Performance Analysis of CoDel and PIE for Saturated TCP Sources

Fabian Schwarzkopf, Sebastian Veith, Michael Menth,
Chair of Communication Networks (KN), University of Tuebingen, Sand 13, 72076 Tuebingen, Germany

***Abstract*—In the recent years, the bufferbloat phenomenon was observed which is mainly due to oversized unmanaged buffers in the Internet. This triggered a new discussion of active queue management (AQM) algorithms in the IETF. "Controlled Delay" (CoDel) and "Proportional Integral controller Enhanced" (PIE) are considered as an alternative to "Random Early Detection" (RED). Their intention is both to take advantage of large buffers for occasional bursts and to limit queueing delays most of the time. Moreover, they are able to cope with varying bandwidth.**

In this paper, we study the performance of CoDel, PIE, and CoDel-ACT, which is an effective modification of CoDel that leads to better performance than CoDel in our studies. We experiment with saturated TCP sources and a fixed-bandwidth bottleneck link and focus on the delay-limiting phase of the algorithms. We investigate the impact of configuration parameters and traffic load on link utilization and queueing delay. We study the timely evolution of queuing delays and drop patterns, and point out significant differences among the algorithms. In particular, we show that CoDel's drop behavior changes over time and may lead to underutilization.

I. Introduction

Active queue management (AQM) mechanisms for the Internet have been discussed since the early '90s [1] and Random Early Detection (RED) [2] has been proposed as a standard [3]. Although implemented in many devices, network operators hesitate to turn RED on because it requires careful adjustment to the specific network environment. In 2009 excessive packet delay was observed in wireless networks [4] due to oversized buffers and persistently full queues. Two years later, this phenomenon was referred to as *bufferbloat* [5]. It underlined the importance of managing buffers. As the community felt that RED is not sufficient to control queues, the IETF working group "Active Queue Management and Packet Scheduling" (aqm)[1] was founded in 2013 and has adopted two major AQM algorithms for potential standardization. "Controlled Delay" (CoDel) [6] which was originally proposed in 2012 [7] and "Proportional Integral controller Enhanced" (PIE) [8] which was originally published in 2013 [9]. Both algorithms have the benefit that they cope with varying bandwidth as they rather control delay than queue size. They achieve that goal with two different strategies. CoDel monitors the actual queueing delay of dequeued packets; if the delay is continuously above a certain threshold for a given interval, CoDel drops packets with excess delay on dequeue. PIE first measures the departure rate on the bottleneck link and predicts the queuing delay. Then, a target delay, the estimated delay, and its recent changes contribute to the calculation of a packet loss probability based on which PIE possibly drops packets on enqueue. Thus, both approaches are significantly different. We also consider a variant of CoDel that modifies the growth of CoDel's internal `count` value as well as the time to wait before dropping mode starts. We refer to this variant by CoDel-ACT (CoDel-adapted-`count`-and-time).

If AQMs are very aggressive, the utilization of a potential bottleneck link may suffer. Thus, there is a tradeoff between low queuing delay and high resource utilization. In this paper, we investigate the three AQM algorithms CoDel, CoDel-ACT, and PIE. We illustrate an initial burst-allowing phase and a succeeding delay-limiting phase in the presence of saturated TCP flows. We investigate the impact of the algorithms and their configuration parameters on queuing delay and utilization for the delay-limiting phase. A time-dependent analysis of the AQM algorithms reveals significant differences among their drop patterns and explains the observed phenomena. The findings contribute to a deeper understanding of the algorithms that are currently under standardization in IETF.

The rest of the paper is structured as follows. Section II reviews the AQM algorithms under study. Section III describes the simulation setup and discusses simulation results, giving statistical evidence and visual insights about the performance behavior and differences of CoDel, CoDel-ACT, and PIE. Section IV reviews related work, and Section V summarizes most important findings of our work and draws conclusions.

II. Algorithms under Study

A. CoDel

For our simulations we implemented the C++-like pseudocode that is given for CoDel in [6]. CoDel equips packets on enqueue with a timestamp so that it can calculate their queuing time on dequeue. The procedure `dodeque()` dequeues a packet and returns a pointer to it (`r.p`) as well as a boolean `r.ok_to_drop` which is true if the queueing time has continuously exceeded the delay threshold `target` for at least `interval` time. The parameters `target` and `interval` are recommended to be set to 5 ms and 100 ms, respectively, and in particular independently of the networking scenario. As CoDel has no other parameters, it can be used without parameter adaptation. The actual logic of CoDel is given in Listing 1 which is performed whenever a packet is dequeued.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/2-1. The authors alone are responsible for the content of this paper.

[1]https://datatracker.ietf.org/wg/aqm/history/

CoDel uses some state variables. The boolean `dropping` indicates whether CoDel may drop packets if two additional conditions are also met. To drop a packet, its `ok_to_drop` must be true, otherwise CoDel leaves its dropping mode. CoDel controls its drop frequency by ensuring that the next packet is not dropped before time `next_drop`. The time between successive values of `next_drop` depends on the persistence of the observed congestion which is tracked by the integer `count`. As the `count` variable becomes large if the queueing delay exceeds `target` for long time, the time between successive values of `next_drop` becomes short which increases the drop rate.

Listing 1. CoDel's dequeue algorithm.

```
Packet* CoDelQueue::deque() {
  double now = clock(); // current time
  dodequeResult r = dodeque();
  if (dropping) {
    if (!r.ok_to_drop) {
      dropping = false;
    }
    while (now >= drop_next && dropping){
      drop(r.p);
      r = dodeque();
      if (!r.ok_to_drop) {
        dropping = false;
      } else {
        count = count + 1;
        drop_next = drop_next + interval/sqrt(count);
      }
    }
  } else if (r.ok_to_drop) {
    drop(r.p);
    r = dodeque();
    dropping = true;
    if (count > 2 && now-drop_next < 8*interval) {  // *
        count = count - 2;                          // *
    } else {                                        // *
        count = 1;                                  // *
    }                                               // *
    drop_next = now + interval/sqrt(count);
  }
  return (r.p);
}
```

As CoDel's algorithm in Listing 1 is not our contribution, we leave its study to the reader. However, we discuss some observation that the reader should understand about the algorithm. CoDel enters its dropping mode only if the queuing delay exceeds `target` for more than `interval` time, and stays in dropping mode until a dequeued packet's delay falls below `target`. After packet loss, CoDel determines the next `next_drop` time, and after subsequent packet loss within a dropping phase, CoDel increments `count`. This leads to increasing drop rates until queueing delay decreases and CoDel leaves its dropping mode. If CoDel reenters the dropping mode, the `count` value is reset to 1 only if the last dropping phase was sufficiently long ago, otherwise `count` is set to an only slightly smaller value than before. Note that CoDel can drop several consecutive packets at once because the next value of `next_drop` is determined relative to the last value of `next_drop` and not relative to the current time unless CoDel just entered the dropping mode.

