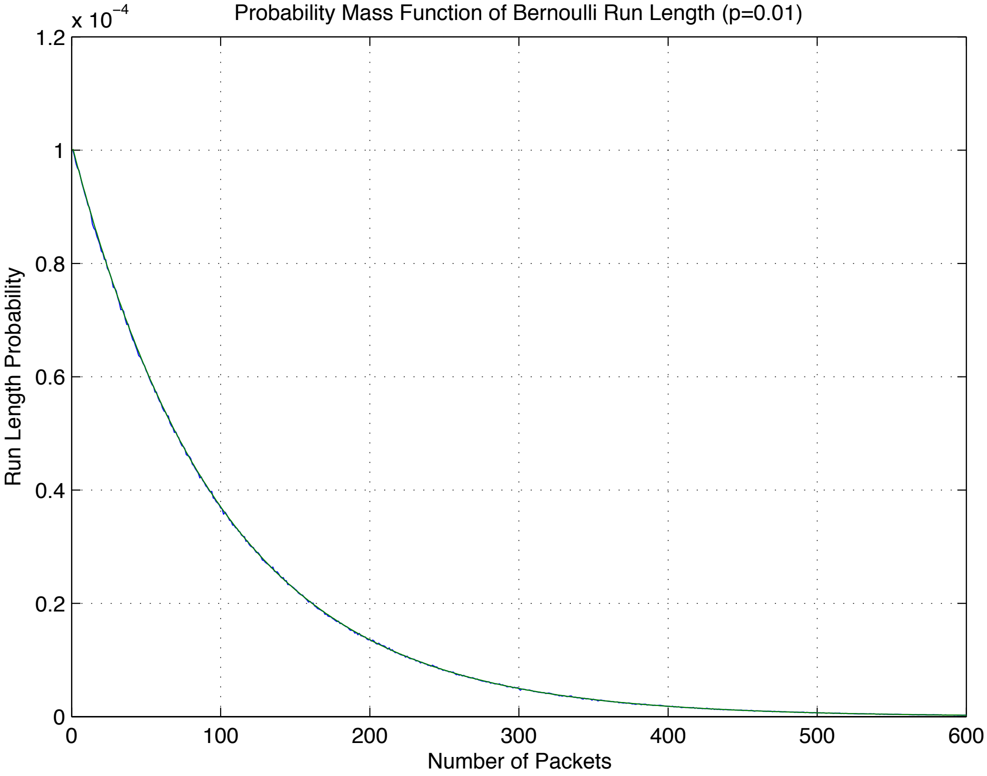

Figure 7 - PMF of Bernoulli Run Length

For another view on this, we can examine the probability mass function of the binomial distribution (the probability of n successes in k Bernoulli trials). For p=0.01 the number of drops in 1000 packets can be approximated by a Gaussian distribution with mean=np=10 and variance=np(1-p)=9.9, as shown in Figure 8.

