Evaluating CoDel, PIE, and HRED AQM Techniques with Load Transients

Ilpo Järvinen and Markku Kojo

Department of Computer Science
University of Helsinki
Email: ilpo.jarvinen@helsinki.fi, markku.kojo@cs.helsinki.fi

***Abstract*—In the past, networks have been mainly optimized for good system throughput but recently achieving low delay has also gained notable traction. Active Queue Management (AQM) has long been recognized necessary for operating Internet routers with shorter standing queues but only limited deployment has occurred. The recent interest in AQM has resulted in new AQM proposals. In this paper we evaluate CoDel (Controlled Delay) and PIE (Proportional Integral controller Enhanced), both being new AQM proposals, and compare the performance against an aggressive RED (Random Early Detection) variant called HRED (Harsh RED). We focus on AQM behavior during load transients typically occurring at the network edge with the common traffic types of today such as Web transactions. We discover that CoDel auto-tuning does not scale well with the load. With the high-end delays experienced, HRED is better than PIE and CoDel when more than a few simultaneous flows share the bottleneck.**

***Index Terms*—Active Queue Management (AQM); TCP; Slow start; Queuing delay**

I. Introduction

Traffic in the Internet of today differs significantly from that of the early Internet. A significant portion of the user-generated traffic is short transfers, for example, Web traffic. Also Web traffic has evolved as the pages have grown in complexity. In order to minimize page download latencies, Web browsers typically allow multiple Web objects to be transferred in parallel but only up to a limit per host or domain [8], [44]. Many Web sites are not satisfied with the latency offered by this off-the-shelf parallelism and resort to a technique called domain sharding in which the Web site artificially inflates the number of hostnames that are used within a single site. Therefore, a Web transaction may consist of a rather large number of parallel object transfers overlapping to various degree.

Transmission Control Protocol (TCP) [40] which acts as the workhorse for a large portion of the Internet traffic, Web traffic included, has been specified decades ago. TCP implements congestion control function [24], [5] that controls how fast data can be injected into the network. It has two major operating modes: slow start and congestion avoidance. Initially, slow start was intended to only transit a TCP flow into equilibrium where from congestion avoidance takes over and continues [24]. However, the current way of using TCP with short transfers has entirely reversed the situation. As typical traffic patterns include many parallel TCP flows each performing slow start, this tends to generate load transients to the network and the flows often even complete before it is possible to reach the equilibrium. Timescale of such load transient is very short, typically from milliseconds to a few seconds depending on the end-to-end path capacity and round-trip time.

Optimizing networks for high throughput has led to an increase in buffer sizes of various network devices because memory capacity has increased together with lower cost associated with larger buffers. In general, for high throughput a larger buffer is better. TCP congestion control continuously drives the network to drop packets after which it backs off and then the process repeats. This TCP probing combined with large, simple FIFO buffers leads to a significant buffer build-up called bufferbloat [16] because drops occur only when the buffer becomes full. The buffer build-up then is visible to all flows sharing the same queue as increasing queuing delay. However, simply making buffers small to optimize for delay is not enough as the buffers also serve an important role in absorbing short-term bursts. Therefore, buffering is often considered harmful only when a standing queue forms.

Often only a trivial queue management using FIFO/drop-tail principles is applied to the buffers in network devices, disregarding any advice on implementing Active Queue Management (AQM) [7]. The purpose of AQM is to enable an Internet router to inform the senders that they need to reduce sending rate before the buffer becomes full, allowing routers to maintain small queues and avoid bufferbloat. The awareness of widespread bufferbloat has recently increased interest in AQM again and resulted in a few new AQM proposals in addition to the traditional AQM techniques such as Random Early Detection (RED) [14]. CoDel (Controlled Delay) [33], [35] and PIE (Proportional Integral controller Enhanced) [36], [37] represent such new state-of-the-art AQM algorithms. Recently also IETF has formed a new AQM working group [1] to focus on better algorithms for managing queues.

In this paper, we perform a simulation study to evaluate how well CoDel and PIE manage load transients as we suspected there would be potential shortcomings in that area. For comparison we use HRED (Harsh RED) [26] which we built earlier on top of RED. Our study significantly differs from earlier studies on PIE and CoDel performance [28], [46], [47], [17], [18], as we focus on very short-term transient effects

978-1-4799-3780-6/14/$31.00 ©2014 IEEE

that are typical with the traffic patterns of today and do not introduce long-running TCP flows at all. The dynamics during load transients have in general received too little attention in AQM research, whereas steady-state behavior is much better covered. The load transients introduced by series of short transfers tend to generate delay spikes that co-existing real-time traffic, such as interactive audio, video, or gaming traffic, sharing the same bottleneck easily experiences as distortions in media quality and lag. In addition, the metrics used in the earlier studies often summarize the results in such a way that transient behavior becomes totally invisible. Our simulation results confirm our suspicions as neither CoDel nor PIE really handles load transients well, and HRED outperforms both in the load transient handling. We also experimented with SFQ CoDel that is a combination of Stochastic Fair Queuing (SFQ) [30] and a CoDel-enabled queue for each SFQ bucket, as it is recommended that CoDel is deployed with multiple queues instead of a single-queue CoDel only [35]. While the results with SFQ CoDel are more promising than with the single-queue CoDel, there are issues in deploying multiple queues with AQM and also some room for improvement with it.

The rest of the paper is organized as follows. In Section II we discuss the related work. Section III introduces the AQM proposals we study and Section IV describes the simulation arrangements. In Section V we present the results, and we continue by discussing additional thoughts on AQM in Section VI. Finally we conclude the paper in Section VII.