B. *CoDel-ACT*

During our studies we discovered obvious deficiencies of CoDel and tried to repair them. A research of existing CoDel variants lead to an algorithm with suitable properties that K. Nichols has suggested in 2012. However, its benefits were not quantified at that time so that it was no longer considered.

CoDel-ACT is a modification of CoDel that addresses the fact that CoDel waits a fixed time `interval` before reentering the dropping mode and that CoDel sets the new value of `count` to `count-2` so that `count` can continuously grow on persistent congestion. To prevent this behavior, K. Nichols suggested two modifications[2]. The first modification requires that the queuing delay exceeds `target` for at least `interval / sqrt(count)` time instead of a fixed duration of `interval` before the procedure `dodeque` sets `r.ok_to_drop` to true for a packet so that CoDel switches to dropping mode. The second modification decays `count` when entering the dropping mode using the following code:

Listing 2. CoDel-ACT's modification to CoDel's dequeue algorithm addressing the *-marked lines in Listing 1.

```
if (count > 2 && now-drop_next < 8*interval) {
  count = count - 2;
  if (count > 126) {
    count = 0.9844 * (count + 2);
  }
} else {
  count = 1;
}
```

The decay applies only if `count` is greater than 126. If so, the function scales `count` with the decay parameter $\frac{126}{128} \approx 0.9844$ rather than just subtracting a fixed value of 2. Thus, this modification reduces `count` more strongly than the current IETF variant in [6] when entering the dropping mode.

C. *PIE*

We implemented PIE according to the appendix of the IETF draft in [8] but corrected some obvious glitches in the pseudocode. The changes comprise the reset of the accumulated drop probability at each drop as well as prevention of a negative drop probability p by limiting its range to [0,1].

PIE updates its drop probability p every `t_update` time (16 ms by default) using the currently observed queueing delay D_q and the control parameter `qdelay_ref` (16 ms by default). On arrival, a packet is dropped with drop probability p unless the queue is obviously not congested which is expressed in [8] through appropriate conditions, e.g., if the current queuing delay is less than `qdelay_ref/2`. We first explain how the current queueing delay D_q is measured and then how p is adapted.

The current delay D_q is estimated by Little's law which requires the departure rate. The latter is obtained by the following concurrent process. If the queue contains more than T_{DQ} bytes and is currently not in a measurement state, then a new departure rate measurement starts at time t_i. The measurement stops at time t_{i+1} when T_{DQ} bytes are dequeued. The current dequeue time Δ is calculated according to Equation (1) and smoothed to obtain the average dequeue time Δ_{avg} according to Equation (2). Then, the departure rate R is computed like in Equation (3) which allows to estimate

[2]The source code of these modifications is available at http://pollere.net/Codel.html, http://pollere.net/Txtdocs/codel.cc, last modified July 24, 2012.

the current queueing delay D_q using Equation (4), given the queue length L_q. If the queue contains less than T_{DQ} bytes, the rate R cannot be updated and the last value of R is reused for the estimation of the current queuing delay D_q.

$$\begin{aligned} \Delta &= t_{i+1} - t_i && (1) \\ \Delta_{avg} &= \frac{1}{4} \cdot \Delta + \frac{3}{4} \cdot \Delta_{avg} && (2) \\ R &= T_{DQ}/\Delta_{avg} && (3) \\ D_q &= L_q/R && (4) \end{aligned}$$

The drop probability p is periodically updated similarly as in Equations (5) and (6) using the current queueing delay D_q, the previous queueing delay D_q^{prev}, and the control parameter qdelay_ref. The equations in [8] include in addition a case analysis of p that we omit for brevity. Equation (5) uses the two factors α and β for which values are recommended in [8]. While CoDel updates its state variable count only in dropping mode, PIE updates its state variable p also when the estimated queueing delay D_q is below the reference value qdelay_ref.

$$p = p + \alpha \cdot (D_q - \texttt{qdelay\_ref}) + \beta \cdot (D_q - D_q^{prev}) \quad (5)$$

$$D_q^{prev} = D_q \quad (6)$$

The draft [8] extended the original algorithm in [9] with some modifications that were proposed by CableLabs simulations. The modifications comprise (1) a de-randomization of drop events and (2) an on/off mechanism of the algorithm. The first modification accumulates the current drop probability p at every packet enqueue using the variable accu_prob. As long as accu_prob is < 0.85, no packets are dropped. If accu_prob becomes > 8.5, then the packet is dropped in any case. If accu_prob ranges between these limits, drops are randomly performed with probability p. Every drop resets accu_prob. The second modification describes the inactivation of PIE with a reset of PIE's internal variables in the absence of congestion. By default, PIE is reactivated if the queue length exceeds one third of the total size. The second modification is irrelevant for our simulations because PIE is never deactivated due to lasting congestion in our experiments.

III. Results

In this section we first describe our simulation setup. Then we study queuing delays with CoDel, CoDel-ACT, and PIE and illustrate that these algorithms exhibit in the presence of saturated TCP sources first a burst-allowing phase and then a delay-limiting phase. In this work we focus on the delay-limiting phase. We first provide evidence about queueing delay distributions depending on the traffic load. Then we study the impact of configuration parameters on queuing delay and utilization of the bottleneck link. As we observe a non-monotone dependency of CoDel's utilization on the traffic load, we perform a time-dependent analysis to better understand the performance behavior of the AQM algorithms. To that end, we investigate how state variables, drop rates, drop patterns, and queuing delays evolve during the delay-limiting phase.

A. Simulator and Network Topology

All simulations were performed with INET 2.4.0 [10] in the OMNet++ network simulation framework 4.4.1 [11]. We used the Network Simulation Cradle 0.5.3 [12] to simulate TCP sources, which facilitates the application of real world network stacks from Linux kernels in simulation programs. All simulations are conducted with Linux kernel 2.6.29.

We simulate clients that are connected to a server over a high-bandwidth link with 1 Gb/s and a shared bottleneck link with 10 Mb/s, which results in a one-sided dumbbell topology. We configured a one-way propagation delay of 0.1 ms for the fast access link and 5 ms for the bottleneck link, which yields a minimum round-trip time (RTT) of about 10 ms.