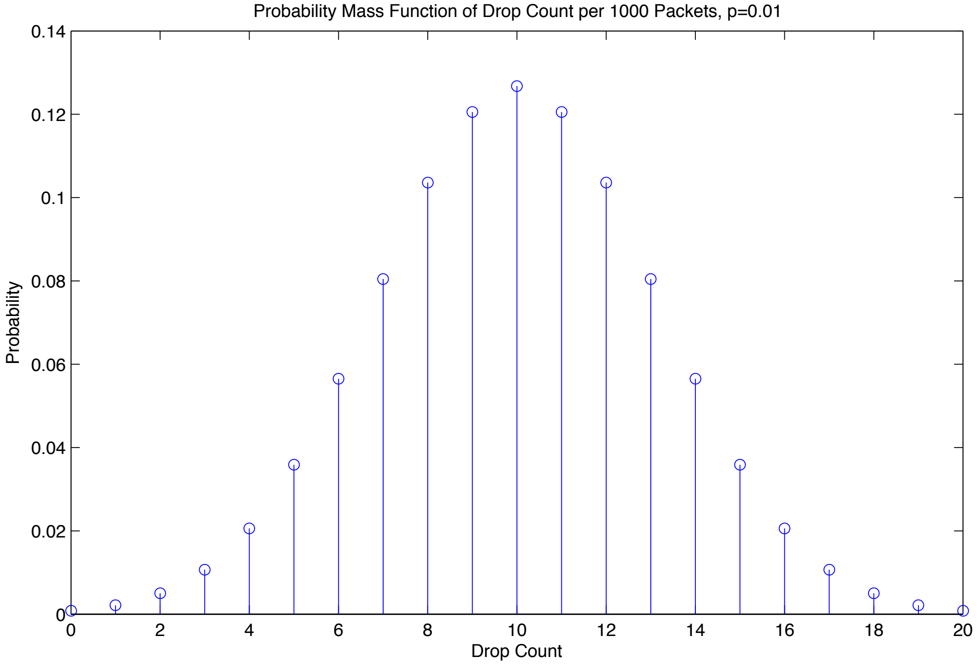

Figure 8 - PMF of Binomial Distribution

Again, it can be seen that significant deviations (e.g. ±50%) from the expected value of 10 drops are quite common. These deviations are a cause for concern because they have the potential to negatively impact performance over short time scales. In particular, performance of a single TCP session is an important metric for assessing system performance, and a single TCP session will be most sensitive to significant deviations in drop probability, with the result being excessive reductions in the sender's congestion window (and thus, poor throughput) if the local drop rate is too high, or excessive increase in the sender's congestion window (and thus, high queuing latency) if the local drop rate is too low. This is especially true for connections with a high bandwidth delay product, where recovery from a reduced congestion window can take many seconds.

To reduce the possibility of these events occurring, DOCSIS-PIE implements a de-randomization function that prevents the extreme excursions in local drop probability. The algorithm simply limits the run-length of the Bernoulli random variable, constraining it to be within 0.85/p and 8.5/p as shown in Figure 9 (for p=0.01). This is implemented by suppressing packet drops until the run length is greater than 0.85/p, and forcing a packet drop if run length equals 8.5/p.

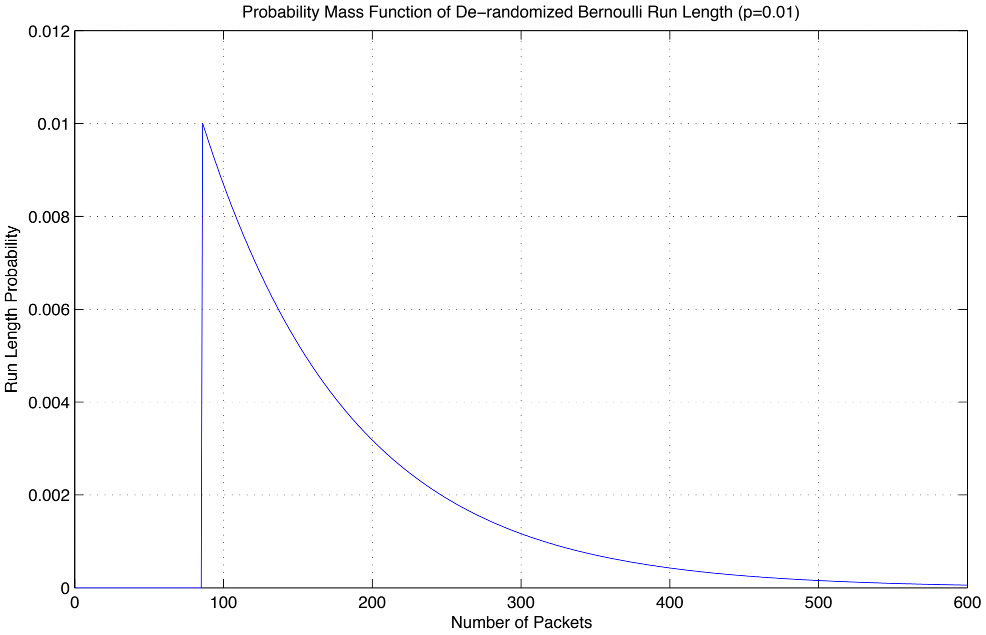

Figure 9 - PMF of De-Randomized Bernoulli Run Length

One side-effect of this simple run-length limitation is that the effective drop rate no longer matches the value of p in the Bernoulli random variable. As is described in more detail in Appendix B, for values of p<0.1, the effective drop probability after de-randomization (p_{eff}) is approximately 0.541*p. This is not concerning, since p is calculated via a feedback control process, so the result is that p settles in to the correct value to produce a p_{eff} that will achieve the desired queuing latency.

Figure 10 shows the resulting probability mass function of the drop count in 1000 packets using the de-randomized approach, here it can be noted that the probability of significant deviations from the expected value is greatly reduced.

