Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM

Naeem Khademi*, Michael Welzl*, Grenville Armitage†, Chamil Kulatunga‡
naeemk@ifi.uio.no, michawe@ifi.uio.no, garmitage@swin.edu.au, chamil@erg.abdn.ac.uk
David Ros§, Gorry Fairhurst‡, Stein Gjessing*, Sebastian Zander¶
dros@simula.no, gorry@erg.abdn.ac.uk, steing@ifi.uio.no, s.zander@murdoch.edu.au

Centre for Advanced Internet Architectures Technical Report 150710A
Swinburne University of Technology
Melbourne, Australia

***Abstract*—CoDel and PIE are recently proposed Active Queue Management (AQM) mechanisms that minimize the time packets spend enqueued at a bottleneck, instantiating shallow, 5 ms to 20 ms buffers with short-term packet burst tolerance. However, shallow buffering causes noticeable TCP performance degradation when a path's underlying round trip time (RTT) heads above 60 ms to 80 ms (not uncommon with cross-continental and inter-continental traffic). Using less-aggressive multiplicative backoffs is known to compensate for shallow bottleneck buffering. We propose ABE: "Alternative Backoff with ECN", which consists of enabling Explicit Congestion Notification (ECN) and letting individual TCP senders use a larger multiplicative decrease factor in reaction to ECN-marks from AQM-enabled bottlenecks. Using a mix of experiments, theory and simulations with standard NewReno and CUBIC flows, we show significant performance gains in lightly-multiplexed scenarios, without losing the delay-reduction benefits of deploying CoDel or PIE. ABE is a sender-side-only modification that can be deployed incrementally (requiring no flag-day) and offers a compelling reason to deploy and enable ECN across the Internet.**

I. Introduction

Recent years have seen increasing mainstream awareness of how critical low latency (delay) is to today's Internet and end users' quality of experience. Expectations are being increasingly driven by interactive applications such as Voice over IP (VoIP), online games, online trading [31] and even time-sensitive, transactional web-based shopping [26].

The delay experienced by any given packet is heavily influenced by routing choices (distance), link speeds (serialisation) and queuing (buffering at bottlenecks during periods of congestion). Increasing network speeds have reduced the relative contribution of serialisation, and therefore placed more focus on the size and management of bottleneck buffers.

A key influence on end user experience is the way bottleneck buffering interacts with capacity estimation techniques of common transport layers. The Internet's reliance on statistical multiplexing requires buffering to absorb transient periods of congestion. Loss-based TCP algorithms will fill a bottleneck's available buffer space before backing-off after the inevitable packet loss (congestion signal). When available buffering is significantly in excess of requirements the resulting queuing delay can far outweigh any delay contributed by distance [19].

Two complementary solutions have emerged—Active Queue Management (AQM) [3] and Explicit Congestion Notification (ECN) [41]. AQM schemes aim to provide earlier feedback about the onset of congestion, and thus reduce buffer filling during congestion. ECN delivers congestion feedback by adding new information to packets in transit, avoiding the detrimental side-effects of dropping packets to signal congestion [46].

The problem we face is that modern AQM schemes can interact badly with the traditional TCP response to congestion notification. A common rule-of-thumb is to allocate buffering at least equivalent to a path's intrinsic 'bandwidth delay product' (BDP) enabling TCP to achieve close to 100% path utilisation. Yet the design goal of AQM schemes, such as Controlled Delay

*Department of Informatics, University of Oslo, Norway
†Centre for Advanced Internet Architectures, Swinburne University of Technology, Australia
‡School of Engineering, University of Aberdeen, United Kingdom
§Simula Research Laboratory, Norway
¶School of Engineering and Information Technology, Murdoch University, Australia

(CoDel) [34], [35] and Proportional-Integral controller Enhanced (PIE) [37], [38], is to effectively instantiate a shallow bottleneck buffer with burst tolerance. So TCP performance suffers once a path's BDP exceeds the bottleneck AQM scheme's effective buffer size, whether congestion is signalled by packet loss or ECN.

A. ABE: "Alternative Backoff with ECN"

We propose a novel way of utilizing ECN that enables ECN deployment with a range of ECN marking behaviours and can be shown to continue to enable low latency and high throughput even for a High BDP path.

Alternative Backoff with ECN (ABE) can be summarised as follows:

- Upon packet loss, a TCP sender reduces its congestion window (*cwnd*) as usual (e.g., NewReno would reduce *cwnd* by 50%, CUBIC by 30%).
- Upon receipt of an ECN mark, a TCP sender reduces *cwnd* by *less* than the usual response for loss.

ABE is based on the intuition that meaningful ECN marks are generated by AQM schemes whose congestion indications are proactively aimed at keeping buffer occupancy low. An ABE sender thus compensates by backing off less, reducing the likelihood of the shallow bottleneck buffer draining to empty. Smaller backoffs will continue to occur as the AQM continues to react, ensuring the traffic does not build a long standing queue.

The idea of backing off less in response to an ECN mark is not new, as the authors of [29] proposed to use a larger multiplicative decrease factor in conjunction with a smaller additive increase factor for ECN in 2002. However [29] assumes the AQM on the bottleneck to be RED [17] (since this work was performed before the introduction of CoDel [34], [35] and PIE [37], [38]) which may not necessarily be deployed to instantiate a shallow buffer – e.g. not necessarily with low marking thresholds. The ABE mechanism differs from the work in [29] since it takes into consideration the default values of the state-of-the-art AQM mechanisms (CoDel and PIE) which aim to instantiate a shallow buffer and also solely relies on updating the multiplicative decrease factor.

ABE is a straightforward sender-side modification that may be deployed incrementally, and requires only that ECN is enabled in both end systems and AQM-managed bottlenecks along the path. An ABE sender's behaviour is simply regular TCP if either the destination or path do not support ECN. Finally, if an ABE sender's flow traverses a bottleneck whose AQM instantiates a relatively deep queue instead, then either packet losses or successive ECN marks will still ensure the sender continues to back off appropriately.

B. ECN past and future

RFC 3168 [41] defined ECN in 2001. It describes a simple marking method, replacing AQM dropping with ECN marking. Since then it has been widely implemented in hosts, but is not widely used [44]. This can partly be attributed to early experiences showing incorrect behaviour by a subset of middleboxes resulted in failure of ECN-enabled TCP connections to 8% of web servers tested in 2000 [36]. This reduced to $< 1\%$ of web servers tested in 2004 [32], yet in 2011 there were still a non-trivial number of paths mangling the ECN field in the IP header [9].

In 2014, extensive active measurements showed the majority of the top million web servers provided server-side ECN negotiation [44]. Wide client-side support for ECN-fallback means only 0.5% of websites suffer additional connection setup latency when fallback per RFC 3168 is correctly implemented.

Baker and Fairhurst [8] recently recommended deployment of AQM with ECN within the Internet. ECN-marking is now supported by default in CoDel and PIE within the Linux kernel and is enabled by default in router firmware, e.g., CeroWRT [1].

In light of increased interest in AQM, ABE provides a new incentive to deploy and enable ECN more widely. ABE is a new method to replace the simple TCP response defined in [41]. ECN support has matured to the point where there is little downside in doing so.

C. Paper structure

The rest of this paper is structured as follows. We motivate ABE in Section II, showing the inter-relationship between TCP backoff, bottleneck buffer size and path BDP, and demonstrating how TCP degrades over CoDel and PIE bottlenecks at plausible domestic and international round-trip times (RTTs). ABE is introduced in Section III. Section IV evaluates ABE using a combination of simulations and real-world experiments; both Linux and FreeBSD systems, considering CUBIC and NewReno congestion controllers, were used, which covers the majority of TCP variants actually deployed in the Internet. Related work is reviewed in Section V. Section VI concludes with a discussion of potential issues and ideas for future work.

II. Problem and Background

TCP congestion control [22] is based on an Additive Increase, Multiplicative Decrease (AIMD) mechanism

for controlling the congestion window. A decrease factor $\beta = 0.5$ has commonly been used in the congestion avoidance phase. However, the convergence of AIMD is due to its multiplicative behaviour, irrespective of the *value* of the backoff factor β; there are no major reasons why the value should be 0.5, and [22] states this choice was a heuristic, conservative one.

The choice of β may have implications in terms of link utilisation. To better understand the problem with AQMs enforcing small queues on long-RTT paths, we revisit how backoff factor, path characteristics and buffer space are intertwined.

A. TCP backoff, path characteristics and link utilisation

Consider a bottleneck link with capacity C and Drop-Tail buffer of size b, traversed by a single long-lived TCP flow. The BDP of the path, $C \times RTT$, will be denoted by δ. We assume that: (a) the TCP flow is operating in congestion avoidance, following a linear additive-increase function with parameter $\alpha = 1$, and multiplicative decrease factor $\beta \in (0, 1)$; (b) transmission is not limited by a small TCP receiver window. Bottleneck utilisation will be denoted as U.