We choose a buffer size for the bottleneck link of 250 KB, i.e., two times the bandwidth delay product at an RTT of 100 ms. We set the buffer so large to study the AQM mechanics with negligible amount of tail drops. We investigate the presented AQM algorithms for control of the queue on the bottleneck link with varying traffic load in terms of 1, 4, 16, and 64 saturated TCP NewReno flows. The flows were randomly started within the first second of a simulation run.

If not mentioned differently, we configure CoDel and PIE with their recommended default parameters: target= 5 ms, interval=100 ms, qdelay_ref=16 ms, and t_update=16 ms. We configure CoDel-ACT with the same parameters as CoDel. While we provide only figures for 10 ms RTT and TCP NewReno in this paper, we run the same experiments with TCP CUBIC and/or 100 ms RTT. Their results are qualitatively the same, but differ in quantity.

B. Illustration of Burst-Allowing and Delay-Limiting Phase

Figure 1 illustrates the queuing delay of consecutive packets at the beginning of a simulation for 1, 4, 16, and 64 concurrent TCP flows. We observe a large spike in the first 1 s – 5 s, but then queuing delay is limited to low values. This is exactly what AQMs should do: they allow that infrequent large bursts can use the available buffer, but they avoid a standing queue under persistent load. We denote these phases as burst-allowing and delay-limiting phase. They can be observed with saturated TCP sources.

With PIE, we experience initially large delays and the duration of such phases are almost independent of the traffic load. It is limited to 1.5 s in all cases. With CoDel, initially large delays increase with traffic load and so does their duration. As a result, CoDel leads to the same queuing delay as PIE for 64 TCP flows, but the duration of large delays is almost 5 s long. CoDel-ACT reveals identical queuing delays as CoDel as their behaviors do not differ within the first few drop phases. From then on, a separate curve for CoDel-ACT is visible in the figure. At the beginning of the simulation, CoDel's and CoDel-ACT's count value is small so that the minimum time between drops interval/sqrt(count) is rather large and initial bursts are cut down only slowly. As CoDel's and CoDel-ACT's drop rate scales with interval, increasing interval also extends the initial phase with large experienced delays. PIE obviously increases its drop rate more

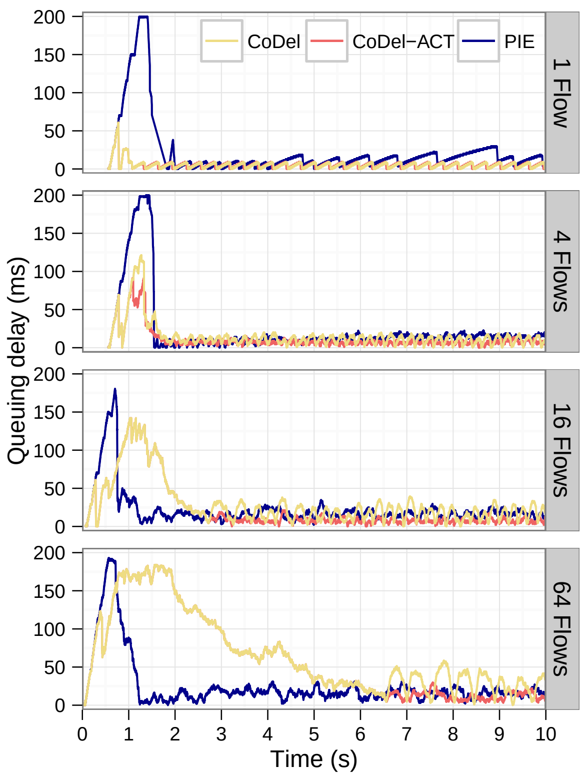


Fig. 1. Time-dependent queuing delay with CoDel, CoDel-ACT, and PIE.

quickly and stops phases of increased delays earlier than CoDel while allowing the use of the same buffer size. The figure also shows that the simulated process is not stationary before 5 s. Therefore, we use only data gathered after 10 s when considering averages, distributions, or quantiles.

C. Queuing Delays in the Delay-Limiting Phase

While Figure 1 illustrates the initial queuing delay, Figure 2 quantifies the long-term queuing behavior of the considered AQM algorithms for 16 concurrent TCP flows as complementary distribution function (CDF) of the queuing delay. With CoDel, about 4% of the packets experience no queuing delay. This is a hint that the bottleneck link may be underutilized. The queuing delay of most other packets is almost equally distributed between 0 ms and 20 ms. With CoDel-ACT, there are fewer packets with no queuing delay, but most packets have a queuing delay less than 10 ms. Moreover, CoDel-ACT centers the queuing delay between 4 ms and 8 ms like a Normal distribution. The same holds for PIE but with a larger mean and variance. Although PIE leads mostly to larger queuing delays than CoDel for 16 flows, the probability for large queuing delays is higher for CoDel than for PIE. As PIE drops on enqueue and the bandwidth is constant in our simulation, the queue length distribution is very similar to the queuing delay distribution. This is different with CoDel and CoDel-ACT as they perform drop on dequeue. Therefore, observed queue lengths are larger which effects that their CDFs have a bias of about 2 – 3 packets to larger values compared to the CDF of the corresponding queuing delay.

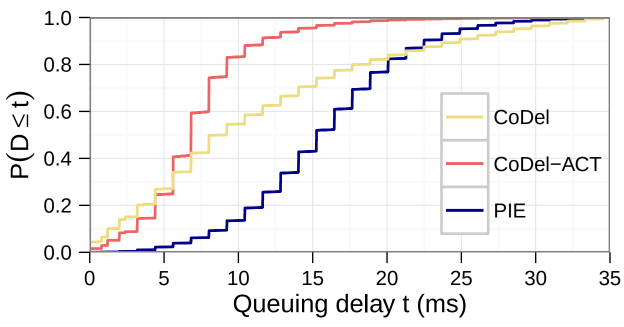


Fig. 2. CDF of the queuing delay for CoDel, CoDel-ACT, and PIE for 16 TCP flows.

Table I shows the mean and the 99% quantile ($Q_{99\%}$) of the queueing delay for different traffic load and for a minimum RTT of 10 ms and 100 ms. In all cases, the mean queueing delay of CoDel and CoDel-ACT increases with more flows while the one of PIE is about 16 ms. The 99% quantiles are significantly larger. While CoDel leads to smaller queueing delay than PIE at low traffic load, it surpasses the one of PIE for many flows. In contrast, CoDel-ACT leads to the least queueing delay in all cases.