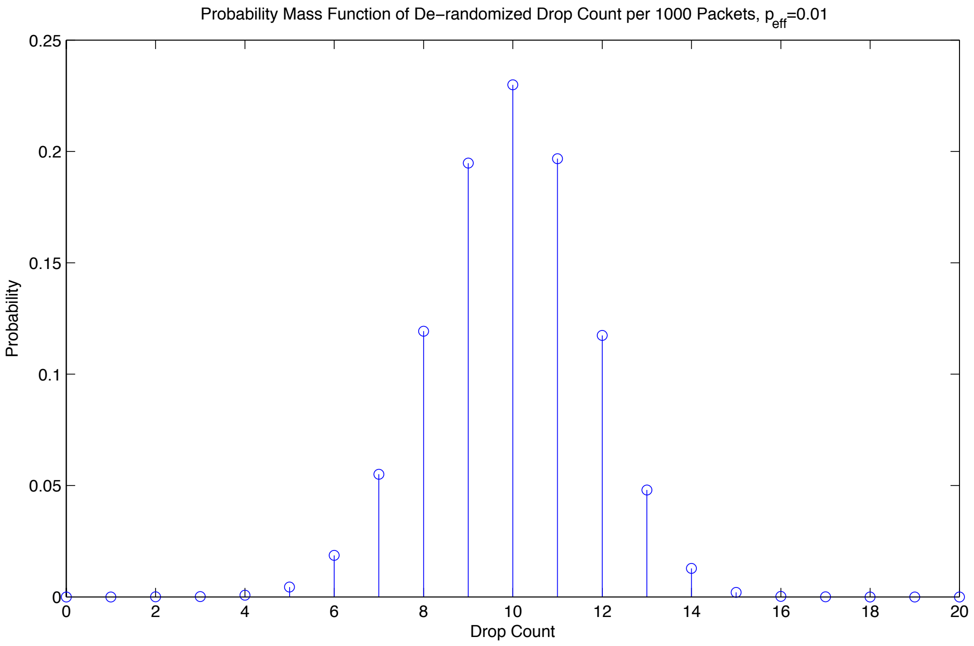

Figure 10 - PMF of De-Randomized Binomial

9 OPERATIONAL ASPECTS

The selection of an AQM algorithm for the DOCSIS specifications included the desire that the algorithm "just work" without tuning or configuration by the operator. We believe that we have accomplished this for a wide variety of use cases with the selection of the DOCSIS-PIE algorithm and choice of default values for AQM parameters. However, in recognition of the possibility that these choices may not be ideal for certain specific use cases, we define some configuration options.

For configuration, the cable modem configuration file (boot file) syntax has been extended or modified in the following ways:

1. A new top-level type-length-value (TLV) parameter that can be used to disable upstream AQM on all service flows
2. A new per-service-flow TLV that can be used to disable AQM on a per-service-flow basis.
3. A new per-service-flow TLV that can be used to set the latency target for the service flow.
4. A default per-service-flow buffer size of 250ms at the Maximum Sustained Traffic Rate.

In addition, the cable modem management information base (MIB) is updated to report the settings for the above configuration parameters, as well as a distinct, per-service-flow counter for AQM dropped packets.

APPENDIX A DEQUEUE RATE TRACKING FOR EXTREMELY RF CONGESTED LINKS

CableLabs conducted experiments on dequeue rate tracking algorithms that could replace the original algorithm in PIE in order to provide a more accurate estimate of congested link capacity. The view was that such an algorithm could be used in conjunction with the token bucket based approach, whereby the estimated link capacity would be used in place of the token bucket rate(s) when the service flow is RF-channel limited.

For these studies, a highly congested link model was used, where the service flow's Maximum Sustained Traffic Rate is 20 Mbps, and the RF link capacity varies between 10 Mbps, 12Mbps, 18Mbps, and 20 Mbps.

Figure 11 and Figure 12 show the accuracy of the PIE dequeue rate tracker (labeled "avg_dq_rate") in the simulation, as well as an early version of a replacement (labeled "new algo."), both compared to the actual link capacity (labeled "Actual Rate"). As was discussed in Section 7, the PIE rate tracking algorithm is unfortunately not well suited for measuring departure rate for DOCSIS links.