Using an argument similar to [6, § 2] but with an arbitrary value for β, the only way to ensure $U = 1$ is with a minimum amount of buffering given by:

$$b \geq \delta \frac{1-\beta}{\beta}. \tag{1}$$

With $\beta = 0.5$, (1) yields the well-known rule-of-thumb for buffer sizing: $b \geq \delta$. The "sweet spot" corresponds to: $b = \delta$; larger b merely results in a longer queue.

Eq. (1) can be rewritten as

$$\beta \geq \frac{\delta}{b+\delta}, \tag{2}$$

where it becomes obvious that, for given path characteristics (δ), a shallower buffer (smaller b) requires a larger β in order for (2) to hold, i.e., to sustain $U = 1$.

Figure 1 illustrates the $U < 1$ case, i.e., when (1)-(2) do not hold. Buffering permits *cwnd* to grow up to a maximum value of $\delta + b$ before packet loss occurs.

Since we assume that $\alpha = 1$, *cwnd* increases by one unit per RTT from $\beta(b + \delta)$ to $b + \delta$, the sawtooth has a slope of 1 and duration between two loss events of $(1 - \beta) \cdot (b + \delta)$. The maximum amount of data D that can be sent in one cycle over the link is given by the area below δ:

$$D = \delta(1-\beta)(b+\delta) \tag{3}$$

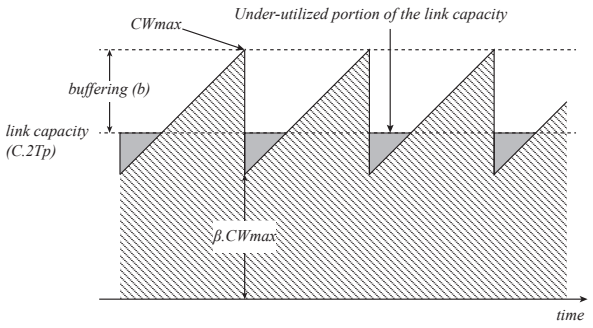


Fig. 1. TCP's sawtooth behaviour.

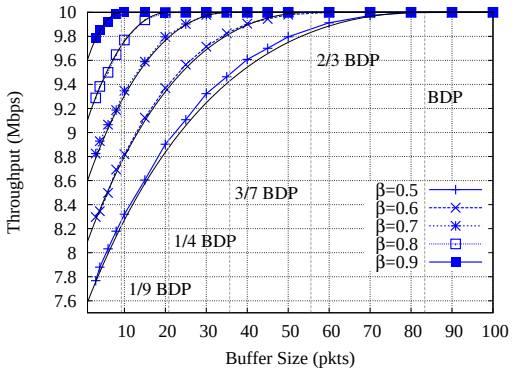


Fig. 2. Throughput (Single NewReno flow @ 10 Mbps, RTT=100 ms) (model and simulation).

Dark triangles correspond to periods where $cwnd < \delta$ (i.e., the TCP sender cannot "fill the pipe"). The buffer is empty, the link is under-utilised, and the amount of data *not* sent in a cycle corresponds to the area $\Delta_{cwnd<\delta}$ of a dark triangle:

$$\Delta_{cwnd<\delta} = \frac{(\delta - \beta(b+\delta))^2}{2} \tag{4}$$

Therefore, we can compute U as:

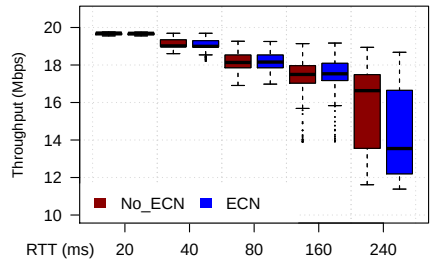
$$U = 1 - \frac{\Delta_{cwnd<\delta}}{D} = 1 - \frac{(\delta - \beta(b+\delta))^2}{2\delta(1-\beta)(b+\delta)} \tag{5}$$

Considering both the cases $U = 1$ and $U < 1$, the throughput, that is $\psi = C \times U$, is given by:

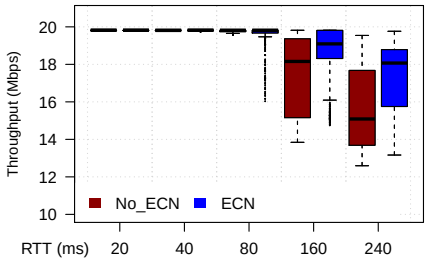
$$\psi = \begin{cases} C & \text{if } b \geq \delta \frac{1-\beta}{\beta}, \\ C\left(1 - \frac{(\delta-\beta(b+\delta))^2}{2\delta(1-\beta)(b+\delta)}\right) & \text{otherwise.} \end{cases} \tag{6}$$

For $b = 0$ and $\beta = 0.5$, this yields $\psi = \frac{3}{4}C$, in line with the model in [5]. The relation between utilisation, backoff factor and buffer size is illustrated in Figure 2, showing values given by both (6) and simulation results.

It also worth noting that (6) holds for all ranges of bottleneck link capacity (C) since the utilised fraction

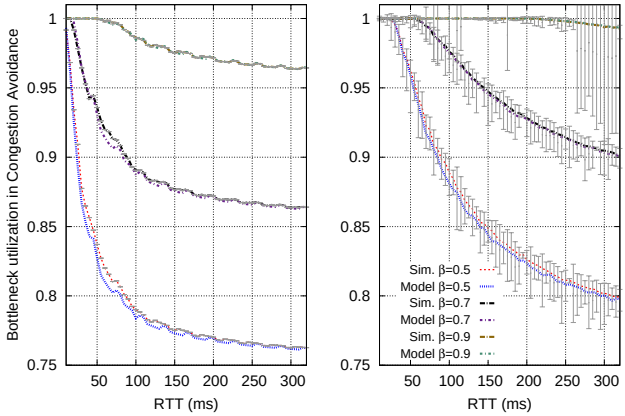


(a) CoDel throughput degrades at higher RTT.



(b) PIE throughput degrades at higher RTT.

Fig. 3. Performance through CoDel and PIE bottlenecks is significantly degraded over high-RTT paths (real-life test).



(a) CoDel (b) PIE

Fig. 4. Model vs. simulation (NewReno @ 20 Mbps); error bars not shown for CoDel because fluctuations were negligible.

of the link capacity (U) derived in (5) is independent of C.

B. Using AQM with a large path RTT

AQM mechanisms try to reduce the queuing delay incurred when DropTail buffers are sized according to (1) (i.e., $b \geq \delta$ when $\beta = 0.5$). CoDel's *target_delay* and PIE's T_{target} parameters control how aggressively they mark or drop packets based on the sojourn time. Small values of *target_delay* or T_{target} do reduce delay but also reduce TCP throughput when the RTT is large [24], [42].

The problem is illustrated by Figure 3, showing the throughput of a single CUBIC flow through a 20 Mbps bottleneck using CoDel or PIE, both with and without ECN, as a function of the path's intrinsic RTT[1]; simulated time was 90 s. Increasing path RTT has a detrimental impact on a CUBIC flow's throughput through both CoDel and PIE bottlenecks[2]. ECN on its own provides limited assistance; simply (naively) replacing drops by marks does not address the problem.

An explanation lies in the way CoDel and PIE both aim for a buffer of 5-20 ms (see § IV-A for the AQM parameters used). For a given bottleneck rate, this target equates to a shallow buffer (b) relative to the BDP (δ) of an entire path. As path RTT increases, the condition in (1) is increasingly violated and utilisation trends below 100%.

Figure 4 compares the median bottleneck utilisation generated both by simulation and Eq. (5) for a TCP NewReno flow in congestion avoidance over an AQM-controlled, 20 Mbps bottleneck with various RTTs and β. Error bars represent standard deviation.

The results are based on the average sending rate over five congestion cycles (saw-teeth) of a simulated flow that lasted 100 cycles. CoDel exhibited negligible variations over multiple runs (too small for error bars), while PIE's probabilistic nature created noticeable variations. At high RTTs with $\beta = 0.9$ PIE would sometimes trigger multiple consecutive backoffs immediately after slow start (dropping *cwnd* to 37% of δ when RTT=300 ms).

The model (Eq. (5)) was applied by tracking the queue length at the time of packet loss (i.e., the largest queue length that CoDel and PIE gave to the flow in each cycle) and taking the median of these values as b.

Figure 4 illustrates that the AQMs seem to roughly behave like a shallow DropTail queue: the utilisation drop with large RTTs is an expected outcome of the small queue granted to the TCP flow, and a larger β is needed to increase utilisation.

C. Selecting the right backoff factor

TCP will clearly benefit from larger β when confronted by shallow queues. But we cannot simply raise β without regard to network conditions. A TCP flow risks creating a significant standing queue if it uses high β on a path whose bottleneck queue is large.

[1] CUBIC's use of $\beta = 0.7$ does not alter the key point.