TABLE I
MEAN AND 99% QUANTILE OF MEASURED PACKET DELAYS (MS).

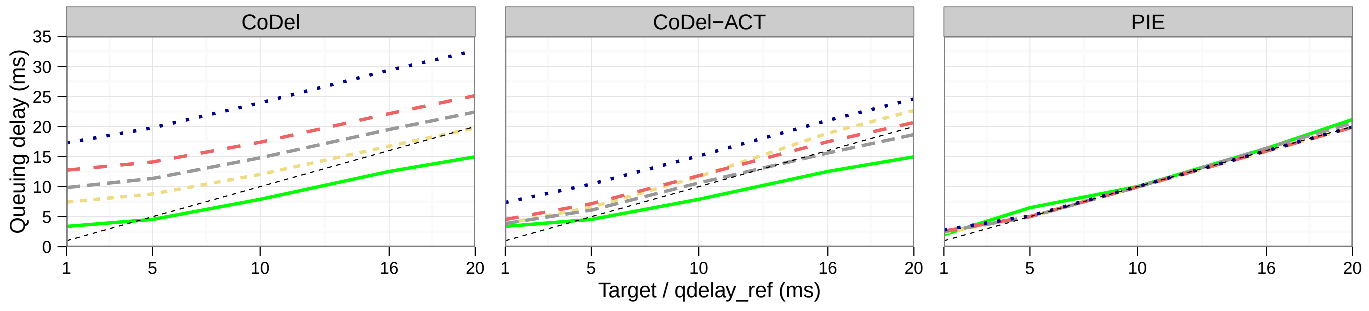
no. TCP flows	CoDel		CoDel-ACT		PIE	
	mean	$Q_{99\%}$	mean	$Q_{99\%}$	mean	$Q_{99\%}$
RTT=10 ms						
1	4.54	9.22	4.54	9.22	16.13	32.14
4	8.79	17.67	6.60	11.63	15.79	24.89
16	14.19	35.75	7.16	15.25	15.95	28.51
64	19.94	44.19	10.36	26.10	15.97	33.30
RTT=100 ms						
1	1.16	6.03	1.15	6.03	8.28	28.93
4	3.46	10.85	3.49	10.85	16.49	38.57
16	6.47	19.28	4.74	12.06	15.75	31.35
64	15.84	36.17	8.63	21.71	15.93	33.76

PIE is configured with `qdelay_ref`=16 ms which results in fact in about 16 ms average queuing delay for all investigated load levels. Therefore, one can think of PIE's `qdelay_ref` as a parameter that leads to a certain average queue length. In contrast, CoDel and CoDel-ACT are configured with `target`=5 ms, but the observed average queuing delays are clearly larger for most load levels. Thus, CoDel's `target` provides only relative control of the average queueing delay.

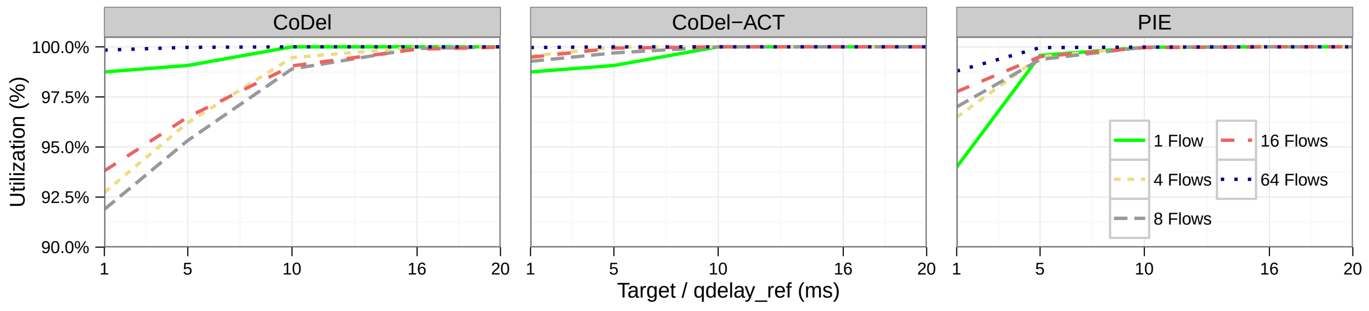
D. Impact of AQM Control Parameters

CoDel, CoDel-ACT, and PIE have two control parameters each: a delay threshold and a time scale parameter. In the following, we investigate their impact on average queuing delay and utilization in the delay-limiting phase for various traffic load. The presented results are mean values over 20 runs.

1) Impact of Delay Thresholds: CoDel and CoDel-ACT use `target` to decide whether a packet's delay is considered as too long so that the algorithms switch to dropping mode after some time and possibly discard such packets. PIE uses `qdelay_ref` to increase or decrease its drop probability p and a fraction of `qdelay_ref` helps PIE to prevent losses



(a) Average queuing delay.



(b) Bottleneck utilization.

Fig. 3. Impact of the delay thresholds `target` for CoDel and CoDel-ACT and `qdelay_ref` for PIE. For CoDel's `interval` and for PIE's `t_update` parameter default values are chosen of 100 ms and 16 ms, respectively. A black dashed line shows the configured delay thresholds `target` and `qdelay_ref`.

in the absence of congestion. We investigate the impact of these delay thresholds during the delay-limiting phase for various traffic load while using the default time scale parameters `interval`=100 ms and `t_update`=16 ms for CoDel, CoDel-ACT, and PIE, respectively.

Figure 3a shows the average queuing delay for the three AQM algorithms depending on the delay threshold. We observe that larger delay thresholds result in longer average queuing delays for all algorithms. With CoDel, larger traffic loads lead to larger average queuing delays that are mostly larger than `target`. This shows again that `target` provides only relative control on queuing delay. We observe similar effects for CoDel-ACT, but the number of TCP flows has less influence on the average queuing delay compared to CoDel. In contrast, PIE is able to control the queuing delay independently of the traffic load. The measured delay almost equals the configured `qdelay_ref`. Moreover, small values of `qdelay_ref` cause smaller queuing delays than CoDel's and CoDel-ACT's `target` value in the presence of many TCP flows. A `target` value of 5 ms for CoDel leads for 16 TCP flows to about the same queuing delay as a `qdelay_ref` value of 16 ms for PIE. Thus, the two recommended default values lead to comparable results in this particular setting.