II. Related Work

Bufferbloat and its problems are described in [16]. Existence of bufferbloat with particular wireless link technologies was measured in [27], [22]. The recent measurements in [3] used peer-to-peer traffic as a vessel to probe the network conditions experienced by a large number of Internet hosts in order to estimate bufferbloat, and the existence of bufferbloat was questioned. However, the use of peer-to-peer flows likely distorts the results significantly because the peer-to-peer traffic is mostly autonomous machine-to-machine traffic whose latency is largely irrelevant. Because the amount of peer-to-peer traffic easily becomes much larger than that of user generated counterpart, it is expected to dominate the result data over the typical user generated traffic. Therefore, any statistical analysis over a full set of packets diminishes the interesting portion of traffic to the invisible upper-end of the cumulative distribution. In addition, the peer-to-peer traffic TCP flows are likely to spend most of the time in the TCP congestion avoidance mode, whereas user generated TCP flows such as Web traffic spend a significant portion of their lifetime in TCP slow start.

In [28] the authors experimented with CoDel, PIE, Adaptive RED (ARED) [12], and a limited set of tests with FQ_CoDel [20]. Both wired and WLAN network setups were used with a varying number of long-running TCP flows. The main conclusion was that also ARED performs reasonably well, or even better than PIE and CoDel. However, with a very low number of TCP flows the goodput with ARED was lower compared to PIE and CoDel because of the bottleneck link under-utilization. PIE performed better than CoDel or ARED when the channel access latency in WLAN was a significant factor.

CableLabs has carried out a simulator-based study [46], [47], [48] on performance of CoDel, CoDel-DT, SFQ CoDel, PIE, and SFQ PIE in order to support the selection of the default AQM to be implemented in Data Over Cable Service Interface Specification (DOCSIS) 3.1 [9] conforming cable modems. Various combinations of Voice-over-IP (VoIP), Web, Constant Bit Rate (CBR), and FTP traffic were tested. TCP congestion control variants under experimentation included Cubic [19], [42], TCP Reno, and low priority LEDBAT [43]. Both CoDel and PIE were shown to provide good or very good latency. In most cases PIE outperformed CoDel. DOCSIS 3.1 choose to not mandate fair queueing due to added implementation complexity [48].

A simulation study on PIE, CoDel, ARED, and FIFO is conducted in [18]. The traffic workload is a mixture of long-lived TCP flows, Web, video streaming, and VoIP traffic. AQM was found to reduce latency at the cost of increase in packet loss rate. CoDel was found to be the most aggressive AQM, whereas PIE performed the best. Unfortunately, the metrics used do not highlight the worst-case delays, and the load itself is not very volatile because of the long-lived TCP flows.

Low priority TCP traffic is tested together with AQM in [17]. The main conclusion was that the use of AQM makes low priority TCP variants to become elevated in priority equal to normal flows.

Data-Center TCP (DCTCP) [2] is another new proposal to address bufferbloat. DCTCP is a combination of AQM, end-host TCP modifications, and enabling Explicit Congestion Notification (ECN) [41] to signal congestion. It tries to leverage existing RED deployment for the AQM by configuring RED to a degenerated mode where RED "average queue" tracks instantaneous queue without any averaging. DCTCP AQM then becomes simply a step function where ECN is used to signal if the queue is above a configurable threshold. Because a large number of ECN marked packets are triggered when the queue exceeds the threshold, the end host compensates for it depending on the severity of the detected congestion. Therefore, DCTCP may back off less on congestion than a traditional TCP would. Less dramatic backoff allows operating shorter queue lengths without under-utilizing the bottleneck on backoff but also introduces danger of unfairness compared with the traditional TCP. In addition to potential fairness problems, DCTCP deployment is challenging because of required changes both to end hosts and routers. Therefore it is different from AQM only solutions such as RED, CoDel, or PIE.

III. Active Queue Management

Using Active Queue Management (AQM) has been recommended over a decade ago [7] but only limited deployment has taken place even though many routers come with RED [14] capability, but it is off by default. The recently introduced

new AQM proposals, CoDel and PIE have also been actively pushed to deployment.

CoDel design goals introduce the notion of "good queue" and "bad queue", and CoDel aims to solve the latter one which is considered to be a persistently full buffer. CoDel monitors the queuing delay of packets in the router and starts reacting if the measured queuing delay is above a *target* delay (5 msecs by default). CoDel discerns good queue from bad queue by delaying response to delay growth by a configurable *interval* (100 msecs by default) after it first detects queuing delay above *target*. Once CoDel enters to the dropping state it maintains a *count* that is increased by one after every drop. CoDel auto-tunes the dropping distance by dividing the *interval* by square root of the *count*. The next drop is scheduled to occur when the current time is past the dropping distance. As such, there is no random dropping in CoDel. The drops are recommended to be performed from the head of the queue in order to reduce the feedback delay to the sender. When the delay falls below *target*, CoDel leaves the dropping state. If dropping state was recently used when re-entering the dropping state, a recall for the *count* is performed immediately instead of starting from scratch. Multiple variants exist for formula that is used to determine the actual value of *count* when the recall occurs.

In all fairness, those advocating CoDel admit that CoDel is not expected to be effective during TCP slow start [45]. The problems with slow start are circumvented by introducing flow isolation with some variant of fair queuing. Therefore, CoDel is recommended to be deployed with a variant of SFQ called Flow Queuing CoDel (FQ_CoDel) [20]. FQ_CoDel implements the shortest queue first optimization [6] for the flows that do not build a queue. As such, FQ_CoDel is not equal to SFQ CoDel but closely related. We believe, however, that enabling any variant of fair queuing may not be practical with many link technologies at least within the near future because the interface to many lower-layer technologies lacks necessary support. Therefore, FQ_CoDel may fail to control the queues efficiently because of a non-cooperative link layer with hidden buffering [29]. Even if the lower layer is able to cooperate, there may be a number of queues at the link layer which need a fair queue enabled CoDel each, leading to increased complexity with an excessive number of queues. There are already real-world examples for not favoring multi-queue AQM over single-queue because of complexity. For example, with cable modems having typically 16 or 32 link-layer service classes, it was concluded that even with small number of queues such as 32 (i.e., 16/32 service classes times 32 queues in total) the performance increase with multi-queue AQM is insufficient to justify the increase in implementation complexity [48]. Obviously the recommended default of 1024 FQ_CoDel queues [20] per service class would be even more unrealistic to be implemented on such hardware. Therefore, we consider single queue AQM testing very important to give insights on how AQM behaves when fair queuing is not practical to deploy. We also believe that FQ_CoDel simply hides undesirable shortcomings by making them infrequent instead of offering a proper solution.