[2] Using DropTail, CUBIC achieves 100% throughput under the same circumstances, albeit with a well-known spike in queuing delay.

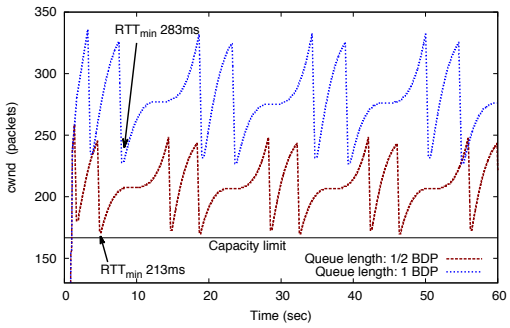


Fig. 5. CUBIC generates a standing queue.

Figure 5 illustrates the issue with Linux CUBIC (using $\beta = 0.7$ since Linux kernel 2.6.25 in 2008) over two sizes of DropTail queue. With a bottleneck rate of 10 Mbps and base RTT of 200 ms, a queue of 1/2×BDP never completely drains whilst a queue of 1×BDP experiences significant and sustained extra delay.

We therefore need a way for end systems to distinguish between the cases of a potentially large DropTail queue and the often much smaller average queue that is under control of an AQM mechanism.

III. Solution

We propose "Alternative Backoff with ECN" (ABE), a mechanism in which TCP backs off with a new value $\beta_{\text{ecn}} > \beta$ instead of the default β in response to an ECN mark, and the back-off in response to loss remains unchanged.

ABE differs from the recommendation in [41], which is that ECN marks should be treated the same way as congestion drops (i.e., same reaction to both at the TCP sender). An ABE sender's behaviour is based on the assumption that an ECN mark is only generated by an AQM mechanism and therefore the queue is likely to be significantly smaller than in case of loss (if this assumption is wrong, another ECN mark or a packet loss is likely to occur, causing TCP to back off further).

The analysis in § II focused on motivating the use of ABE during congestion avoidance, however, having a different response to marks at the end of slow start can also be beneficial, as we will see next.

A. ABE in Slow Start

TCP doubles *cwnd* every RTT during the slow start phase and enters the congestion avoidance phase after the first packet loss or mark [41] by halving *cwnd* (using β=0.5).[3] The rationale behind this is to fall back to the last *cwnd* that successfully transmitted all its data.

Depending on how a certain multiple of TCP's initial window is aligned with the path BDP and the queue length, slow start can exceed the available capacity with an overshoot that may be as large as almost one BDP or as small as one packet. While halving the rate is the only way to reliably terminate the overshoot of one flow within one RTT, it can create significant under-utilisation, depending on how large that overshoot was. This problem is more pronounced for short flows such as common web traffic which may terminate with a *cwnd* of just above 0.5×BDP, shortly after slow start. Recent trends [2] show a substantial increase in the length of short flows, increasing the probability that they terminate not during slow start but rather shortly after it, during the congestion avoidance phase, making the impact of the choice $\beta_{\text{ecn}} = \beta$ in slow start more profound (§ IV-D). Also, SPDY and HTTP/2 [10], [43] reuse existing connections for multiple objects, further increasing the probability of web flows to leave slow start.

This issue is depicted in Figure 6. The solid lines illustrate two single NewReno flows, with intrinsic RTTs of 160 ms and 240 ms, when $\beta_{\text{ecn}} = \beta = 0.5$ is used. The flow with RTT=240 ms suffers from under-utilisation: after slow-start overshoot, a small *cwnd* results in a slow increase during congestion avoidance. The flow with RTT=160 ms was luckier after slow start, but then the default β_{ecn} wastes capacity in congestion avoidance.

Dashed lines in Figure 6 show the same scenarios with $\beta_{\text{ecn}} = 0.9$. In both cases, *cwnd* is just below the BDP soon after slow start, which prevents the bottleneck from severe under-utilisation, reducing the completion time of short flows such as the ones common in web traffic. However, a large overshoot results in *cwnd* being reduced in multiple steps, which increases the latency over multiple RTTs after slow-start (however, the number of these RTTs is bounded by $log_{\beta_{\text{ecn}}}(0.5)$).

In § IV-D we evaluate the costs versus benefits of using a higher β_{ecn} at the end of slow start and demonstrate that its gains are significant.

IV. Evaluation

The experimental setup for both simulations and real-world tests is described in detail in Appendix A.

[3] This is the case for both NewReno and CUBIC. While the latest implementation of CUBIC includes Hystart [20] (a mechanism that tries to prevent slow start's overshoot by examining the delays of ACK arrivals), if an overshoot happens it causes *cwnd* to be halved.

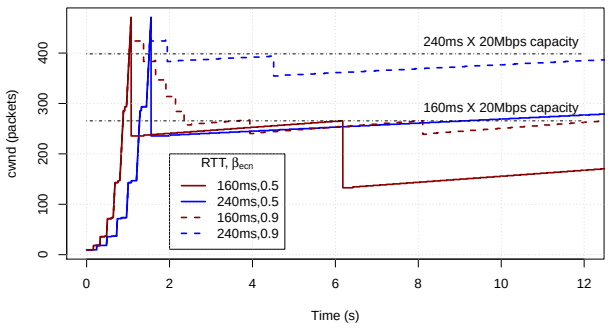


Fig. 6. The effect of overshoot at the end of slow-start for $\beta_{ecn} = \{0.5, 0.9\}$ @ 20 Mbps (real-life test).

A. AQM algorithms and parameters

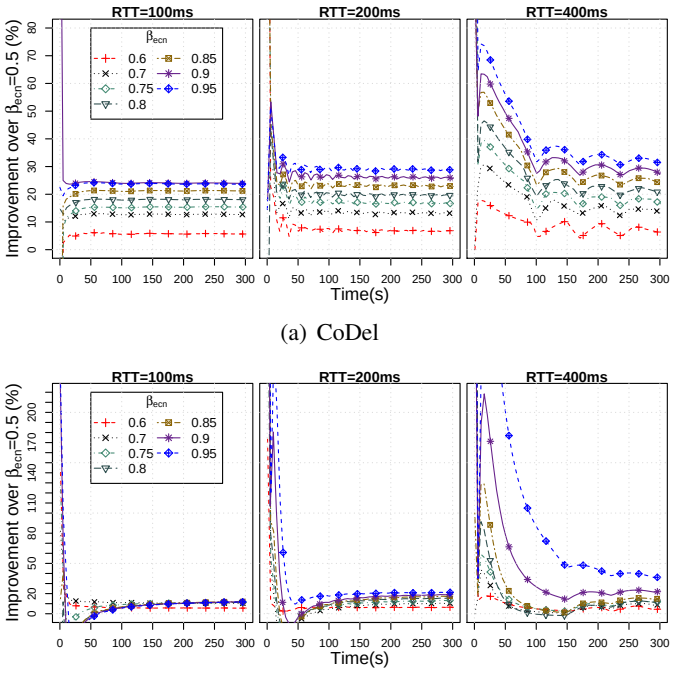
Both CoDel and PIE were used in all experiments. We did not run tests with FQ_CoDel [21] because it shares some of CoDel's issues, and it may introduce other issues that are beyond the scope of this paper (e.g., problems with classifying flows due to use of IPsec or tunnels such as VPNs).

For CoDel and PIE, in all experiments, we used the code that was shipped with Linux 3.17.4 and the ns-2 code that is publicly available, and applied the Linux default parameters everywhere. CoDel's *interval* and *target* were set to 100 ms and 5 ms, respectively, while PIE's *target* and *tupdate* were set to 20 ms and 30 ms. PIE's *alpha* is 0.125 and *beta* is 1.25. A parameter called *max_burst* is described in [39], with a default value of 150 ms. This parameter allows bursts of a given size to pass without being affected by PIE's operation, is 100 ms by default in the simulation code and is hard coded as such in the Linux implementation. *bytemode* was turned off in Linux and accordingly *queue_in_bytes_* was disabled in ns-2.

None of the specifications mention the maximum physical queue length – this is not a parameter of the AQM itself. In the Linux `man` pages, *limit* is 1000 packets.

In Linux kernel 3.17.4, packets on ECN-enabled flows are dropped rather than marked when PIE's drop/mark probability exceeds 10%, yet we could find no published literature providing a rationale for such behaviour. One might argue it provides a defense against non-responsive, ECN-enabled flows. However, here we recommend disabling it due to its detrimental impact on well-behaved ECN-enabled flows[4].

[4]See Appendix B for experiments supporting this choice.



(a) CoDel

(b) PIE

Fig. 7. Improvement over β_{ecn}=0.5 (bytes departed) for a single NewReno flow @ 10 Mbps.