Figure 3b shows the utilization for the three AQM algorithms. CoDel's utilization is below 100% for small values of `target`, and ranges between 95% – 100% for its default parameter of 5 ms depending on the traffic load. CoDel-ACT improves the utilization compared to CoDel while the observed queuing delay is even shorter. The default parameter `target`=5 ms leads to almost 100% utilization for most traffic loads. Also PIE's utilization suffers from a small delay threshold, but the recommended `qdelay_ref`=16 ms achieves 100% utilization for all investigated loads. A closer look at CoDel's utilization reveals that it is high for 1 and 64 TCP flows, but it is low for 4, 8, and 16 TCP flows. This non-monotone order is non-intuitive, therefore, we investigate and explain that issue further in Section III-E.

2) Impact of Time Scale Parameters: CoDel's dynamics scale with the time parameter `interval`. First, the dropping mode is triggered after `interval` time if all packets experienced too long delay within that time. Second, the minimum time within a drop phase is a dynamic fraction of `interval`. And third, the duration for which CoDel remembers the `count` state parameter of the last drop phase also depends on `interval`. PIE's dynamics scale with `t_update` because after that period PIE regularly adjusts its drop probability. We investigate the impact of the time scale parameters during the delay-limiting phase for various traffic load while using the default delay thresholds `target`=5 ms and `qdelay_ref`=16 ms, respectively. We report findings but omit figures due to space limitations.

CoDel's and CoDel-ACT's queuing delay increases linearly with `interval` and clearly depends on traffic load. For 4 or more flows, CoDel-ACT leads to about half the queuing delay compared to CoDel. PIE causes a queuing delay of 16 ms for `t_update`=16 ms and all investigated traffic loads. This value is lower for smaller values of `t_update` but only slightly larger for larger values up to `t_update`=150 ms. In addition, we also observe some dependence on the traffic load in these ranges.

The utilization of CoDel heavily depends on `interval`. It is between 98.5% and 100% for small values of `interval` around 25 ms, takes a minimum of 95% – 100% at `interval`=100 ms, and values between 99% and 100% at `interval`=200 ms. Again, 1 and 64 TCP flows lead to higher utilization than 4, 8, and 16 TCP flows. CoDel-ACT shows increasing utilization for increasing values of `interval`. A single TCP flow always leads to lower utilization than other traffic loads. At `interval`=125 ms or larger, 100% utilization is reached for all traffic loads. PIE achieves 100% utilization for `t_update`=16 ms or larger. Smaller values of `t_update` cause lower utilizations which also depend on traffic load.

E. Time-Dependent Analysis of the Delay Limiting Phase

In the remainder of this work, we study the time-dependent behavior of the AQM algorithms.We first point out that their state variables, drop rates, drop patterns, and queuing delays behave differently and depend on traffic load. We show how loss patterns and queuing delays even change over time for CoDel which also affects utilization.

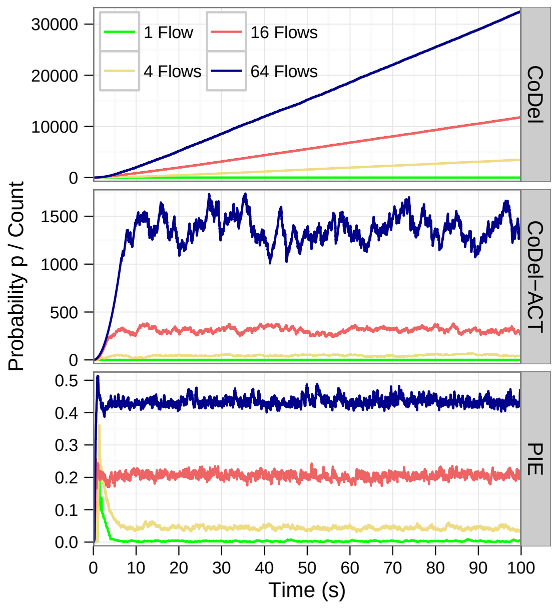


Fig. 4. Evolution of state variables over time.

1) State Variables: CoDel's and CoDel-ACT's state variable `count` and as well as PIE's drop probability p memorize the recently experienced congestion to some degree. We investigate how they evolve over time. Figure 4 shows a time series of CoDel's and CoDel-ACT's `count` variable as well as PIE's drop probability p for different numbers of concurrent TCP flows. The data is taken from the first 100 s of a single simulation run. CoDel's `count` variable increases linearly over time with a slope depending on the traffic load. An exception is the transmission of a single TCP flow for which `count` remains very low. The boundless growth of the `count` variable is due to CoDel's algorithm and the saturated TCP sources. The `count` variable increases during a drop phase with every consecutive drop. When CoDel leaves its dropping mode, the rate increase of several saturated TCP sources is large so that the queue length rises quickly again. As a result, the next dropping mode is triggered before a duration of `8*interval` has past since the last `next_drop` instant. Therefore, CoDel does not reset its `count` variable to 1 so that `count` can increase without bounds. An exception is the experiment with a single TCP flow where it obviously takes longer until CoDel switches to dropping mode again.

CoDel-ACT's `count` variable behaves differently over time. Like with CoDel, it stays low for the transmission of a single TCP flow. However, it oscillates around values 50, 300, and 1300 for higher traffic load and does not continuously grow. On the one hand, CoDel-ACT returns to dropping mode more quickly than CoDel because the time during which increased queuing delays must be observed before CoDel-ACT switches to dropping mode is `interval/sqrt(count)` instead of `interval` so that CoDel-ACT does not reset `count` to 1, either. On the other hand, CoDel-ACT sets `count` to $0.9844 \cdot$ `count` at the beginning of a new dropping mode, which is lower than `count-2` for `count` values larger than 128. That feature effects that CoDel-ACT's `count` value is bounded in contrast to the one of CoDel. Average `count` values depend on the traffic load because the number of TCP flows governs the overall rate increase of the traffic aggregate when connections are in congestion avoidance phase.

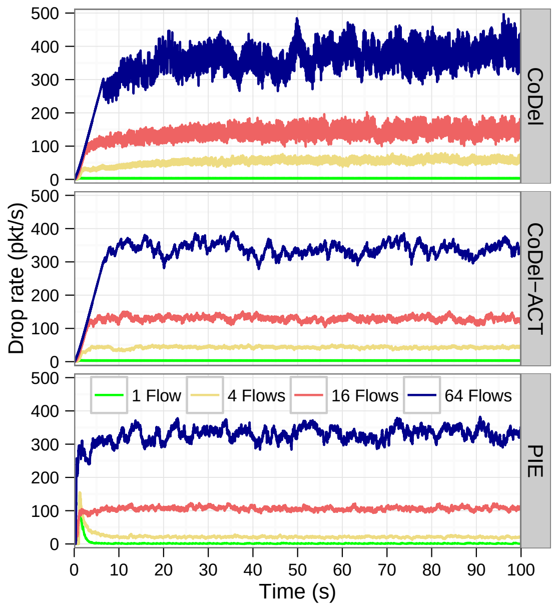


Fig. 5. Time-dependent drop rates.