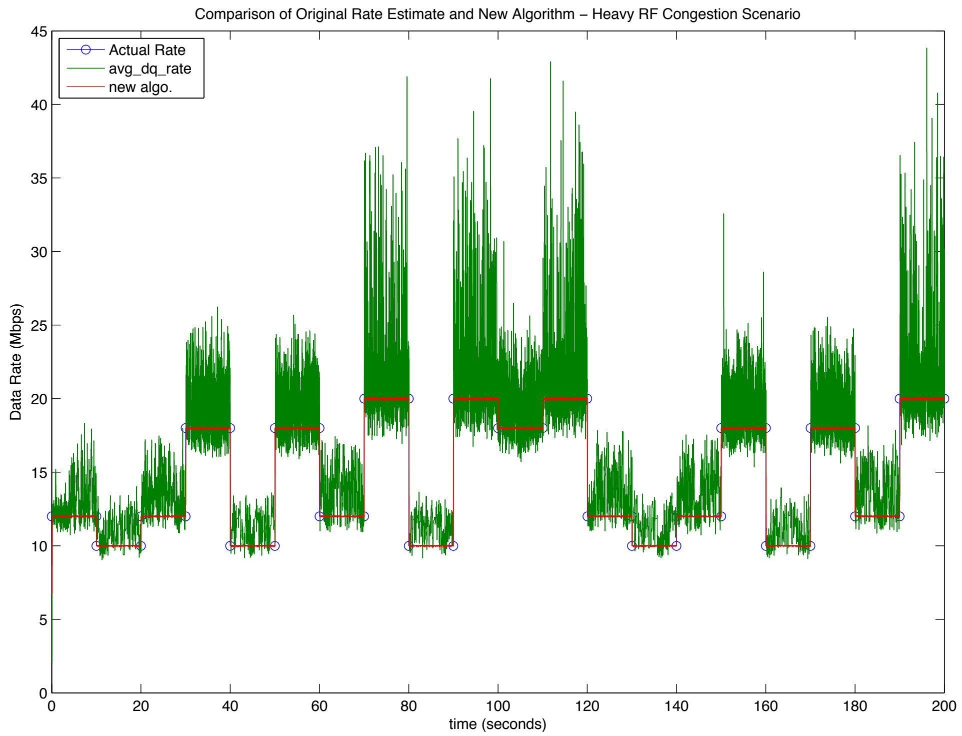

Figure 11 - Default PIE Dequeue Rate Estimator

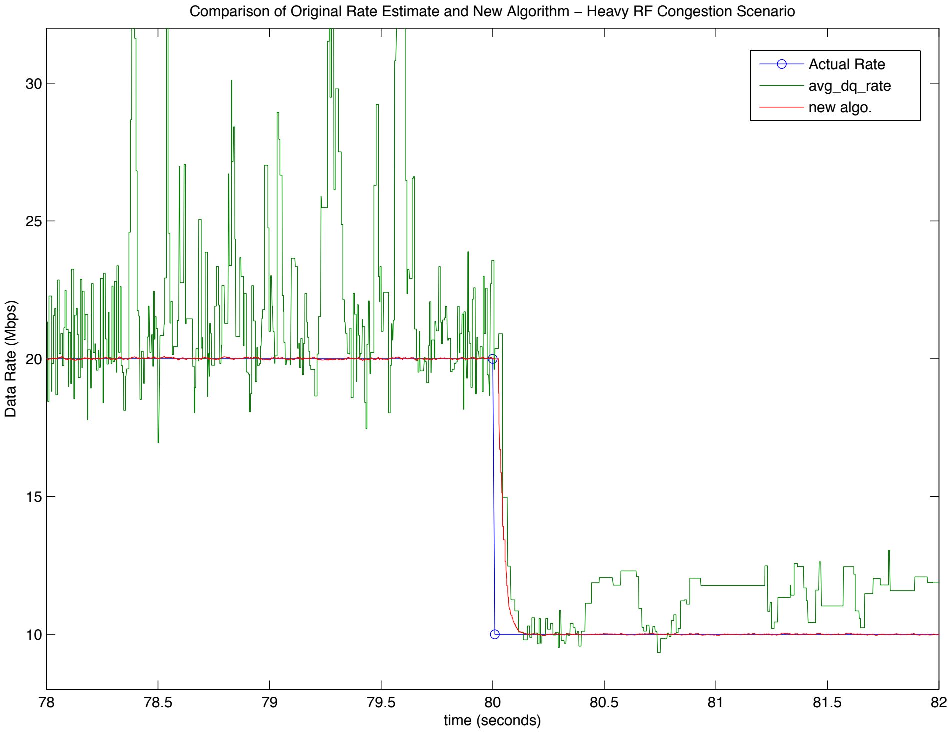

Figure 12 - Default PIE Dequeue Rate Estimator - detail

Through further analysis and simulation, the "new algorithm" shown in Figure 11 and Figure 12 was refined to reduce computational complexity and further improve accuracy. The simulation results for this algorithm are shown in Figure 13 and Figure 14.