PIE is based on Proportional Integral controller (PI) [21]. With PI, not only current delay is used to control the drop probability but also the delay trend is considered. PIE provides a solution to determine current delay ($delay_{curr}$) without per packet calculation and adds burst allowance to avoid drops with short-term bursts. PIE maintains estimate for departure rate. When there are more than departure rate threshold (*dq_threshold*) packets in the queue, a measurement cycle is started which ends once threshold number of packets have been transmitted. At the end of the cycle, the departure rate is calculated by dividing the amount of data transmitted by the elapsed time. The average departure rate is then calculated using exponentially weighted moving average. A periodic timer with *tUpdate* interval is used to update the drop probability p:

$$delay_{curr} = \frac{queuelength}{departurerate_{average}}$$

$$p = p + \alpha \cdot (delay_{curr} - delay_{ref}) + \beta \cdot (delay_{curr} - delay_{old})$$

The calculated drop probability p is used on arriving packets. Reference delay $delay_{ref}$ is the target delay for PIE. The α and β determine how much a deviation from reference delay and delay trend affect the drop probability, respectively. PIE auto-tunes α and β based on drop probability to scale the response timescale. PIE also implements burst allowance; if a short-term burst occurs, no random dropping is performed as long as there is burst allowance quota remaining. The quota is reduced whenever a departure rate measurement is successful. If the delay is below the reference delay, the burst allowance quota is reset to *max_burst*.

HRED [26] is an aggressive RED variant that is designed TCP slow start in mind. The recommended RED parametrization [13] is not useful with load transients. Adaptive RED (ARED) [12], [15] is also likely to be too slow to respond resulting in similar behavior as with regular RED because state is only updated using relatively infrequent timer while the slow start keeps increasing the load constantly. Usually such a slow AQM response leads to fallback using tail-drop [26]. Instead of auto-tuning the dropping probability slowly such as with ARED, HRED immediately starts with aggressive dropping. HRED is parametrized so that a drop will take place no later than a defined point in slow start. Once the dropping starts, HRED tends to drop rather aggressively which helps when multiple flows are in the common-mode slow start because they are likely to experience some drops each leading to back off one RTT later. Although dropping is heavy, HRED still retains benefits of random dropping which makes it to provide rather good fairness in most of the cases.

As HRED is not able to discern the reason for queue excursions, it always responds as if slow start would be in progress. Occasionally this can lead to under-utilization. However, similar problem might also affect other AQM techniques if operating point is set such that a really low target delay is used. This is due to the well know trade-off in AQM between delay and throughput; while operating close to minimum

TABLE I: AQM parameters

CoDel [32], [34]		PIE [31]		HRED	
target	5 msecs	delay_ref	20 msecs	th_{max}	40 pkts
interval	100 msecs	α	0.125	th_{min}	3 pkts
		β	1.25	w_q	0.02/0.04
		max_burst	100 msecs	max_p	0.65
		dq_threshold	5/10 pkts		
		tUpdate	30/60 msecs		

TABLE II: 95th percentile of the queuing delay (secs)

	Flows			
	1+1	**2+2**	**4+4**	**6+6**
CoDel	0.090	0.126	0.170	0.246
PIEupdate	0.234	0.252	0.294	0.332
PIEthresh	0.198	0.216	0.246	0.270
HRED	0.120	0.126	0.132	0.132
HRED (aggressive)	0.096	0.096	0.108	0.109
SFQ CoDel	0.060	0.057	0.054	0.066

delays, TCP congestion control is less likely to fully utilize the bottleneck link because it halves the number of packets in flight (congestion window) when AQM drops a packet.

IV. Test Setup

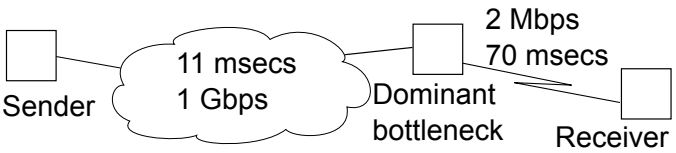

Fig. 1: Test setup

We use ns2 simulator [23] to conduct the experiments. The workload consist of a number of simultaneous TCP flows. The flows are started in groups. First group of flows starts at zero seconds and the second group start is delayed slightly. The delay is uniformly distributed between 0 and 0.65 seconds. All TCP flows within a single group start at the same point of time. The transfer size for the flows within a single group is 360 kbytes in total, that is, the total size is divided by the number of flows in the group to get the transfer size for a single TCP flow. We chose this workload close to the worst-case TCP slow start in mind. That is, when the TCP flows perform a common-mode slow start, the combined effect is likely to be somewhat higher than when the TCP slow starts are spread out as with typical Web traffic. The number of TCP flows in a group is 1, 2, 4, or 6, which is well within what a typical browser may use [8]. In order to limit the number of test cases, we decided to limit the number of flows to 6+6 flows even though real browsers may use even higher number of parallel connections. The selected transfer size resembles the composition of a typical Web page [39], however, we exclude modelling the effects of DNS queries or internal dependencies within the Web page. We believe our model is still good enough to cover the area close to the worst case with reasonable accuracy.

The TCP flows use ns2 *Sack1* TCP. TCP initial window [4] is three segments, the initial RTO is three seconds, and TCP delayed ACKs [10] are in use.