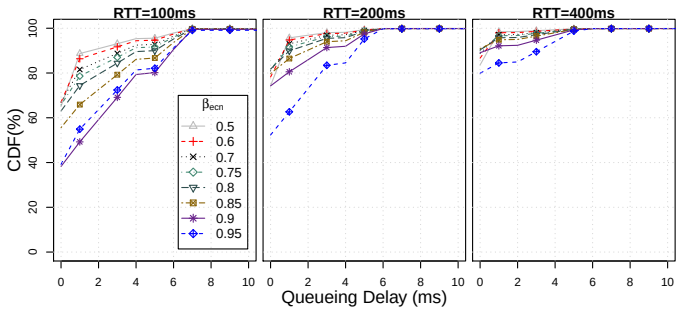
B. Latency vs. throughput for a single flow

Finding the right β_{ecn} value requires investigating a trade-off between latency and throughput. Generally, the larger the multiplication factor (β) the higher the throughput, but also the latency. However, the ECN congestion signal is a notification of the presence of an AQM mechanism on the bottleneck. The state-of-the-art AQM schemes already mark at low buffering thresholds, limiting the latency increase that can be caused by a larger than standard β.

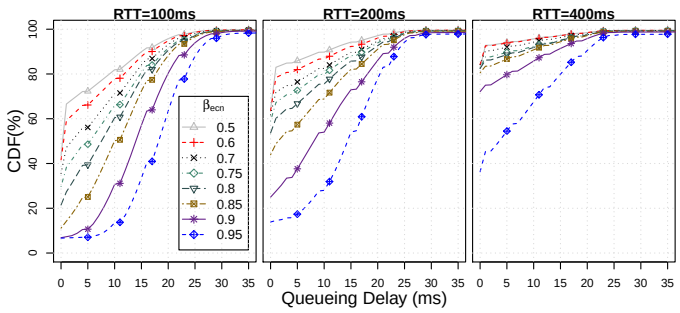
We investigated this trade-off for different β_{ecn} values for a single long-lived TCP NewReno flow using simulations with CoDel and PIE, and for a set of RTT values (100 ms, 200 ms and 400 ms) as presented in Figures 7 and 8.

Figure 7 shows that significant gains in throughput can be achieved as β_{ecn} increases with both PIE and CoDel, especially during the period right after slow-start and for larger RTTs. Moreover, a higher β_{ecn} improves the throughput of longer transfers compared to β_{ecn}=0.5, especially on large RTT paths—e.g. with an RTT of 400 ms, by 13%, 24% and 31% (CoDel) and 9%, 16% and 37% (PIE) for β_{ecn} values of 0.7, 0.85 and 0.95, respectively.

Figure 8 shows the CDF of queuing delay for the

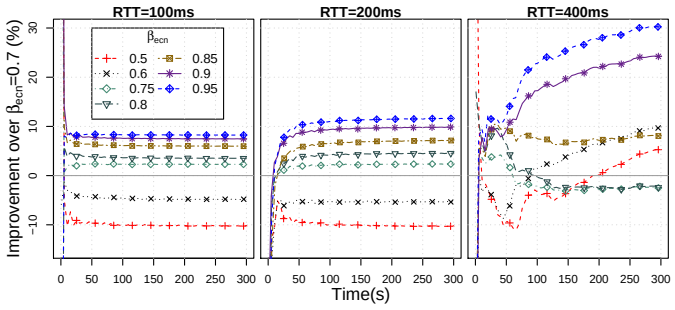


(a) CoDel

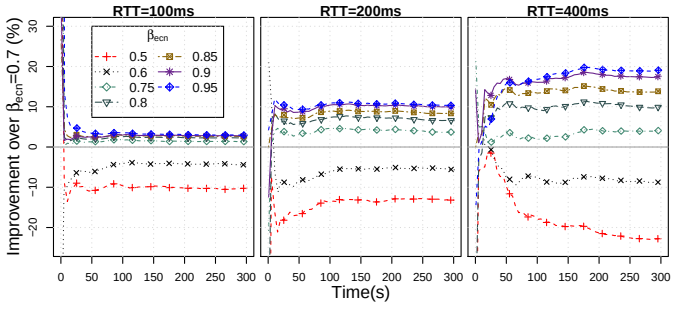


(b) PIE

Fig. 8. CDF of queuing delay for a single NewReno flow @ 10 Mbps.



(a) CoDel



(b) PIE

Fig. 9. Improvement over β_{ecn}=0.7 (bytes departed) for a single CUBIC flow @ 10 Mbps.

scenario of Figure 7. CoDel is able to keep queuing delay close to its default target delay parameter of 5 ms for all ranges of β_{ecn} (Figure 8(a)). In case of PIE (Figure 8(b)), the latency distribution becomes somewhat more heavy-tailed with increasing β_{ecn}, although the 90% percentile stays below PIE's default target of 20 ms for the range $0.5 \leq \beta_{\text{ecn}} \leq 0.85$. This is due to the burst-absorbing and random nature of PIE in contrast to the deterministic dropping policy of CoDel [25].

When RTT=100 ms, while CoDel adds 1 ms/4 ms to the median/90th percentile for β_{ecn}=0.95 compared to β_{ecn}=0.5, PIE adds 18 ms/11 ms to the median/90th percentile for the same scenario which is slightly significant relative to the intrinsic RTT. However, with β_{ecn} of 0.7 and 0.85, this can be reduced to 2 ms/3 ms and 10 ms/6 ms respectively.

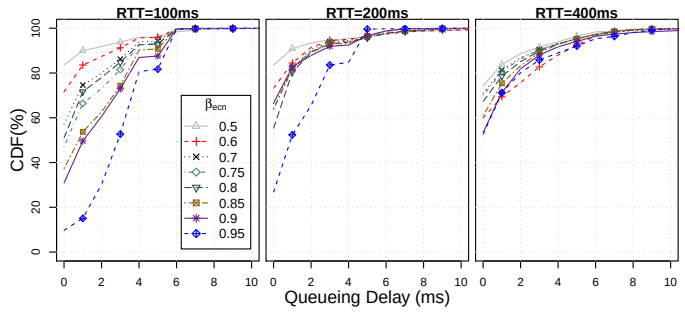
We can deduce that for a NewReno flow, a β_{ecn} value in the range $0.7 \leq \beta_{\text{ecn}} \leq 0.85$ would be an optimal trade-off between latency and throughput.

We also evaluated the above scenarios for a single CUBIC flow as presented in Figure 9 and Figure 10. Similar trends as for NewReno can be observed with CUBIC with regards to the achieved throughput gain for the period after slow-start and short flows for $0.7 \leq \beta_{\text{ecn}}$ although with a less profound gain due to the already aggressive nature of CUBIC's additive increase and β_{ecn}=0.7 (Figure 9). The trend for the queuing delay is also similar to that of NewReno (Figure 10).

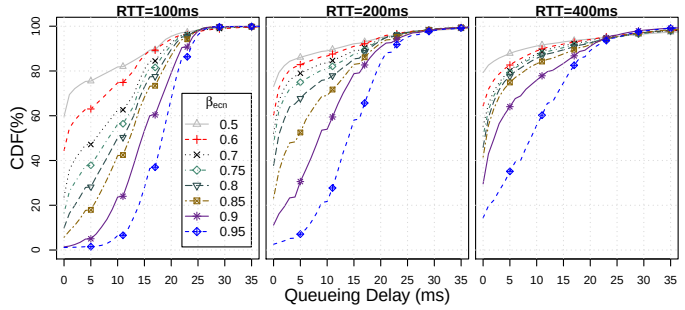
In terms of long-term throughput the exception is only for the scenario of CoDel on a path with large RTT where, surprisingly, an improvement of 5% and 10% can be gained by choosing β_{ecn} of 0.5 and 0.6 instead of the default 0.7 and using any value of $0.75 \leq \beta_{\text{ecn}} \leq 0.8$ will decrease the long-term throughput. However using a higher than default β_{ecn} will always lead to a better gain in throughput for all other RTT scenarios with CoDel (Figure 9(a)). In case of PIE, the trend in all scenarios is consistent with NewReno gaining 14% and 19% over the default β_{ecn} for 0.85 and 0.95 values respectively (Figure 9(b)).

Choosing these β_{ecn} values comes at the cost of an increase in the median/90th percentile of queuing delay equal to 6 ms/3 ms (PIE, β_{ecn}=0.85) and 12 ms/5 ms (PIE, β_{ecn}=0.95) and 1 ms/1 ms (CoDel, β_{ecn}=0.85) and 3 ms/2 ms (CoDel, β_{ecn}=0.95) when RTT=100 ms (Figure 10), which is negligible compared to the gain in throughput.

We can deduce that for a CUBIC flow, a β_{ecn} value in the range of $0.85 \leq \beta_{\text{ecn}} \leq 0.95$ would be an optimal trade-off between latency and throughput.