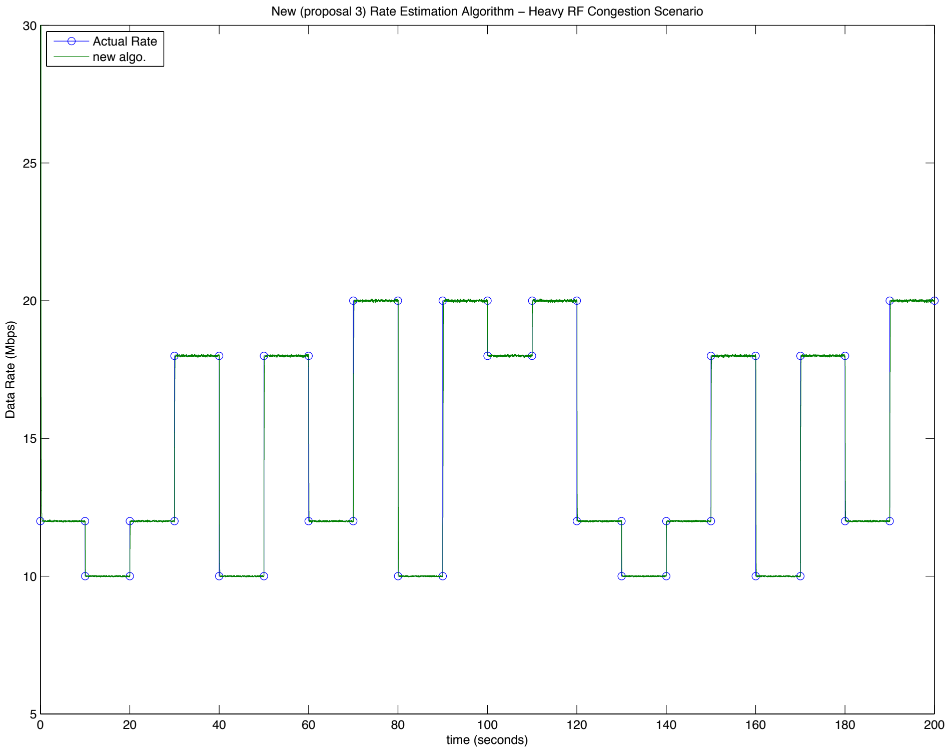

Figure 13 - Proposed DOCSIS Technology-aware Dequeue Rate Estimator

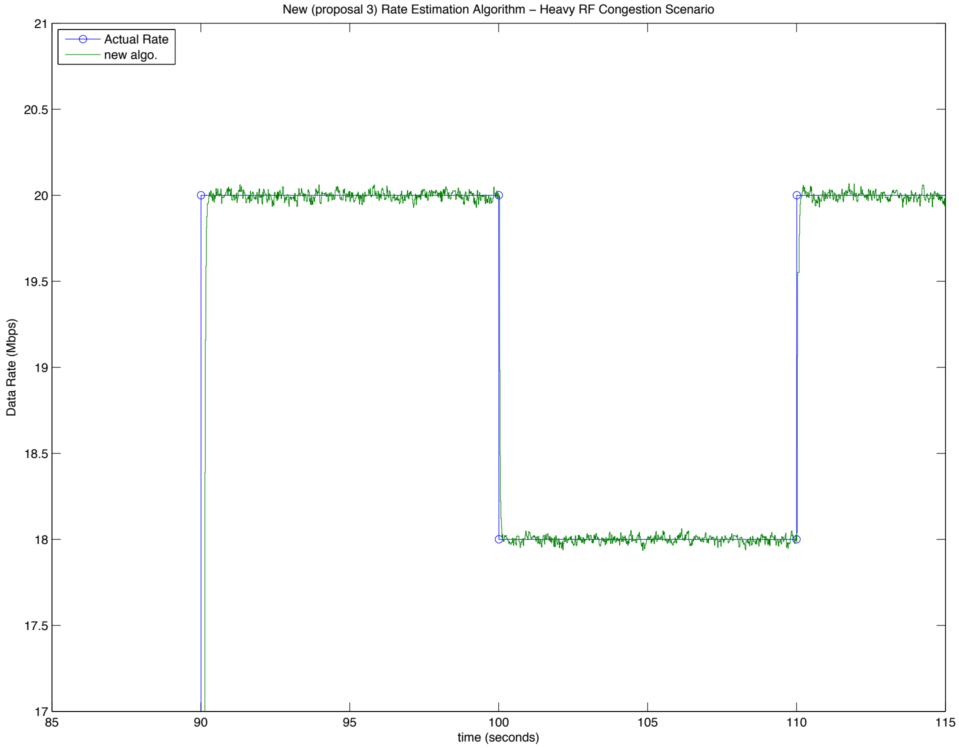

Figure 14 - Proposed DOCSIS Technology-aware Dequeue Rate Estimator - detail

The algorithm is as follows:

```
On Initialization:
        dq_count_ = 0;
        pending_ = true;
        rate_estimate = Maximum Sustained Traffic Rate

On each MAP arrival:
dq_count_ += granted_bytes;
if (acked_req_bytes == 0) pending_ = false;

On each drop probability calculation instance (every 16 ms):
        inst_rate = 8 * dq_count_ / 0.016;
        if (pending_ || (inst_rate > rate_estimate)) {
                rate_estimate = 0.5 * rate_estimate + 0.5 * inst_rate;
        }
        dq_count_ = 0;
        pending_ = true;

    where:
        granted_bytes = number of bytes granted to the service flow in the current MAP
        acked_req_bytes = requested bytes that have been acknowledged but not
                granted (pending grants)
```

This algorithm tracks the congested rate sufficiently accurately as long as the modem continues to make requests. If the modem ceases making requests (e.g. due to lack of user traffic) the congested channel rate estimate ("rate_estimate") can become stale. Development of a method to detect and handle this condition is left for future study.

As mentioned above, the congested channel rate estimate would be used as follows:

```
    b = current tokens from MSR token bucket

    pr_limited = min(rate_estimate, peak_rate);
    msr_limited = min(rate_estimate, max_sustained_rate);

    if (bytes_in_queue <= b) {
        qdelay = bytes_in_queue / pr_limited;
    } else {