Figure 1 shows the test setup. The access link at the network edge that is the bottleneck for the workload is 2 Mbps with 70 msecs one-way propagation delay. The link is error free. On the Internet side, there is an additional 11 msecs delay for the rest of the path. We also experimented with 101 msecs delay for the rest of the path besides of 11 msecs delay to see if our results are also valid with a larger delay. The AQM parameters are show in Table I. For PIE there are three variants: one with default parameters of 10 packets for the departure rate threshold and 30 msecs for the update interval (*PIE*), and two other variants where either the departure rate threshold is changed to 5 packets (*PIEthresh*) or the update interval is changed to 60 msecs (*PIEupdate*). With HRED we use the original HRED parametrization with $w_q = 0.02$ and also introduce a more aggressive variant with $w_q = 0.04$ that trades off delay at the cost of utilization. The physical buffer size is set to 100 packets to allow AQM to operate without falling back to FIFO behavior during load transients. In addition, we also experiment with SFQ CoDel using the same parameters as with CoDel.

V. Results

In the result analysis we focus on the queuing delays observed in the bottleneck router queue. Figure 2 shows the cumulative distribution function for queuing delay experienced by the TCP packets with 4+4 TCP flows for different AQM techniques. Up to median, all AQMs yield relatively low queuing delay up to 24 msecs only. With other workloads having different number of simultaneous TCP flows the median delay does not exceed 24 msecs either, regardless of the AQM technique in use. Interactive traffic, however, is affected by high delay spikes because the deadlines with interactive traffic are short and the low delay periods usually cannot offset the harm caused by the periods with higher delay. Therefore, we turn our attention to the high-end queuing delays that occur during each testcase.

Table II shows the 95th percentile of queuing delay with different AQM algorithms and number of flows and Figure 3 shows the median and quartiles (25th and 75th percentiles) of the maximum queuing delays observed over 100 replications with different AQM algorithms and number of flows. The bars represent median and the error bars show quartiles. In practice, the high-end delays with our workload occur mostly because of the load transients caused by TCP slow start. During slow start TCP increases the load exponentially on each round trip, resulting in packet bursts of increasing size that pile up at the bottleneck router queue and require a quick response from the AQM mechanisms.

CoDel is promising with 1+1 TCP flows. However, when the number of flows increases, both 95th percentile and maximum delay increase significantly. The behavior is due to the too slowly responding auto-calibration of the drop distance in CoDel. The auto-calibration is only based on *interval* and square root of *count*. As neither of these is in a direct

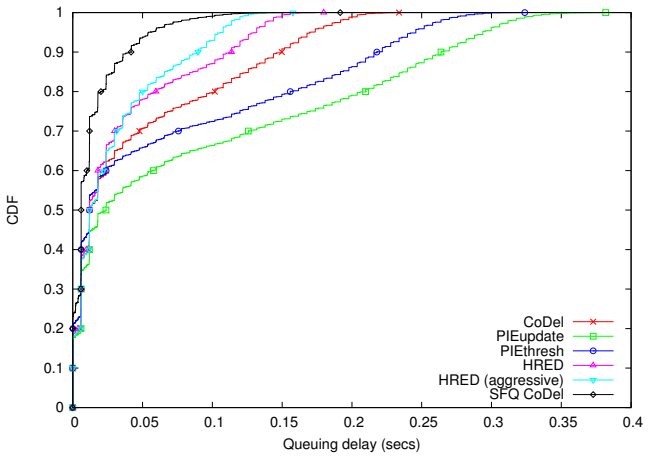

Fig. 2: Cumulative Distribution Function for queuing delay with 4+4 TCP flows (over 100 replications)

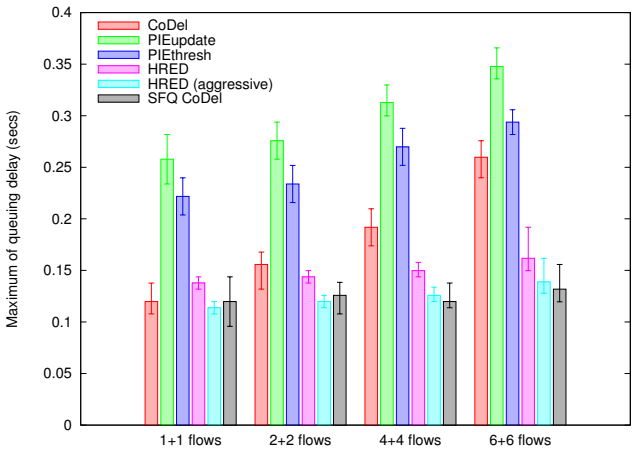

Fig. 3: The median and quartiles of maximum queuing delay with different AQM algorithms (over 100 replications)

relationship to the current load, CoDel dropping is not affected by the heaviness of the load until through secondary effects of the slowly growing *count*. However, the secondary effects occur too late to be effective against common-mode slow start.

Our first impression with PIE was all but encouraging. We found out that with default parameters PIE does not do anything. After careful analysis, we discovered that PIE fails to operate when the link rate is low enough because it cannot measure the departure rate. When the link rate is low, the update timer of PIE always expires before departure rate threshold amount of packets can be transferred. As a result, PIE remains in the burst allowance mode throughout the whole test. The continuous burst allowance effectively makes the behavior equal to FIFO which is why we decided to omit the PIE with default parameters from the results discussion. This contradicts with the claim that "threshold is not crucial for the system's stability" [36]. In order to solve this problem, we had two options: either to make the update timeout longer or to reduce the threshold which both have downsides. We decided to include both variants in our experiments because no clear guidelines exist on reparametrizing PIE. We show results separately for both PIE variants (*PIEupdate* and *PIEthresh*) but refer to them together as PIE. In addition, we ignore the tradeoffs related to setting these two parameters and the diverging results with the different parameter values as we want to focus on comparing different AQMs with each other.

PIE has notably higher high-end delays compared to CoDel, but when the load increases it retains the performance level better compared to CoDel. Even though the drop probability in PIE is affected by the load, the response is too slow due to several reasons. PIE implements burst allowance up to 100 msecs bursts by default which delays its response. Response is further delayed as update to the drop probability is done only periodically and the load is accounted only on the next update. In addition, PIE applies a correction based on the delay trend which helps when the queue is increasing. However, a draining transient queue is detected as downwards trend and it works against PIE until slow start has reached a window size that overloads the bottleneck (i.e., causes persistent queue). Any downward trend during slow start is a transient because the packets and the corresponding load remain on the end-to-end path even when they are not instrumentable in the bottleneck router queue. Therefore, PIE misdetects the early clues of the slow start which only show up as transient queuing delay spikes.