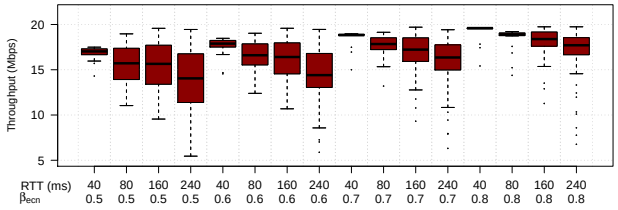


(a) CoDel

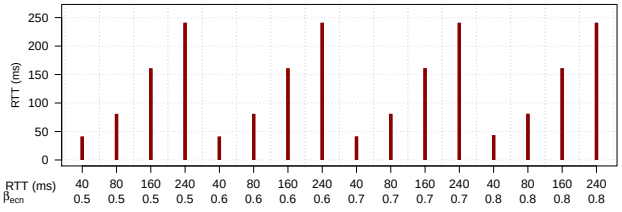


(b) PIE

Fig. 10. CDF of queuing delay for a single CUBIC flow on @ 10 Mbps.

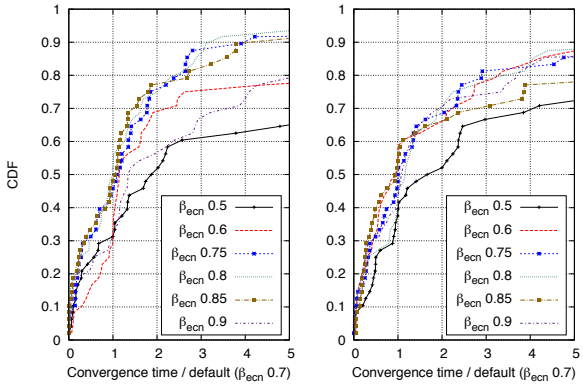


(a) Increased β_{ecn} improves throughput at higher RTTs



(b) Increased β_{ecn} causes insignificant change in median RTT

Fig. 11. Increasing β_{ecn} (single NewReno flow over CoDel bottleneck @ 20 Mbps).



(a) CoDel (b) PIE

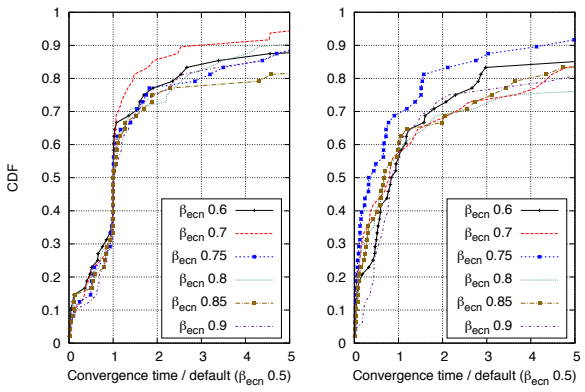
Fig. 12. Ratio of convergence time to a test with default β_{ecn}, CUBIC.

C. *Bulk transfers*

Figure 11(a) illustrates how increasing β_{ecn} (0.5, 0.6, 0.7 and 0.8) allows a FreeBSD NewReno flow to recover some of the throughput otherwise lost when running through a CoDel bottleneck at a large RTT. At the same time, Figure 11(b) shows that the resulting RTT is basically unaffected.

We also investigated how β_{ecn} affects the time that similar flows need to converge to a fair rate allocation. Intuitively, one might believe that using a larger β_{ecn} increases the time needed by a new flow to reach its fair share because already converged flows would give up less of their bandwidth when they hit the capacity limit. This is not necessarily correct. Simplifying, if we call the rates (or congestion windows) of two flows X and Y, perfect fairness is achieved if their ratio X/Y is 1. Then, if a congestion event affects both flows at the same time, this ratio becomes $\frac{X}{\beta_{\text{ecn}}} * \frac{\beta_{\text{ecn}}}{Y}$, rendering the value of β_{ecn} irrelevant for convergence to fairness. The ratio will obviously become different if only one flow is affected – but by this argument, convergence to fairness should mainly depend on how often each flow is affected, and not on how often congestion happens in total. In the absence of a per-flow scheduler like FQ_CoDel, this depends on how queuing dynamics affect flow synchronization, which may itself not depend heavily on the value of β_{ecn} used by all flows.

We measured convergence by calculating Jain's Fairness Index [23] of the cumulative number of bytes transferred per flow from the time a new flow entered, and noting the time it took until this index reached a threshold (which, for the graphs presented, was set to 0.95). We tested using two, four and ten flows with equal RTTs, starting with a delay of 30 seconds in between, and did not find any consistent improvement or disadvantage from changing β_{ecn}. Therefore we opted to explore the parameter space with a set of more controlled experiments of only two flows, using different capacity



(a) CoDel (b) PIE

Fig. 13. Ratio of convergence time to a test with default β_{ecn}, NewReno.

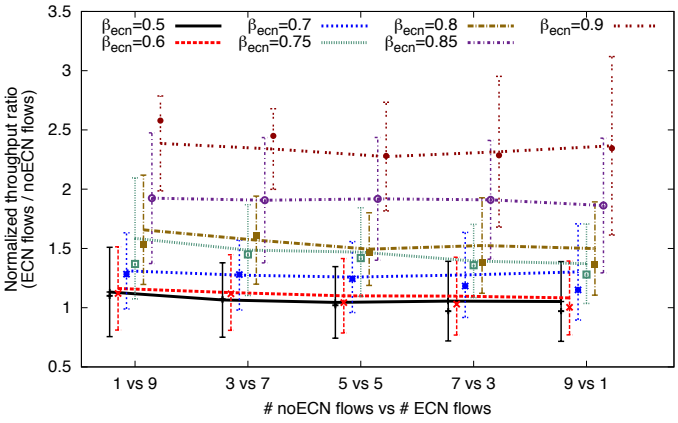


Fig. 14. Normalized throughput ratio between two groups of 10 flows @ 10 Mbps with a CoDel queue using several RTTs. Lines pass through the arithmetic mean, dots indicate the median.

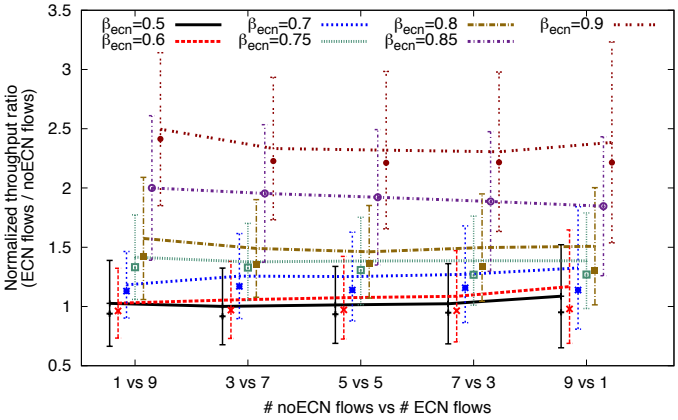


Fig. 15. Normalized throughput ratio between two groups of 10 flows @ 10 Mbps with a PIE queue using several RTTs. Lines pass through the arithmetic mean, dots indicate the median.

and RTT values for every test (all combinations of 1, 5, 10, 20, 40, 100 Mbps and 10, 20, 40, 60, 80, 160, 240, 320 ms). Figures 12 and 13 show the convergence time from 485 90-second two-flow experiments (97 for every beta value, including 0.5; we also tested values 0.55, 0.65, 0.75, 0.85 and 0.95 with similar results and omit these lines for clarity) each for NewReno and CUBIC. The simulation time was long enough to ensure convergence in all experiments.

Convergence time is shown as the ratio of the time taken in an experiment with similar conditions but the default behaviour of CUBIC ($\beta_{\text{ecn}} = 0.7$) and NewReno ($\beta_{\text{ecn}} = 0.5$), respectively, i.e. a value of 2 means that it took twice as long for a flow to converge than the default case. The distribution is heavy-tailed because there were some extreme outliers – not only very large times for the depicted β_{ecn} values, but also very small ones for $\beta_{\text{ecn}} = 0.5$. Some were caused by the initially explained scheduling (one flow was simply "unlucky" because it was perpetually affected by a congestion mark). In other cases, the default flow saw slow start ending at just the right value, creating an outlier for all non-default values of β_{ecn}.

Unsurprisingly, a significantly *lower* β_{ecn} than CUBIC's default consistently increases its convergence time. All β_{ecn} values can sometimes cause even faster convergence than the default, but for CUBIC no value of β_{ecn} could significantly improve it (generally, less than half of the cases converged faster than the default case). This is somewhat different with NewReno, where at least with PIE the value $\beta_{\text{ecn}} = 0.75$ seemed to have consistently improved convergence.

The point of this evaluation was to see if convergence time would be significantly (and consistently) harmed by changing β_{ecn}. Our results let us conclude that this is not the case.