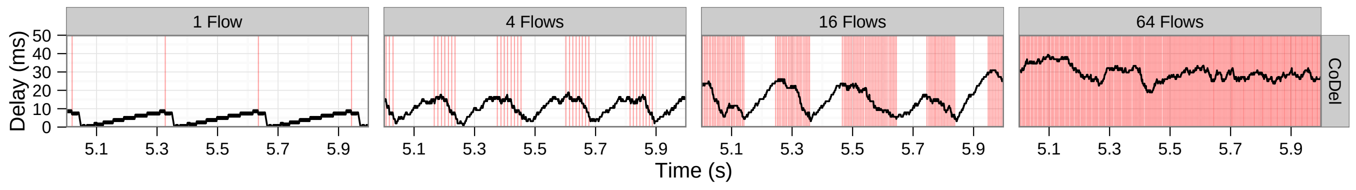
PIE's drop probability p oscillates around values that depend on the traffic load. In the presence of a single flow, the drop probability is almost zero. At the beginning of a congestion phase, the drop probability p reaches its operating point faster than `count` values do and reveals an initial overshoot before returning to a stationary level.

2) Drop Rates: Figure 5 visualizes time-dependent drop rates for the three AQM algorithms. We calculated them by applying the time-exponentially weighted moving average (TEWMA) [13] with a memory of 500 ms to individual packet losses. All drop rates oscillate around values that depend on the traffic load. For 64 flows, CoDel takes about 20 s until its stationary drop rate is reached. CoDel-ACT reaches that level already after 6 s and PIE does so within 2 s. The slow increase of drop rates for CoDel and CoDel-ACT is caused by the rather slow increase of the `count` value after simulation start. PIE produces the least drop rates, followed by CoDel-ACT and CoDel with the largest drop rates. The broad curve of CoDel's drop rate reveals oscillations with higher amplitude than those of CoDel-ACT and PIE. This is due to CoDel's extreme alternating drop and non-drop phases which are illustrated next.

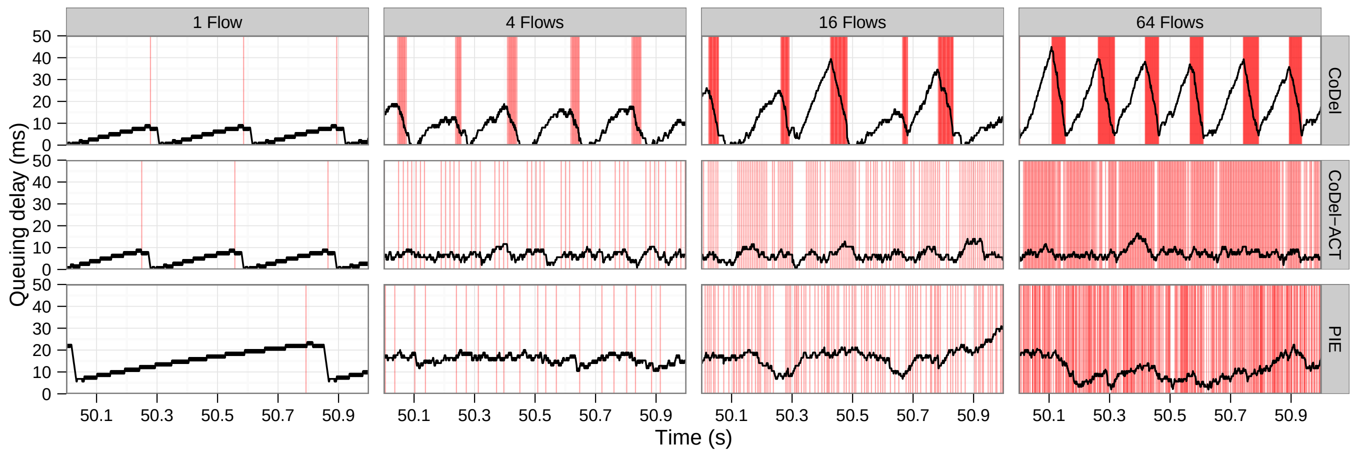
3) Loss Patterns and Queuing Delays: The plots in Figure 6 show the time series of queuing delays of consecutive packets as a curve and packet losses as vertical lines. Intervals of 1 s duration after 5 s, 50 s, and 500 s simulation time are provided. As CoDel-ACT's and PIE's results after 5 s and 500 s almost equal those after 50 s, we omit them in the figure.

a) Comparison of Loss Patterns and Queueing Delays after 50 ms: Figure 6b compares loss patterns and delays of the three algorithms after 50 s. In case of a single flow, they all drop only a single packet after observation of too long delays. The packet loss causes a decrease in traffic rate and experienced queuing delay shortly after. The time between losses is the same for CoDel and CoDel-ACT, but it is almost 3 times longer for PIE. As a consequence, PIE loses fewer packets and achieves higher resource utilization than CoDel variants (see Figure 3b). PIE drops packets on enqueue while CoDel and CoDel-ACT drop packets on dequeue. This leads to a larger interval between packet loss and delay reduction for PIE in the figure.

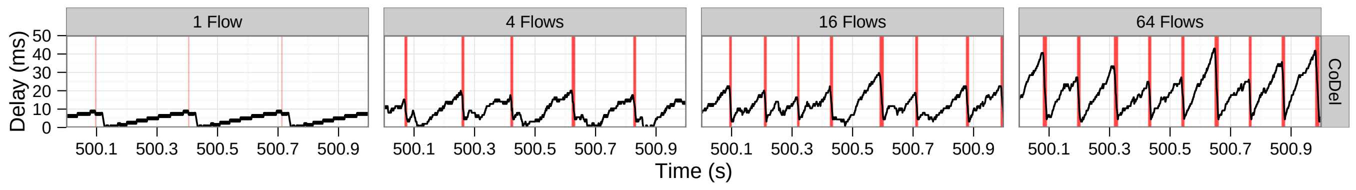
We now consider multiple concurrent TCP flows. CoDel exhibits drop phases alternating with non-drop phases. The drop phases are rather short and end when the queuing delay of dequeued packets falls below `target`. CoDel drains a long queue within 10 ms – 60 ms by discarding packets on dequeue. The high density of consecutive vertical lines reveals very high drop rates within drop phases. They are enabled by the very large `count` values of about 2500, 6000, and 16250 after 50 s persistent traffic load of 4, 16, and 64 flows. They lead to a minimum time between consecutive packet losses of 2.0 ms, 1.3 ms, and 0.8 ms. As the transmission of a



(a) Simulation interval 5 – 6 s.