        qdelay = ((bytes_in_queue - b) / msr_limited +  b / pr_limited);
    }
```

A.1 Comparison to Token Bucket based approach

Using this algorithm, the modem can track congested channel rate and use it to generate an estimate of queuing latency. A simpler alternative is to ignore the effects of channel congestion and simply assume that the service flow is always rate-shaper limited. The prevailing view among network operators is that prolonged channel congestion is relatively rare in current system deployments. This may be due to the fact that offered maximum sustained rates are increasing faster than consumption, so average activity levels are falling. Nonetheless, we simulated the performance of the combined token-bucket and rate-tracking algorithm and compared it to using token-bucket rate shaper alone. We simulated a heavy traffic scenario both in the case of no RF congestion (where the two would be expected to provide identical performance) and an extreme RF congestion scenario - one where the user is only able to achieve 30% of their max sustained rate. The results are shown in Figure 15, where the token-bucket-only approach is labeled "b" and the combined token-bucket and rate-tracking approach is labeled "a4.5". While it is clear that the rate-tracking approach is able to provide better performance in the extreme congestion case than utilizing only the token bucket, the view of cable operators was that the likelihood of experiencing such an extreme degradation of link performance was very low, and given that service is already significantly degraded, optimizing the performance for such an unlikely situation was not needed and that performance of the token-bucket only algorithm was sufficient.

