UNIVERSITY OF OSLO
Department of Informatics

The New AQM Kids on the Block: Much Ado About Nothing?

Technical Report 434

Naeem Khademi

David Ros

Michael Welzl

ISBN 82-7368-399-0
ISSN 0806-3036

23 October 2013

The New AQM Kids on the Block: Much Ado About Nothing?

Naeem Khademi, David Ros[*], and Michael Welzl

Networks and Distributed Systems Group
Technical Report 434
Department of Informatics
University of Oslo
{naeemk,michawe}@ifi.uio.no, david.ros@telecom-bretagne.eu

Abstract

Active Queue Management (AQM) design has again come into the spotlight of network operators, vendors and OS developers. This reflects the growing concern and sensitivity about the end-to-end latency perceived by today's Internet users. Indeed, delays on the order of seconds have become common due to the deployment of excessively-sized FIFO/DropTail buffers at the edge of many networks. CoDel and PIE are two AQM mechanisms that have recently been presented and discussed at the IRTF and the IETF. However, to the best of our knowledge, they have not yet been thoroughly evaluated or compared against each other except by simulation. We set thus to perform an experimental evaluation using real-world implementations, in both wired and wireless testbeds. We have in addition compared them with a decade-old variant of RED called Adaptive RED, which shares with CoDel and PIE the goal of "knob-free" operation. Surprisingly, in many instances results were much more favorable towards Adaptive RED. We do not call into question the need for new AQMs, however, there are lessons yet to be learned from old designs.

1 Introduction

End-to-end latency is one of the most important problems of the Internet today. Much awareness has been raised about this issue by the "Bufferbloat" project [22, 1], highlighting how common it is to experience buffer-induced latencies ranging from a few hundred milliseconds up to several seconds at the network's edge – e.g., in ADSL broadband modems, DOCSIS cable modems and 802.11 APs [16]. Ubiquitously deployed loss-based TCP congestion control mechanisms (e.g., standard SACK and Linux's CUBIC) tend to *persistently* fill any buffer; over-buffered devices can therefore worsen the user-perceived Internet performance, most specifically for latency-sensitive applications such as real-time interactive multimedia, online gaming and even web browsing, especially when they share the bottleneck queue with long-lived TCP connections.

[*]David Ros is with Télécom Bretagne–Institut Mines-Télécom.

Table 1: Published evaluation results of new AQMs; peer-reviewed publications are shown in bold face.

	CoDel	PIE	FQ_CoDel
Wired, sim	**[32][23]**[41] [40]	[40]	[40]
Wired, real-life	**[23]** ✓	✓	✓
Wireless (any)	✓	✓	-

Active Queue Management (AQM) is generally regarded as the best approach to solve this problem. The primary goals of any AQM mechanism are: (a) to let the buffer absorb packet bursts while preventing it from sustaining long standing queues; (b) to break any synchronization between flows. If possible, AQM mechanisms (which we shorten to "AQMs" in this paper) should also be able to protect flows from being starved by other more aggressive or misbehaving flows, as well as to support Explicit Congestion Notification (ECN).

While the history of AQM design begins around two decades ago with Random Early Detection (RED) by Floyd and Jacobson [20], it is only recently that the necessity of widespread AQM adoption has been fully realized by vendors and service providers. AQMs have occasionally been deployed in the Internet's core [2] or in ISPs' infrastructures [16], but in general there has been no extensive usage of AQMs in devices operating at the last hop of the access networks, even when it has been long known that these devices often form the bottleneck of an end-to-end path. The lack of deployment of AQMs has often been attributed to the difficulty of correctly tuning the parameters of RED, the only mechanism that was described in an IETF Request for Comments [13] and that has been most implemented by vendors.

Recently, two new AQMs, CoDel [32] and PIE [36] have been proposed, along with a combination of CoDel with stochastic fair queuing (FQ_CoDel) [21]. They promise to tackle the latency problem in lightly-to-moderately multiplexed environments. Thus, they are suitable candidates for implementation by devices operating on access links where a handful of flows might be sharing the bottleneck queue at any instant.

Despite claims of promising results achieved by these new AQMs, there is little published work evaluating their performance. Exceptions are two (non peer-reviewed) preliminary studies from CableLabs [41, 40] which assess AQM performance on DOCSIS 3.0 modems and rely solely on ns-2 simulation results. Also, [23] conducts a preliminary investigation on the interaction of different AQMs and low-priority "scavenger" congestion control using real-life tests. In addition, the initial CoDel paper [32] includes a simulation study comparing CoDel with traditional RED. While [41, 23] study CoDel only, [40] also includes PIE and FQ_CoDel. Assertions about the performance of these AQM schemes on access links have not yet been fully substantiated by exhaustive real-life experiments. Moreover, no investigation is yet available in the literature regarding the performance in Wi-Fi networks where latency is more prominent.

Table 1 gives an overview of the significant AQM evaluation coverage omissions; as one can see, only [40] appears in all three columns, i.e. it is the only work that compares the three mechanisms, albeit only using simulations in wired scenarios, and the current paper fills almost all the holes in the table. We also

sought answers to a number of interesting questions: given that one of the major contributions of the new mechanisms appears to be their parameterless operation, and given that an old "auto-tuning" variant of RED ("Adaptive RED (ARED)" [19]) exists, how much better are CoDel, FQ_CoDel and PIE with respect to ARED? How do all these mechanisms operate with ECN? Are they truly insensitive to the values of some "magic numbers", or are these factors actually just parameters in disguise?

Summarizing, to the best of our knowledge, this paper is the first to:

1. Thoroughly evaluate the design goals and compare the performance of CoDel, PIE and ARED in a generic access link scenario using real-life tests.
2. Investigate their performance in a variety of 802.11 wireless scenarios.
3. Investigate AQM's interaction with SFQ in FQ_CoDel.
4. Investigate their interaction with ECN.

This work is meant to be a step towards a better understanding of these AQMs. Assuming that the design of the mechanisms under evaluation was guided by TCP dynamics, we focus only on the basic operation of these schemes with bulk TCP transfers. For this reason, we do not investigate traffic that is more typical of multimedia applications, e.g. bursty TCP and/or unresponsive UDP flows; this is left for future work.

The rest of this paper is organized as follows. Section 2 provides an overview of the AQMs studied in this paper. Section 3 presents the experimental setup used in the evaluations. An assessment of the chosen AQMs, in a variety of wired and wireless (802.11) scenarios, is the subject of Sections 4 and 5, respectively. Section 6 studies the interaction of CoDel with SFQ scheduling, and the performance with ECN is investigated in Section 7. Finally, Section 8 concludes the paper.

2 AQMs For Low Latency

The most commonly mentioned reason for the lack of AQM deployment in today's Internet is the known difficulty in correctly setting the parameters of RED. Other mechanisms, while being less dependent on parameter settings (e.g., CHOKE [35]), did not have the support of the IETF behind them, and this may be why they played a minor role (if any) in the adoption of AQMs in the Internet. It is therefore essential that a newly standardized[1] AQM requires very little parameter tuning, or, better yet, none at all.

Despite their authors' "no-knob" claim, both CoDel and PIE in fact maintain a set of parameters (with recommended default values) that do affect their performance. On the other hand, ARED adaptively sets most of RED's parameters based on a target average queue as an input parameter. We can identify two key parameters that are common to all three mechanisms:

[1] The formation of a new IETF Working Group on AQM has been approved by the IESG on 26 September 2013.

Target delay: CoDel and PIE both maintain a value called *target_delay*. However this parameter is used in different ways in these two AQMs. While CoDel starts dropping packets when the queuing delay has been above *target_delay* for a certain amount of time, PIE continuously updates its dropping probability based on the difference between the current queuing delay and *target_delay*. Although ARED does not explicitly maintain a *target_delay* value, when ARED is used in a fixed bandwidth scenario, it is possible to derive its corresponding *target_queuing* from a given target delay.