Not to our big surprise, HRED performs the best among the single-queue configurations because it was designed to ensure rapid response with hard deadline for dropping. The slightly more aggressive setup with HRED is even better. With 1+1 flows, the high-end delays are slightly higher with HRED and roughly equal with aggressive HRED compared to CoDel. However, HRED is able to retain the performance level significantly better than Codel when the number of simultaneous TCP flows increases. With CoDel the high-end delay more than doubles for 6+6 flows while with HRED the high-end delays stay roughly at the same level which makes HRED even more stable than PIE with increasing load. The inter-quartile range for maximum delay with HRED (see Figure 3) is small up to 4+4 flows indicating that HRED response is relatively similar in most of the cases.

Similar to HRED, the high-end delays remain roughly at the same level with SFQ CoDel regardless the number of TCP flows. With SFQ CoDel the 95th percentiles of the delay are lower and the medians of the maximum delay are roughly at the same level compared to HRED. This indicates that the flow isolation using SFQ works reasonably well. Table III shows the maximum queuing delay out of 100 replications, that is, the worst-case delay, in each test case. These worst-case delays follow mostly the same pattern as the other high-end delay statistics. While the median of the maximum delay with SFQ CoDel is roughly equal to that of the single-queue CoDel in the case of 1+1 TCP flows, the worst-case delay for SFQ CoDel with 1+1 TCP flows is not as good as with single-queue CoDel because two separate CoDel queues respond slower to

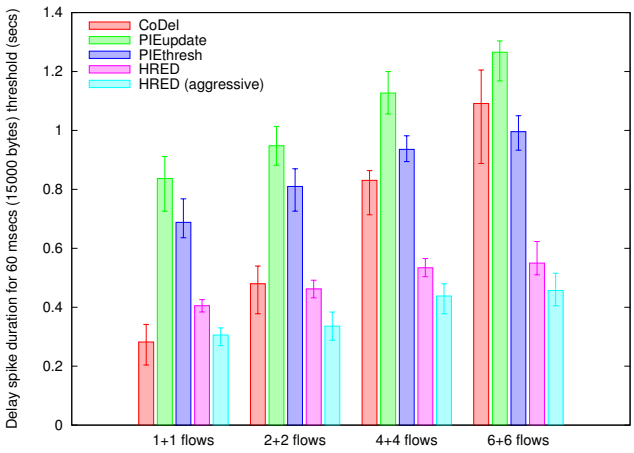

Fig. 4: The median and quartiles of delay spike duration with more than 60 msecs threshold worth of queue in the router queue for different AQM algorithms (over 100 replications)

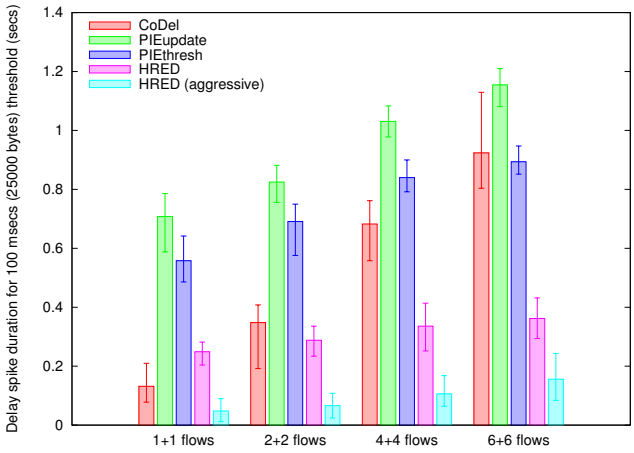

Fig. 5: The median and quartiles of delay spike duration with more than 100 msecs threshold worth of queue in the router queue for different AQM algorithms (over 100 replications)

TABLE III: Maximum queuing delay (secs)

	Flows			
	1+1	**2+2**	**4+4**	**6+6**
CoDel	0.150	0.186	0.234	0.296
PIEupdate	0.336	0.342	0.378	0.402
PIEthresh	0.276	0.294	0.324	0.366
HRED	0.150	0.168	0.174	0.216
HRED (aggressive)	0.156	0.134	0.152	0.192
SFQ CoDel	0.180	0.180	0.174	0.222

a load transient than a single-queue CoDel that shares the state for these TCP flows. With 2+2 TCP flows the worst-case delay is roughly equal to that of single-queue CoDel. Only when the number of flows increases beyond 2+2, SFQ CoDel clearly lowers the worst-case delay compared with CoDel. Moreover, HRED provides lower worst-case delay compared to SFQ CoDel in most cases but obviously SFQ would make the worst case less likely to occur because of hashing the flows usually places them into different SFQ buckets. Only when an interactive flow unluckily would share a queue, the worst-case delay affects also it.

When we took a closer look into the traces, we made an interesting additional observation related to the packets from the TCP initial windows [4]. The packets in the HRED router queue during the highest delay spikes with 6+6 flows included a significant fraction of packets belonging to the initial windows of the flows, while with the other single-queue AQMs a negligible fraction of initial window packets were present during the highest delay spikes. This confirms our observation that other AQMs except HRED react too slowly and allow highest delay spikes to arise after the initial round trip of TCP slow start, while the highest delays with HRED are heavily intensified by the not congestion controlled packets in the TCP initial window.