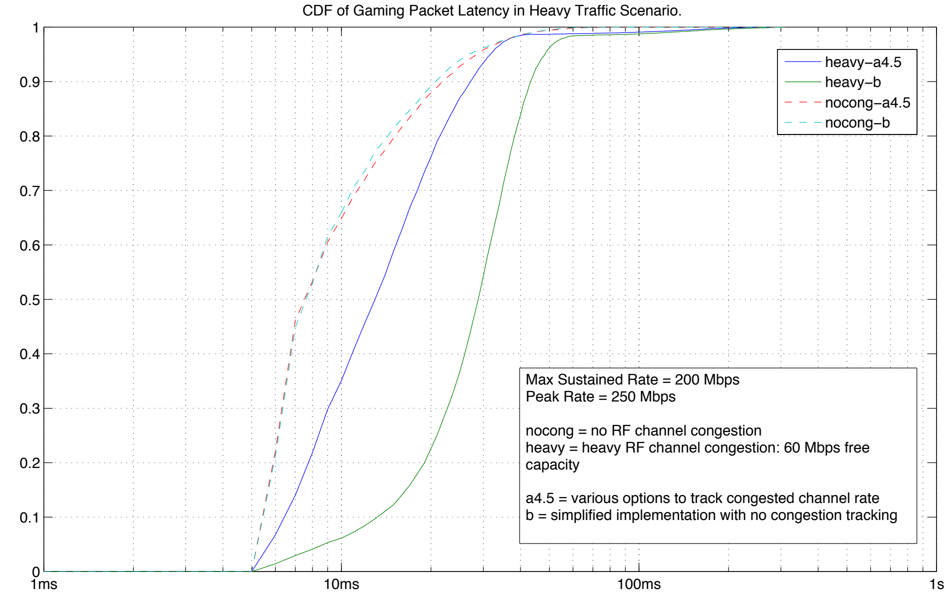

Figure 15 - Comparison of Rate Tracking Approach to Non-Rate Tracking Approach

APPENDIX B DE-RANDOMIZER BEHAVIOR

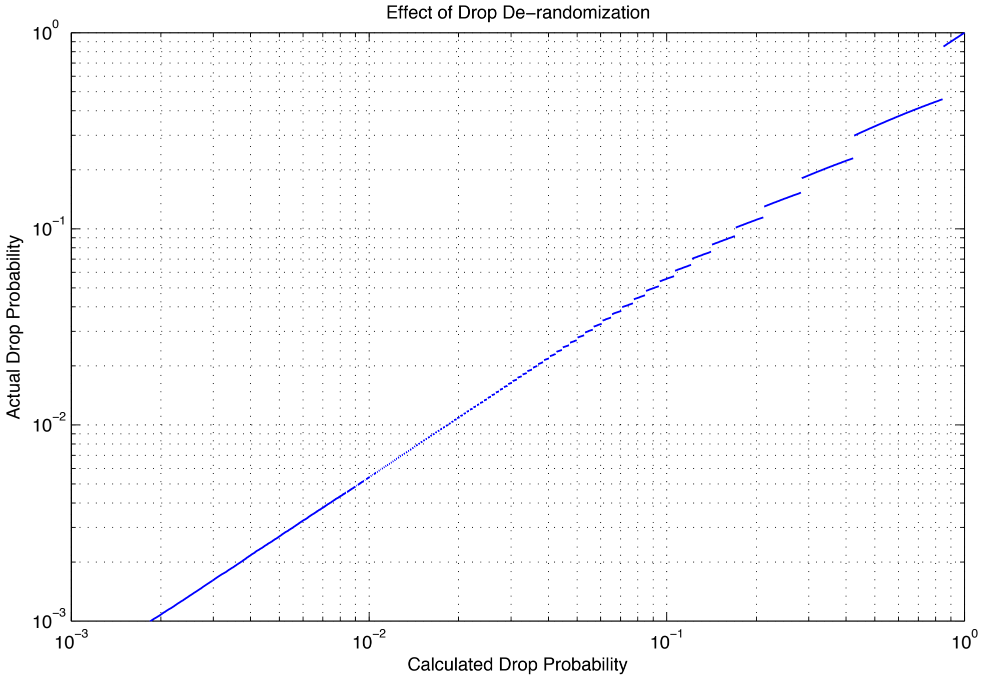

Figure 16 - Drop De-randomizer Behavior

As Figure 16 shows, the actual drop probability produced by the algorithm is typically less than what might be inferred by the value calculated for the "drop probability" variable p. For values of $p < 10^{-1}$, the actual drop probability (p_{eff}) is approximately 0.541*calculated. For high values of drop probability (i.e. values greater than 10^{-1}) there are discontinuities in the relationship that become more pronounced.

In traditional control systems theory, these discontinuities may be cause for concern, since they will produce oscillations in the steady state behavior. For example, the algorithm is unable to produce a calculated drop probability that would result in an actual drop probability of 0.6. If the stable operating point of the system were to require an actual drop probability of 0.6, the AQM would oscillate between values above ($p_{eff} > 0.85$) and below ($p_{eff} < 0.46$) the desired rate. This behavior can be seen experimentally if the AQM is driven with unresponsive traffic at a rate 2.5x the output rate.