Interval: Most AQMs require a certain time interval to update their dropping/marking probability or decide whether to discard the incoming/outgoing packet(s). The semantics of this interval differs from one AQM to another. CoDel [32] uses the interval value to decide how long the queuing latency can stay above *target_delay* before switching to *dropping mode* (Section 2.1). On the other hand, PIE and ARED use their interval to update the dropping/marking probability (Section 2.2 and Section 2.3).

Next, we provide a brief overview of the AQMs that are investigated in the rest of this paper. In addition to the two recent PIE and CoDel AQMs that adaptively try to keep the latency low, we have also included ARED both as a baseline for comparison and also because, to the best of our knowledge, ARED's performance has never been evaluated against the recent AQM mechanisms in the context of bufferbloat on access links.

2.1 CoDel

CoDel [32, 33] tries to detect a standing queue by tracking, using timestamps, the minimum queuing delay (or sojourn delay) packets experience in a fixed-duration interval, set to 100 ms by default.

CoDel assumes that a small *target* standing queue delay is tolerable so as to achieve good link utilization. When the minimum queuing delay has exceeded the *target_delay* value during at least one interval, a packet is dropped from the tail of the queue and a control law is used to set the next dropping time. Using the well-known relationship of dropping rate to TCP throughput [30], this time interval is decreased in inverse proportion to the square root of the number of drops since the dropping state was entered, to ensure that TCP does not underutilize the link.

When the queuing delay goes below *target_delay*, the controller stops dropping packets and exits the dropping state. In addition, no drops are carried out if the queue contains fewer than an MTU's worth of bytes. Additional logic avoids re-entering the dropping state too early after exiting it, and CoDel resumes the dropping state at a recent control level, if one exists. CoDel only enters the dropping state when the minimum queuing delay has exceeded *target_delay* for an interval long enough to absorb normal packet bursts. This ensures that a burst of packets will not experience packet drops as long as the burst can be cleared from the queue within a reasonable period.

A variant of CoDel with stochastic fair queuing ("FQ_CoDel") has been available since Linux kernel version 3.5 to provide better fairness and flow isolation. Flow isolation in FQ_CoDel better protects flows from the impact of non-responsive flows such as constant bit-rate (CBR) multimedia traffic.

Table 2: PIE parameters in [7].

PIE Parameter	Default value
t_{update}	30 ms
T_{target}	20 ms
α	2
β	20

2.2 PIE

The Proportional Integral controller Enhanced (PIE) AQM [36] randomly drops a packet at the onset of congestion. Similar to CoDel, it uses queuing latency instead of the more commonly used queue length. It uses the trend of latency over time (increasing or decreasing) to determine the congestion level.

Different from CoDel that drops packets on departure (dequeue time) and requires timestamping, PIE drops packets on arrival (enqueuing time) with probability p and does not require timestamping, making it a more lightweight mechanism. Every t_{update} time units, PIE estimates the current queuing delay using Little's law in Eq. 1 and derives p based on Eq. 2:

$$E[T] = N/\mu \tag{1}$$

$$p = p + \alpha * (E[T] - T_{target}) + \beta * (E[T] - E[T]_{old}) \tag{2}$$

where N is the current queue length, μ is the draining rate of the queue, $E[T]$ represents the current estimated queuing delay, $E[T]_{old}$ represents the previous iteration's estimation of the queuing delay, and T_{target} is the target queuing delay.

The drop probability calculation incorporates the direction in which the delay is moving by employing a classic Proportional Integral (PI) controller design, similar to the one used in [24]. The α factor determines how the deviation of current latency from T_{target} affects the drop probability. The β factor makes additional adjustments depending on whether the latency trend is positive or negative. The default values of PIE parameters based on the Linux implementation in [7] are shown in Table 2.

2.3 ARED

One of the main historical challenges with the deployment of RED [20] by network operators has been the tuning of its parameters, so that the average queue length stays approximately around a desired *target_queuing* value[2] in the presence of different levels of congestion, when RED's performance is hard to estimate in advance. This has been a discouraging factor for network operators when considering to deploy RED on congested routers, where predictable queuing latency is important. Adaptive RED (ARED) [19] solves this problem by dynamically adjusting RED's maximum drop probability (p_{max}). ARED observes the average queue length ($\bar{N}$) to infer whether to make RED more or less aggressive.

[2] Bear in mind that *target_queuing* can correspond to a certain *target_delay* in fixed bandwidth scenarios.

Table 3: ARED parameters in [19].

ARED Parameter	Default value
$interval$	500 ms
α	$min(0.01, p_{max}/4)$
β	0.9

Similar to RED, ARED keeps two thresholds (th_min and th_max) which, to correlate with a single $target_queuing$ value, are set to $0.5 * target_queuing$ and 1.5 * $target_queuing$ in accordance with the rules in [19]. If $\bar{N}$ oscillates below th_min, early detection is too aggressive. On the other hand, if $\bar{N}$ oscillates above th_max, early detection is too conservative. Using an Additive Increase / Multiplicative Decrease (AIMD) policy (see Table 3), ARED adaptively changes p_{max} so that the average queue length oscillates around $(th_max + th_min)/2$. ARED updates p_{max} periodically after every *interval* (500 ms by default).

Although ARED [19] was proposed almost a decade and half ago, along with plenty of other RED variants [39, 17, 10, 29, 34] and even if it has been available since Linux kernel version 3.3, its performance has not been thoroughly investigated yet.

3 Performance Evaluation Setup

While the "older generation" of AQM mechanisms were designed with a general Internet router in mind, the two new AQMs (CoDel and PIE) are designed to overcome the bufferbloat problem on access links, by keeping queuing delay low. It is vital to investigate the deployment of AQM mechanisms where they belong. For instance, an AQM designed to perform on a 1000 Hz OS kernel cannot achieve buffering latency requirements that are common in data centers. The experimental network testbed setup that we used in this paper is therefore based on the assumption of AQM deployment on access links.

3.1 Experimental Setup

Two types of topologies are used in our real-life experiments: A) wired and B) wireless. A detailed summary of the hardware used in these experiments is given in Table 4.

3.1.1 Wired Topology

Our wired testbed setup consists of a dumbbell network topology with two sets of Dell OptiPlex GX620 machines acting as senders and receivers in a dumbbell topology. The senders and receivers are connected via two routers; the first router is acting as an *AQM router* implementing AQM on its bottleneck (egress) interface. All senders are connected to the AQM router's ingress interface using a switched network with 100 Mbps Ethernet links. The second router is acting as a delay node on both forward and reverse traffic, providing an RTT_{base} for the traffic traversing it using the $ipfw$ dummynet module [5]. This router is connected to the receivers with switched 100 Mbps links.

The egress interface of the AQM router (bottleneck interface) and the ingress interface of the second router are set to 10 Mbps advertised mode using *ethtool* [3] with auto-negotiation being on to establish the bottleneck link.

3.1.2 Wireless Topology

Our wireless testbed setup, depicted in Figure 1, is similar to the dumbbell topology in the wired setup with the difference that the AQM router acts as an 802.11 Access Point (AP) using *hostapd* [4], and either senders or receivers can be associated with it as required. All nodes are equipped with 802.11 Atheros-based AR5001X+ NIC chipsets. The AP's Wi-Fi interface implements AQM, and this interface and all other wireless nodes' NICs are set to 802.11g mode with the default *Minstrel* rate adaptation mechanism enabled [6]. All wireless nodes are stacked clustered machines with close proximity and operate on a free channel frequency where there is no (or minimal) interference from other SSIDs. All Ethernet links on the topology are set to 100 Mbps.