Figures 4 and 5 show the delay spike duration during which the queue length remains continuously above 60 and 100 msecs threshold (15000 and 25000 bytes), respectively. SFQ CoDel is omitted because there is no single queue where to calculate this metric from. The delay spike duration indicates the longest continuous period where the queue remains above the given value during a test run. In practice there was usually just one major delay spike per test run. The trends are similar to those seen with the maximum queuing delay in Figure 3. This is as expected because longer delay implies more packets to drain until the delay returns to a low level again. Here, the interesting metric is the duration of the delay spike that interactive traffic would experience as a period of harmful jitter, because codecs used for interactive media can typically only conceal a limited amount of jitter for short periods of time. Therefore, long-lasting spikes are likely to cause quality reduction or even gaps to the interactive media playback. In the Figures 4 and 5, we see that the delay spikes are rather long-lasting regardless of AQM technique used. Typically the delay spikes exceeding 60 msecs threshold last clearly for more than 200 msecs and the delay spikes exceeding 100 msecs threshold last for around 130 msecs or longer except with aggressive HRED that encounters notably shorter delay spike durations. With a larger number of simultaneous TCP flows PIE and CoDel encounter delay spikes that last a significantly longer duration compared to HRED.

Figures 6 and 7 show the response times for the first and second group of TCP flows, respectively. The response time for a group is defined as the time between sending the first TCP SYN and receiving the last ACK packet for the flow completing last within the group. The response times are generally slightly longer in the second group because the later starting flows are likely to get more affected by the earlier starting flows as expected. Within each group the response times remain roughly at the same level regardless of the AQM technique in use. This also means that little or no harm is caused by any of the AQM techniques on the response times for the TCP flows.

Figure 8 shows the Jain's fairness index [25] calculated

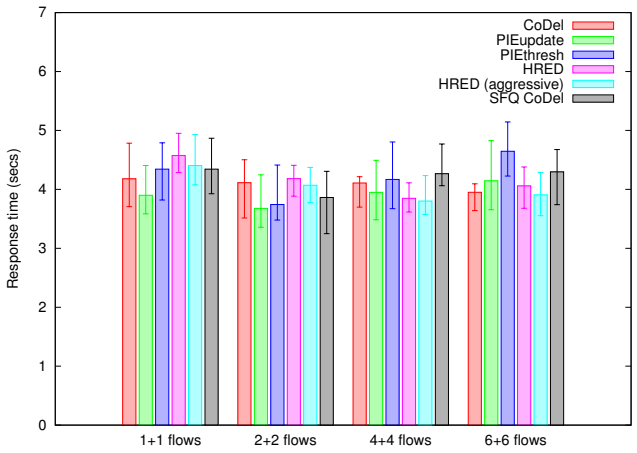

Fig. 6: The median and quartiles of response time for the first group of flows (over 100 replications)

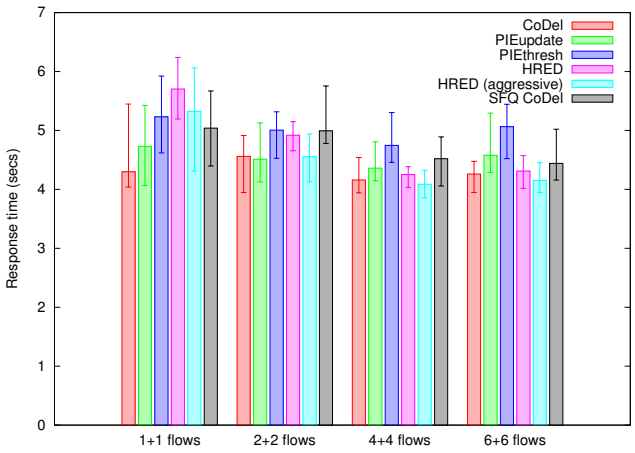

Fig. 7: The median and quartiles of response time for the second group of flows (over 100 replications)

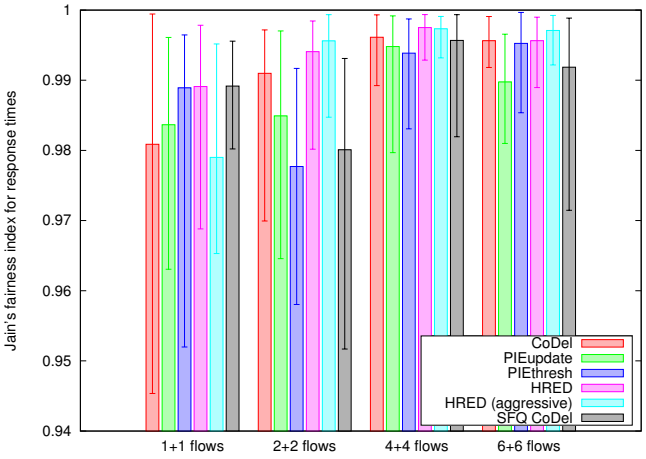

Fig. 8: The median and quartiles of Jain's Fairness index between the response times of the first and second group of flows (over 100 replications)

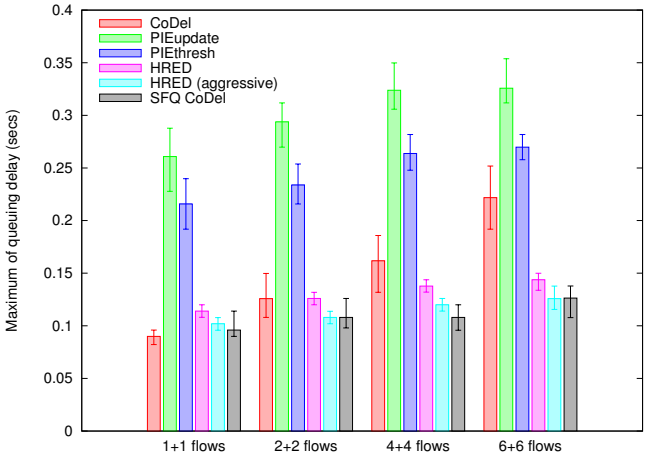

Fig. 9: The median and quartiles of maximum queuing delay with longer rest of the path delay (101 msecs) using different AQM algorithms (over 100 replications)

for the response times of the first and second group of flows. In general, the fairness is very good regardless of the AQM technique. Somewhat surprisingly, SFQ CoDel does not provide the best fairness for the response times even though it has flow isolation.