Next, we investigated the long-term impact that flows with a large β_{ecn} value can have on legacy traffic that does not support ECN. To study the worst case, we tested CUBIC flows with various β_{ecn} values in competition with NewReno flows that do not support ECN. Figures 14 and 15 show the throughput ratio for various combinations of 10 flows across a 10 Mbps bottleneck link using CoDel and PIE. The plotted value was normalized by weighting it with the number of flows in each group. Each test lasted five minutes, of which we cut the first two seconds to remove slow start. We carried out every test 3 times, using RTTs of 20, 80, 160 and 320 ms; with the largest RTT, this duration included 133 TCP

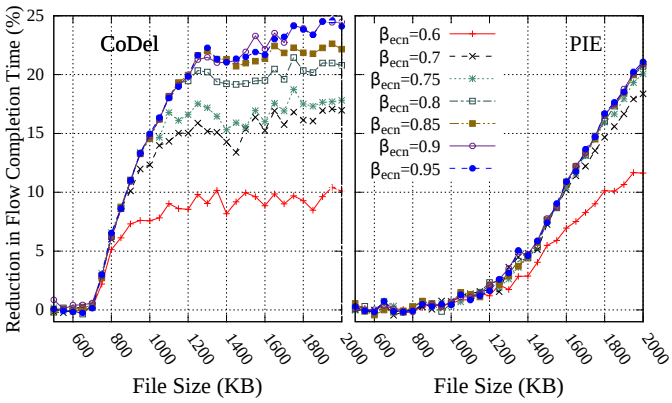


Fig. 16. Percentage reduction in FCT (compared to using $\beta_{\text{ecn}} = 0.5$) with different flow sizes over a 10 Mbps link with 100 ms base RTT

sawteeth.

The difference between CoDel and PIE is marginal, and generally a larger β_{ecn} value increased the throughput ratio, as could be expected. Our goal was to assess the potential "danger" of upgrading β_{ecn} on the Internet. The increase in the throughput ratio is limited to a factor of 3 for the already quite extreme value of β_{ecn}=0.9. The expected throughput difference is about double for β_{ecn}=0.85, and smaller values seem to be even closer together. This is CUBIC, where β_{ecn}=0.8 has been used on the Internet for several years and β_{ecn}=0.7 is now the default value in the Linux kernel.

D. *Short flows*

Much of the Internet's traffic consists of short flows (i.e. web traffic, e-mails etc). Recent trends show that the average web page size is increasing but most web flows still have a size of less than 2 MB [2]. So flows that do not terminate during slow-start may terminate shortly after, and the value of β_{ecn} at the end of slow-start will have an impact on the load time of the corresponding pages.

We simulated one TCP NewReno (Initial Window=3) flow over a 10 Mbps bottleneck with different flow sizes ranging from 500 KB to 2000 KB and a base RTT of 100 ms. Figure 16 shows the percentage reduction in Flow Completion Time (FCT) for different β_{ecn} values compared to using β_{ecn}=0.5. TCP flows started randomly within the first second of simulation time and the results were averaged over 1000 different seed values.

According to Figure 16, there is a reduction in FCT when the file size is larger than 700 KB for both PIE and CoDel. A flow with a size of less than 700 KB ends during or just after slow-start (for the selected link speed and RTT). Therefore changing β_{ecn} has no impact on the performance of very small files.

However, when a file is larger than 700 KB it may experience one or more cwnd reductions after slow-start overshoot. These consecutive reductions (up to one per RTT) slow down the sending rate to match the link speed. In Figure 17, PIE (target delay=20 ms) reduces the cwnd four times (from 448 packets to 29 packets) with β_{ecn}=0.5 (the cwnd is reduced suddenly as opposed to a smooth reduction which is shown in the figure with ns-2). This quickly drains the standing queue developed during the exponential growth. With β_{ecn}=0.9, a larger standing queue is created, and it takes only three reductions (up to 324 packets) until the file transfer is completed. When β_{ecn} is changed from 0.5 to 0.9 with CoDel (target delay=5 ms), it results a number of cwnd reductions from 2 (from 190 packets to 49 packets) to 8 (up to 84 packets).

In both schemes β_{ecn}=0.5 brings the cwnd much below the target link rate (the BDP of 83 packets), resulting in poor link utilisation with 100 ms RTT. This is particularly significant with large RTTs where the cwnd growth rate becomes very slow. Small reductions with a large β_{ecn} bring the cwnd more precisely to the bottleneck rate, resulting in better bottleneck utilisation. This yields a better FCT for short flows with a moderate file size.

In the case of Figure 17, it reduces FCT by 300 ms with PIE and by 570 ms with CoDel. However, the increased number of ECN-marked packets with β_{ecn}=0.9 due to a larger standing queue does not have a negative impact on FCT when used with an ECN-capable transport.

The improvement in FCT with CoDel is greater for file sizes below 2000 KB compared to PIE. CoDel overshoots less than PIE and marks earlier, creating a smaller standing queue. As a result, a large β_{ecn} causes less improvement in PIE than in CoDel.

We simulated the same scenario for different RTTs (100 ms, 200 ms and 400 ms) keeping the file size at 2000 KB. In Figure 18, PIE shows a better performance for the whole range of RTTs. PIE also randomises packet drops, making FCT improvement consistent. The improvement of PIE saturates at β_{ecn} near 0.75.

With CoDel, the performance is not consistent for all RTTs if flow is terminated after the slow-start. It shows the same number of cwnd reductions for all different seed values due to the deterministic behaviour of packet drops with a single flow. Because one reduction with β_{ecn}=0.5 is more significant than with β_{ecn}=0.9, results

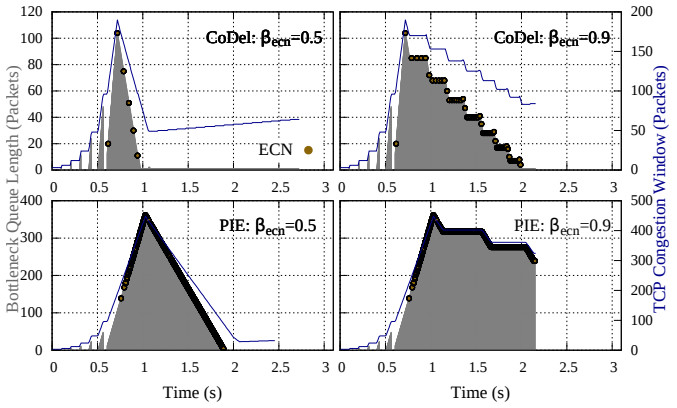


Fig. 17. Bottleneck queue and cwnd behaviour with one TCP NewReno flow. A large β_{ecn} results in precise cwnd adaptation (file size 2000 KB)

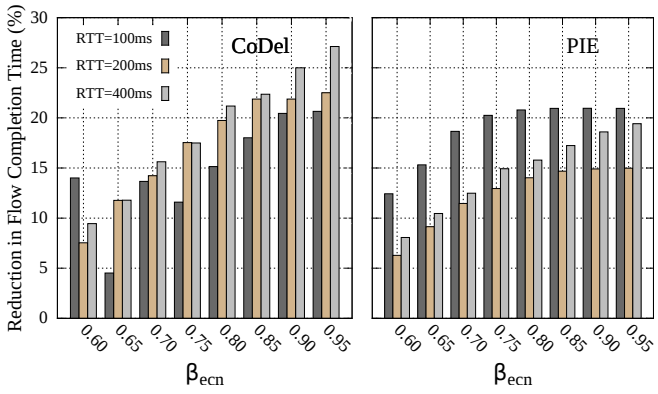


Fig. 18. FCT reduction performance with three (100 ms, 200 ms, 400 ms) different RTTs (file size 2000 KB)

are not consistent with CoDel depending on how many reductions have occurred and depending on the mismatch in cwnd under each RTT. Therefore to minimise this impact we randomized the RTT by +/-10 ms for each selected RTT in our simulations.

V. Related Work

Since its inception, ECN has attracted much interest in the research community. One obvious question that papers have tried to answer was: can we get "more bang for the bits"? Given that there are two ECN bits in the IP header, they can be re-defined to distinguish between multiple levels of congestion [15], [47], or used as defined, but marked with a different rule that must be known to the end hosts [45], [40]. LT-TCP [18] proposed using ECN to distinguish between random packet losses on wireless links and congestion; NF-TCP [7] suggests using it as a means to detect congestion early and back off more aggressively than standard TCP, thereby reducing the delay caused for competing applications. The potential applications are broad – yet, none of these methods were designed to interoperate with the currently deployed mix of standards-compliant ECN-capable and -incapable routers and end hosts such that it could be gradually introduced in the Internet.

"Congestion Exposure" (ConEx) [13], which is based on Re-ECN [12], uses traffic shapers to hold users accountable for the congestion they cause on the whole path (as opposed to giving most capacity to the host running the most aggressive congestion control). ConEx is gradually deployable, but needs significant changes to the infrastructure (hosts must be modified to give information to the network, traffic shapers must be deployed).