3.1.3 Generic Setup

All nodes are synchronized using *ntp*. Unless otherwise noted, each experiment consists of 60 sec (wired tests) or 180 sec (wireless tests) of $iperf$ traffic between each sender/receiver pair. We have chosen a longer traffic period (180 sec) for wireless tests to mitigate the effect of randomness in the wireless channel. Each run is repeated 10 times and the presented results are the averages of all runs. RTT_{base} is set to 100 ms by default unless otherwise noted. TCP-Segmentation-Offload (TSO) engine is turned off by default on all nodes' NICs. The bottleneck's maximum queue size (*txqueuelen*) is set to 1000 packets and its Byte-Queue-Limit (BQL) value is set to 1514 bytes.

TCP's appropriate-byte-counting (ABC) [9], SACK [12] and D-SACK options [11] are enabled. TCP's transmit and receiver buffers are set large enough to be able fully utilize bandwith×delay ($C.RTT_{base}$) on the bottleneck path and the congestion control algorithm is set to SACK (named *reno* in Linux). We used the Hierarchical Token Bucket (HTB) Linux queuing discipline to set the AQM on the bottleneck interface. The AQM *qdisc* size is also set equal to *txqueuelen* (1000 packets) for all experiment runs.

All traffic is dumped using *tcpdump* on all interfaces in the network. We use the Synthetic Packet Pairs (SPP) tool [8] to precisely measure the actual per-packet RTT perceived on the path. To measure the TCP throughput/goodput, *tcptrace* is used. CoDel and ARED's modules are available by default in Linux 3.10.4 kernel while we used the PIE module code available in [7] for our experiments. All AQMs use their default values unless otherwise noted. In case of ARED, the parameters are calculated based on the recommendations in [19] except when *target_delay*=1 ms where *th_min* and *th_max* are set to 1 MTU and 2 MTUs accordingly.

4 Evaluating AQM Designs

To better understand the basic behavior of CoDel, PIE and ARED, we conducted a first set of experiments using only a single TCP flow and observed the trade-off between RTT and goodput in a given period. For target delays

Table 4: Experimental network testbed setup

<table>
<tr><td>Model</td><td>Dell OptiPlex GX620</td></tr>
<tr><td>CPU</td><td>Intel(R) Pentium(R) 4 CPU 3.00 GHz</td></tr>
<tr><td>RAM</td><td>1 GB PC2-4200 (533 MHz)</td></tr>
<tr><td>Ethernet</td><td>Broadcom NetXtreme BCM5751
RTL-8139 (AQM interface)
RTL8111/8168B (Dummynet router)</td></tr>
<tr><td>Ethernet driver</td><td>tg3
8139too (AQM interface)
r8168 (Dummynet router)</td></tr>
<tr><td>802.11 b/g</td><td>D-Link DWL-G520 AR5001X+</td></tr>
<tr><td>802.11 driver</td><td>ath5k</td></tr>
<tr><td>OS kernel</td><td>Linux 3.8.2 (FC14)
Linux 3.10.4 (AQM router) (FC16)</td></tr>
</table>

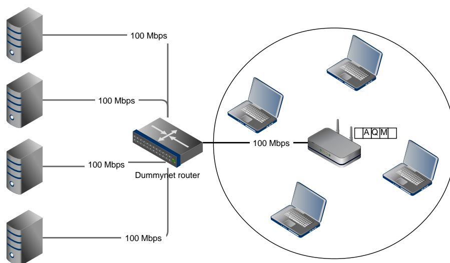
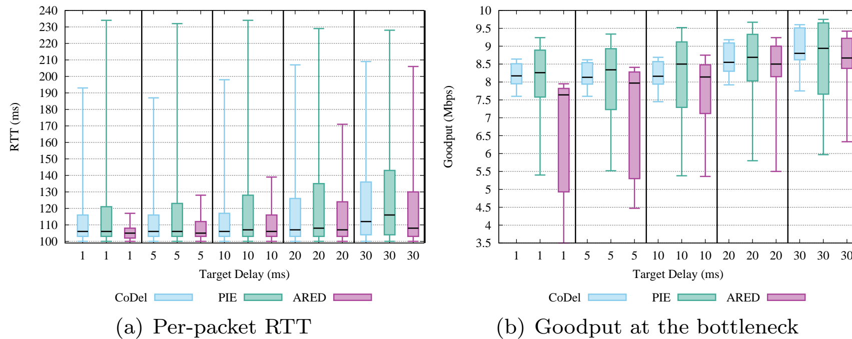


Figure 1: 802.11 experimental network topology

in the range of 1 to 30 ms, CoDel and PIE's total goodputs fall in the ranges of 8.20∼8.94 Mbps and 8.20∼8.85 Mbps while ARED achieves 7.06∼8.71 Mbps for a single TCP flow when RTT_{base}=100 ms. The total goodput does not reveal any information about the *dynamics* of an AQM; given that it is no longer uncommon to use TCP for interactive real-time applications [14], and that non-TCP real-time applications are affected by the dynamics of a competing TCP [38], we report on the goodput distribution seen over 5-second intervals rather than the total goodput in the remainder of this paper.

Figure 2(a) shows the per-packet RTT distribution of CoDel, PIE and Adaptive RED for a single TCP flow and for various target delay values in the range from 1 to 30 ms. Bottom and top of whisker-box plots show 10th and 90th percentiles, respectively. For target delay values above 1 ms, Figure 2(a) shows that all AQMs can maintain a median queuing delay $\leq target_delay$. However, CoDel and PIE produce higher maximum and longer distribution tail of queuing delay than Adaptive RED.

Figure 2(b) shows the TCP goodput measured at the bottleneck link for the same experiments. It shows that CoDel achieves the best goodput among the three AQMs, while ARED suffers from low link utilization for lower target delay values of ≤10 ms when only a single TCP flow is present at the bottleneck. While this is indicative to understand the basic behavior of AQMs, it is not a realistic assumption that a single TCP flow will often be present on access links since most common applications and web browsers use parallel connections (long-lived or short-lived). This even more pronounced in multi-user environments



(a) Per-packet RTT

(b) Goodput at the bottleneck

Figure 2: Per-packet RTT and bottleneck's per 5-sec TCP goodput for a single TCP flow (RTT_{base}=100 ms). Bottom and top of whisker-box plots show 10th and 90th percentiles accordingly.

such as public wired or Wi-Fi networks at hotels, airports, etc. Therefore it is more realistic to consider the performance under different congestion levels and traffic loads.

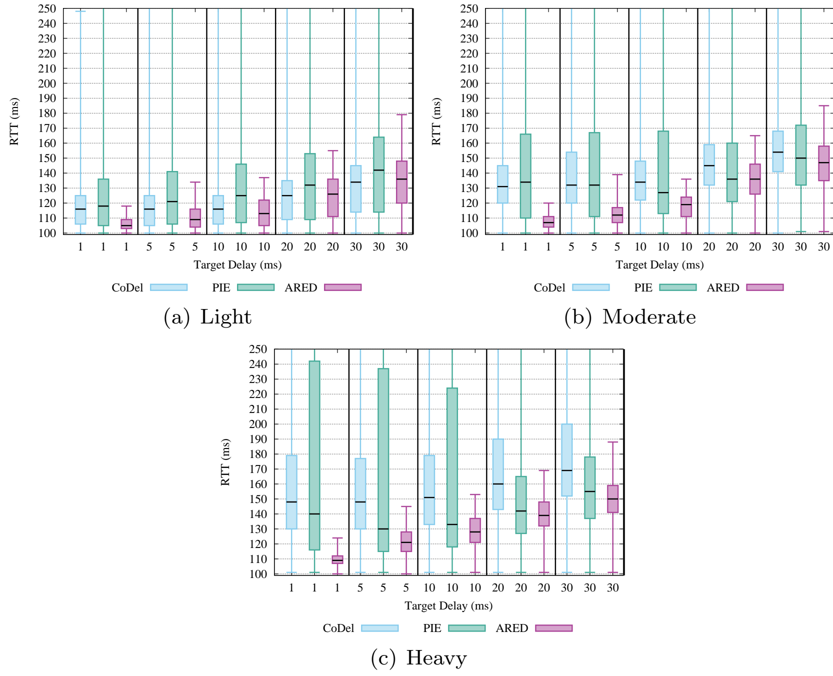
4.1 Parameter Sensitivity

In this section, we assess how the mechanisms under consideration fare when varying their parameter settings. Such AQMs have two key parameters, and their tuning affects their performance under different network conditions (see Section 2). We investigate how they behave in scenarios with different number of flows (i.e., different congestion levels) and RTT_{base} values, and for different parameter settings for $target_delay$ and the interval.