(b) Simulation interval 50 – 51 s.



(c) Simulation interval 500 – 501 s.

Fig. 6. Time-dependent queuing delays on a shared bottleneck link with 10 Mb/s; packet losses are marked as vertical lines.

packet takes only 1.2056 ms for the bottleneck bandwidth of 10 Mb/s, even two packets may be dropped at once with 64 flows so that one vertical line possibly represent two packet drops. CoDel causes long non-drop phases by waiting at least `interval` time after increased queuing delays are observed in a non-drop phase before dropping mode is triggered again. Therefore, non-drop phases are at least `interval` time long. The non-drop phases are longer for a few TCP flows than for many TCP flows because the overall rate needs more time to recover and to anew drive the queue into congestion. CoDel-ACT also exhibits drop phases alternating with non-drop phases. However, drop phases show lower drop rates compared to CoDel, and non-drop phases are shorter. The time between drops in a drop phase is larger than with CoDel because CoDel-ACT's `count` variable is around 50, 300, and 1300 for 4, 16, and 64 flows, which leads to at least 14.1 ms, 5.8 ms, and 2.8 ms between packet drops. Therefore, CoDel-ACT takes longer to sufficiently reduce the queue length which extends the drop phase compared to CoDel. The shorter non-drop phases are facilitated by the fact that CoDel-ACT must wait only `interval/sqrt(count)` time after observing extended queuing delays before it switches to dropping mode. They are around 18.3 ms, 5.8 ms, and 2.8 ms long, i.e., some non-drop phases are very short and can be recognized only by reduced queuing delay. Thus, CoDel-ACT restarts dropping very fast and leads to shorter delays than CoDel. The time between drop phases decreases with increasing number of TCP flows for the same reason as with CoDel. PIE drops packets continuously over time and almost randomly. However, slightly decreased and increased queuing delays correlate with slightly decreased and increased drop rates.

A comparison of resulting queuing delays shows significant differences among the algorithms. With CoDel-ACT, the queuing delay oscillates with moderate amplitude around a short value, with PIE with a moderate amplitude around a larger value, and with CoDel with a large amplitude around a load-dependent value. This explains why CoDel leads to mean queuing delays that are mostly smaller than those of PIE in Table I, but may cause significantly larger quantiles.

A closer analysis of CoDel's queuing times reveals that with a single flow, the queuing delay is always positive. With 4 and 16 flows, the queuing delay of some packets is zero, and with 64 flows, queuing delay is again mostly positive. Zero queuing delays may indicate an idle link, which causes underutilization. This explains the non-monotone dependence of the utilization on the traffic load observed in Figure 3b. The reason why a single flow avoids zero queuing delays in contrast to 4 or 16 flows is that CoDel's `count` does not rise for a single flow so that consecutive drops are spaced sufficiently far apart. Therefore, CoDel drops only a single packet before leaving the dropping mode instead of draining the queue.

b) CoDel's Varying Drop Behavior over Time: Figures 6a–6c reveal that CoDel's drop behavior changes over time: drop phases are long after 5 s, shorter after 50 s, and very short after 500 s. This is due to increasing `count` values which cause larger drop rates. We consider 16 flows. Figure 6a shows that queuing delays after 5 s simulation time become short at the end of a drop phase but stay above zero. Thus, the queue never empties and the bottleneck link is well utilized. Drop rates after 5 s are low enough that the overall traffic rate is reduced carefully so that it matches approximately the link rate at the end of the drop phase. As a result, the queue is not fully drained and underutilization does not occur. Figure 6b shows that queuing delays after 50 s sometimes fall down to zero at the end of a drop phase. This means that the queue is sometimes empty and the link may be idle, leading to underutilization. Drop rates are larger than after 5 s so that the overall traffic rate is reduced too much and falls below the link rate at the end of the drop phase. The remaining queue content may not suffice to fully fill the link until the traffic rate turns up again, and cause the link to run idle for short time. Figure 6c illustrates that drop phases after 500 s stop with larger queuing delays than after 50 s and that zero queuing delays are mostly avoided. This leads to improved utilization after 500 s. After 500 s simulation time, drop rates are so large that multiple packets are dropped at once and the queue is drained faster than a single RTT so that the traffic rate is not yet reduced at the end of the drop phase. This avoids underutilization for two reasons. First, when CoDel stops dropping, the queue holds more packets with a queueing time less than `target` than after 50 s. Thus, there is more remaining data. Second, the queue increases again after the end of a drop phase until the effect of reduced traffic rates becomes visible at the bottleneck link. It causes the queuing delay to first slightly increase and then decrease before increasing again.

4) Evolution of Utilization and Queuing Delay: We studied the evolution of time-dependent utilization averaged over 1 s in the presence of 16 TCP flows and report results without figures. PIE and CoDel-ACT achieve about 100% utilization during the entire simulation. In contrast, CoDel's utilization starts with 100%, falls down to 95% after 30 s simulation, and slowly increases to 99% after 500 s. We define the variant CoDel-count-n which has `count` set to the constant value n. CoDel-count-600 leads to almost 100% utilization, CoDel-6000 to 95%, and CoDel-60000 to 99%. This confirms that utilization of CoDel depends on `count` in a non-monotone way and, therefore, changes over time if `count` increases from small to large values.

IV. Related Work

The bufferbloat phenomenon has been reported in [5]. While some authors are rather doubtful about its prevalence and impact [14], [15], bufferbloat has been demonstrated in cellular networks [16]. The authors of [17] pointed out many sources contributing to Internet latency and countermeasures.

In [7], CoDel was suggested to control queueing delay independently of buffer size, RTTs, bottleneck bandwidth, and even under varying bottleneck bandwidth. A comparison with RED was also provided. Similar results are reported in [18] from an actual Linux testbed. AQMs may be combined with scheduling mechanisms [19]. For instance, CoDel is mostly recommended to be combined with stochastic fair queueing (SFQ) to isolate flows against each other [20].