Some schemes were defined for use within a pre-configured operator domain. In "Pre-Congestion Notification" (PCN) [33], the ECN bits are used to inform DiffServ ingress routers of incipient congestion to better support admission control. Data Center TCP (DCTCP) [4] is another proposal to update TCP's ECN response. This has been shown to offer measurable benefits when network devices update their ECN marking behaviour. Only the software in hosts needs to change for DCTCP—deployment in routers can be achieved via an unusual configuration of RED to operate on the instantaneous queue length. However, in contrast to our proposal, this is only currently considered safe when all network devices along a path use the updated marking rules, and hence is only enabled by default for network paths with an RTT less than 10 ms [11], the primary reason why this has only been specified for Data Center usage.

A recent IETF presentation [28] discussed the possibility of gradually introducing a DCTCP-like scheme in the Internet; this is very close in spirit to the work presented in this paper. However, just like DCTCP, it requires a change to the receiver to more precisely feed back the number of ECN marks at a greater precision than today (i.e., more than one signal per RTT). Incorporating such feedback in TCP is ongoing IETF work [27]. Because marking the instantaneous queue length provides faster feedback, such an approach should theoretically be expected to perform better than the marking that we propose, provided it can also be combined with a sender behaviour that strikes the correct balance between efficiency across updated routers and compatibility across old routers. We regard both this paper and [28] as necessary investigations of the problem

space before a decision can be made regarding an update of the ECN standard.

Compared to DCTCP, our response to ECN is still based on a reduction each RTT, rather than per marked segment, but our method releases the network from the deployment pre-requisites of DCTCP. However, we note that a network device configured for DCTCP can also provide appropriate marking for our method, as can other router marking policies. This lack of sensitivity to marking removes a key obstacle to ECN deployment.

The idea of using values of $\beta \neq 0.5$ is at the basis of proposals for improving performance of long-lived TCP flows in high-speed networks. CUBIC is one example of such congestion controllers tailored to high-speed links, but other similar schemes can be found in the literature. For instance, H-TCP [30] uses $\beta \in (0.5, 0.8)$, and the value of β is adapted as a function of measured minimum and maximum RTTs. In a similar vein, after loss is experienced TCP Westwood [14] sets *cwnd* to the product of the estimated available capacity and the lowest observed RTT—thus, the equivalent β is variable and dependent on network conditions. High-Speed TCP (HSTCP) [16] adapts β in the range $(0.5, 1)$, with β taking higher values for larger *cwnd* above a threshold. In these proposals, the rationale behind the choice of a larger β is to allow for a faster recovery of *cwnd* after loss[5]; something that, with "standard" congestion control, can take many RTTs over paths with large BDP.

VI. Concluding remarks

This paper proposes and motivates Alternative Backoff with ECN (ABE), a simple change in a TCP sender's reaction to observing ECN congestion marks. We show that ABE can bring important performance improvements, with a low cost in terms of implementation effort. ABE achieves this performance using the ECN marking specified in RFC 3168 for routers/middleboxes and the TCP receiver. It also defaults to a conservative behaviour whenever ECN is not supported along the path, ensuring it is incrementally deployable.

As our results in § IV show, the choice of β_{ecn} is important but not overly critical, in the sense that ABE seems robust and offers performance improvements across a range of values of β_{ecn}. Setting $\beta_{\text{ecn}} \in [0.7, 0.85]$ for NewReno and $\beta_{\text{ecn}} \approx 0.85$ for CUBIC seems to provide reasonable trade-offs between latency, throughput and fairness across a wide range of scenarios.

[5] Adaptation of the window-increase parameter α is also used by these mechanisms for such purpose.

We expect methods like ABE to encourage usage of ECN, and as this usage increases, we believe the time will become ripe to revisit proposals for alternative ECN marking, along the lines of Section V. When use becomes widespread, router/middlebox manufacturers will have an incentive to implement these improved ECN specifications to further optimise performance. Hosts using ABE will then also be able to update their β_{ecn}.

Our study explored two well-known auto-tuning AQM methods with their standard pre-set parameter values (§ IV-A). To limit the problem space, we did not investigate the impact of changing the CoDel and PIE parameters. We also used the same β value in both slow start and congestion avoidance. Intuitively, one may think that using more aggressive AQM parameters for marking should advocate a higher β value. However, we expect there is a limit to how aggressively an AQM scheme can react, before even packets in a natural packet burst are punished by ECN marks or drop. Such questions concerning AQM tuning should be investigated in future work.

ECN-enabled routers need to protect themselves from overload by unresponsive traffic. RFC 3168 recommended routers to avoid high levels of ECN marking, assuming this was an indication of congestion collapse, and therefore encouraged drop under these conditions, as an appropriate response to overload. However, DCTCP and other modern AQM schemes use a low marking threshold, where this assumption becomes invalid, and can significantly impact performance (§ B). We therefore agree with the recommendation in [8], which encourages new research into more appropriate overload protection methods.

The use of ECN must be initiated by the TCP client. This makes ABE immediately applicable for use cases where a host updated with ABE initiates a connection and then transmits data, such as uploads to the Cloud, Web 2.0 applications, etc. In use cases where a client initiates a connection for data sent by a server, such as web surfing, ABE requires the server to be updated. An example of the expected performance with ECN, but without ABE is shown in Figure 3. Given such results and the increasing ability of routers to correctly pass ECN-enabled packets, no significant disadvantage is to be expected in this case, and the impact of ABE will increase as servers introduce support for it.

In conclusion, results have been presented from experiments, theory and simulation that show real benefit can be derived from updating TCP's ECN response, and we assert that this can provide a significant performance

incentive towards greater use of the ECN across the Internet.

VII. Acknowledgements

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.

Appendix

A. Simulations

All simulations presented in this paper are based on ns-2.35, using "Linux TCP" updated to use the pluggable congestion controls shipped in Linux kernel 3.17.4.

B. Experimental testbed

The results in Figures 3, 5, 6, 11, 12, 13, 14 and 15 were created using TEACUP [48] driving a physical testbed of source and destination hosts in two separate subnets either side of a PC-based router (the classic dumbell topology).

The 64-bit end hosts[6] are booted into FreeBSD 10.1-RELEASE or openSUSE 13.2 for NewReno and CUBIC trials respectively. Dynamic TCP state is logged with SIFTR under FreeBSD (event driven) and web10g under Linux (polled). We use iperf to source and sink traffic, and use tcpdump to capture the traffic actually seen 'on the wire' at each host. Modified FreeBSD NewReno and Linux CUBIC congestion control modules were implemented so that loss-triggered β and ECN-triggered β_{ecn} could be separately set and altered via sysctls.

The two subnets are linked by a 64-bit software router[7] based on openSUSE 13.2. The router provides a configurable rate-shaping bottleneck with CoDel or PIE AQM as required. The router also separately emulates configurable artificial delay in each direction.

Shaping, AQM and delay/loss emulation is on the router's egress NIC in each direction. The hierarchical token bucket (HTB) queuing discipline is used for rate-shaping, with the desired AQM queuing discipline (e.g. pfifo, codel) as leaf node. After the bottleneck stage, additional (configurable) fixed delay is emulated with netem. We loop traffic through pseudo interfaces (intermediate function block, or IFB, devices) to cleanly separate the rate-shaping/AQM from the delay/loss instantiated by netem. This ensures the bottleneck buffering occurs under the control of the selected AQM.

Linux kernel 3.17.4 is used on all hosts and the router. To get high timer precision for queuing and delay emulation the router's kernel is patched and recompiled to tick at 10000Hz.

Each PC has an additional NIC attached to an independent control network. A separate 'command and control' host uses ssh over this control network to coordinate and launch repeated trials. This includes configuration of source and destination hosts (enabling or disabling ECN, disabling of hardware segmentation offloading, setting specific TCP congestion control algorithms and β_{ecn}, etc), configuration of the bottleneck (rates, path delays, AQM, and ECN on or off), initiation of trials (launching one or more instances of iperf at particular times with particular iperf parameters) and subsequent data retrieval from participating machines.

A key issue is that PIE's average drop/mark probability can be driven high for reasons that do not indicate a high level of congestion or misbehavior of a non-responsive TCP flow in the short term. We explore this with a simulation of a PIE bottleneck carrying 20 long-lived TCP flows running for 300 s (all started within the first second of simulation time) and a short flow starting after 50 s with different values of β_{ecn}.

We turned off PIE's default drop behaviour for Figure 19(a), which shows that PIE's drop/mark probability can frequently rise significantly above 10% for $\beta_{\text{ecn}} > 0.7$ flows. We further observed that PIE's drop/mark probability would even rise above 10% for $\beta_{\text{ecn}} = 0.5$ flows during normal slow-start. Yet queuing delays remain bounded, as expected of a PIE bottleneck.

Figure 19(b) shows the same experiments with default behaviour re-enabled. PIE's dropping probability is controlled under 10% but the queuing delay is increased with larger β_{ecn}.