4.1.1 Target Delay

Figure 3 shows the per-packet RTT distributions of CoDel, PIE and Adaptive RED for lightly, moderately and heavily congested link scenarios respectively, and for various $target_delay$ values ranging from 1 to 30 ms (such range includes PIE's 20 ms and CoDel's 5 ms defaults). We define *lightly*, *moderately* and *heavily* congested link scenarios as when 4, 16 and 64 TCP flows associated with 4 sender/receiver pairs are sharing the bottleneck, respectively (i.e., 1, 4 or 16 flows per sender-receiver pair). Since RTT_{base} is set to 100 ms, the difference between values on whisker-box plots and 100 ms correspond to queuing delay at the bottleneck. We refer to this as "queuing delay" hereafter.

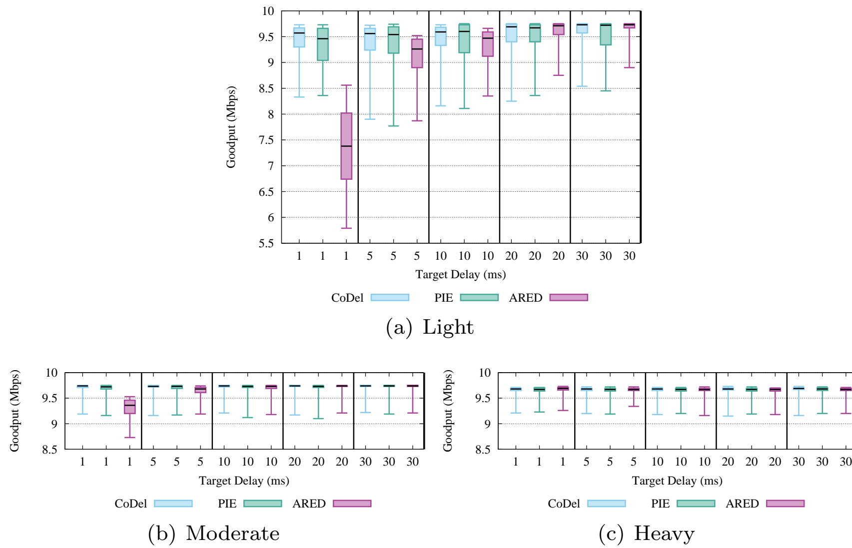
Here we can make several observations: CoDel and PIE's median, 10th and 90th percentiles of queuing delay increase proportionally to the level of congestion at the bottleneck, whereas Adaptive RED better controls the median latency at closer levels to $target_delay$ for all extents of congestion. While Adaptive RED's median and percentiles of queuing delay decrease proportionally to the decrease of $target_delay$, PIE and CoDel's median and percentiles don't exhibit a close correlation with the choice of $target_delay$ more specifically for higher congestion levels. It is also observable that in general, and more clearly under high congestion, PIE and CoDel show longer queuing delay distribution



(a) Light (b) Moderate

(c) Heavy

Figure 3: Per-packet RTT. Light, moderate and heavy congestion scenarios (4 senders and RTT_{base}=100 ms); bottom and top of whisker-box plots show 10th and 90th percentiles, respectively.



(a) Light

(b) Moderate (c) Heavy

Figure 4: TCP goodput at the bottleneck link (per 5-sec intervals). Light, Moderate and Heavy congestion scenarios (4 senders and RTT_{base}=100 ms); bottom and top of whisker-box plots show 10th and 90th percentiles, respectively.

tails (the difference between 10th and 90th percentiles), meaning more fluctuations in RTTs. CoDel's distribution tail length stays roughly equal for different target values within each congestion level scenario, while PIE's distribution tail tend to increase with decrease in target delay when the link becomes more congested. PIE in particular yields very large fluctuations in delay under load, for small values of $target_delay$ ($\leq$ 10 ms) (see Figure 3(c)). Adaptive RED, on the other hand, exhibits much shorter (and also decreases with decease in $target_delay$) distribution tails, meaning more controlled queuing delay.

Figure 4 shows the achieved goodput of AQMs for different congestion levels and target delays related to the scenarios in Figures 3(a), 3(b) and 3(c).

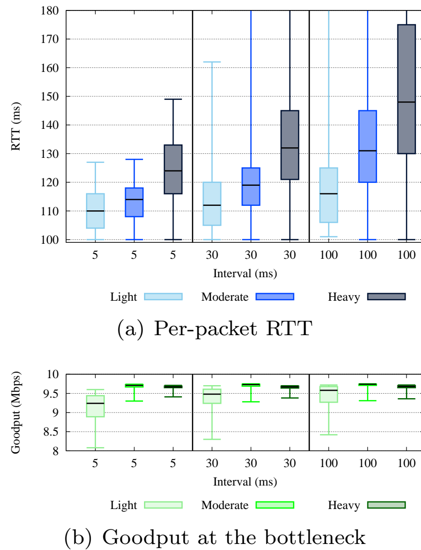
In the lightly congested scenario, Adaptive RED's median goodput drops to 7.38 Mbps when target delay is set to the extremely low value of 1 ms, otherwise it always stays between 9.26 and 9.73 Mbps for target delays from 5 to 30 ms. This is close to PIE and CoDel's median goodput which are around 9.46∼9.72 Mbps and 9.57∼9.73 Mbps for all target delay values respectively. Similarly, in the moderately congested scenario, Adaptive RED achieves almost exactly the same goodput as PIE and CoDel for all values of target delay except for the extreme low target of 1 ms. This similarity is even more obvious in the highly congested scenario where all AQMs yield ≈9.7 Mbps as the maximum achievable goodput on the bottleneck link.

4.1.2 Interval Time

CoDel and PIE use an update interval time that can be adjusted from userspace. ARED however uses a static fixed interval time of 500 ms, and therefore we only study CoDel and PIE in this part. As explained in Sections 2.1 and 2.2, CoDel enters its dropping state if the minimum queuing delay exceeds target delay for a duration of one *interval* while PIE uses an interval (t_{update}) to estimate the current queuing delay and update the drop probability. Again we stress the basic difference between CoDel and PIE's interval semantics; we vary these parameters but do not consider them to have a similar meaning. We consider three time-granularities at which AQMs might perform: fine (5 ms), medium (30 ms) and coarse (100 ms) relative to the RTT_{base} of 100 ms. This set of values incorporates CoDel's (100 ms) and PIE's (30 ms) default interval values as well.

CoDel's dropping-mode interval Figure 5 shows CoDel's performance when its dropping-mode interval is set to the above values and for different levels of congestion. While this parameter obviously controls a trade-off between delay and goodput, the figure indicates that a smaller value than the default one could be recommendable: for RTT_{base}=100 ms, using a smaller interval than the default 100 ms leads to significant improvement in median queuing delay (37.5%∼54.8%) while compromising 3.56% of median goodput only when congestion level is light (Figure 5(b)).