We also experiments with larger delays for the rest of the path delay in order to validate that our results hold for a larger set of parameters. Figure 9 shows the median and quartiles of the maximum queuing delays with different AQM algorithms when the rest of the path delay was set to 101 msecs. A quick comparison to Figure 3 reveals that the relative order for the AQM techniques remain the same and even the numerical values for the delays are roughly on the same level regardless of the rest of the path delay. Similarly with the delay spike duration shown in Figure 10 compared to Figure 4, the relative order between AQM techniques and trends are retained, and there are only small changes in the numerical values mainly in favor to CoDel and HRED.

VI. Discussion

While HRED performed reasonably well in our study compared with CoDel or PIE it would be challenging to parametrize it correctly for the case where the link rate is expected to vary. HRED depends on proper calibration of the time-constant for the low-pass filter in RED exponentially weighted moving average of queue length and it would need to be tuned on-the-fly if the link rate is changing constantly. In order to have even better behavior, the effective end-to-end delay that is combination of all flows through the router would be needed but it is closely related to measuring the current link load which remains an open challenge [38].

We believe it would be possible to make PIE more responsive than what was seen in our results. However, even though parameter tweaking is encouraged [37], no guidelines exist

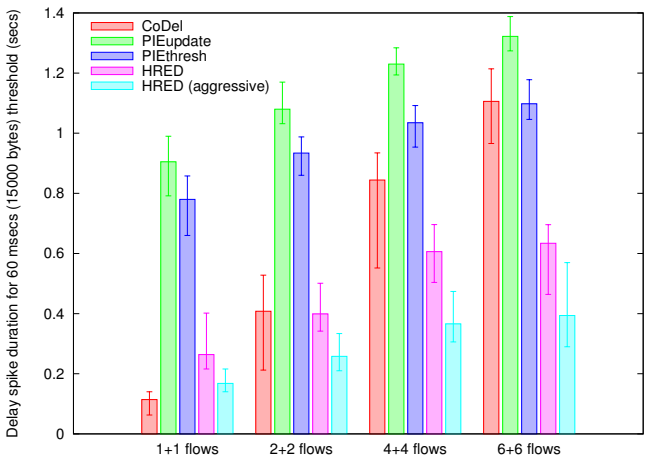

Fig. 10: The median and quartiles of delay spike duration with more than 60 msecs threshold worth of queue in the router queue for different AQM algorithms when the rest of the path delay was 101 msecs (over 100 replications)

on how to do it. Besides, the key point with both CoDel and PIE was to lower the bar for deployment by removing complex AQM configuration step, which has long seen to be a disincentive for RED deployment.

CoDel *count* recall has seen multiple revisions. In all, the duration during which previous value is used is in relation to *interval*. The default value with ns2 and Linux kernel CoDel implementation is 16 times *interval* (1.6 seconds with 100 msecs default interval) which makes it unlikely that next Web user transaction would reuse the previous state because end user usually processes the information for some time before issuing the next action that triggers more Web requests. In addition, many variants exist on which formula is used to calculate the *count* value that is recalled. However, for this particular workload we are using, it is likely that larger count is always better than smaller one because slow start requires a response from router. The response needs to be sufficiently heavy compared with the number of flows that are slow starting at the same time. We also implemented the recall variant that matches what CoDel Linux implementation is using, but there was no noticeable change in the delay results. If we would have modelled full Web transaction with DNS lookups and dependencies within the Web page, the recall variant might have had some effect. However, it is still questionable whether using the recalled count from the previous state for the new flows that are slow starting would be any better than selecting a big initial count directly regardless of history. Studying the effects of the count recall in CoDel would be an interesting topic on its own.

AQM deployment faces a significant challenge because some link layers lack necessary interfaces. The interface might not provide any way to control the queuing below the IP layer but instead buffers large amount of packets at link layer leaving zero or meaningless queue size to the IP-layer queue. As AQM operates at the IP layer, this is likely to impact the deployment of most AQM techniques on networks which are based on such link technologies. Also, fair queuing together with AQM cannot schedule packets to allow low latency for some traffic if the packets share a long queue at the lower layer. Occasionally rate shaping is attempted in order to circumvent the lack of link-layer support which moves the bottleneck to the shaper. However, it cannot work correctly when the link rate of the actual bottleneck varies, which is typical, for example, with wireless links. As such, it is not true nor elegant solution.

Hashing flows to buckets used in SFQ is know to be subject to collisions due to limited number of buckets. In case of a collision, an interactive flow is subject to similar delay as another flow that is hashed to the same bucket. Thus, the number of buckets used by [20] and chosen for Linux FQ_CoDel implementation is large (by default 1024 buckets). However, a large number of queues causes problems because even if a link-layer support is provided for AQM, SFQ on top of link-layer technologies where lower layer implements multiple queues becomes complex. With such link-layer technologies, each link-layer queue requires a separate set of SFQ queues, leading to a very large total number of queues at the IP layer. A real world example with excessive number of queues is discussed in [48] where 32 service classes would require 32k queues in total (with the 1024 default) and even total of 1024 queues with only 32 queues per service class was considered not justified. Sharing queues between classes, while possible, would have increased complexity too much. An alternative approach to reduce collision probabilities is to combine hashing with multiple buckets per hash similar to approaches used in caching [11].

VII. Conclusions

In this paper we simulated the behavior of CoDel and PIE AQM proposals during load transients that occur because of TCP slow start and compared them against the slow-start optimized HRED. The results show that latency-wise CoDel performance will not scale when the number of flows increases. PIE performed worst regardless of the number of flows but scales better than CoDel when the number of flows increases. Therefore, PIE would likely provide better response in case of extreme overload compared with CoDel but our study only covered up to 6+6 TCP flows. Among the single-queue controllers, HRED provides the best performance except with 1+1 flows where CoDel performance is equal to or slightly better than with HRED. In addition, with HRED the high-end queuing delay is not affected by the number of flows almost at all. SFQ CoDel also provides good results. However, its deployment is subject to additional challenges.