The authors of [21] compared the performance of CoDel and RED using simulations and evaluated queueing delay and throughput with different buffer sizes. They concluded that RED is also able to control the queue at a reasonable length but does not extend the transmission time of files because it leads to fewer packet drops than CoDel. Interactions between CoDel and LEDBAT have been studied in [22]. The authors of [23] have presented a software-defined implementation of RED and CoDel in an FPGA to support 10 Gb/s links.

In [9], PIE was presented and it was shown that it is able to control delay while maintaining high utilization during different congestion levels. The work compared CoDel and PIE, and showed that CoDel is not able to control queue delay under heavy load. In [24], CableLabs simulated both CoDel and PIE in DOCSIS cable modems. They demonstrated that actively managed buffers reduce the queueing delay. Another simulation of the DOCSIS cable modems showed that CoDel has problems to adjust to a sudden change in bottleneck rate under unresponsive loads [25]. Another comparison between PIE and CoDel for DOCSIS is provided in [26].

In [27], the impact of the main parameters of CoDel and PIE is analyzed. The authors compared CoDel, PIE, and Adaptive RED (ARED) in a testbed environment at an RTT of 100 ms. They used the originally published PIE algorithm which differs from the newer one used in our work.

V. Conclusion

In this work we have investigated the utilization and queueing delay of the three AQM algorithms CoDel, CoDel-ACT, and PIE for saturated TCP flows and a bottleneck with constant bitrate. We first illustrated the existence of a burst-allowing phase and a delay-limiting phase in the presence of saturated TCP sources. Then, we showed that the AQM algorithms lead to different queuing delay and arrival rate distributions. While PIE is able to keep the average queueing delay at its configured `qdelay_ref` parameter, CoDel's average queuing delay depends both on its `target` parameter and the traffic load. We investigated the impact of configuration parameters on average queueing delay and utilization. CoDel-ACT leads to higher utilization and less queueing than CoDel while PIE leads to better utilization at the expense of increased queueing delay. We performed a time-dependent analysis of the algorithms' delay-limiting phase. We investigated how state variables `count` and p, and drop rates evolve over time which revealed a boundless growth of CoDel's `count` variable for saturated TCP sources. An analysis of drop patterns and queueing delays showed significant differences in the operation of the three algorithms. CoDel's drop behavior changes over time due to increasing `count` and leads to potential underutilization. CoDel-ACT is designed to avoid such behavior and leads to shorter queueing delays. PIE does not exhibit such behavior by design.

Our work contributes to the understanding of novel AQM algorithms currently discussed for standardization in IETF, and potentially to their improvement. We revealed some disadvantageous properties of CoDel compared to CoDel-ACT or PIE. However, the experiments were limited to a single bottleneck with constant bitrate of 10 Mb/s and saturated TCP flows. To recommend one of the three AQM algorithms, further studies are needed, in particular with different traffic models and varying bandwidths.

References

[1] R. Adams, "Active Queue Management: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013.

[2] 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.

[3] B. Braden et al., "RFC2309: Recommendations on Queue Management and Congestion Avoidance in the Internet," Apr. 1998.

[4] B. Turner, "Has AT&T Wireless data congestion been self-inflicted?" http://blogs.broughturner.com/2009/10/is-att-wireless-data-congestion-selfinflicted.html.

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

[6] K. Nichols et al., "Controlled Delay Active Queue Management," draft-ietf-aqm-codel-00, Oct. 2014.

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

[8] R. Pan et al., "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," draft-ietf-aqm-pie-00, Oct. 2014.

[9] ——, "PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem," in *IEEE Workshop on High Performance Switching and Routing (HPSR)*, 2013.

[10] A. Varga, "INET-2.4.0 released," http://inet.omnetpp.org/, Jun. 2014.

[11] ——, "OMNeT++ 4.4.1 released," http://www.omnetpp.org/, Oct. 2014.

[12] S. Wand, "Network Simulation Cradle," http://research.wand.net.nz/software/nsc.php, 2012.

[13] R. Martin and M. Menth, "Improving the Timeliness of Rate Measurements," in *GI/ITG Conf. MMB*, 2004.

[14] O. Hohlfeld et al., "BufferBloat: How Relevant? A QoE Perspective on Buffer Sizing," http://downloads.ohohlfeld.com/paper/bufferbloat-qoe-tr.pdf, TU Berlin, Tech. Rep. 2012-11, Nov. 2012.

[15] M. Allman, "Comments on Bufferbloat," *ACM SIGCOMM Computer Communications Review*, vol. 43, no. 1, Jan. 2013.

[16] H. Jiang et al., "Understanding Bufferbloat in Cellular Networks," in *Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet)*, Aug. 2012.

[17] B. Briscoe et al., "Reducing Internet Latency: A Survey of Techniques and their Merits," *to appear in IEEE Communications Surveys & Tutorials*, 2016.

[18] T. Hoiland-Jorgensen, "Battling Bufferbloat – An Experimental Comparison of Four Approaches to Queue Management in Linux," http://rudar.ruc.dk/handle/1800/9322, Roskilde Univ., Tech. Rep., Dec. 2012.

[19] F. Baker and R. Pan, "RFC7806: On Queuing, Marking, and Dropping," Apr. 2016.

[20] T. Hoeiland-Joergensen et al., "FlowQueue-Codel," http://tools.ietf.org/html/draft-ietf-aqm-fq-codel, Mar. 2016.

[21] N. Kuhn et al., "Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot?" in *Capacity Sharing Workshop (CSWS)*, 2014.

[22] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. D. Taht, "Fighting the Bufferbloat: on the Coexistence of AQM and Low Priority Congestion Control," in *Workshop on Traffic Measurement and Analysis*, 2013.

[23] A. Sivaraman et al., "No Silver Bullet: Extending SDN to the Data Plane," in *ACM HotNets*, Nov. 2013.

[24] G. White, "Active Queue Management in Docsis 3.X Cable Modems," http://www.cablelabs.com/wp-content/uploads/2014/06/DOCSIS-AQM_May2014.pdf, Cable Television Laboratories, Inc., Tech. Rep., May 2014.

[25] R. Pan et al., "QoE: As Easy As PIE," http://www.nctatechnicalpapers.com/Paper/2013/2013-qoe-as-easy-as-pie, National Cable and Telecommunications Association, Inc., Tech. Rep., 2013.

[26] J. Martin, G. Hong, and J. Westall, "Managing Fairness and Application Performance with Active Queue Management in DOCSIS-based Cable," in *Capacity Sharing Workshop (CSWS)*, 2014.

[27] N. Khademi et al., "The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PI," in *IEEE Global Internet Symposium*, 2014.