PIE's t_{update} interval Figure 6 shows PIE's performance for different values of t_{update}. As expected, PIE achieves better queuing delays with fine-grained intervals and better goodput with coarse-grained intervals. The queuing delay's trend is more obvious in the lightly congested scenario but becomes less predictable as the congestion level increases (Figure 6(a)). In the lightly congested



(a) Per-packet RTT

(b) Goodput at the bottleneck

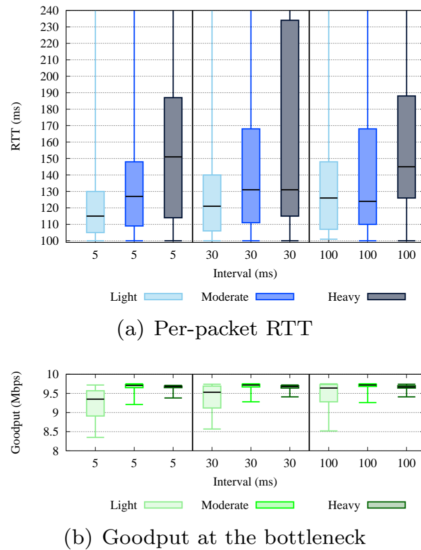
Figure 5: Per-packet RTT and bottleneck's per 5-sec TCP goodput. Varying CoDel's dropping-mode interval (4 senders, RTT_{base}=100 ms and $target_delay$=5 ms).

scenario, PIE's median queuing latency can be improved by 28.6% by reducing the default t_{update} from 30 ms to 5 ms while compromising only 1.86% of median goodput (Figure 6(b)).

4.1.3 Sensitivity to Base RTT

RTT_{base} plays an important role for the TCP performance coupled with an AQM mechanism. For instance, CoDel's default dropping-mode interval allows it to absorb a burst size of an interval worth of data, making it suitable for scenarios when RTT_{base} is reasonably close to 100 ms (or lower). The authors of [32] claim that excellent performance can be achieved for RTT_{base} in the range 10–300 ms, but no evaluation results are provided to sustain this assertion. In this part, we provide evaluation results for three common values of RTTs: 1) short RTTs (5 ms) common for intra-city/state transfers; 2) medium RTTs (100 ms) typical for continental and most inter-continental internet paths; 3) long RTTs (500 ms) common for few inter-continental paths, some developing countries and satellite links. We ran the experiments with 500 ms RTT_{base} for 300 sec to capture the long-term TCP *cwnd*'s sawtooth behavior while other experiments were ran for 60 sec as explained in Section 3.1.3. We consider the queuing delay as the difference between RTT_{base} and experienced per-packet RTT assuming the processing delay is negligible.

Figure 7 shows per-packet queuing delay for different RTT_{base} values. It can be observed that with most AQMs, better queuing delay can be achieved when RTT_{base} is large while the goodput level deteriorates in such scenarios (Figure 8). In these cases, the longer feedback loop causes a longer convergence time to the available capacity and larger *cwnd* sawtooth for large RTT_{base}, which in turn leads to the longer periods of link under-utilization when packet drops



(a) Per-packet RTT

(b) Goodput at the bottleneck

Figure 6: Per-packet RTT and bottleneck's per 5-sec TCP goodput. Varying PIE's t_{update} interval (4 senders, RTT_{base}=100 ms and $target_delay$=5 ms).

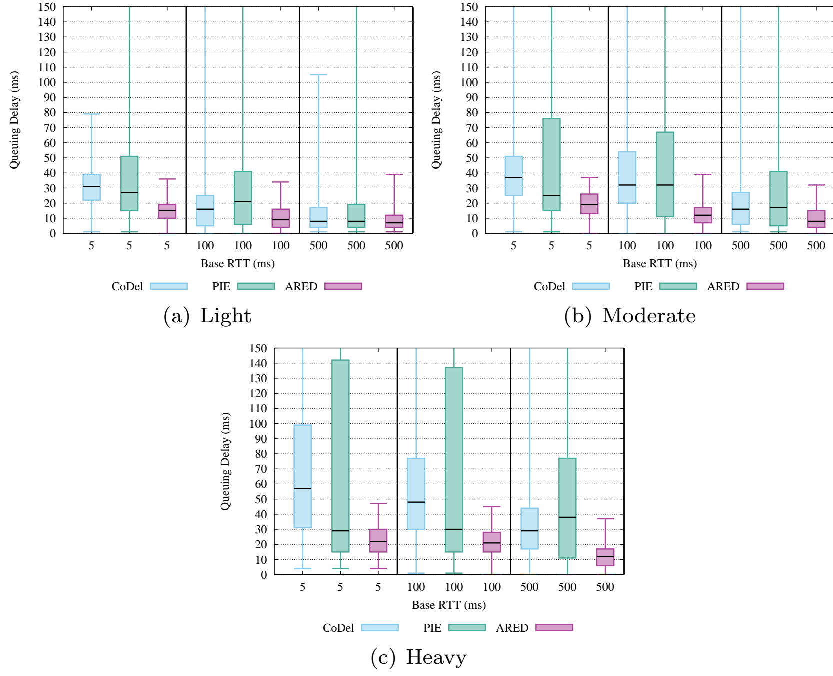
occur. It is also observable that for short RTT_{base}, ARED is able to achieve significantly better queuing delay than CoDel and PIE with almost exactly the same goodput level for all levels of congestion. For intermediate (100 ms) and longer (500 ms) RTT_{base} ARED performs significantly better than CoDel and PIE in terms of queuing delay while losing little goodput in light and moderate congestion scenarios respectively.

5 802.11 WLANs

Today's prevalent Wi-Fi networks remain a big challenge for controlling latency on the Internet. This is largely due to the multi-rate behavior of 802.11 WLANs, with bandwidth fluctuations that are caused by the rate adaptation mechanism that reacts to noise and to some extent to contention [26, 27]. It is long known – and confirmed by Figure 10 – that 802.11 rate adaptation can induce end-to-end latencies in the order of hundreds of milliseconds to multiple seconds, in particular in the presence of uplink traffic. Since much of the Wi-Fi network's latency is caused by channel access contention, it has – to the best of our knowledge – not yet being investigated to what extent AQMs can play a role in mitigating latency.

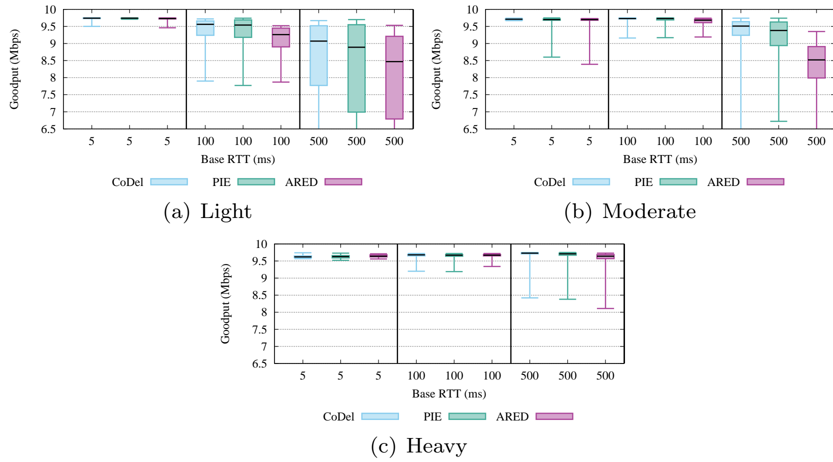
A bottleneck can emerge on an Access Point (AP)'s Wi-Fi interface, but it can also move to the wired interface of modems/CPEs behind the AP, depending on the traffic type and channel condition. While the location of a bottleneck may often be hard to predict, the most interesting scenario for this evaluation is a *public* Wi-Fi network common in airports, hotels and corporations where the provided Internet bandwidth supersedes the maximum Wi-Fi bandwidth – e.g. an 802.11g AP connected to a CPE subscribed for 100 Mbps.

In a *public* Wi-Fi scenario, the AP's wireless interface can become a bottle-



(a) Light
(b) Moderate
(c) Heavy

Figure 7: Per-packet queuing delay. Varying RTT_{base} at different congestion levels (4 senders and $target_delay$=5 ms).



(a) Light
(b) Moderate
(c) Heavy

Figure 8: TCP goodput at the bottleneck link (per 5-sec intervals). Varying RTT_{base} at different congestion levels (4 senders and $target_delay$=5 ms).