We also discovered that PIE with default parameters does not work with low-rate links because PIE fails to measure departure rate within a single update interval. Without successful departure rate measurement, PIE assumes the link to be uncongested all the time. We believe that this burst allowance problem with PIE may affect measurements also in other studies if the link rate in use is low enough. This is especially problematic if the link rate varies because then PIE

would work intermittently which is hard to notice compared with our case where PIE did not do anything.

REFERENCES

[1] "Active Queue Management and Packet Scheduling (aqm)." [Online]. Available: https://datatracker.ietf.org/wg/aqm/charter

[2] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. SIGCOMM '10*, Aug. 2010.

[3] M. Allman, "Comments on Bufferbloat," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 30–37, Jan. 2012.

[4] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390, Oct. 2002.

[5] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Sep. 2009.

[6] T. Bonald, L. Muscariello, and N. Ostallo, "Self-Prioritization of Audio and Video Traffic," in *Proc. IEEE International Conference on Communications (ICC 2011)*, Jun. 2011.

[7] B. Braden *et al.*, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, Apr. 1998.

[8] Browserscope. [Online]. Available: http://www.browserscope.org/?category=network&v=1

[9] CableLabs, "MAC and Upper Layer Protocols Interface Specification," Data-Over-Cable Service Interface Specifications DOCSIS 3.1, Oct. 2013.

[10] D. Clark, "Window and Acknowledgement Strategy in TCP," RFC 813, Jul. 1982.

[11] T. Cloonan, J. Allen, T. Cotter, and B. Widrevitz, "Minimizing Bufferbloat and Optimizing Packet Stream Performance in DOCSIS 3.0 CMs and CMTSs," in *SCTE Cable-tec EXPO '13*, Oct. 2013.

[12] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A Self-Configuring RED Gateway," in *Proc. INFOCOM '99*, Mar. 1999.

[13] S. Floyd, "RED: Discussions of Setting Parameters," Nov. 1997. [Online]. Available: http://www.icir.org/floyd/REDparameters.txt

[14] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[15] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," ICSI, Tech. Rep., Aug. 2001.

[16] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Queue*, vol. 9, no. 11, Nov. 2011.

[17] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Taht, "Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control," in *Proc. 5th IEEE International Traffic Monitoring and Analysis Workshop (TMA'13)*, Apr. 2013.

[18] E. Grigorescu, C. Kulatunga, and G. Fairhurst, "Evaluation of the Impact of Packet Drops due to AQM over Capacity Limited Paths," in *Proc. Capacity Sharing Workshop (CSWS '13)*, Oct. 2013.

[19] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[20] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-CoDel," Internet Draft, Mar. 2014, Work in progress.

[21] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proc. INFOCOM 2001*, Apr. 2001.

[22] J. Huang *et al.*, "An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance," in *Proc. SIGCOMM '13*, Aug. 2013.

[23] ISI at University of South California, "The network simulator – ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns

[24] V. Jacobson, "Congestion Avoidance and Control," in *Proc. SIGCOMM '88*, Aug. 1988, pp. 314–329.

[25] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," Tech. Rep. DEC TR-301, Sep. 1984.

[26] I. Järvinen, Y. Ding, A. Nyrhinen, and M. Kojo, "Harsh RED: Improving RED for Limited Aggregate Traffic," in *Proc. 26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012)*, Mar. 2012.

[27] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee, "Understanding Bufferbloat in Cellular Networks," in *Proc. Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet)*, Aug. 2012.

[28] N. Khademi, D. Ros, and M. Welzl, "The New AQM Kids on the Block: Much Ado About Nothing?" University of Oslo, Department of Informatics, Tech. Rep. 434, Oct. 2013.

[29] M. Kojo, I. Järvinen, H. Tschofenig, and A. Y. Ding, "Supporting Low Latency near the Network Edge and with Challenging Link Technologies," in *Workshop on Reducing Internet Latency*, Sep. 2013.

[30] P. McKenney, "Stochastic Fairness Queueing," in *Proc. INFOCOM '90*, vol. 2, Jun. 1990, pp. 733–740.

[31] P. Natarajan, R. Pan, and C. Piglione, "Proportional Integral Controller - Enhanced (PIE) AQM Implementation," 2013. [Online]. Available: ftp://ftpeng.cisco.com/fred/ropan

[32] K. Nichols, "CoDel ns2 code," 2012. [Online]. Available: http://www.pollere.net/CoDel-ns-2.35.patch

[33] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, May 2012.

[34] ——, "SFQ CoDel ns2 code," 2012. [Online]. Available: https://github.com/dtaht/ns2.git

[35] ——, "Controlled Delay Active Queue Management," Internet Draft, Mar. 2014, Work in progress.

[36] R. Pan *et al.*, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," in *Proc. 2013 IEEE Conference on High Performance Switching and Routing*, Jul. 2013.

[37] ——, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," Internet Draft, Feb. 2014, Work in progress.

[38] D. Papadimitriou, M. Welzl, M. Scharf, and B. Briscoe, "Open Research Issues in Internet Congestion Control," RFC 6077, Feb. 2011.

[39] R. Peon and W. Chan, "SPDY Essentials," Google Tech Talk, Dec. 2011.

[40] J. Postel, "Transmission Control Protocol," RFC 793, Sep. 1981.

[41] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001.

[42] I. Rhee, L. Xu, and S. Ha, "CUBIC for Fast Long-Distance Networks," Internet Society, Internet Draft, Aug. 2008, Work in progress.

[43] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, Dec. 2012.

[44] S. Souders, "Roundup on Parallel Connections," Mar. 2008. [Online]. Available: http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/

[45] D. Täht, "Inside Codel and Fq Codel," in *Standford University Networking Seminar*, Jan. 2013.

[46] G. White, "Active Queue Management Algorithms for DOCSIS 3.0," CableLabs, Tech. Rep., Apr. 2013.

[47] G. White and J. Padden, "Preliminary Study of CoDel AQM in a DOCSIS Network," CableLabs, Tech. Rep., Nov. 2012.

[48] G. White, "Active Queue Management in DOCSIS 3.x Cable Modems," CableLabs, White paper, May 2014.