In the context of a real network, these discontinuities cause little problem for a couple of reasons. First, steady-state conditions where the AQM-induced drop probability is greater than 10^{-1} are not generally expected. With a TCP traffic load, packet loss rates consistently greater than 10^{-1} can occur as a result of the congestion avoidance algorithm, but this will generally happen only if the upstream rate is extremely limited (e.g. below 100 kbps [Padhye]), which is unlikely for DOCSIS 3.x. With a UDP (or other non-responsive) traffic load, packet loss rates consistently greater than 10^{-1} can occur if the traffic rate is consistently more than 10% greater than the upstream rate, but this is also unlikely except in the event of a runaway process. Second, the performance of many applications will likely be impacted more due to the packet loss itself rather than due to any fluctuations in packet drop rate resulting from these discontinuities. Third, real-world traffic rarely looks like a steady-state condition, so any fluctuation in drop probability resulting from these discontinuities is likely to be masked by fluctuations in traffic load.

APPENDIX C RATE SHAPING IN DOCSIS SERVICE FLOWS

It is typical that upstream and downstream Service Flows used for cable broadband Internet access are configured with a Maximum Sustained Traffic Rate. This QoS parameter rate-shapes the traffic onto the DOCSIS link, and is the main parameter that defines the service offering. Additionally, it is common that upstream and downstream Service Flows are configured with a Maximum Traffic Burst and a Peak Traffic Rate. These parameters allow the service to burst at a higher (sometimes significantly higher) rate than is defined in the Maximum Sustained Traffic Rate, for the amount of bytes configured in Maximum Traffic Burst, as long as the long-term average data rate remains at or below the Maximum Sustained Traffic Rate.

Mathematically, what is enforced is that the traffic placed on the DOCSIS link in the time interval (t_1,t_2) complies with the following rate shaping equations:

$$\mathrm{TxBytes}(t_1,t_2) <= (t_2-t_1)*R/8 + B$$

$$\mathrm{TxBytes}(t_1,t_2) <= (t_2-t_1)*P/8 + 1522$$

for all values $t_2>t_1$, where:

R = Maximum Sustained Traffic Rate (bps)

P = Peak Traffic Rate (bps)

B = Maximum Traffic Burst (bytes)

The result of this configuration is that the link rate available to the Service Flow varies based on the pattern of load. If the load that the Service Flow places on the link is less than the Maximum Sustained Traffic Rate, the Service Flow "earns" credit that it can then use (should the load increase) to burst at the Peak Traffic Rate. This dynamic is important since these rate changes (particularly the decrease in data rate once the traffic burst credit is exhausted) can induce a step function increase in buffering latency.

APPENDIX D REFERENCES

[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.

[LEDBAT] D. Rossi, C. Testa, S. Valenti, L. Muscariello, LEDBAT: the new BitTorrent congestion control protocol, International Conference on Computer Communications and Networks (ICCCN 2010), Zurich, Switzerland, August, 2010.

[Hoeiland] T. Hoeiland-Jorgensen, P. McKenney, D. Täht, J. Gettys, E. Dumazet, "FlowQueue-Codel", IETF draft-hoeiland-joergensen-aqm-fq-codel

[Nichols] K. Nichols, V. Jacobsen, "Controlled Delay Active Queue Management", IETF draft-nichols-tsvwg-codel

[Padhye] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP Throughput: A Simple Model and its Empirical Validation, Proceedings of the ACM SIGCOMM Conference on Network Architectures and Protocols, pp. 303--314, Vancouver, CA, 1998

[Pan] R. Pan, et al., "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", IETF draft-pan-tsvwg-pie

[Pollere] Notes on the Algorithm and its Implementation, http://pollere.net/CoDelnotes.html

[Shreedhar] M. Shreedhar, G. Varghese, "Efficient fair queueing using deficit round robin". ACM SIGCOMM Computer Communication Review, October 1995.

[White] G. White, J. Padden, "Preliminary Study of CoDel AQM in a DOCSIS Network", http://www.cablelabs.com/downloads/pubs/PreliminaryStudyOfCoDelAQM_DOCSISNetwork.pdf

[White2] G. White, D. Rice, "Active Queue Management Algorithms for DOCSIS 3.0", http://www.cablelabs.com/downloads/pubs/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf

[White3] G. White, R. Pan, "A PIE-based AQM for DOCSIS Cable Modems", IETF draft-white-aqm-docsis-pie