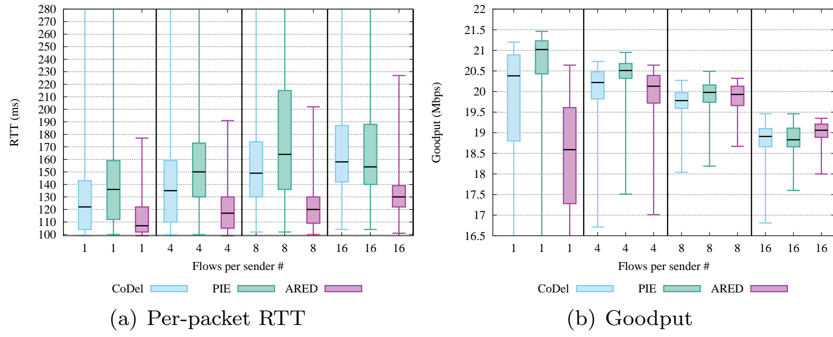
neck for downlink traffic. We studied the effect of deploying AQM on an AP's wireless interface using real-life experiments. Figure 9 shows the per-packet RTT and goodput of downlink traffic for different AQMs. Since the available bandwidth varies in Wi-Fi networks, we used the maximum achievable TCP throughput on an 802.11g in perfect transmission (27 Mbps) as a parameter to calculate ARED's *target_delay*. Figure 9(a) shows that ARED achieves significantly better latency than CoDel and PIE while it achieves less goodput (median of 18.6 Mbps versus 20.4 Mbps and 21 Mbps for CoDel and PIE) when the congestion level is light (Figure 9(b)). Goodput of ARED quickly improves as the congestion level increases.

This shows that, almost similar to their performance in wired networks, AQMs can control latency well on wireless interface when the dominant type of traffic is downlink. This is because in downlink scenarios, few stations are actively trying to access the channel while the AP is in charge of transmitting all data packets, resulting in the rate adaptation to rarely trigger the use of lower bit-rates when the channel condition is good and stable (the downlink and uplink Wi-Fi performance has already been thoroughly investigated in our previous works [27, 25]).

However, the presence of uplink traffic can lead to significantly high latencies in *public* WLANs. Here, the channel access latency becomes the major source of delay. Figure 10 shows the uplink performance in a mixed traffic scenario with an equal number of uploading and downloading flows. It can be observed that even in light congestion the median RTT increases to 440 ms$\sim$530 ms for all AQMs. This delay would be noticeable by the users of most interactive multimedia applications that might compete with such TCP flows. The median RTT increases to 1.15 sec and 1.18 sec for CoDel and ARED in the presence of moderate congestion, and it increases further as the congestion level increases, while PIE's controls the median RTT to stay at an almost constant level.

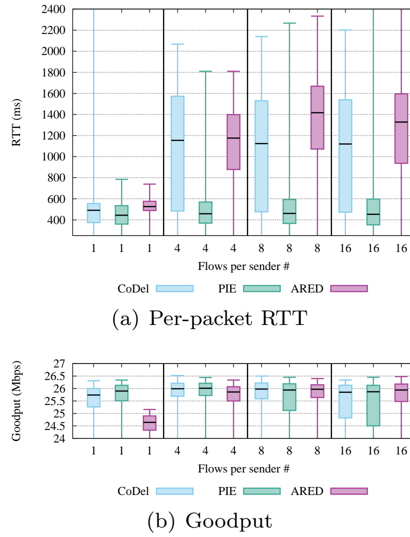
A detailed investigation of the packet traces revealed the origin of this significant increase of delay with CoDel and PIE: the downlink ACKs that are caused by the uplink traffic share the bottleneck with the data packets at the AQM queue (AP's wireless interface). The AQM queue is then backlogged with these data packets and ACKs, and frequent ACK drops by an AQM lead to excessive end-to-end delays for the uplink traffic in addition to an already high channel access delay that prolongs the ACK departures from the AQM queue. This causes the AQM to react further to the accumulated ACKs in the buffer, making TCP's feedback loop exceedingly long.

The RTT that is measured by the TCP uplink sender gets prolonged when ACKs at the head of the queue are dropped (CoDel). Until a timeout occurs (which, by itself, is affected by the duration and variation of the measured RTT), uplink data packets will be transmitted on the channel without the sender being notified about congestion until the AP finally gets the chance to deliver one of its ACKs (note that the AP has the same chance to access the channel as all other stations). In this regards PIE works best, as it is the AQM that least aggressively reacts to queue growth in our evaluations, as opposed to ARED which drops rather aggressively.



(a) Per-packet RTT (b) Goodput

Figure 9: Per-packet RTT and TCP goodput at the bottleneck link (per 5-sec intervals). 802.11 downlink traffic scenario for 4 senders, RTT_{base}=100 ms and $target_delay$=5 ms.



(a) Per-packet RTT

(b) Goodput

Figure 10: Uplink's Per-packet RTT and TCP goodput at the bottleneck link (per 5-sec intervals). 802.11 mixed traffic scenario for 4 senders, RTT_{base}=100 ms and $target_delay$=5 ms.

6 FQ_CoDel: Blending SFQ and AQM

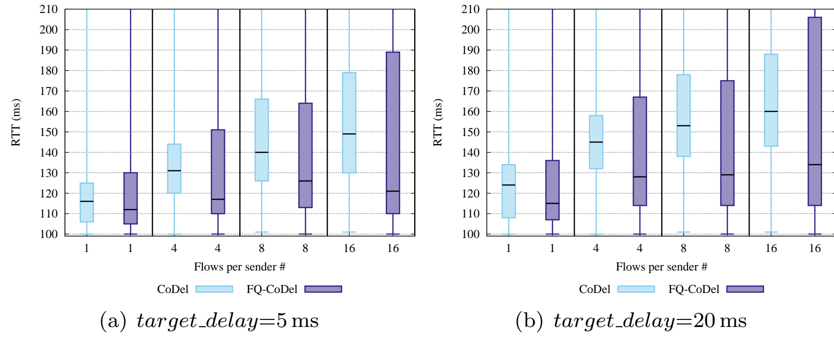
A combination of CoDel and stochastic fair queuing has been proposed [21] and implemented in Linux since kernel 3.5. In general, any AQM can be combined with fair queuing mechanisms such as stochastic fair queuing (SFQ) [31]. SFQ uses a hash function to classify the incoming packets into one of a set of flows, and each set will be directed to a sub-queue. Each sub-queue will be shared thus by one or more hashed TCP flows. To provide fair share of bandwidth between each sub-queue, SFQ dequeues the packets from the sub-queues in round-robin fashion while using FIFO/DropTail within each sub-queue.

SFQ improves performance by two means: 1) it provides fairness among sets of flows and 2) it helps with isolating flows in the presence of non-responsive traffic. In contrast to Fair Queuing (FQ) [15], the fairness provided by SFQ is not always 100% and it depends on the number of TCP flows hashed to each sub-queue. Nevertheless, SFQ is often favored over FQ because it can achieve a great degree of fairness without requiring large amounts of memory and processing, by keeping a limited-size hash table. In addition, SFQ ensures that the other flows don't get starved and improves their latency in the presence of flows that are non-responsive to congestion. These conditions make SFQ a favorable candidate in combination with any AQM at sub-queue level. FQ_CoDel is an attempt to mix the benefits of both SFQ and CoDel.

Figure 11 compares the per-packet RTT obtained by CoDel and FQ_CoDel when *target_delay* is set to 5 ms and 20 ms respectively, for different numbers of TCP flows. It can be noticed that with CoDel, median, 10th and 90th percentiles of RTT increase proportionally as the number of TCP flows increases (Figures 11(a) and 11(b)). On the other hand, FQ_Codel manages to keep the median and 10th percentiles of RTT significantly lower than CoDel (16%–66% reduction in 10th percentile queuing delay and 25%–57% reduction in median queuing delay, when *target_delay*=5 ms).