We experience a large number of CE packets with large β_{ecn} in the case of Figure 19(a) and counting them in the PIE algorithm increases the marking probability. Hence PIE creates a low queuing delay with responsive TCP flows. In the case of Figure 19(b), large β_{ecn} results in dropping more packets and controlling the probability. Use of large β_{ecn} with less that 10% marking probability increases queuing delay.

It seems clear that a drop/mark probability of 10% is far too low a threshold to begin dropping ECN-enabled

[6] HP Compaq dc7800, 4GB RAM, 2.33GHz Intel® Core™2 Duo CPU, Intel 82574L Gigabit NIC for test traffic and 82566DM-2 Gigabit NIC for control traffic

[7] Supermicro X8STi, 4GB RAM, 2.80GHz Intel® Core™i7 CPU, 2 x Intel 82576 Gigabit NICs for test traffic and a 82574L Gigabit NIC for control traffic

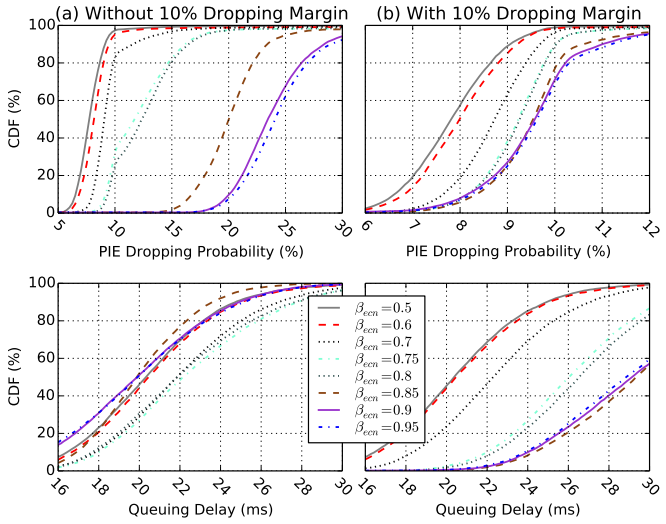


Fig. 19. PIE without and with 10% dropping threshold (20 long-lived TCP flows).

packets. It is detrimental to short flows (where slow start is a significant portion of a flow's overall lifetime) and to flows using higher β_{ecn} (such as flows originating from ABE-enabled senders), and appears to provide little benefit. Therefore, for the rest of our PIE experiments we disabled this default behaviour.

REFERENCES

[1] CeroWrt Project. http://www.bufferbloat.net/projects/cerowrt.

[2] HTTP Archive. http://httparchive.org/trends.php.

[3] R. Adams. Active Queue Management: A Survey. *IEEE Communications Surveys and Tutorials*, 15(3):1425–1476, 2013.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proc. ACM SIGCOMM*, pages 63–74, New Delhi, India, 2010.

[5] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel TCP Sockets: Simple Model, Throughput and Validation. In *Proc. IEEE INFOCOM*, pages 1 –12, Apr 2006.

[6] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proc. ACM SIGCOMM*, pages 281–292, Portland (OR), USA, Sep 2004.

[7] M. Arumaithurai, X. Fu, and K. Ramakrishnan. NF-TCP: Network Friendly TCP. In *17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6, 2010.

[8] F. Baker (ed.) and G. Fairhurst (ed.). IETF recommendations regarding Active Queue Management. Internet Draft draft-ietf-aqm-recommendation, work in progress, Jan 2015.

[9] S. Bauer, R. Beverly, and A. Berger. Measuring the State of ECN Readiness in Servers, Clients,and Routers. In *Proc. ACM IMC*, pages 171–180, 2011.

[10] M. Belshe and R. Peon. SPDY Protocol. Internet Draft draft-mbelshe-httpbis-spdy, work in progress, Feb 2012.

[11] S. Bensley, L. Eggert, and D. Thaler. Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters. Internet-Draft draft-bensley-tcpm-dctcp, Feb 2014.

[12] B. Briscoe, A. Jacquet, C. D. Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing Congestion Response in an Internetwork Using Re-Feedback. *Proc. ACM SIGCOMM, Computer Communication Review*, 35(4):277–288, Aug 2005.

[13] B. Briscoe, R. Woundy, and A. Cooper. Congestion Exposure (ConEx) Concepts and Use Cases. RFC 6789 (Informational), Dec 2012.

[14] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of ACM MOBICOM*, pages 287–297, Rome, Jul 2001.

[15] A. Durresi, L. Barolli, R. Jain, and M. Takizawa. Congestion Control Using Multilevel Explicit Congestion Notification. *Journal of Information Processing Society of Japan*, 48(2):514–526, Feb 2007.

[16] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec 2003.

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

[18] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle. Loss-Tolerant TCP (LT-TCP): Implementation and Experimental Evaluation. In *Military Communications Conference - MILCOM*, pages 1–6, 2012.

[19] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11):40:40–40:54, Nov 2011.

[20] S. Ha and I. Rhee. Taming the Elephants: New TCP Slow Start. *Computer Networks*, 55(9):2092–2110, Jun 2011.

[21] T. Hoeiland-Joergensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet. FlowQueue-Codel. Internet Draft draft-ietf-aqm-fq-codel, work in progress, Dec 2014.

[22] V. Jacobson. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM, pages 314–329, New York, NY, USA, 1988. ACM.

[23] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical report TR-301, DEC Research, Sep 1984.

[24] N. Khademi, D. Ros, and M. Welzl. The New AQM Kids on the Block: Much Ado about Nothing? Technical Report 434, University of Oslo, Dept. of Informatics, Oct 2013.

[25] N. Khademi, D. Ros, and M. Welzl. The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 85–90, April 2014.

[26] R. Kohavi and R. Longbotham. Online experiments: Lessons learned. *IEEE Computer Magazine*, 40(9):103–105, 2007.

[27] M. Kuehlewind and R. Scheffenegger. Problem Statement and Requirements for a More Accurate ECN Feedback. Internet-Draft (work in progress) draft-ietf-tcpm-accecn-reqs-04, Oct 2013.

[28] M. Kuehlewind, D. Wagner, J. M. R. Espinosa, and B. Briscoe. Immediate ECN. Presentation at the 88th IETF meeting.

[29] M. Kwon and S. Fahmy. TCP Increase/Decrease Behavior with Explicit Congestion Notification (ECN). In *IEEE ICC*, New York, New York, USA, May 2002.

[30] D. Leith and R. Shorten. H-TCP: TCP for High-speed and Long-distance Networks. In *Proceedings of PFLDnet 2004*, Argonne (IL), USA, Feb 2004.

[31] J. Loveless, S. Stoikov, and R. Waeber. Online Algorithms in High-frequency Trading. *Commun. ACM*, 56(10):50–56, Oct 2013.

[32] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Computer Communication Review*, 35(2):37–52, Apr 2005.

[33] M. Menth, B. Briscoe, and T. Tsou. Precongestion Notification: New QoS Support for Differentiated Services IP Networks. *Communications Magazine, IEEE*, 50(3):94–103, 2012.

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

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

[36] J. Padhye and S. Floyd. Identifying the TCP Behavior of Web Servers. In *In ACM SIGCOMM*, 2000.

[37] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *HPSR*, pages 148–155. IEEE, 2013.

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

[39] R. Pang, P. Natarajan, F. Baker, B. VerSteeg, M. S. Prabhu, C. Piglione, V. Subramanian, and G. White. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. Internet-Draft draft-ietf-aqm-pie-00, Oct 2014.

[40] I. A. Qazi, L. L. H. Andrew, and T. Znati. Congestion Control with Multipacket Feedback. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2012.

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

[42] J. Schwardmann, D. Wagner, and M. Kühlewind. Evaluation of ARED, CoDel and PIE. *20th Eunice Open European Summer School and Conference*, 2014.

[43] B. Thomas, R. Jurdak, and I. Atkinson. SPDYing up the Web. *Commun. of ACM*, 55(12):64–73, Dec 2012.

[44] B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffenegger. Enabling Internet-wide Deployment of Explicit Congestion Notification. In *Proceedings of the 2015 Passive and Active Measurement Conference*, New York, 03/2015 2015.

[45] N. Vasic, S. Kuntimaddi, and D. Kostic. One Bit is Enough: A Framework for Deploying Explicit Feedback Congestion Control Protocols. In *First International Communication Systems and Networks and Workshops (COMSNETS)*, pages 1–9, 2009.

[46] M. Welzl and G. Fairhurst. The Benefits and Pitfalls of Using Explicit Congestion Notification (ECN). Internet-Draft draft-ietf-aqm-ecn-benefits, Oct 2014.

[47] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One More Bit is Enough. *IEEE/ACM Trans. Netw.*, 16(6):1281–1294, Dec 2008.

[48] S. Zander and G. Armitage. TEACUP v1.0 - A System for Automated TCP Testbed Experiments. CAIA Technical Report, http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf, 29 May 2015.