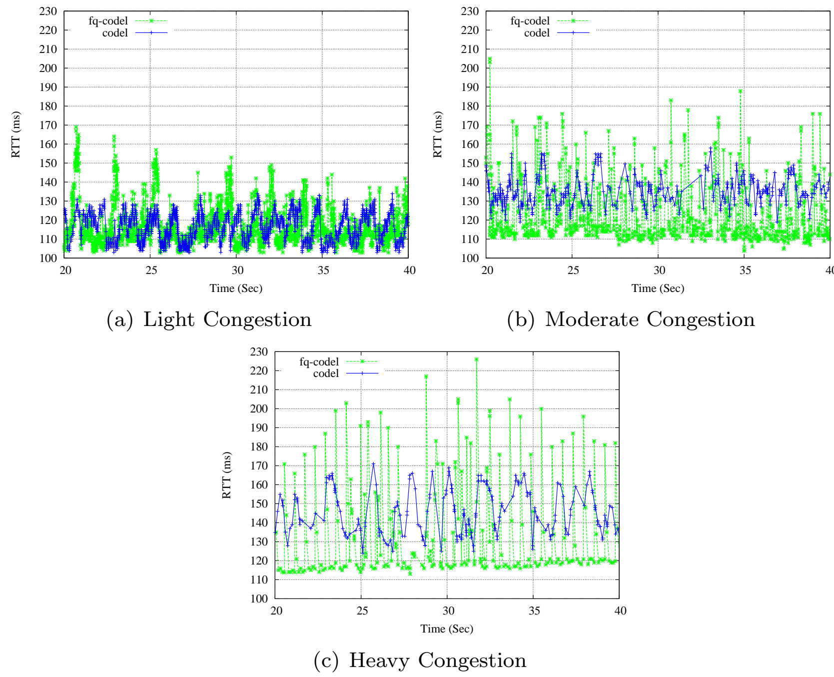
However, it can also be noted that the upper part of the queuing delay distribution tail (90th percentile) may sometimes slightly increase in some cases with FQ_CoDel (up to 13%–20% for *target_delay*=5 ms and 6%–20% for *target_delay*= 20 ms). This means lower queuing delay with FQ_CoDel in general with occasional higher delay spikes (i.e., higher jitter) compared to CoDel. We have investigated this in further detail by observing delay changes in time. Figure 12 shows the per-packet RTT of a sample TCP flow when competing with other flows in different congestion levels. It shows that FQ_CoDel achieves a lower latency than CoDel in general, at the expense of a larger jitter than with CoDel. This trend intensifies as the congestion level increases. This can be justified by the round-robin behavior of FQ_CoDel's dequeue function. Since FQ_CoDel serves every flow in round-robin order, packets belonging to the flow(s) that temporarily have the largest sub-queue will be served the last in the system if not being dropped, which leads to occasional higher spikes in per-packet latencies.

Overall, the use of stochastic fair queuing combined with an AQM should improve the *median* latency on the access links, especially with traffic flows that are non responsive to congestion (e.g., bursty TCP traffic, or CBR multimedia traffic with no congestion control). Studying SFQ and AQM's interaction with these types of traffic is out of this paper's scope. We aim to expand the set of SFQ-AQM mechanisms as future work.



(a) $target_delay$=5 ms (b) $target_delay$=20 ms

Figure 11: Per-packet RTT. Comparison between CoDel and FQ_CoDel. 4 senders and RTT_{base}=100 ms.



(a) Light Congestion (b) Moderate Congestion

(c) Heavy Congestion

Figure 12: Per-packet RTT samples of a single flow. 4 senders, $target_delay$=5 ms and RTT_{base}=100 ms.

Table 5: Marking rate (with ECN) to dropping rate (without ECN) ratio (4 senders).

Flows per sender	CoDel	PIE	Adaptive RED
1	1.256	1.156	6.621
4	1.356	1.106	3.465
8	1.719	1.591	4.303
16	6.117	6.569	3.873

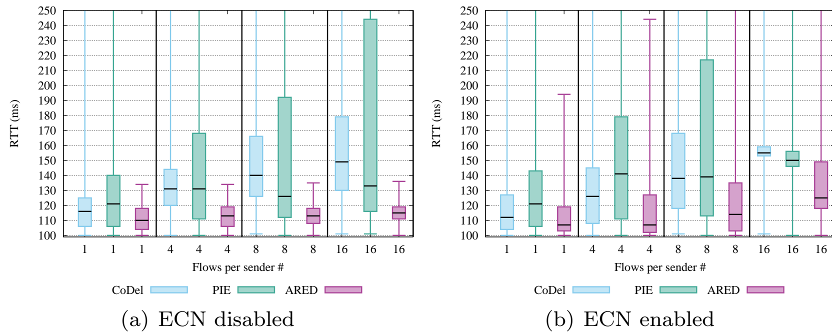
7 ECN

Explicit Congestion Notification (ECN) [18] allows routers using AQM to mark packets belonging to ECN-capable flows in case of congestion instead of dropping them. Without ECN, a TCP sender relies on receiving three DupACKs to infer that congestion has happened and a packet has supposedly been dropped. Since these three DupACKs are caused by packets succeeding the dropped packet, they must reach the receiver in the same RTT in order to avoid an extra RTT of delay before the sender can react. Moreover, since ECN reduces packet loss, it also reduces head-of-line (HOL) blocking delay that occurs in TCP when a receiver misses a data chunk of an otherwise consecutive block of data, or the significant delay that ensues when the dropped packet is a TCP SYN [28]. This gives ECN the potential to improve the user-perceived performance of many applications, e.g. by reducing page load times in web transfers.

Although ECN was proposed almost two decades ago and was perceived to coexist with RED [20], it has not been deployed or turned on in most of the Internet routers and therefore there has been little incentive for end-hosts to use ECN. The lack of AQM deployment has also been a discouraging factor for ECN deployment. Although most Linux implementations of the AQMs studied in this paper provide basic ECN support, we are not aware of any investigations of how ECN interacts with the AQMs that we consider in this paper.

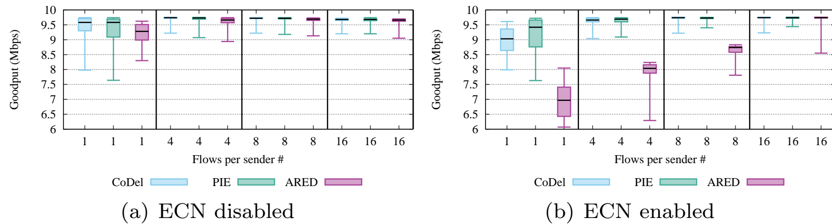
The common way to implement ECN marking (as we could confirm for all the AQMs investigated in this paper) conforms to this recommendation from RFC 3168 [37]: *"For a router, the CE codepoint of an ECN-Capable packet SHOULD only be set if the router would otherwise have dropped the packet as an indication of congestion to the end nodes."* Marking a packet instead of dropping it lets the packet stay in the queue until it is being served, which changes the metrics that AQMs use to mark/drop packets – typically the queue size or queuing delay. Hence, turning on ECN by simply marking packets that would otherwise have been dropped without changing anything else in the AQM behavior affects the AQM's marking/dropping decision process, which could lead to a higher marking/dropping probability for the same or other flows.

Figure 13 shows RTTs in a scenario with ECN-capable flows versus another scenario without ECN. In accordance with the description above, it shows a significant difference between the two scenarios. While ARED generally yields a better RTT than CoDel and PIE, it is quite badly affected by ECN as the congestion level increases. In a heavily congested scenario (16 flows per sender), also PIE and CoDel see a more pronounced impact, as its queues are constantly backlogged with CE-marked packets, leading to a very narrow RTT distribution tail (Figure 13(b)).



(a) ECN disabled (b) ECN enabled

Figure 13: Per-packet RTT for ECN-capable and non-ECN flows. 4 senders, $target_delay$=5 ms and RTT_{base}=100 ms.



(a) ECN disabled (b) ECN enabled

Figure 14: TCP goodput at the bottleneck link (per 5-sec intervals) for ECN-capable and non-ECN flows. 4 senders, $target_delay$=5 ms and RTT_{base}=100 ms.

Another result of AQM's interaction with ECN can be observed in Figure 14 where ARED's goodput drops down significantly when the congestion level *decreases*. This is due to the aggressive dropping/marking behavior of ARED in response to the increase in average queue size. ARED tends to CE-mark more packets than other AQMs when average queue size grows because of enqueuing already CE-marked packets. This leads to undershooting link-utilization in light to moderate congestion levels. Table 5 provides the marking rate to dropping rate ratio of all AQMs, showing that with ECN, ARED CE-marks approximately 3.5∼6.6 times more packets than it drops without ECN, while CoDel and PIE's marking to dropping ratio stays around 1.1∼1.6 for low to moderate congestion levels.

Based on the above observations we recommend that AQMs should modify their dropping/marking decision process to incorporate the impact that CE-marked packets on their measurements. This could perhaps be done by trying to exclude CE-marked packets from all calculations – e.g. ARED could exclude CE-marked packets from its average queue size calculation, and CoDel and PIE could update their queuing delay measurement to exclude the delays caused by CE-marked packets.

8 Concluding Remarks

Mark Twain must have thought of (A)RED when he said: “The reports of my death have been greatly exaggerated”. Indeed, in our evaluations, ARED only performed worse than CoDel or PIE in very few scenarios: when the number of flows on the bottleneck link was very small; in the public 802.11 WLAN scenario with both up- and downlink traffic, where the AP is connected to a broadband service with bandwidths that are higher than the WLAN bandwidth; with ECN, assuming a common implementation which only marks packets that it would otherwise drop, but does not change anything else about the AQM rules as ECN is turned on (which, as we have explained in Section 7, does not seem to be a good way to use ECN).

ARED outperformed both CoDel and PIE in *all other* situations, and most notably regarding the metric that these mechanisms strive to optimize: delay. We therefore consider it a promising direction for future work to improve ARED such that the issues above are solved, and, similar to FQ_CoDel, combine it with SFQ (SFQ_RED has already been implemented in the Linux kernel).

As explained in the outset and shown in Table 1, there are only very few published results on the performance of the AQMs discussed in this paper, and we therefore consider our work as a first fundamental step towards a better understanding of their performance. By its nature, such a first step has its limitations, and so this paper only focused on bulk TCP transfers. Our plans for future work therefore also includes more realistic traffic types, as well as simulations to stretch environment parameters to conditions that cannot be produced with our testbed.

9 Acknowledgement

This work was partly funded by the European Community under its 7th Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

References

[1] Bufferbloat. http://www.bufferbloat.net/.

[2] Cisco WRED Guide. http://www.cisco.com/en/US/docs/ios/qos/configuration/guide/config_wred.pdf.

[3] Ethtool. http://linux.die.net/man/8/ethtool.

[4] Hostapd. http://hostap.epitest.fi/hostapd/.

[5] IPFW. http://info.iet.unipi.it/~luigi/dummynet/.

[6] Minstrel. http://madwifi-project.org/browser/madwifi/trunk/ath_rate/minstrel/minstrel.txt.

[7] PIE Linux code (Cisco). ftp://ftpeng.cisco.com/pie/linux_code/.

[8] Synthetic Packet Pairs. http://caia.swin.edu.au/tools/spp/.

[9] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental), Feb. 2003.

[10] S. Athuraliya, S. Low, V. Li, and Q. Yin. REM: Active Queue Management. *Network, IEEE*, 15(3):48–53, 2001.

[11] E. Blanton and M. Allman. Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. RFC 3708 (Experimental), Feb. 2004.

[12] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, and Y. Nishida. A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP. RFC 6675 (Proposed Standard), Aug. 2012.

[13] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309 (Informational), Apr. 1998.

[14] E. Brosh, S. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne. The delay-friendliness of tcp for real-time traffic. *Networking, IEEE/ACM Transactions on*, 18(5):1478–1491, 2010.

[15] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Symposium Proceedings on Communications Architectures & Protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[16] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 43–56, New York, NY, USA, 2007. ACM.

[17] W.-c. Feng, K. G. Shin, D. D. Kandlur, and D. Saha. The BLUE Active Queue Management Algorithms. *IEEE/ACM Trans. Netw.*, 10(4):513–528, Aug. 2002.

[18] S. Floyd. TCP and Explicit Congestion Notification. *SIGCOMM Comput. Commun. Rev.*, 24(5):8–23, Oct. 1994.

[19] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Technical report, 2001.

[20] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, Aug. 1993.

[21] J. Gettys. fq_codel status. Presentation at the 87th IETF meeting.

[22] J. Gettys. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing*, 15(3):96, 2011.

[23] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Taht. Fighting the Bufferbloat: On the Coexistence of AQM and Low Priority Congestion Control. 2013.

[24] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1726–1734 vol.3, 2001.

[25] N. Khademi and M. Othman. Size-based and direction-based tcp fairness issues in ieee 802.11 wlans. *EURASIP J. Wirel. Commun. Netw.*, 2010:49:1–49:13, Apr. 2010.

[26] N. Khademi, M. Welzl, and R. L. Cigno. On the Uplink Performance of TCP in Multi-rate 802.11 WLANs. In *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part II*, NETWORKING'11, pages 368–378, Berlin, Heidelberg, 2011. Springer-Verlag.

[27] N. Khademi, M. Welzl, and S. Gjessing. Experimental Evaluation of TCP Performance in Multi-rate 802.11 WLANs. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–9, 2012.

[28] A. Kuzmanovic. The power of explicit congestion notification. *SIGCOMM Comput. Commun. Rev.*, 35(4):61–72, Aug. 2005.

[29] S. Liu, T. Basar, and R. Srikant. Exponential-RED: A Stabilizing AQM Scheme for Low- and High-speed TCP Protocols. *Networking, IEEE/ACM Transactions on*, 13(5):1068–1081, 2005.

[30] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[31] P. McKenney. Stochastic Fairness Queueing. In *INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE*, pages 733–740 vol.2, 1990.

[32] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 10(5):20:20–20:34, May 2012.

[33] K. Nichols and V. Jacobson. Controlled Delay Active Queue Management. Internet-Draft draft-nichols-tsvwg-codel-01.txt, IETF Secretariat, Feb. 2013.

[34] T. Ott, T. V. Lakshman, and L. Wong. Sred: stabilized red. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1346–1355 vol.3, 1999.

[35] R. Pan, B. Prabhakar, and K. Psounis. Choke - A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 942–951 vol.2, 2000.

[36] R. Pang. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. Internet-Draft draft-pan-tsvwg-pie.txt, IETF Secretariat, June 2013.

[37] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), Sept. 2001. Updated by RFCs 4301, 6040.

[38] L. Stewart, G. Armitage, and A. Huebner. Collateral damage: The impact of optimised tcp variants on real-time traffic latency in consumer broadband environments. In L. Fratta, H. Schulzrinne, Y. Takahashi, and O. Spaniol, editors, *NETWORKING 2009*, volume 5550 of *Lecture Notes in Computer Science*, pages 392–403. Springer Berlin Heidelberg, 2009.

[39] J. Sun, K.-T. Ko, G. Chen, S. Chan, and M. Zukerman. PD-RED: to Improve the Performance of RED. *Communications Letters, IEEE*, 7(8):406–408, 2003.

[40] G. White. A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks. Technical Report, CableLabs, Apr. 2013.

[41] G. White and J. Padden. Preliminary Study of CoDel AQM in a DOCSIS Network. Technical Report, CableLabs, Nov. 2012.